Single Chip Hardware Support for Rasterization and Texture Mapping

Hans-Josef Ackermann
Fraunhofer Institute for Computer Graphics
Darmstadt, Germany

Abstract

Today's interactive 3D-applications on PCs demand efficient hardware support for functionality, e.g. shading and texture mapping. In this paper, I present an ASIC that integrates most of the 3D-related functionality defined in Intel's de-facto standard 3DR. As the chip was designed for real time environmental simulation systems, the main focus has been on texture mapping, which provides the most natural appearance at a moderate effort level. To avoid artifacts during texture mapping, the chip performs bi- or tri-linear blending on a MIPmap structure. Texture addresses are calculated perspective correct. A crucial problem concerning the tri-linear blending is the necessary data bandwidth between ASIC and the texture buffer. Therefore, I discuss several memory types and architectures for the texture buffer depending on performance, price and board space requirements. A short overview of different system architectures using the ASIC concludes the paper.

1 Introduction

That which just recently occurred was foreseen years ago: After a continuous increase in computational power and system resources, the PC has become a platform for real time 3D-graphics applications such as environmental simulation for training education and entertainment. Games as mass products are pushing the market, preparing it for hardware-supported 3D rendering. Such hardware support is still necessary as the quality of software solutions is still poor in respect to resolution and image quality. One feature which is gaining more and more in importance is texture mapping. For optimum appearance with lowest possible modeling costs, texture mapping is a key feature of real time graphics systems today. Due to performance reasons, most hardware and software realizations of texture mapping on entry level platforms like PCs do not support true perspective mapping and texture blending. This results in bad artifacts when doing animation. Thus, components which overcome this problem will be standard components of graphics systems within a few years. First products like the GL*i*NT chips by 3Dlabs or the Real3D/100 chipset by Lockeed Martin are available or announced. Compared to the REX chip presented in this paper, the GL*i*NT chip (300SX) has a much lower texture mapping functionality and rendering performance. The Real3D chip set has almost the same 3D functionality, has less performance and consists of several chips. Both GL*i*NT and Real3D/100 are designed for a certain typical system environment. They integrate bus interfaces and video controllers, which is efficient for small and effective systems. Contrarily, the REX chip was designed for highest and scalable rendering performance with more flexibility for system design.

During the development of the REX chip, Intel published its 3DR specification [7], which is an attempt to define and standardize the functionality of the pixel-oriented part of the rendering pipeline. Looking at this „abstract rendering engine“ as it is called by Intel, we found that it fits the 3D functionality of the REX chip very well. This gave us the confirmation, that we are on the right track.

2 The Rendering and Texture ASIC

The Rendering and Texture ASIC (REX) has been designed to provide an efficient, highly integrated support for shading and texture mapping. It is based on the TRIA Chip described in [1] and the concepts of the PRE-Chip described in [2]. The REX-chip consequently makes use of pipelining and on-chip parallelism to deliver a shaded and texture mapped pixel every clock cycle. To gain even more performance, multiple REX chips can be employed without additional initialization overhead.

2.1 Functionality of the REX Chip

The REX chip has the following characteristics:

Primitives:

- Triangles
- Quadrilaterals with upper edge parallel to the span

Sampling algorithms:

- Exact point sampling (24 bit address space)
- Area sampling with 16 stochastically distributed sample points and up to four visible fragments per pixel

Pixel operations:

- Flat shading
- Gouraud shading
- Z-buffering
- Alpha blending for transparency
- Texture mapping, perspective correct, 1, 4 (bi-linear blending) or 8 samples (tri-linear blending) per pixel, MIPmap approach
- Nine mixing modes to support lighting effects
- Support for accumulation buffer (16 times oversampling)

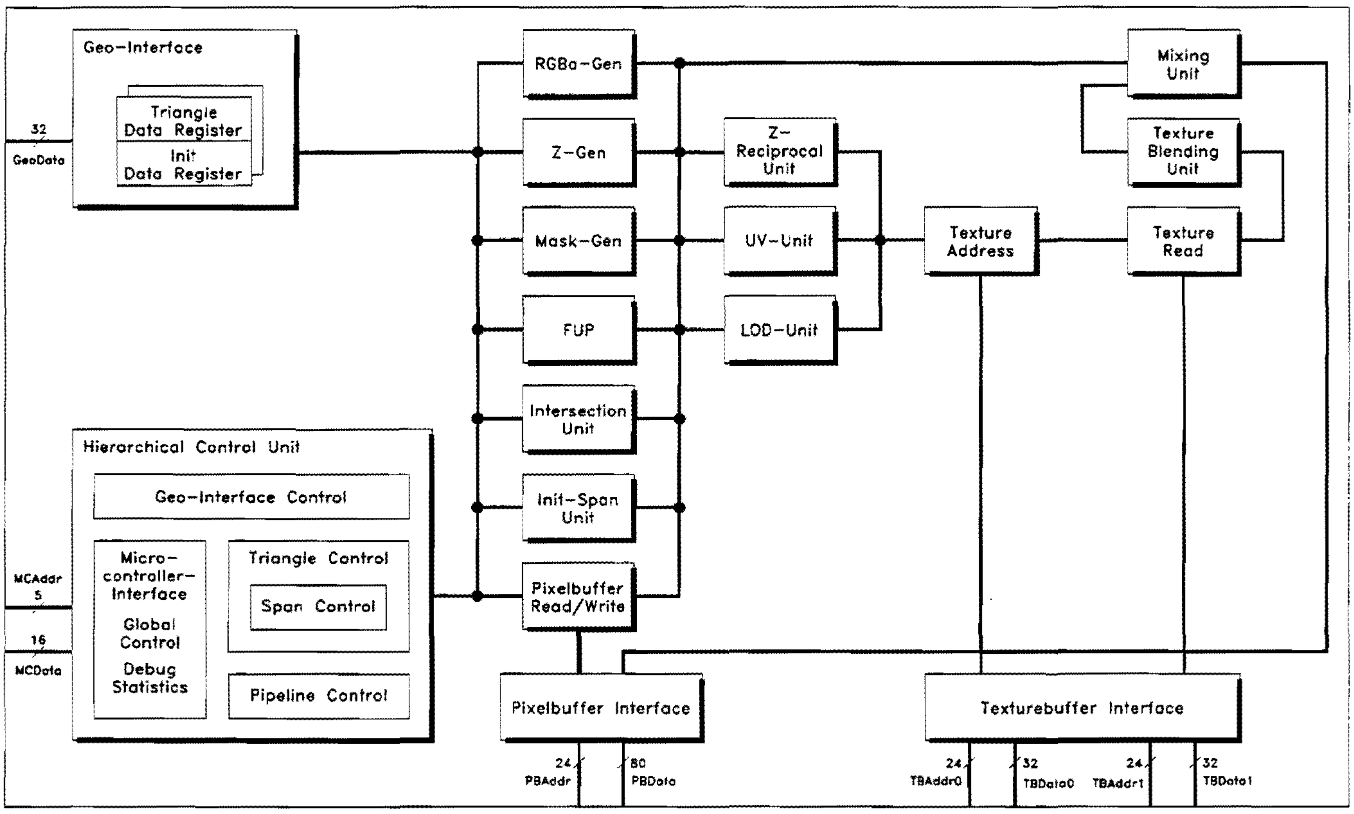

Figure 1: Block Diagram of the REX Chip

Provisions for chip-level parallelism:

- Up to 32 REX-chips can work on the same data set in a scan-line interleaved architecture

Interfaces:

- 32 bit data interface for primitive data
- 16 bit microcontroller interface
- 80 bit pixel buffer interface
- 64 bit texture buffer interface

Provisions for testing and debugging:

- Full access to internal RAM and control and diagnostic registers via the microcontroller interface
- Boundary scan interface
- Two scanpaths including all internal registers

Figure 1 shows the block diagram of the REX chip. The chip has a hierarchical control structure realized by interactive state machines. The data paths are realized as parallel pipelines of different lengths. Shift register stages are provided to equalize the different lengths so that values, which belong together, arrive at the mixing stage at the same clock cycle. When accesses to external interfaces can not be carried out pipelined within a clock cycle, parts of the pipeline or the whole pipeline is halted. This can be necessary during area sampling or during tri-linear blending, when all 8 samples have to be loaded from the texture buffer. To prevent the pixel buffer latencies from degrading performance, the pixel buffer interface contains FIFO memories for the different logical parts of the pixel buffer word.

2.2 Description of the Building Blocks

In the following section, I describe the different basic building blocks of the REX chip focusing on the realization of the texture mapping functionality.

2.2.1 The Control Structure

The global control and initialization of the chip is done through the microcontroller interface. Before normal operation, internal status registers which control the operation modes and several RAMs which are used as lookup tables have to be loaded. This is done using the microcontroller interface. The interface provides a 16 bit data path and a 5 bit address room with direct access to some important control/status registers as well as to registers which contain statistical data. The statistical data are necessary for a real time simulation system to recognize an overload condition in the viewing subsystem. All other memory locations inside the chip and the memory interfaces are accessed through an address/data register logic. This logic is controlled by a state machine providing auto-increment modes for the address generation and taking care of the format conversion of the data words. There are five 16 bit data registers which can hold a 80 bit pixel buffer word. In addition to the initialization tasks, the chip can be operated in a debug mode where several kinds of steps, like pixel step, span step, segment step and triangle step are executed under the control of the external microcontroller.

The control unit of the geometry interface takes care of the loading of the data sets from an external FIFO memory. In single-Rex mode, its state depends on the availability of data in the FIFO and the status of the generation pipeline. In multi-Rex

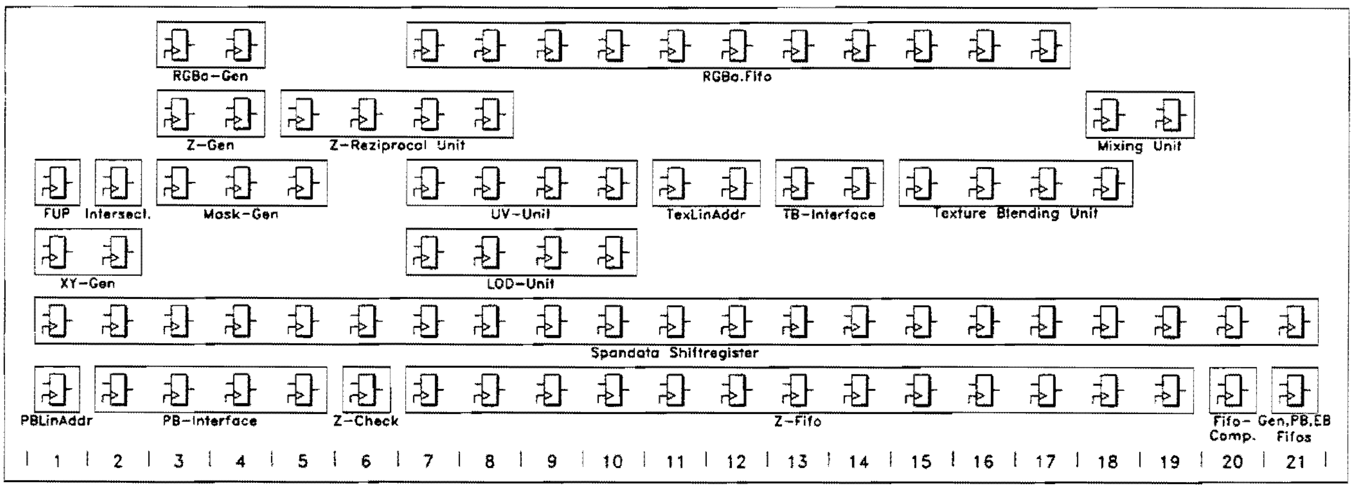

Figure 2: The Pipeline Structure of the REX Chip

mode, the geometry control of the master device observes whether all slave devices are ready to receive new geometry data as well.

The triangle control unit deals with all functions which have to be performed per triangle. Its most important task is the loading of primitive data from the setup registers in the geometry interface to the internal working registers. Triangle- and span-relevant data are loaded independently to enable the pipeline to process a new triangle when pixels of the last span of the actual triangle are still generated. The triangle control unit initializes the segment process which decides in which part of a triangle the current span is. The init span process determines the starting values for the span control unit.

The main task of the span control unit is to generate the x-address of the actual pixel. Furthermore, pixel data, which are invalid due to certain conditions like opacity, are marked to prevent operations such as tri-linear blending which could halt the pipeline, from dropping the performance unnecessarily. Data of two triangles can be present in the pipeline at a time. Corresponding data are marked with an ID bit. In this case the geometry unit has to assure, that the processed triangles do not overlap to avoid data inconsistencies. Otherwise, the pipeline has to run empty before values of the next triangle are generated.

The REX chip as shown in figure 2 consists of several processing pipelines which operate in parallel. The maximum pipeline length is 21 stages. Intermediate results and parameters from all pipelined units which are not used immediately are stored in the span data shift register. The pipeline has to be halted by the pipeline control unit under several conditions. The most likely of these conditions is that the pixel buffer FIFOs run full due to refresh cycles when using DRAMs or due to accesses to the extended buffer when performing area sampling. If more than one access to the texture buffer is necessary (tri-linear blending) or if the texture buffer is not ready due to refresh cycles, the parts of the pipeline which are not related to texture mapping are halted on a cycle by cycle basis.

2.2.2 The Geometry Interface

The geometry interface stores the data sets which are read from an external FIFO memory. The length of the data sets and the meaning of the single values are different depending on the type of data set. The shortest data set comprises 22 32 bit words; the longest data set consists of 45 words. Due to the buffering of the parameters in the geometry interface as well as in the processing units, initialization and processing can work in parallel. For accumulation buffer [5] operation, pseudo statistic subpixel offsets [9], [4] from a table are added to the vertices x,y addresses.

2.2.3 The Rasterization Unit

The rasterization unit determines the addresses of pixels which belong to a primitive represented by its corresponding data set. The rasterization unit can be operated in point sampling mode and in area sampling mode. Rasterization is done in ascending scanline order (top down). It always starts from the vertex with the lowest y value (*pTop*). The edge with the biggest difference in y direction is the active edge and determines the scanning direction. The coordinates of the starting points and endpoints of the scanlines are calculated in the init span unit by adding the edge slopes recursively.

In point sampling mode, pixels are only considered if their center points lie within the geometric boundaries of the triangle. Pixels which lie on the boundary are handled in such a way that, in the case of triangles sharing edges, all edge pixels are set only once. This prevents errors during alpha blending and reduces the number of frame buffer accesses. The algorithm has already been realized in the TRIA chip described in [1].

In area sampling mode, the situation is much more complex. To avoid aliasing effects, all pixels which are partially covered by a triangle are taken into account. For all pixels, the area which is covered by the actual triangle has to be decided. The basic idea is to define 16 stochastically distributed sampling points within a pixel and test which of these sampling points are covered by the triangle. ? [illegible]

During area sampling, the FUP unit analyses the type of scanline which is to be rasterized. There are basically seven different types of scanlines depending on whether they include one, two, three or none of the vertices. This information together with the scanning direction (left to right or right to left) and the sign of the edge slope determines the sampling point within the actual pixel (which is not always the center point as in point sampling). The coordinates of the sampling point are passed to the r,g,b,α and the z generator which calculate the respective values. Furthermore, formula and parameters for the calculation of the intersection points are determined and passed to the intersection

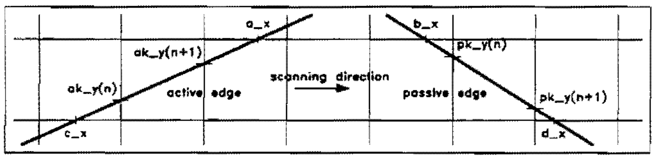

Figure 3: Intersection Points

unit. Most of the functionality of the FUP unit, which handles more than 150 different cases, is realized in 8 lookup tables which keeps the algorithm flexible up to a certain extent.

The intersection unit determines the intersection points between the boundaries of a pixel and the edges of the triangle, which is rasterized (figure 3). Formulas which are applied according to the decision of the FUP unit are of the form:

$$intersect\_y = frac(p\_y) + (1 - frac(p\_x)) \cdot dy\_dx$$

The fractional part of the x,y addresses of the intersection points are rounded to 3 bits. They are concatenated and form a 6 bit address to a lookup table which contains 64 16 bit mask words. Each bit location in the mask words determines whether the corresponding sampling point is covered or not. Depending on the scanning direction and the sign of the edge slope, the bits of the mask must be inverted to get the covered sample points. In case more than one edge is contained in the pixel, the masks are generated for each edge and then logically ANDed to form the final mask.

↓ depth comparison?

2.2.4 The r,g,b,α Generator

In the area sampling mode, the distance of successive sampling points does not always equal 1 due to the edge pixels. Thus, simply adding a fixed increment for r,g,b,α and z is not possible. Multiplication operations are necessary in any case if the starting or end point of a scanline does not lie on a pixel center. In order to have a homogeneous formula and a fixed pipeline length for the calculation, we decided to determine the color, alpha and z values in the most general way which is by using the plane equation:

$$rel\_x \cdot dv\_x + rel\_y \cdot dv\_y + pTop\_v$$

where *rel_x* and *rel_y* are the x,y coordinates relative to the coordinates of *pTop*, *dv_x* and *dv_y* are the increments of v (representing r,g,b,α and z) in x and y direction and *pTop_v* is the value of v at *pTop*. *pTop* is the reference vertex of the triangle. Figure 4 shows a block diagram of the r generator. The g,b,α generators are identical to the r generator.

2.2.4 The z Generator

The z value at the sampling location is calculated in the same way as r,g,b and α using the plane equation. The only deviation from the block diagram in figure 4 are the wider data paths. In point sampling mode, only one read cycle to the pixel buffer and one write cycle depending on the result of the z comparison take place. In area sampling, the situation is much more complex as there are up to four fractions per pixel which have to be considered.

2.2.5 The z Reciprocal Unit

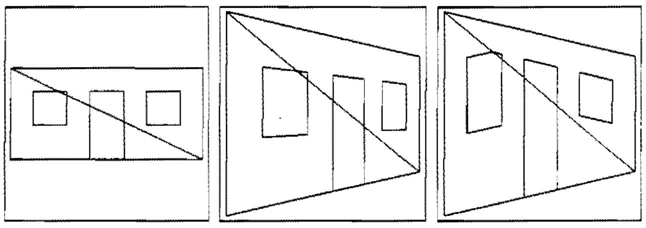
Figure 5a-c: a - Original Texture; b - Perspective Mapping; c - Linear Mapping

During their flow through the geometry part of the rendering pipeline as defined by graphics standards like PHIGS PLUS, all vertices are subject to a perspective transformation. During rasterization, linear interpolation of colors or texture addresses, which is commonly done, leads to perspective distortion as shown in figure 5c. The distortion is as big as the z differences between the vertices of a triangle are. It can be easily noticed on large planes textured with a regular pattern. A possible solution to this problem could be to keep the triangles small. This, however, would burden the geometry stage of the rendering pipeline and would decrease the attractiveness of texture mapping, which gives a very natural appearance at moderate geometry costs. To avoid this, we decided to realize a true perspective texture mapping which requires a division per pixel. The division is realized as multiplication with the reciprocal value which is determined in the z reciprocal unit shown in figure 6.

The reciprocal value is calculated based on the iteration algorithm by Newton/Raphson. The formula is:

$$za_{n+1} = za_n \cdot (2 - z \cdot za_n)$$

with z being the value of which the reciprocal value is wanted and za being the iterated value. For za_0 a seed value is used.

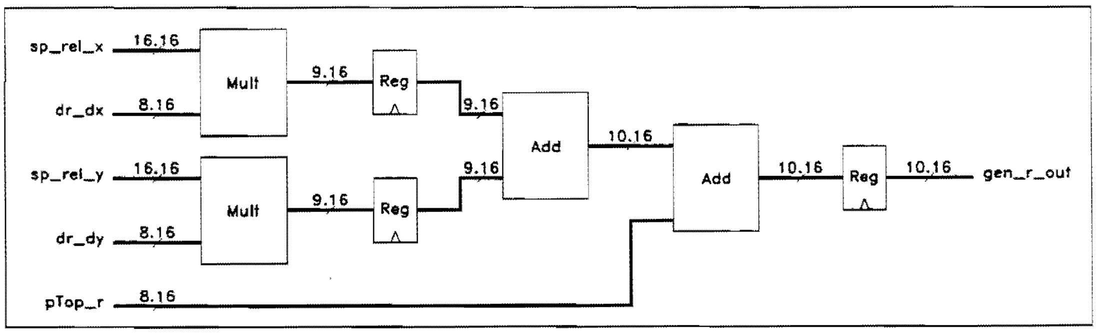

Figure 4: The r Generator

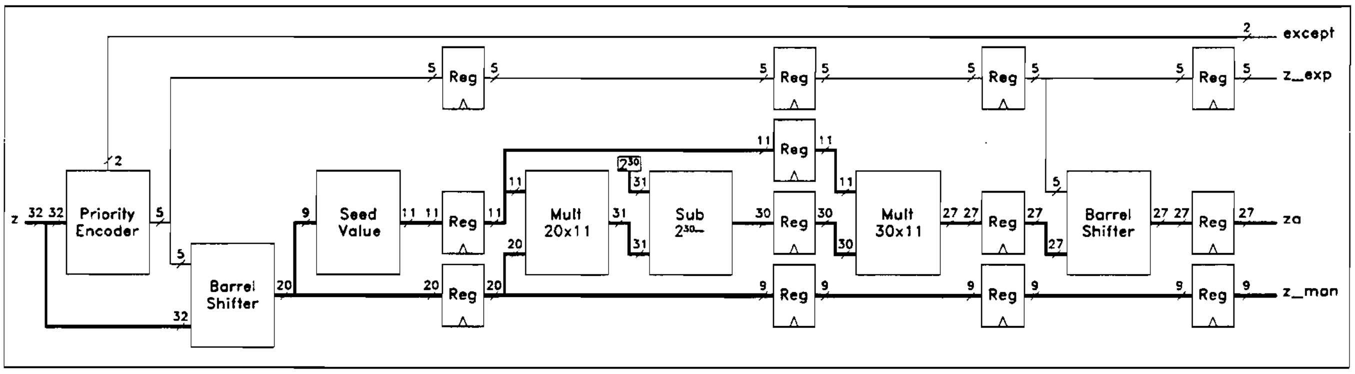

Figure 6: Block Diagram of the z Reciprocal Unit

The algorithm requires normalized floating point numbers. Therefore, the fixed point number z is converted first. The nine most significant bits of the mantissa are used to determine a seed value. The iteration is done one time, doubling the number of significant bits, which is enough for our purposes. In the end, the resulting floating point number is reconverted to a fixed point representation. The floating point representation of z is passed to the LOD unit where it is used to determine the correct level of detail from the MIPmap representation of the texture.

2.2.6 The Texture Unit

The texture unit is the most complex processing unit within the REX chip. It consists of the uv generator, the LOD unit, the texture address unit, the texture buffer interface and the texture blending unit.

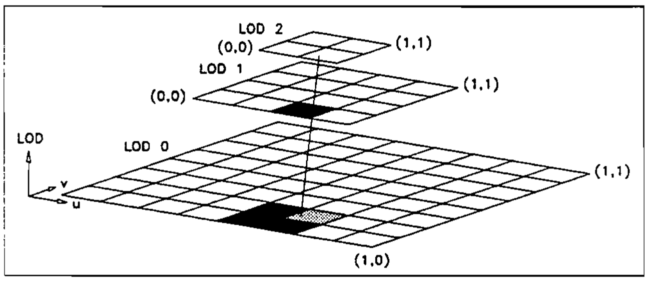

Figure 7: The MIPmap Structure

The texture unit works on prefiltered texture maps which are called MIPmaps [11]. In this representation, in addition to the image with the intrinsic resolution, filtered images with lower resolutions called levels of detail (LODs) are available. The filtering is done by simply accumulating the color values of four adjacent pixels of the higher resolution image and averaging the result to compute one (figure 7). In our realization, the maximum size for a texture is 1024 by 1024 texels. This size limits the possible number of LODs to 11, where LOD0 is equivalent to a base texture of 1024 by 1024 texels. Since it does not make sense to have any LOD of any texture available, the two values BESTLEVEL and WORSTLEVEL, included in the data set of textured triangles, indicate the available LODs. The LOD which has to be used depends on the z value of the sampling point. The use of MIPmaps avoids aliases which are due to subsampling. Artifacts which are due to horizontal or vertical movements of the viewer or objects in a texture mapped scene can be reduced by using bi-linear blending instead of point sampling the texture [6]. In this mode, the four texels which are nearest to the u,v address of the sampling point are taken into account. Their values are weighted according to their distance to the sampling point and summed up. The result is scaled to the color range and assigned to the pixel. In the bi-linear blending mode, the four nearest pixels are chosen from the nearest LOD. When zooming into a scene, the change of the different LODs can be recognized. This effect can be reduced by tri-linear blending. Tri-linear blending consists of two bi-linear blending operations with texels from the two nearest LODs. The results of the bi-linear blending operations are weighted according to the z coordinate of the sampling point. The weighting function determines the resulting blending effect. Thus, it should be kept flexible.

2.2.6.1 The uv Generator

Using the x and y coordinates from the actual sampling point, the za value from the z reciprocal unit, and a set of eight coefficients *a11*-*a22* which are precalculated and included in each parameter set, the uv generator calculates the normalized uv values according to the following equations:

$$u_n = za\cdot[(a11\cdot sp\_x)+(a12\cdot sp\_y)+a13]+a14$$
$$v_n = za\cdot[(a21\cdot sp\_x)+(a22\cdot sp\_y)+a23]+a24$$

u_n and v_n are multiplied with the size of the LOD calculated by the LOD unit in order to receive the final uv values which are used for the point sampling mode of texture mapping. For bi- and tri-linear blending, the uv values of the four nearest texels per LOD are calculated depending on the fractional parts of the uv values. The corresponding weighting coefficients are read from a lookup table.

2.2.6.2 The LOD Unit

During point sampling and bi-linear blending, the LOD unit determines the LOD which is nearest to the sampling point. During tri-linear blending, the two LODs which embrace the sampling point are taken into account. In the tri-linear blending mode, weighting factors for the LODs are read from a lookup table which is addressed by the fractional part of the calculated LOD value. The basic equation to determine the LOD is:

$$LOD = ld(clod\cdot za^2)$$

Since za is available in normalized floating point format from the z reciprocal unit, the whole equation is calculated in floating point which reduces the complexity of the realization. If the calculated LOD exceeds the range given by BESTLEVEL and WORSTLEVEL, it is saturated to the respective value. In the tri-linear blending mode, the following equations determine the two LODs which are used:

$$LOD_{low} = min(WORSTLEVEL, max(BESTLEVEL, int(LOD)))$$

$$LOD_{hi} = min(WORSTLEVEL, max(BESTLEVEL, int(LOD)+1))$$

2.2.6.3 The Texture Address Unit

The texture address unit determines the physical addresses for the access to the texture buffer. Besides the uv values from the uv generator and the LOD, the texture color depth (8, 16 ore 32 bits) and the 24 bit base address of the actual texture map are considered for the calculation.

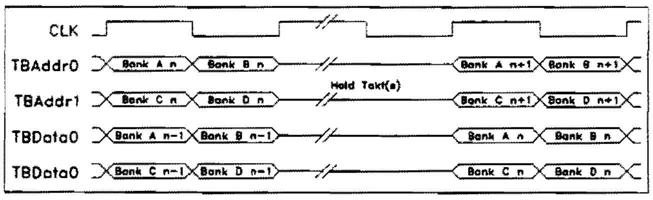

Figure 8: Timing Diagram of the Texture Buffer Interface

2.2.6.4 The Texture Buffer Interface

The texture buffer interface consists of two independent 32 bit data paths with 24 bits of address space each. The attached memory must be organized in such a way that four arbitrary, adjacent 32 bit texels can be read in one access. This can be achieved by interleaving the memory two times in the horizontal and two times in the vertical direction. In 16 bit or 8 bit texture modes, the memory stores two or four texels per word. Starting from their base address, the LODs of a texture are stored consecutively in the memory beginning with the best LOD available. The interface runs at 80 MHz. During one period of the 40 MHz chip clock, addresses for all four banks are present at the two independent address busses and can be latched externally. After a certain number of wait cycles, depending on memory speed, the four texel values appear on the data busses and are latched inside the chip. Figure 8 shows the basic timing of the texture buffer. To support DRAMs, a unit detects page misses based on the array size of the texture memory. This enables an external controller to start precharge cycles as early as possible. Furthermore, an internal logic keeps track of refresh cycles and decides whether refresh cycles have to be triggered. The refresh logic tries to trigger refresh cycles during periods when the texture memory is not used. If a certain number of refresh cycles (determined by the type of memory device) is pending, the texture pipeline is halted and the refresh cycles are executed. The values which determine the behavior of the refresh logic and the number of wait states necessary for page mode and random cycles are controlled from the microcontroller interface. When fast SRAMs are used to realize the texture buffer, no wait states are necessary. When using DRAMs, one or two wait states in page mode and three to five wait states in random mode are inserted, depending on the device speed. Wait states halt the generator pipeline as well as the texture pipeline. During tri-linear blending, the pipeline is halted for each pixel, because eight values have to be loaded in two cycles. An exception to this rule takes place if values which have been loaded in the previous cycle can be reused. For this purpose, the last values are cached in the texture buffer interface.

2.2.6.5 The Texture Blending Unit

The texture may be represented in eight different formats as shown in the following table. Depending on the purpose of texture, color values (r,g,b), intensity (i) and alpha values (α) are used.

According to the texture format, multiplexers within the texture blending unit connect the components of the texture words to the appropriate calculation units. During bi-linear blending, the components of the four texel values are weighted with the factors determined by the LOD unit and summed up. The calculations

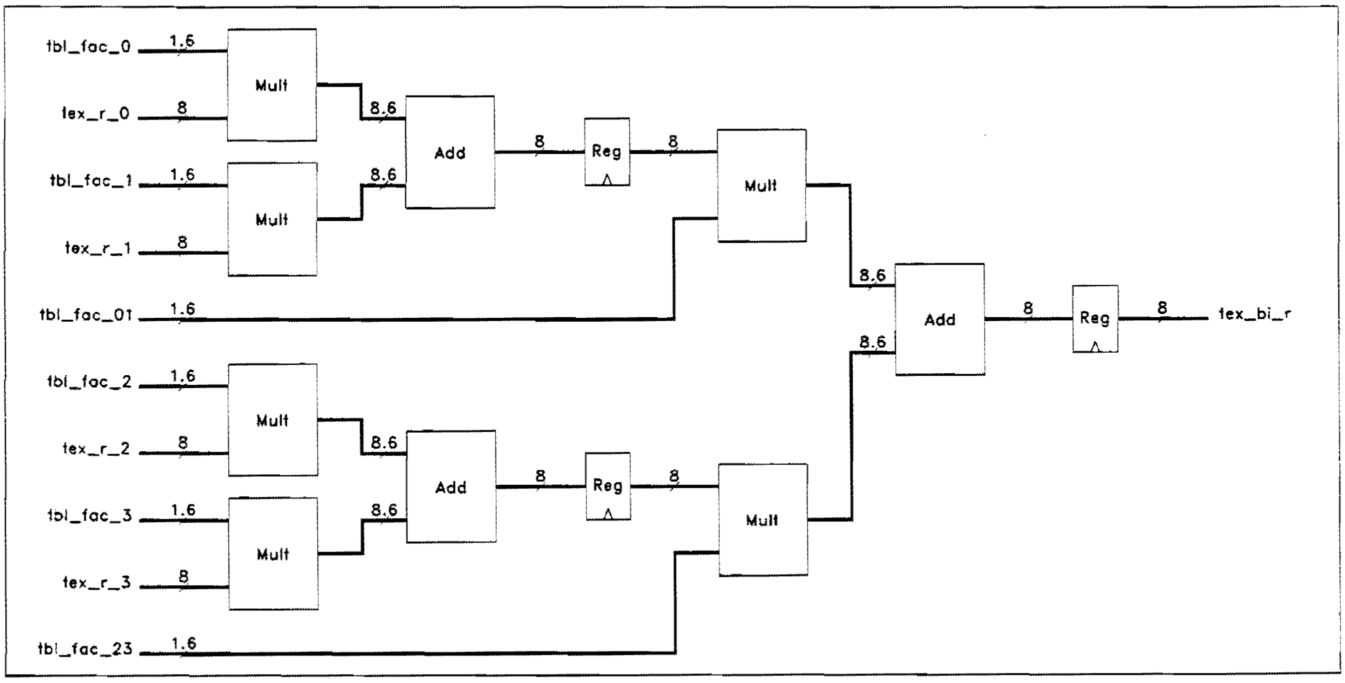

Figure 9: Block Diagram of the Texture Blending Unit

for r,g,b and α are done in parallel. For each component, six multiplications and four additions are necessary (figure 9). For tri-linear blending, four texels of the first LOD are processed in this way and latched. The four texels of the second LOD are blended consecutively in the same manner. The resulting values are weighted with the LOD blending factors and summed up to determine the final values.

texel width	contents	format
32	r,g,b,α	8,8,8,8
24	r,g,b	8,8,8
16	r,g,b	5,6,5
16	i,α	8,8
16	r,g,b,α	4,4,4,4
8	i	8
8	α	8
8	i,α	7,1

Table 1: Texel Formats

2.2.7 The Mixing Unit

The mixing unit combines the results of the r,g,b,α generator pipeline and the texture pipeline to achieve lighting effects on textured surfaces. Furthermore, a secondary color defined in the triangle data set can be included into the mixing for fading effects. We realized nine different mixing modes:

1. $mix\_c = gen\_c * t\_i; \quad mix\_\alpha = gen\_\alpha;$
2. $mix\_c = gen\_c, \quad mix\_\alpha = gen\_\alpha * t\_i;$
3. $mix\_c = gen\_c * t\_i; \quad mix\_\alpha = gen\_\alpha * t\_i;$
4. $mix\_c = gen\_c * t\_c; \quad mix\_\alpha = gen\_\alpha;$
5. $mix\_c = gen\_c * t\_c, \quad mix\_\alpha = gen\_a * t\_\alpha;$
6. $mix\_c = gen\_c + t\_i * sec\_c, \quad mix\_\alpha = gen\_\alpha + t\_i * sec\_\alpha;$
7. $mix\_c = gen\_c, \quad mix\_\alpha = gen\_\alpha + t\_\alpha * (sec\_\alpha - gen\_\alpha);$
8. $mix\_c = gen\_c + t\_i * (sec\_c - gen\_c), \quad mix\_\alpha = gen\_\alpha;$
9. $mix\_c = gen\_c + t\_i * (sec\_c - gen\_c), \quad mix\_\alpha\ gen\_\alpha + t\_i * (sec\_\alpha - gen\_\alpha);$

In the equations, c corresponds to the color components r,g,b, whereas i corresponds to an intensity. The gen values are from the generator pipeline, whereas the t values are from the texture pipeline. As a last operation, the resulting values are saturated to the maximum and minimum values to prevent color over- or underflows. In the accumulation buffer mode, the accumulation of the generated and the memory value takes place in the mixing unit. The necessary weighting factors correspond to the reciprocal of the supersampling factor which is 4 or 16. Thus, the weighting can be achieved by simply shifting the operands.

2.2.8 The Pixel Buffer Interface

The pixel buffer has a word width of 80 bits and has a linear addressing scheme. It consists of two logical regions, the real pixel buffer and the extended buffer. In point sampling mode, the pixel buffer stores all pixel related data of a respective address. These are the color values r,g,b, the alpha value α, the depth value z and some status information. In the supersampling mode, r,g,b,α are expanded by four bits each, to store the accumulated intermediate values. In the area sampling mode, the 16 bits of the area mask are stored additionally. Furthermore, for each pixel, up to four fractions are stored in the extended buffer which is addressed dynamically. To mark pixels with more than one fraction, a pointer bit in the status of the pixel buffer entry is set.

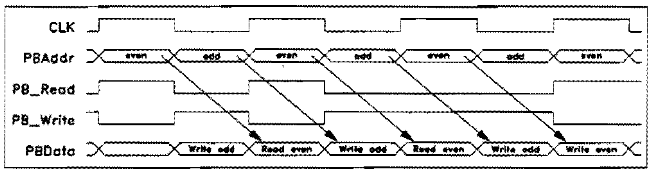

Figure 10: Timing Diagram of the Pixel Buffer Interface

In this case, the r,g,b,α and z values are replaced by a pointer to one or two addresses in the extended buffer, where two fractions are stored consecutively at the first address and one or two at the second address, providing three or four fractions exist. The number of existing fractions is coded in the status of the pixel buffer entry. To ease the z decision which is much more complex in the area sampling mode than in the point sampling mode, a status bit indicates whether one of the datasets in the extended buffer belongs to a fraction covering the whole pixel. In this case, the z value and the position of the full pixel are stored in the pixel buffer entry. If the actually generated fraction is behind the full pixel, it is rejected and no accesses to the extended buffer occur. In the case of more than four fractions per pixel, a smooth bit is set, which indicates this situation. In a postfiltering process, such pixels can be blended with surrounding pixels to minimize aliasing. The same situation takes place when the extended buffer is full and no fractions can be stored.

Like the texture buffer interface, the pixel buffer interface works at 80 MHz. The pixel buffer is assumed to consist of two banks, each containing either even or odd addresses. During the high state of the 40 MHz chip clock, the even addresses are output, during the low state, the odd addresses. The type of access (read or write) is indicated by two signals. The addresses are latched externally. Assuming fast SRAMs for the pixel buffer, read data are latched in the chip one clock cycle after the address appears. Write data are output at the next edge of the clock and are removed at the second following edge of the 40 MHz clock. Figure 10 shows the basic timing of the pixelbuffer interface.

The pixel buffer interface receives its data from three FIFOs which contain 80 bit data words and 24 bit addresses. One of the FIFOs contains only full pixels and first fractions. The second contains additional fractions where pointers have to be stored in existing pixel buffer entries. The third FIFO, which consists of two independent blocks for even and odd addresses, contains data for the extended buffer. The subdivision assures, that independent of the available time slice (even or odd), a value can be written. During each period of the chip clock, only one read cycle can be executed. Therefore, read cycles have priority over write cycles. Since the external memory access frequency is assumed to be double that of the internal pipeline, two write cycles can be performed per chip clock cycle. If no read cycle is necessary due to an invalid pixel in the pipeline, two write cycles are executed. The decision from which FIFO the data are written is made based on the filling level of the FIFOs, the triangle ID and a round-robin strategy. If the triangle ID bit changes within a FIFO, the other two FIFOs are served first. If one of the FIFOs is filled, the generator pipeline is halted until the filling level is once again below a certain threshold.

2.3 Performance of the REX Chip

Analyzing the performance of polygon rendering hardware, two different parameters must be considered. The first one is the performance in polygons, namely triangles, per second which can be rendered, assuming the data sets are available. Assuming further, that the triangles are small, the initialization time for the chip determines the performance limits. The initialization time mainly depends on the number of parameters in the data set of a triangle. Thus, for different shading and texturing modes, different performance values turn out, as shown in table 2.

rendering mode	triangles/s	triangles $\leq$ pixels
flat shading	900,000	34
flat shading + texture	540,000	64
Gouraud shading	660,000	50
Gouraud shading + texture	440,000	80

Table 2: Maximum Triangle Performance Values

The second important performance parameter is the number of pixels which can be generated per second. This parameter depends on the chosen sampling algorithm, the texture blending mode, the speed of the attached memory and the clock frequency of the internal pipeline. Clock frequency and memory speed influence the performance linearly. When DRAMs are used, there is some additional loss in performance due to refresh cycles and random accesses. For point sampled texturing, only one access per pixel is necessary. For bi-linear blending, the required four values can be read pipelined in one clock cycle as well. Using tri-linear blending of 32 bit textures, two clock cycles are necessary unless data from the internal cache registers can be used. When 16 bit or 8 bit textures are used, the probability of cache hits is high as more texels are read per cycle. Assuming no memory wait cycles are necessary in the point sampling mode at 40 MHz chip clock, the following performances occur:

rendering mode (including z buffering)	Mpixels/s
flat shading	40
Gouraud shading	40
texture mapping, 1 sample	40
texture mapping, 4 samples	40
texture mapping, 8 samples (16 bit)	40
texture mapping, 8 samples (32 bit)	20

Table 3: Maximum Pixel Generation Rates

In the area sampling mode, the performance additionally depends on the data. The factor which mainly influences the performance is the depth complexity of the rendered scene. This is due to the fact that the number of fractions per pixel determines the number of accesses to the extended buffer. Under worst case conditions, with four existing fractions, one read and one write to the pixel buffer and four reads and one write to the extended buffer are necessary. The whole process with the comparisons and decisions takes about 20 cycles. This delay influences performance only in the instance of many such pixels occurring consecutively or if the pixel buffer FIFOs are empty. Otherwise, the interface is kept busy with data from the FIFOs.

Another factor which influences performance during area sampling is the ratio between edge pixels and pixels set during point sampling. In scenes with small triangles, a significant rise in rendering time can be noticed due to the higher number of pixels which must be processed. Since the REX chip was designed for real time applications, it keeps track of statistical information within a frame. This information can be accessed via the microcontroller interface. Thus, the system can react to overload situations by reducing the scene complexity.

2.4 Status of the REX Chip

The design has been described using VHDL. Functional simulation and synthesis have been done using the Synopsys VHDL simulator and silicon compiler. Test stimuli and strobes were generated from a software model built from an existing software implementation. The netlist generated by the synthesis tool was transferred to the LSI Logic design environment and simulated with the CMDE simulator. The results of the behavioral and netlist simulation were compared to ensure the correctness of the design flow. The gate count is about 450,000 gate equivalents with about 280,000 logic gates, 40,000 gates for registers and 130,000 gates for internal RAMs. The utilization is about 45%. Due to power dissipation and utilization reasons, we migrated from the LSI 300K 0.6µ 5V technology to the 500K 0.5µ 3.3V technology with a die size of 16,5 by 16,5 mm. The chip was designed for a worst case operating frequency of 40 MHz. Two internal PLL-circuits compensate clock skew and buffer delays and generate the 80 MHz frequency for the buffer interface operation. The chip has a power dissipation of about 5W. It is packaged in a 39 x 39 CPGA package with 447 pins. The signal pin count is 304.

The netlist of the chip has been delivered to the manufacturer for layout. We expect first silicon to be available at the end of '95.

3 Texture Memory Considerations

The texture memory is a part of the rendering system which mainly influences performance and determines costs. In order to keep one MIPmap 32 bit texture with a LOD0 resolution of 1024 by 1024 texels in the texture memory, at least 5,33 Mbytes are necessary. Due to the fixed array sizes of memory devices, typically 8 Mbytes are used. To prevent texture buffer updates from dropping rendering performance, the texture buffer should be realized as a double buffer. Both buffers together add up to 16 Mbytes, which is not even too extreme a demand. Assuming a memory architecture with an interleave of four, a read cycle time of 25 ns would be appropriate. Furthermore, the most restrictive requirement is that the random accesses do not influence performance. To fulfill these requirements, using ultra fast 15 ns SRAMs is still the only possible way. Such RAMs are available with a complexity of 1 MBit. So 16 Mbytes correspond to 128 chips. Independent of the fact that such RAMs are rather expensive, they use a lot of boardspace, too much for compact systems. like in the PC environment.

DRAMs are available in higher complexity and at reasonable lower costs. Besides the more complex interface and the necessity of refresh cycles, DRAMs are substantially slower than SRAMs. Due to the architecture of a DRAM, there are two different access times. The random access time starts from applying the row address. Including the precharge time between subsequent cycles, the cycle time for a typical DRAM is about 100 ns. When data are accessed within a row of the memory array, the access is faster. This mode is called page mode. The page mode cycle time is typically about 60 ns. Achieving the same performance under worst case conditions (random access) as with SRAMs, requires a four times higher interleave factor. A higher interleave factor again means higher chip count and a more complex circuitry, decreasing the advantages of the DRAMs.

Recently, a number of enhanced memory devices based on DRAMs have been presented.

Hyper page mode devices have the same characteristics as ordinary DRAMs. The cycle time in page mode has been reduced to about 20 to 30 ns which means double performance compared to conventional page mode DRAMs. The random access time could be reduced only slightly to 90 ns.

EDO DRAMs (Extended Data Out DRAMs) have key timing parameters similar to that of hyper page mode DRAMs. Additionally, EDO DRAMs have internal buffers to keep output data stable during CAS precharge time. This feature relaxes the timing and removes the need for external buffers.

EDRAMs (Enhanced DRAMs) contain four internal banks with one line of cache for each. They have a random cycle time of 65 ns, a cache cycle time of 15 ns and a cache to cache cycle time of 25 ns. Up to now, only chips with a complexity of 4 Mbit are available.

SDRAMs (Synchronous DRAMs) and SGRAMs (Synchronous Graphics RAMs) belong to another group of memory devices. They are clocked and have an architecture comprising of two internal banks with cache memory for each bank. Operating the banks alternately, precharge times can be hidden. SDRAMs have a faster cache cycle time (10 ns) but a slower random access time (100 ns) compared to EDRAMs. SDRAMs and SGRAMs are burst oriented. This means that only ascending addresses within a page can be read at full speed. SDRAMs are available or announced in complexities up to 64 Mbit. SGRAMs are announced only in a 128K by 32 bit organization.

RDRAMs (Rambus DRAMS) are also burst oriented. Unlike all other devices, RDRAMs have a special high speed (250 MHz), low voltage swing electrical interface with a proprietary command protocol. Burst access is extremely fast (2 ns). The single read cycle time within the cache is 40 ns. The random access time is 112 ns due to cache miss latencies. The significant difference between burst performance and performance in the other operation modes results from the protocol overhead. RDRAMs require a special controller and a fixed board layout. Due to the high operation frequency the power consumption of the controller is high.

MDRAMs (Multi bank DRAMs) have been one of the latest developements in memory architectures. They exploit the idea of multiple banks on chip, to hide latencies and to increase the hit rate when accessing nonsequential data. The smallest chip comprises 16 banks of 32 rows by 256 columns by 32 bit, summing up to 4 Mbit. The most complex chip announced contains 72 of such banks. MDRAMs are clocked and have a 16 bit multiplexed address/data bus. Data are transferred on both edges of the clock which can be up to 166 MHz. The random access time is about 54 ns. The burst acces time is 6 ns. The access time between activated banks is 18 ns. Control of the MDRAMs is different and more complex than that of ordinary DRAMs. An external controller has to take care of precharge, activate and refresh cycles. The electrical interface is CMOS or TTL compatible.

Taking into account the different characteristics of the described memory devices, there is no optimum solution. If price and board space are of minor importance, SRAMs deliver optimum performance for random access. Among the other devices standard DRAMs and DRAMs with hyper page mode are the cheapest solution if performance is of less importance. EDRAMs offer a very good page mode performance and an acceptable random access time at moderate costs. Due to their limited complexity they require medium board space. MDRAMs seem to be a true alternative to SRAMs. They combine excellent page mode cycle time with acceptable random access time. Due to the internal architecture, the hit rate is high, compared to the other devices with internal cache. Board space requirements of the MDRAMs are moderate. Costs are expected to be similar to that of standard DRAMs. More details are discussed in [10].

4 System Considerations

Using the REX chip, a wide variety of system requirements can be covered. To design a system utilizing a REX chip or multiple REX chips, the most important question is which performance should be reached. This determines the number of REX chips in the system. A multiple REX approach with 32 REX chips increases the pixel generation rate almost linearly up to 1.28 Gpixels/s. On the other hand, it does not increase the triangle performance. Nevertheless, a triangle performance of 400,000 to 900,000 is already rather high and requires a multiprocessor geometry pipeline to be reached as system performance. A drawback of the multiple REX approach, as with all similar approaches, is the necessity of supplying multiple texture buffers as well. The effects of choosing texture memory in terms of performance, boardspace and costs have already been outlined in the previous paragraph. When using the REX chip in the area sampling mode, the up to four fractions per pixel have to be blended after the generation of a frame has been finished. For this purpose, another ASIC, the Color Blending Unit (CBU) has been designed. The blending is done at the frequency of the video output. Systems for multi-media applications such as the integration of video and graphics using the REX chip have been described in [8] and [3].

5 Conclusions and Future Work

I presented a rendering chip which integrates high performance shading and texture mapping functionality compatible with Intel's 3DR specification. The chip was mainly designed for real time applications. The chip integrates support for chip level parallelism allowing for a scaleable system performance.

Future work will focus on two different areas. The most important was already discussed in the section Texture Memory Considerations. It is necessary to reduce memory costs without a significant drop in performance. Therefore, new architectures and memory types must be investigated.

The second topic is the integration of a CPU kernel in order to calculate the necessary data structures from the standard vertex representation internally. This would improve the interface efficiency and increase system performance.

7 Acknowledgements

The REX chip was realized in cooperation with STN Atlas Electronics, Bremen and the Institute for Integrated Circuits, Erlangen. The work was funded in part by the German Ministry for Education and Research (BMBF).

8 References

[1] Ackermann, H.-J., Hornung, C.: An Architecture for a High Performance Rendering Engine. In A. Kaufman, (Ed.): *Rendering, Visualization and Rasterization Hardware*, Springer-Verlag, Berlin, 1993, pp.157-174.

[2] Ackermann, H.-J., Hornung, C.: The Triangle Shading Engine. In R.L. Grimsdale, A. Kaufman, (Eds.): *Advances in Computer Graphics Hardware V*, Springer-Verlag, Berlin, 1991, pp.3-13.

[3] Ackermann, H.-J., Osterfeld, U.: Integration of Live Video and Computer Graphics for Video Effect Generation. In W. Straßer, (Ed.): Proceedings of the ninth Eurographics Workshop on Graphics Hardware, Oslo, September 1994, pp.109-114.

[4] Barkans, A.: Hardware-Assisted Polygon Antialiasing, *Computer Graphics and Application*, 11(1), January 1991, pp. 80-88.

[5] Haeberli, P., Akeley, K.: The Accumulation Buffer: Hardware Support for High-Quality Rendering. Computer Graphics, Vol.24 Nr.4, August 1990, pp. 309-318.

[6] Heckbert, P. S.: Survey of Texture Mapping. *Computer Graphics & Applications*, 6(11), November 1986, pp. 56-67.

[7] Intel Corporation: Intel 3DR/RE Graphics Programming Manual, revision 002, April 1995.

[8] Jäger, M., Osterfeld, U., Ackermann, H.-J., Hornung, C.: Building a Multimedia ISDN PC, *Computer Graphics and Application*, 13(5), September 1993, pp. 24-33.

[9] Lathrop, O., Kirk, D., Voorhies, D.: Accurate Rendering by Subpixel Addressing. IEEE Computer Graphics & Applications, September 1990, pp. 45-53.

[10] Schrödter, F.: Diploma Thesis, Technical University Darmstadt, September 1995.

[11] Williams, L.: Pyramidal Parametrics. *Computer Graphics*, Proc. SIGGRAPH'83, 17(3), July 1983, pp. 1-11.