A Multipurpose Hardware Shader

Josef Pöpsel, Eckard Tikwinski

ABSTRACT For the last few years the state of the art in producing three-dimensional Gouraud shaded graphics has been the use of Gouraud shading hardware. Combined with z-buffering it enables graphic workstations to provide real time display. The disadvantages of this approach are the relatively poor quality of Gouraud shaded images and its still very high cost, which so far prohibit real-time applications running on standard PCs. This article describes a universal approach to provide a very powerful graphic unit using minimal hardware at very low cost. This graphic unit will not only support Gouraud shading, but also such methods as 2D-texturing [3], solid texturing [10], normal (bump) texturing [2], shadow mapping [15, 13] and Phong shading [11] as well as a combination of these methods (shade trees [5]).

1 Introduction

In parallel with the increase in hardware power within the last 20 years, graphics quality has been enhanced. The simple line drawings which were common some years ago have been succeeded by today's standards for shaded objects (PHIGS+/PEX) [6]. For real-time applications (e.g., simulation) only Gouraud shading [8] combined with the z-buffer algorithm are currently possible. Other algorithms (Scan-Line algorithm [9], Raytracing [14,1] or Radiosity [7]) cannot be used because of the tremendous computing power they require [4]. In contrast to Gouraud shading, which is distinguished by the very simple and steady algorithm it uses, the other methods mentioned above cannot (yet) be realized in hardware. However, the disadvantage of Gouraud shading is the poor quality that can be achieved (Machband effects, poor detailing level), which is far from photo realism. In recent years various methods have been published to evade this drawback:

- With 2D-texturing [3], a two-dimensional picture is 'glued' onto the shaded surface. Every single point of the picture of an object is, after the shading, modulated with the colour of the texture at the corresponding place. An advertising pillar is a good example for 2D-texturing.

- With 3D-texturing (solid texturing) [10] the displayed object is 'milled from the block'. After shading the object, the colour is modulated with a function which depends on the object's three-dimensional coordinates. A marble pillar is an example for 3D-texturing.

- With normal texturing (bump texturing) [2], the normal of a texture modulates the normal of a point of the object. Thereafter the point is illuminated. Modulating the normal produces optical effects such as a golf ball surface. It has to be pointed out here that normal texturing can only be used in combination with Phong shading [11], because the value of the normal vector must be known for each pixel.

- A very general method is provided by so-called shade trees [5], where a freely definable function is called for each point, which alters the appearance of the object. The input parameters of this function are calculated by linear interpolation inside the polygons of the object.

This list of special methods can be substantially extended.

In the end, all of these methods have something in common: they interpolate over a polygon a number of parameters (some use more, others less parameters) and call a more or less complex function to calculate the colour of the pixel.

2 Concept

If we define $C|_{x,y}$ as the colour of the pixel (x,y), then

$$C|_{x,y} = f(P_1(x,y), P_2(x,y), ..P_n(x,y))$$

where f represents the function used by the displaying method. P_i stands for the i-th of the n input parameters, where $P_i(x,y)$ is determined by the values at the polygon corners.

Below, two examples are given for Gouraud shading with and without 2D-texture:

- For Gouraud shading with $n = 4$ (r, g, b, z) we can write

$$C|_{x,y} = f\begin{pmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{pmatrix}$$

$$\text{with} \quad f = \begin{pmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{pmatrix} \quad \text{for} \quad z(x,y) > \textit{z-buffer(x,y)}$$

- For Gouraud shading with 2D-texture and $n = 6$ (r, g, b, z, t_u, t_v)

$$C|_{x,y} = f\begin{pmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{pmatrix}$$

$$\text{with} \quad f\begin{pmatrix} k * r(x,y) + \overline{k} * TB_{r,g,b}(t_u, t_v) \\ k * g(x,y) + \overline{k} * TB_{r,g,b}(t_u, t_v) \\ k * b(x,y) + \overline{k} * TB_{r,g,b}(t_u, t_v) \end{pmatrix} \quad \text{for} \quad z(x,y) > \textit{z-buffer(x,y)}$$

where $TB_{r,g,b}(t_u, t_v)$ represents the colour of the texture at position (t_u, t_v), k and $\overline{k}$ are modulation constants $(k + \overline{k} \equiv 1)$.

If one separates the evaluation of the shading function f from the interpolation, the calculation can be performed by low-cost signal processors with sufficient speed. However, a precondition is that the front-end hardware interpolator preprocesses the data especially for the signal processor to free the signal processor from address calculation and similar administration tasks. For very simple functions the signal processor is not required at all. Using Gouraud shading, for example, without any textures, the pixel colour is identical with the results of the interpolation. The required z-buffer comparison is performed by hardware.

3 The Universal Shader

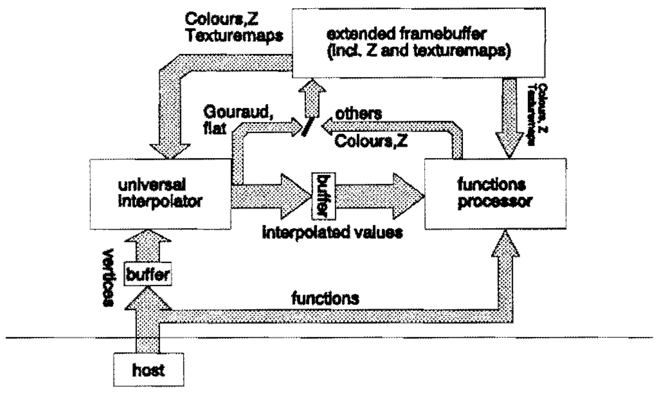


Fig. 1. Data flow of the universal shader

Figure 1 shows a data flow chart containing essentially four elements:

- host computer
- universal interpolator
- functions processor
- extended frame-buffer

1. The host computer transfers the polygon corner vertices to the universal interpolator as well as a particular shading function (as program code) to the functions processor. This shading function remains constant over a great number of polygons.

2. From the vertices of the polygon, the universal interpolator evaluates the parameters for all pixels inside the polygon by bilinear interpolation. These parameters and all necessary information are then passed on to the functions processor.

   There is also a direct link to the extended frame-buffer. Therefore, the interpolation results of the Gouraud and flat shading can be transferred directly to the frame-buffer, by-passing the functions processor.

   The universal interpolator can be realized in hardware (ASIC) with minimal effort. It mainly consists of a microprogram controlled arithmetic unit and control logic. The initialization and programming of the microcode is performed by another signal processor, which has to be regarded functionally as a part of the interpolator. The interpolator also has two identical interfaces to the frame-buffer to allow access to two different areas without the need to reload addresses when changing from one area to the other.

   Particularly for Gouraud shading there are four parallel adders, which can deliver the colour and z-value for one pixel per clock cycle.

3. The functions processor is a standard signal processor which calculates the shading functions.

4. The extended frame-buffer contains, in addition to the actual display memory (double-buffered, 24 bit), the z-buffer including z-comparator as well as memory space for texture-maps or, generally speaking, for two-dimensional arrays (e.g., normals for bump texturing). Using address shifting, this memory area can also be utilized as frame-buffer (hardware panning).

The links

$$\text{host} \rightarrow \text{universal interpolator}$$

and

$$\text{universal interpolator} \rightarrow \text{functions processor}$$

are realized using FIFOs, in order to be able to continuously load the individual function blocks.

For a better understanding of the universal shader we now go more into detail using the refined data flow shown in Figure 2.

The universal interpolator, which is the kernel of the universal shader, consists of three logical components:

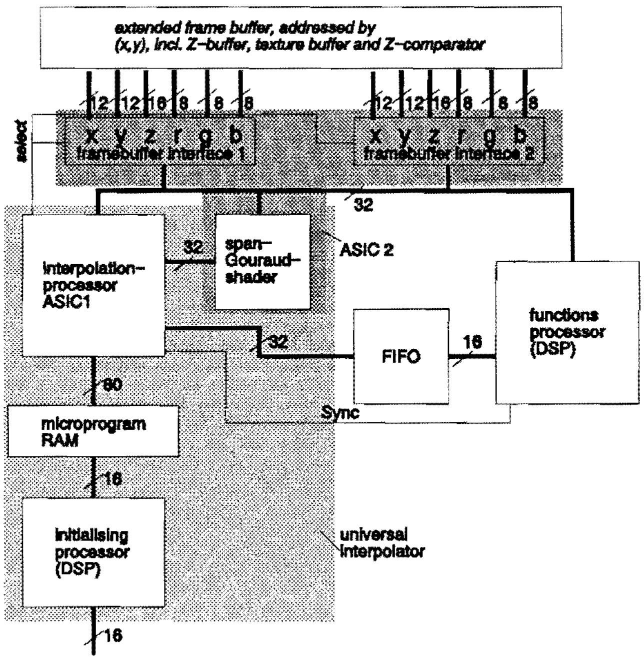
- One microcode programmable, highly parallel processor, whose structure is optimized particularly for the many interpolation operations required (ASIC, four-level pipelined, 20 MHz)
- The initialization signal processor which provides the microcode program for the interpolation processor after transferring the required variable parameters evaluated from the corner point information of the polygons to be displayed.
- A span Gouraud shader, which can, on its own, interpolate n times in parallel r, g, b and z. In the contrary to the interpolation processor, which can perform only one interpolation per clock, the span processor needs only one clock per pixel. Using pure Gouraud shading, the span shader is used as accelerator (ASIC, 20 million pixels/sec).

There is a twofold link between the universal interpolator and the extended frame-buffer. Each interface has registers for R,G,B (with pre-boundary check and clipper for 0-255) and Z, as well as up and down counters for x and y. The doubled interface is particularly useful for source-destination operations like window move.Either of the two interfaces can be selected by the functions processor as well, instead of the universal interpolator, to give the functions processor full access to the extended frame-buffer. Both interfaces are integrated into the span shader ASIC.

The dual-ported RAM linking the universal interpolator and functions processor has a width of 32 bits and a depth of 1024 words.

3.1 The Interpolation Processor

The interpolation processor consists of an 80-bit wide, 1024 deep dual-ported RAM (combined microprogram memory and buffer), an arithmetic unit and a control unit. Every instruction is performed in one clock cycle with the exception of conditional jumps (2 clock cycles). One instruction performs up to six parallel operations. One microprogram word (80 bits) contains data as well as instruction code. The data part is split into the two segments DW1 and DW2 (each 32 bits wide). The remaining 16 bits represent the code for



extended frame buffer, addressed by (x,y), incl. Z-buffer, texture buffer and Z-comparator
12 12 16 8 8 8
x y z r g b
framebuffer interface 1
framebuffer interface 2
select
32
span-Gouraud-shader
ASIC 2
interpolation-processor ASIC1
FIFO
16
functions processor (DSP)
Sync
80
microprogram RAM
16
initialising processor (DSP)
universal interpolator
16

Fig. 2. Refined data flow chart of the universal shader

the available instructions, which are divided into six groups for the respective six parallel instructions.

The word format is shown in Table 1:

Table 1. Microprogram word format

bits	0..31	32..63	64..67	68	69..70	71..75	76..79
contents	data word 1	data word 2	primary instruction	frame-buffer interface select	frame-buffer access type	interface register select	interface counter manipulation
name	DW1	DW2	PRI	FIS	FZA	IRS	ICM

Primary Instructions (PRI, 4 bits):

The following primary instructions have been implemented so far:

IP (interpolate)	performs 32-bit signed addition DW1:=DW1+DW2
ISPC (interpolate span count)	if last IP instruction did not cause a carry from bit 15 to 16, same as IP, else DW1:=DW1+DW2-$10000
MV (move)	copy data if bit31(DW1) and bit31(DW2) then program[DW1].DW2:=program[DW2].DW2 else if not bit31(DW1) and bit31(DW2) then program[DW1].DW2:=program[DW2].DW1 else if bit31(DW1) and not bit31(DW2) then program[DW1].DW1:=program[DW2].DW2 else if not bit31(DW1) and not bit31(DW2) then program[DW1].DW2:=program[DW2].DW2

WC
(write constant)
	if bit31(DW2)
	then program[DW2].DW2:=DW1
	  else program[DW2].DW1:=DW1

RX, RY, RZ, RR, RG, RB (read X,Y,Z,R,G,B)	read values from the interface selected in DW1 (starting from bit 16, bit 0-15:=0)

REP
(repeat)
	repeat function
	DW1:=DW1-1;
	if DW1 $\geq$ 0 then goto program[DW2]

SYNC1	wait until the SYNC1 instruction is overwritten by the initializing signal processor (DSP). This is used for synchronization of the initializing DSP with the interpolation processor.
SYNC2	sends the instruction to activate the sync-signal to the functions processor and waits for this event. This is used to synchronize the functions processor with the interpolation processor.
'_'	no primary instruction is performed.

Frame-buffer interface select (FIS, 1 bit):

Possible values: 1,2

Selects first or second interface between the universal interpolator and extended frame-buffer, which will be used for succeeding operations.

Frame-buffer access type (FZA, 2 bits):

Possible values: FW, FWZ, FR, -

FW	Writes the values of the selected interface into the frame-buffer. No z-comparison is performed.
FWZ	Same as FW, but z-comparison is performed.
FR	Reads the extended frame-buffer and writes to both (!) interfaces.
-	No frame-buffer access.

Frame-buffer accesses, together with frame-buffer interface select, are always, without exception, executed as the first step in the microprogram (highest priority).

Interface register select (IRS, 4 bits):

Possible values: WX, WY, WZ, WR, WG, WB; SR, SG, SB, SZ, SDR, SDG, SDB, SDZ; SN; SN8; FP; -

WX-WB	Writes DW1 into the respective registers of the interface selected by FIS, before (!) the primary instruction is executed. Only bits 16-32 (digits to the left of the decimal point) are transmitted. R, G, B are limited to the range of 0-$FF.
SR-SDZ	Writes DW1 into the respective registers of the span Gouraud shader, before (!) the primary instruction is executed. SR-SZ are the starting values of the span, SDR-SDZ represent the gradients of R, G, B and Z within the span.
SN	Transmits the number of pixels to be interpolated (=DW1) to the span shader. SN also synchronizes the interpolation processor with the span shader. Writing SN into the span shader causes n-times execution of the span interpolation, if the span shader is ready. Otherwise the interpolation processor will wait. This is executed before (!) the primary instruction.
SN8	Used for fast erasing of the extended frame-buffer (80 million pixels/sec). DW1 blocks sized 4*2 pixels are written. The selected interface is used as frame-buffer address, where the values for SR, SG, SB, SZ are written to the pixels $(x*4, y)..(x*4+3, y), (x*4, y+1)..(x*4+3, y+1)$. The values for SR-SZ are not altered.
FP	Transmits DW1 to the buffer between interpolation processor and functions processor before (!) the primary instruction is executed.
'-'	No register is selected.

The instructions FP and '-' are not coded with the four IRS-bits (lack of available bits), instead the four ICM-bits are used (see below).

Interface counter manipulation (ICM, 4 bits):

Possible values: X+, Y+, X-, Y-, XY+, XY-, X+Y-, X-Y+, -

The counters of the selected interfaces are increased respectively decreased after (!) execution of the primary instruction (lowest priority).

Thus a program step contains the instruction combinations shown in Table 2.

Table 2. Instruction combinations

DW1	DW2	PRI	FIS	FZA	IRS	ICM
initial value	increment	IP, IPSC	1	FW	WX, WY, WZ	X+, Y+
			2	FWZ	WR, WG, WB	X-, Y-
src addr	dst addr	MV	‘-’	FR	SR, SG, SB	XY+, XY-
constant	dst addr	WC		‘-’	DSR, DSG, DSB	X+Y-, X-Y+
repeat counter	jump	REP			SN, SN8	‘-’
		SYNC1			FP	
					‘-’	
		SYNC2				
		‘-’				

Using the described instructions, it possible to generate optimized programs for the interpolation processor, which can become quite simple if the generation is supported by software. Those programs are stored in the initialisation processor and, when required, copied (by the initialization processor) into the 80-bit dual-ported RAM (after the variable data are entered). The initialization processor regards the dual-ported RAM as organized into 16-bit $\times 5120$, so that the data can be manipulated quite easily.

The overall hardware requirements are modest, compared with the achieved flexibility and execution speed. Apart from the frame- and z-buffer, which are always required, there are two ASICS, two low-cost standard DSPs, 10 kbyte very fast dual-ported RAM (20 ns) as microprogram memory and buffer, and 4 kbyte dual-ported RAM (50 ns) as buffer between interpolation processor and functions processor.

3.2 Examples

Finally, some sample programs are given to demonstrate the performance power of the interpolation processor. Data which are entered into the microprogram by the initialization processor are underlined.

Example 1: Draw a window with constant colour and constant Z

(Start with X,Y, width Nx, height Ny, colour (R,G,B), z-value Z)

Line	Label	DW1	DW2	PRI	FIS	FZA	IRS	ICM
1		R.0	0	-	-	-	SR	-
2		G.0	0	-	-	-	SG	-
3		B.0	0	-	-	-	SB	-
4		Z.0	0	-	-	-	SZ	-
5		X.0	0	-	-	-	WX	-
6	Loop:	Y.0	2.0	IP	1	-	WY	-
7		Nx/4.0	0	-	1	-	SN8	-
8		Ny/2.0	Loop	REP	-	-	-	-

Lines 1-4	write R,G,B and Z into the span shader
Line 5	writes x into interface 1
Line 6	writes y into interface 1 and then increments y by 2
Line 7	generates Nx/4 pixel blocks (4*2) next to each other in the desired colour
Line 8	causes Ny/2-times repetition. The functions processor stays idle during this action.

The above program can clear a 1280 × 1024 pixel screen within 1.6 ms.

Example 2: Window copying

Starting from (X1,Y1) to (X2,Y2), width Nx, height Ny. Windows may overlap, yet with restrictions.

Line	Label	DW1	DW2	PRI	FIS	FZA	IRS	ICM
1		Ny.0	Rep2	WC	-	-	-	-
2		Y1.0	0	-	1	-	WY	-
3		Y2.0	0	-	2	-	WY	-
4	Edge:	Nx.0	Repl	WC	-	-	-	-
5		X1.0	0	-	1	-	WX	-
6		X2.0	0	-	2	-	WX	-
7	Span:	0	0	-	1	FR	-	X+
8	Repl:	0	Span	REP	2	FW	-	X+
9		0	0	-	1	-	-	Y+
10	Rep2:	0	Edge	REP	2	-	-	Y+

The inner loop (lines 7,8) needs 3 clock cycles. Thus, using a 20 MHz clock cycle, a window of 100 × 100 pixels can be copied in 1.5 ms.

Example 3: Gouraud shaded triangle

(base line horizontal, top point above base line, top point colour (R,G,B), top point coordinates (X,Y,Z), height Yh, variations of the left edge dEX,dER,dEG,dEB,dEZ, variation of the triangle width dN, variations within a span dSR,dSG,dSB,dSZ (remains constant (!) within a triangle for all spans)).

Line	Label	DW1	DW2	PRI	FIS	FZA	IRS	ICM
1		dSR	0	-	-	-	SDR	-
2		dSG	0	-	-	-	SDG	-
3		dSB	0	-	-	-	SDB	-
4		dSZ	0	-	-	-	SDZ	-
5		Y.0	0	-	1	-	WY	-
6	Edge:	Z.5	dEZ	IP	-	-	SZ	-
7		R.5	dER	IP	-	-	SR	-
8		G.5	dEG	IP	-	-	SG	-
9		B.5	dEB	IP	-	-	SB	-
10		X.5	dEX	IP	1	-	WX	-
11		0.5	dN	IPSC	-	-	SN	-
12		Yh.0	Edge	REP	-	-	-	Y+

Lines 1-4	load delta-registers of the span shader.
Line 5	sets Y start address.
Lines 6-10	write the starting values of the spans into the span processor and then perform the interpolation (edge interpolation).
Line 11	writes the number of pixels for the current span into the span processor and then starts the span interpolation. The functions processor stays idle during this action, as in the previous example.

One triangle containing 100 pixels (e.g., X=5, Y=0, Yh=10, dN=2.0, dEx=-1.0) is calculated within 116 clock cycles. If one assumes that the microprogram parameters for the next triangle can be generated by the initialization processor within this time, then the described hardware will be able to draw approximately 172 000 triangles/sec.

Example 4: Gouraud shading + transparency + texture

This example demonstrates how shading methods which, up to now, have had to be implemented in software (if at all), and therefore were very time-consuming, can be performed in considerably less time. A good example for such an application is a coloured air-balloon.

To simplify the matter, only the inner loop of a triangle is shown, which is required for one span.

Line	Label	DW1	DW2	PRI	FIS	FZA	IRS	ICM
n	Span:	ZS.	dZS	IP			FP	
n+1		0	0	RZ	1	FR	WX	
n+2		texX.	dtexX	IP	2		WY	
n+3		texY.	dtexY	IP			FP	
n+4		RS.	dRS	IP			FP	
n+5		0	0	RR	1		FP	
n+6		0	0	RR	2	FR	FP	
n+7		GS.	dGS	IP			FP	
n+8		0	0	RG	1		FP	
n+9		0	0	RG	2		FP	
n+10		BS.0	dBS	IP			FP	
n+11		0	0	RB	1		FP	
n+12		0	0	RB	2		FP	
n+13		NX	Span	REP	1			X+

n	Z-interpolation, then, after transfer to the functions processor, reading of the z-value from the extended frame-buffer. Thereafter transfer to the functions processor.
n+1	Reading of the z-value from the extended frame-buffer, then transfer to the functions processor.
n+2, n+3	Writing the texture coordinates in interface 2, then interpolation of the texture coordinates.
n+4	Transfer of the red component (Gouraud), then interpolation of 'red'.
n+5	Reading of the frame-buffer (red component of the background), then transfer to the functions processor.
n+6	Reading of the frame-buffer (red component of the texture), then transfer to the functions processor.
n+7..n+9	Steps n+4..n+6 for green component.
n+10..n+12	Steps n+4..n+6 for blue component.
n+13	Repeat n..n+12 for all pixels within one span.

The program, running on the functions processor to perform the span interpolation, looks like this:

```
read z1
read z2
if z1 > z2 then begin
      read rGour
      evaluate r:=rGour*k1
      read rBack
      evaluate r:=r+rBack*k2
      read rTexture
      evaluate r:=r+rTexture*k3
      ... the same steps for g
      ... the same steps for b
      write z1,r,g,b, into frame-buffer
end else begin
      read 9 parameters
end
```

Because the signal processor used can access memory parallel to executing arithmetic operations, and, furthermore, can perform the operation $x := x + k \times x\prime$ in one single clock cycle (MAC), it takes only 15 clock cycles (100 ns each) for one pixel. The universal interpolator needs 14 clock cycles (50 ns each) for the equivalent action. This leaves the functions processor as the bottleneck. However, using this quite costly shading method, it is still possible to achieve a pixel-rate of approximately 650000/sec (about 6500 triangles/sec).

The list of methods can be extended at will.

4 Summary

In this paper a circuit was introduced which makes it possible very flexibly and also very efficiently to draw polygons by any given, freely programmable shading function. This is achieved using a combination of special hardware and low-priced signal processors. Taking into account the advantages of ASICs, the circuitry can be realized in a PC-environment at very low cost. With the hardware described, de facto standards like RenderMan [12], which are very demanding concerning the flexibility of the hardware shader, can possibly run in real time.

Acknowledgements

I wish to thank U. Claussen and Ch. Hornung for their ideas contributing to the development of the conception. Acknowledgements to K. G. Vits for the translation, and, last but not least, many thanks to B. Groth who created the necessary productive environment by the supply of coffee and biscuits.

References

[1] Amanatides, J.: Ray tracing with cones. *Computer Graphics* 18(3): 129-135, July 1984.

[2] Blinn, J. F.: Simulation of wrinkled surfaces. *Computer Graphics* 12(3): 286-292, August 1978.

[3] Blinn, J. F. and Newell, M. E.: Texture and reflection in computer generated images. *Communications of the ACM* 19(10):542-547, October 1976.

[4] Claussen, U.: Verfahren zur schnellen Beleuchtungs- und Schattierungsberechnung. *Dissertationsarbeit*, Fakultät Physik, Universität Tübingen, Germany 1990.

[5] Cook, R. L.: Shade trees. *Computer Graphics* 18(3): 223-231, July 1984.

[6] van Dam, A.: PHIGS+ functional description revision 3.0. *Computer Graphics* 22(3): 125-218, July 1988.

[7] Greenberg, D. P., Cohen, M. F. and Torrance. K. E.: Radiosity: a method for computing global illumination. *The Visual Computer* 2: 291-297, 1986.

[8] Gouraud, H.: Continuous shading of curved surfaces. *IEEE Transaction on Computers* C-20(6):623-628, June 1971.

[9] Newman, W. M. and Sproull, R. F.: *Principles of interactive computer graphics*. Springer, 1986.

[10] Peachey, D. R.: Solid texturing of complex surfaces. *Computer Graphics* 19(3): 279-286, July 1986.

[11] Phong, B. T.: Illumination for computer generated images. *Communications of the ACM* 18(6): 311-317, June 1975.

[12] *The RenderMan Interface.* Pixar, San Rafael, CA, edition 3.0, May 1988.

[13] Reeves, W. T., Salesin, D. H. and Cook, R. L.: Rendering antialiased shadows with depth maps. *Computer Graphics* 21(4): 283-291, July 1987.

[14] Whitted, T.: An improved illumination model for shaded display. *Communications of the ACM* 23(6): 343-349, June 1980.

[15] Williams, L.: Casting curved shadows on curved surfaces. *Computer Graphics* 12(3): 270-274, August 1978.