Graphics Algorithms on Field Programmable Function Arrays

Jaap Smit and Marco Bosma

University of Twente
Department of Electrical Engineering EF9250
POBox 217, 7500AE Enschede, The Netherlands
e-mail: jaap@nt.el.utwente.nl,
bosma@nt.el.utwente.nl

Abstract

The amount of energy consumed in basic CMOS building blocks, like external RAM, external bus-structures, multipliers, local (cache) memory and on chip bus-structures, is analyzed thoroughly to find ways for substantial improvement of the power consumption of high speed graphics algorithms. A Field Programmable Function Array capable of low-power execution of a wide range of algorithms is introduced. Aspects of the compilation of the volume rendering algorithm to this architecture are discussed.

1. Background

The amount of time needed for the rendering of a *dense* 256^3 volumetric dataset, using a 0.5mm CMOS RISC processor lies typically in the order of 2 minutes. The power dissipated by such a processor is mostly at least 5 Watt. I.e. an amount of 600 Joule is needed, when memory access, dissipation in the cache, and many other similar issues are neglected, for the rendering of one single image. This RISC processor architecture consumes hence 15 KWatt for rendering speeds of 25 frames/sec. It is the objective of most software approaches to reduce this amount using conditional code in order to obtain a speed-up when a *sparse* dataset should be rendered. The availability of a low-power graphics processor offers opportunities which are frequently overseen. It can be shown that an amount of energy of at least 2.26 J is needed in an 1μm CMOS process to render a single image on a volume renderer with an 8-bit accuracy. This figure includes all fundamental aspects of the algorithm, like for instance access to an external memory, on-chip caching etc. This indicates that the *whole* rendering process can be performed about *300 times faster* in a 1μm CMOS process, when compared to a modern RISC-processor realized in a 0.5μm CMOS process, for a given power budget. This speed-difference will be even greater when memory-access and cache access would be taken into consideration for the RISC processor and when both realizations would use the same technology. Lack of programmability and high price due to insufficient market coverage are the major problems with dedicated hardware solutions. This is why a true low power alternative is introduced: the Field Programmable Function Array.

This FPFA can be used to solve the problem of dense Volume Rendering [1], [2], [3], in real-time at extremely low power levels. References [4] and [5] describe dedicated systems and chips which are freely programmable for a wide range of graphics applications. A description of low power design from the point of view of *automatic synthesis* is given in [6]. The material presented is heavily dependent on the underlying energy consumption figures of the library. Moreover, no attempt is made to find a universal lower bound. Up to date information about low-power VLSI design is given in [7]. It introduces the basic concepts from technological concepts to circuit and system level implications. Reference [8] shows that synchronous VLSI designs, like for instance adders, consume less energy than their asynchronous variants. An up to date comparison on the algorithmic level including higher order, cubic spline-based, interpolation methods is given in [9].

2. Introduction to low-power concepts

It is of fundamental importance to understand that there exists a *lower bound* for the amount of energy needed to execute a given algorithm in a given, say 1μm 5V CMOS process. This lower bound is the overall energy complexity: $\mathcal{E}_{tot}$. It equals the minimal value of the sum of the energy consumed due to arithmetic $\mathcal{E}_{ar}$, wiring $\mathcal{E}_{wiring}$ and data-storage $\mathcal{E}_{RAM}$. Any technology can be sufficiently calibrated as an energy reference for most

arithmetic applications through the calculation of the average energy needed to perform the addition of the three bits, a, b, and cin of a full-adder. One technique for the description of larger arithmetic blocks, introduced in [11] uses a ripple factor ϱ to represent any extra energy due to ripples within an arithmetic element, like an adder or a multiplier. The ripple factor depends on the architecture of the arithmetic element. It takes values between 1.5 .. 1.8 for adders ranging from 8 to 32 bits, and 2 .. 2.5 for multipliers ranging from 8x8 to 32x32 bits. The formulas describing the minimal amount of energy consumed by an adder and a multiplier are:

$$\mathcal{E}_{\_m\_adder} = m\, \varrho_{ripple}\, \mathcal{E}_{fa}$$
$$\mathcal{E}_{\_mxn\_mul} = m\, n\, \varrho_{mul}\, (\mathcal{E}_{fa} + \mathcal{E}_{and})$$

With $\mathcal{E}_{fa} = 2.41$ pJ and $\mathcal{E}_{and} = 0.35$ pJ for the 1µm, 5V CMOS process introduced earlier.

The energy needed for on-chip communication: $\mathcal{E}_{wiring}$ is derived from the energy needed to switch a (metal-1) wire of 1m length:

$\mathcal{E}_{/m} = 1.44$ nJ, in the same process.

The energy needed for on-chip data-storage depends on the length $\mathcal{D}_{RAM}$ of the wires within the memory system, the probability P_t of a data transfer, the width $\mathcal{W}$ of the databus, the relative efficiency η_{ov} corresponding to the overhead associated to the presence of $\mathcal{A}$ address bits, and $\mathcal{W}$ databits, $\eta_{ov} = \mathcal{W} / (\mathcal{W} + \mathcal{A} + 2)$ and the relative efficiency η_{acc}, describing the energy-overhead involved in the data transport within the random access memory system. This results in an amount of energy needed for one read or write operation which equals:

$$\mathcal{E}_{RAM} = \mathcal{D}_{RAM}\, \mathcal{W}\, P_t\, \mathcal{E}_{/m} / (\eta_{ov}\, \eta_{acc})$$

3. Energy Complexity Aspects

Volume rendering can be seen as a three step process: 1) the calculation of gradients and opacity from grey-values at the sample position, 2) the calculation of the color at the sample position from the gradient and the light position and 3) the composition of the final color from color and opacity values along a ray. We refer for reasons of brevity to [12] for a high quality algorithm for the calculation of interpolated grey-values and gradients and to [13] for a table-based algorithm for the calculation of the color from the gradient and the light position.

The most direct way to tackle the issue of energ complexity of an algorithm is to estimate the amount c energy for the most expensive operations whil neglecting those parts which contribute less than sa 20% to the total amount of energy needed.

3.1 Energy Consumption of external RAMs

Let us take this approach and first concentrate on th aspect of access to the dataset. The contents of on selected row is written into a second level RAM on th border of the chip as shown in Figure 1. This RAM i read sequentially from the chips core to its pins at a rat of 1 byte each 2ns over its IO-pins as indicated in th figure.

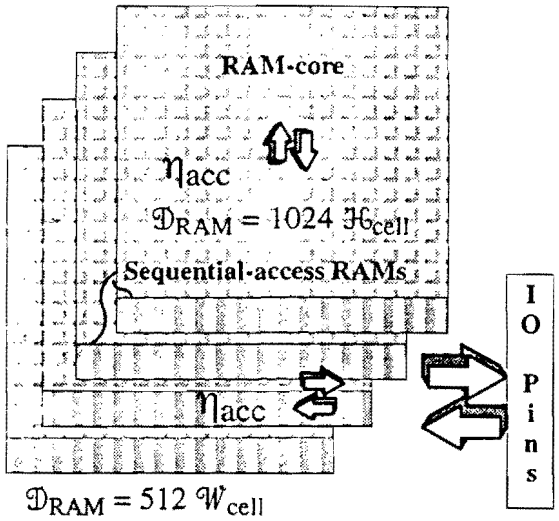

Fig. 1: Traditional VRAM architecture

Figure 2 shows the amount of energy needed for th transfer of 4 to 512 bytes of data from a tradition RAM, organized as 8 memories on one chip each 102 bits deep and 512 bits wide.

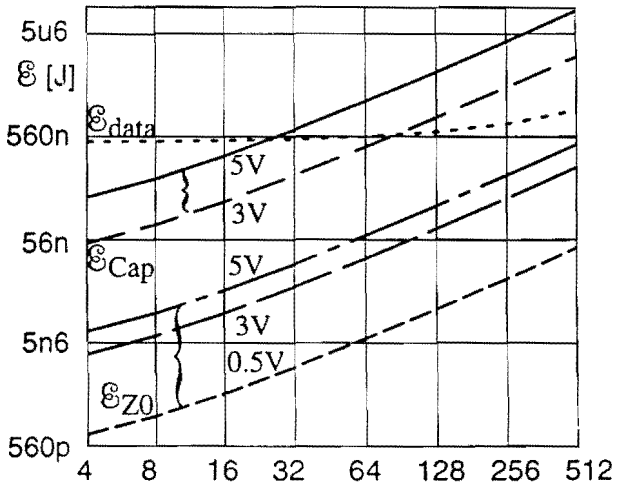

Fig. 2: Energy consumption, RAM-core & IO

There are two contributions: $\mathcal{E}_{data}$ describes the amount of energy needed to transfer the data from the RAM-core to the IO-interface, whereas $\mathcal{E}_{Cap}$ and $\mathcal{E}_{Z0}$ represent the amount of energy needed to transfer the data over capacitive IO-pins, using either a capacitive load or a loaded transmission-line interface with a 500mV voltage swing. Two graphs are given for these cases, one for VDD = 3V and one for VDD = 5V. Note that $\mathcal{E}_{data}$ depends hardly on the length of the burst, i.e. almost all energy is taken by the row selection (RAS), whereas the column select (CAS) takes negligible energy. The most effective interface is apparently the transmission-line interface with a 3V supply voltage. The access efficiency used in the formula's is 1/8. This is a value which can be built with some ease for most RAMs.

3.2 Energy Consumption of Multipliers

The amount of energy for a single 8x8 multiplication is:

$$\mathcal{E}_{\_mul}(8, 8) = 8 \times 8\, \varrho_{mul}\, (\mathcal{E}_{fa} + \mathcal{E}_{and}) = 240pJ$$

in a 1µm 5V process. I.e. one row-access to the external RAM takes as much energy as 2290 8x8 multiplications in a 1µm 5V CMOS process. This amount will grow rapidly when a better process, like for instance 0.5 µm 3V, is selected. One may however argue that this value is indicative for many future processes as well as the process used for the external RAM chips will evolve in basically the same way as the process to be used for the graphics chip.

3.3 Energy Consumption of on-chip RAMs

The process of front-to back composition demands that gradients and grey-values are known at the sample locations along a ray. This implies random access to at least 8 surrounding voxels for the grey-value interpolation and another 12 voxels for the calculation of accurate gradients with the intermediate gradient algorithm introduced in [12]. I.e. 20 RAM locations should be loaded and 28 multiplications are needed to calculate the grey-value and the gradient at the voxel location. It is extremely important that the accesses to the RAM are not made to an external RAM, as one should at least make over 2000 multiplications before the energy involved in the access to one single random location in the external RAM comes in the same order of magnitude as the multiplications. Hence we insert an on-chip cache memory between the graphics chip and the arithmetic part of the renderer to improve the overall efficiency.

How many words may this cache memory have to be energy efficient with respect to the arithmetic operations ?

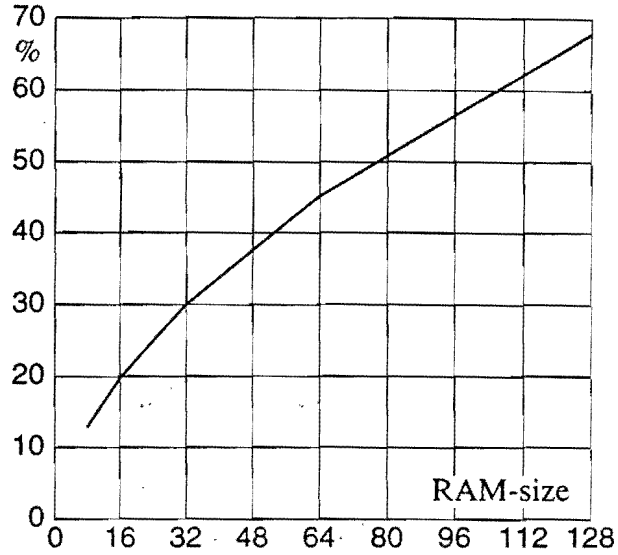

Fig. 3: $\mathcal{E}_{\_ram} / \mathcal{E}_{\_mul}$ in %

The amount of energy dissipated by the access to an 8 x 64 bit RAM with η_{acc} = 1/8 is:

$$\mathcal{E}_{\_ram}(8, 64) = 108pJ$$

This is about 45% of the amount of energy needed for an 8x8 multiplication. Figure 3 shows the ratio of $\mathcal{E}_{\_ram}/\mathcal{E}_{\_mul}$ for ram-sizes ranging from 8 bytes to 128 bytes. I.e. the largest acceptable cache size to be used in a low-power graphics engine in conjunction with one multiplier, is at most 64. A tri-linear interpolation, to be used for the calculation of the grey-value or a component of the gradient, uses 7 multiplications and 8 random accesses to a memory. An efficient configuration for such a tri-linear interpolator uses 8 interleaved RAMs with even-odd addressing which can be read simultaneously to perform 4 linear interpolations in x, followed by 2 linear interpolations in y, followed by 1 linear interpolation in z. Such a tri-linear interpolator uses 34% of the energy for RAM-access and the remaining 66% for arithmetic.

3.4 Energy Consumption due to wiring

The wires connected to an arithmetic block have, by definition, a length $\mathcal{R}_{pd}$ when the amount of energy dissipated in the wiring equals the amount of energy

dissipated in the arithmetic block. Hence, for a multiplier, we have:

$$\mathcal{R}_{pd} = \mathcal{E}_{\_mul} / (32\ \mathcal{E}_{/m}) = 5.21 \text{ mm}$$

There will hence be 20% overhead due to the wiring of the hardware which executes the algorithm, when the multipliers involved are connected to cells in their environment with wires over which their operands are supplied which exceed 1.04 mm in length.

4 The Field Programmable Function Array

The idea behind the Field Programmable Function Array, which is currently designed at the University of Twente, is to provide all the features for low-power operation of an arbitrary algorithm in such a way that minimal overhead is included. Hence, there will just be a minimal controller. Instead, all instructions from an inner-loop of the program are compiled into a graph, which is formed through the static application of control-bits to many identical registered arithmetic and logic units (RALUs) with two adders, boolean functions, and a multiplication, as well as some dedicated functions. The 64-words deep register banks, are either controlled by an address generator or a RALU. A total amount of 100 ... 500 RALUs will fit on a single chip. They will be interconnected over short distances with multiplexers connected to all four sides of a basic RALU block. Moreover, long distance on-chip busses are used to transfer interleaved data at a high (2ns) transfer rate, using low-voltage (e.g. 0.5 V) differential bus connections with the IO/interface, which includes local RAM-storage, or with a direct connection to a RAMBUS RAM. Figure 4 shows a possible floorplan of the FPFA.

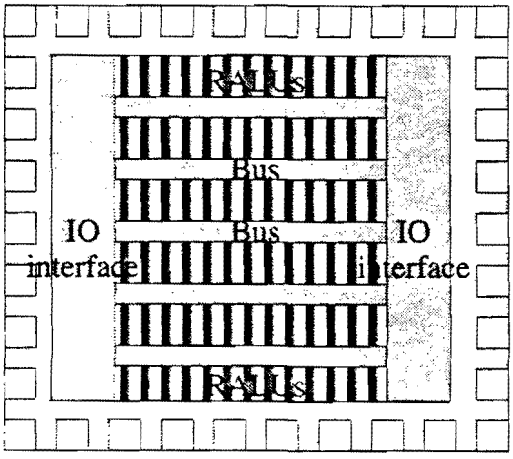

Fig. 4: Possible FPFA Floorplan

5 Programming the FPFA

There are so many RALUs on a single chip that it is much more natural to interconnect them in such a way that a single expression, or even a sequence of expressions, maps onto a small graph.

5.1 A tri-linear interpolator

Figure 5 shows how the technique of compilation into graphs works out for a tri-linear interpolator used to resample grey-values or calculate intermediate difference gradients in a volume rendering or texture mapping application.

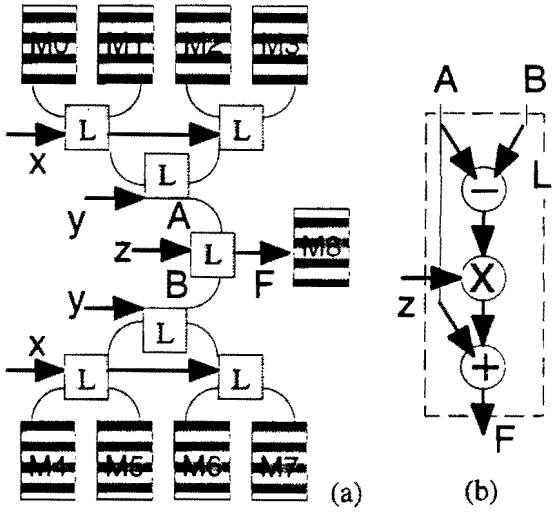

Fig. 5: Tri-linear interpolator realization

In Figure 5a, 7 ALUs and 8 register banks are used to form one tri-linear interpolator. An additional register-bank M8 may be used to collect the results or output. Figure 5b shows the functionality programmed into each of the 1D linear interpolators L. The tri-linear interpolator is controlled from a 3D address with integer part (X,Y,Z) which is used to address the memory banks, whereas the fractional part (x,y,z) is fed into the ALUs. The interleaved memories M0, ... M7, are located along the fast bus which can deliver inputs at 8x the speed of these memories.

Important design issues for an FPFA are the location o the interconnect, the optimal amount of interconnect the inclusion of rudimentary hardware to ease specia functions, like *bit-reversal* primitives used to suppor Fast Fourier Transforms, used in Fourier Volum Rendering, simple functions to enable normalizatio operations etc.

The datapath will be pipelined at the level o RAM-access and ALU-operations. This has a consequence that the tri-linear interpolator has a latenc of 4 clock-cycles for a write operation to M8 and 5 cloc

cycles for completion of the algorithm. A preferred method to take this into account is to initialize the counter which addresses M8 with the negative value of the latency, or to propagate a busy bit, to be used as cin on the address counter of M8. The techniques proposed make it possible to compile expressions in a straightforward way into a graph, provided that there are sufficient local busses within, c.q. between the ALUs to provide the functionality.

5.2 Compilation of optimized code

In volume rendering it is not necessary to compute gradients and colors when the opacity at the sample location is zero, or when the accumulated opacity reaches one. This issue is solved on a traditional computer through the use of if-statements. On an FPFA it is much more natural to generate all addresses along the ray and to conditionally collect all locations at which these operations should be executed in an output RAM. Once this RAM is full, one may switch the configuration of the graph and execute the remaining algorithm on the desired address combinations only.

5.3 Random access to external RAM

One very effective technique in volume rendering uses a collection of pre-specified 3D regions, stored together with the local minimum and maximum of the grey-value within the region. This technique can be very well used in conjunction with an FPFA as well. The main processor for which the FPFA acts as a service processor can simply select those regions for which the min- and max- grey-value give rise to a visible object and issue a load command to the FPFAs IO-interface. The FPFA will start rendering this region, by loading the whole region in one burst, when it has finished executing all previous tasks. The RALUs will be idle during the burst data transfer, which takes, depending on the application, just a small fraction of the busy-time of the RALUs.

5.4 Execution of arbitrary functions

Can the FPFA handle arbitrary functions like sin (x) or 1 / sqrt ($x^2 + y^2 + z^2$) ? The answer is definitely yes.

Functions like sin (x) can be calculated at an arbitrary x location using a table driven approach, in which 128 values of the function are stored in two memory banks. The desired function value can be extracted at an arbitrary location even in-between the samples using a linear interpolator. The calculation of 1 / sqrt (x) is as simple, provided that x is first normalized to [0.5, 1) to avoid problems with the singularity at x = 0. This normalization step involves the computation of a proper power of 2 which scales x to the desired interval. The hardware which can perform this operation is rather simple. This is why this normalization/denormalization circuit which extracts an appropriate number from one of the ALUs operands, will be included in all ALUs. The actual scaling operation may be done either on a (barrel-) shifter or on a multiplier. It is likely that most applications will show a substantially higher demand for multipliers than for shifters, hence it will be a preferred technique to map the shift operation on any of the multipliers available.

This discussion shows how important the selection of additional arithmetic operations is for the construction of a datapath which is optimal for a wide range of applications.

6 Complexity issues

It takes from 8 to 12 RALUs to build one tri-linear interpolator including an address generator and an output buffer. A total of 4 tri-linear interpolators is needed to resample a 3D volume and to compute volume-gradients using the intermediate difference method described in [12]. It is on an FPFA preferred to compute all intermediate differences in x, y, and z on a local subvolume and to address these subvolumes with an offset of 0.5 voxel distance in the respective direction of differentiation. A tri-linear interpolator can then be used to compute the gradient with good frequency response fidelity at the sample location. The use of 4 tri-linear interpolators connected to 4 banks of 8 memories makes it possible to calculate the grey-value and the gradient at one sample location each clock-cycle. The on-chip reflectance map technique [13] can be mapped onto the FPFA as well. Its mapping is however considered to be sub-optimal as a total of 24 memory blocks are needed for the 1.5 KByte reflectance map. Moreover, just one bi-linear interpolator, which needs 3 multipliers, is needed in that approach. For reasons of routability one may use 6 bi-linear interpolators, each connected to four memories. The calculation of the light intensity can however be done in a much more appropriate way using two 1D tables, which model the intensity of the highlight, which can be addressed simultaneously with the proper angle.

A volume rendering application which applies a step-function ranging from 0 to 1 as alpha-threshold will visualize an iso-surface. This application can always use early ray termination and it needs to

calculate just one gradient on each ray. It can be programmed in a totally different way on the FPFA, to take advantage of this fact.

It is a considerable advantage of the concept of the FPFA that one can change its functionality on the fly. Compare this with a dedicated engine. For instance a surface renderer which can map textures needs the presence of texture mapping hardware, even when this feature is not used. In contrast, the texture mapping functionality can be dynamically loaded on an FPFA. This gives the user the unique opportunity to use sheer arithmetic power as needed, at extremely low levels of energy dissipation and in the most optimal form.

7 Conclusions

The concept of the FPFA extends the ideas of the Field Programmable Gate Array from logic functions to arithmetic functions. It has been shown that there is a high risk that on-chip memories and long interconnect are dominating the performance of a dedicated graphics chip. The energy models used in the introduction were derived independent from any specific library. As a consequence, the conclusions presented are independent of specific design system constraints. A study of a wide range of libraries and on-chip RAMs has given raise to a lot of disappointment in the past, as the on-chip memories, which are needed for most graphics applications, showed to be worse than assumed. This has as implication that not only the *architecture* should be adapted to low-power, the *library* needs adaptation as well. As long as this issue has not been settled by the foundries, we will have to work either with sub-optimal FPFAs or with partially full-custom designs. The high regularity of FPFAs and their high performance for graphics applications makes the extra effort however worth-wile.

8 References

[1] H.Pfister, A. Kaufman and F.Wessels, "Towards a Scalable Architecture for Real-Time Volume Rendering", *Proceedings of the 10th Eurographics Workshop on Graphics Hardware*, volume EG95HW, pp 123-130. EuroGraphics Workshop Proceedings Series, 1995.

[2] G.Knittel, "A Scalable Architecture for Volume Rendering", *Proceedings of the 9th Eurographics Workshop on Graphics Hardware*, volume EG94HW, pp 58-69. EuroGraphics Workshop Proceedings Series, 1994.

[3] J. Lichterman, "Design of a Fast Voxel Processor for Parallel Volume Visualization", *Proceedings of the 10th Eurographics Workshop on Graphics Hardware*, volume EG95HW, pp 83-92. EuroGraphics Workshop Proceedings Series, 1995.

[4] S.E.Molnar, J. Eyles, and J.Poulton, "Pixelflow: High-Speed Rendering Using Image Composition", *Computer Graphics*, 26(2) pp. 231-240, July 1992.

[5] TMS320C80 Multimedia Video Processor (MVP), Texas Instruments, Digital Signal Processing Products series, 1994.

[6] A.P. Chandrakasan and R.W. Brodersen, *Low Power Digital CMOS design*, Kluwer Academic Publishers, Norwell MA, 02061 USA and Dordrecht the Netherlands.

[7] A. Bellaouar and M. I. Elmasr, *Low Power Digital VLSI Design*, Kluwer Academic Publishers, Norwell MA, 02061 USA and Dordrecht the Netherlands.

[8] D.J. Kinniment, J.D. Garside, and B. Gao, "A Comparison of Power Consumption in Some CMOS Adder Circuits", *Proceedings of PATMOS*, Eds C. Piguet, W. Nebel, pp. 106-118. ISBN 3-8142-0526-X

[9] M. Bentum, *Interactive Visualization of Volume Data*, Phd. Thesis, University of Twente. ISBN 90-9008788-5.

[10] M. Bosma and J. Terwisscha van Scheltinga, *Efficient Super Resolution Volume Rendering*, Masters thesis, University of Twente, August 1995.

[11] J. Smit and J.H. Huisken, "On the energy complexity of the FFT", *Proceedings of PATMOS*, Eds C. Piguet, W. Nebel, pp. 119-132. ISBN 3-8142-0526-X

[12] M. Bosma, J.Smit, and J. Terwisscha van Scheltinga, "Superresolution Volume Rendering Hardware", *Proceedings of the 10th Eurographics Workshop on Graphics Hardware*, volume EG95HW, pp 117-122. EuroGraphics Workshop Proceedings Series, 1995.

[13] J. Terwisscha van Scheltinga, J.Smit, and M. Bosma, "Design of an on-chip reflectance map", *Proceedings of the 10th Eurographics Workshop on Graphics Hardware*, volume EG95HW, pp. 51-55. EuroGraphics Workshop Proceedings Series, 1995.