

A New Simple and Efficient Antialiasing with Subpixel Masks

Andreas Schilling
Universität Tübingen
Bundesrepublik Deutschland *

Abstract

Antialiasing of edges is often performed with the help of subpixel masks that indicate which parts of the pixel are covered by the object that has to be drawn. For this purpose, subpixel masks have to be generated during the scan conversion of an image. This paper introduces a new algorithm for creating subpixel masks that avoids some problems of traditional algorithms, like aliasing of high frequencies or blinking of small moving objects. The new algorithm can be implemented by lookup tables that make use of the inherent symmetry of the algorithm. The results are compared with conventional supersampling¹. A hardware implementation is described.

CR Categories and Subject Descriptors: I.3.1 [Computer Graphics]: Hardware Architecture - *raster display devices*; I.3.3 [Computer Graphics]: Picture/Image generation - *display algorithms*

Additional Key Words and Phrases: antialiasing, exact area subpixel algorithm.

1 Introduction

The use of a subpixel mask for antialiasing purposes is very common [6, 7, 3, 9, 1]. It has several advantages compared with other antialiasing techniques. First, it preserves spatial information that is lost, when other methods are used. This information is very important

if more than two objects contribute to a pixel. Second, it can be determined quite easily. At the end of the rendering process, the final brightness of the pixel can be calculated easily by adding up the contributions from the subpixels.

However, the subpixel mask is usually computed in a way that introduces avoidable errors. In the following section, the artifacts that occur when using supersampling are shown. Analyzing these artifacts leads to a simple algorithm, the Exact Area Subpixel Algorithm, that can avoid the described disadvantages by combining the benefits of the subpixel mask with the calculation of the exact pixel coverage.

2 The Problem

The easiest and therefore most common way to determine the subpixel mask is by sampling at the subpixel centers. The subpixel is on, if the subpixel center is inside the polygon, otherwise it is off (Of course, the results of this calculation can be precomputed and stored in a lookup table).

The problems with that method are obvious. Consider for example the following case (Fig. 1). The pixels consist of 4×4 subpixels. An object with a horizontal lower edge is moving slowly downward across the pixel. Nothing happens, until the edge reaches the top-most line of subpixel centers. As soon as the subpixel centers are reached, all the four upper subpixels are switched on at the same time. As a result, the brightness of the pixel is increased in four big steps instead of the 16 steps, which we would like to obtain with 16 subpixels. This is shown in Fig. 2 a), where the number of subpixels that are set is plotted as a function of the exact area, covered by the polygon.

The same applies for diagonal lines. In Fig. 2 b) we see the result for edges with a slope of 45° . Fig. 2 c) represents the ideal function that we would like to obtain. This function is achieved with the algorithm introduced in the next section.

If the objects are very small (smaller than a subpixel),

*Wilhelm-Schickard-Institut für Informatik, Graphisch-Interaktive Systeme, Auf der Morgenstelle 10/C9, 7400 Tübingen, E-mail: andreas@gris.informatik.uni-tuebingen.de.

¹The experiences, described here were gained in a research project, partly supported by the Commission of the European Communities through the ESPRIT II-Project SPIRIT, Project No. 2484.

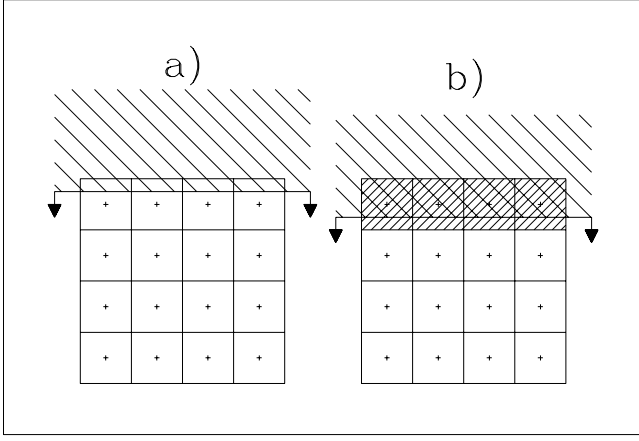


Figure 1: Problems with oversampling: horizontal edge moving downward over a pixel, consisting of 4×4 subpixels. The first line of subpixels is switched on simultaneously.

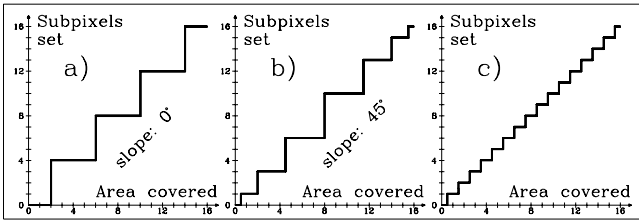


Figure 2: Number of set subpixels vs. area covered. a) Result of supersampling with a horizontal or vertical line, b) Supersampling with a diagonal line, c) Ideal function.

the effect of the errors of the conventional approach is especially bothersome. The whole object appears and disappears again as it moves across the screen. A thin line or the end of a skinny triangle appears as a dashed line (Fig. 3).

A way to minimize aliases, generated by regular supersampling, is the use of stochastic sampling [5, 1, 4]. Although the most annoying artifact (alias effect) is replaced by an artifact (noise) that is more tolerable, other objectionable effects, like the blinking of small moving objects or holes in thin lines etc. are not dealt with correctly.

3 Exact Area Subpixel Algorithm

Our new approach, therefore does not sample at the subpixel centers. Instead, the exact portion of the pixel area is calculated. But in contrast to other approaches that also calculate this area, but then have to store it as an extra value for later processing [3], we convert the area into a subpixel count that represents the area portion best. In the following, the algorithm is explained for 4×4 subpixels, later on, a generalization for $n \times n$

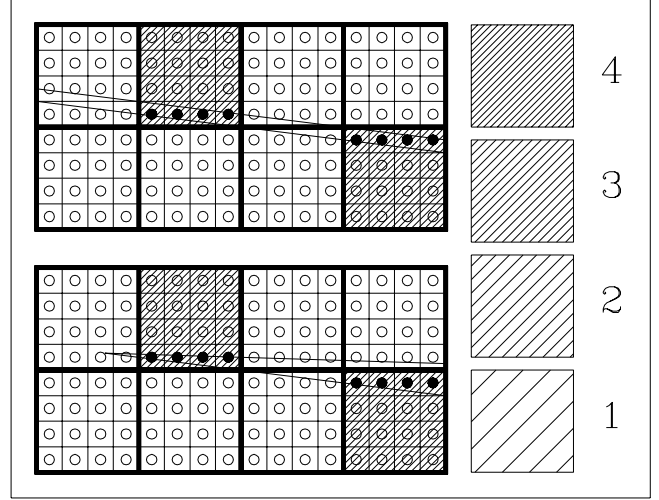


Figure 3: Problems with oversampling (thin line and sharp triangle). Each pixel consists of 4×4 subpixels. Subpixels that are set are indicated by filled circles. The density of the hatching indicates the final pixel brightness.

subpixels is presented. Let us look again at the horizontal edge, mentioned above. With the new approach, we get a coverage of one subpixel, when the line is $1/16$ of the pixel width under the upper pixel edge (in fact we get one subpixel between a coverage of $1/32$ and $3/32$ of the pixel area). If the line has reached $2/16$, we get a coverage of 2 subpixels instead of 0 or 4 with the old method.

Now, the only problem that remains to be solved is to find the right locations for those subpixels. A simple observation can help us. If we look at an edge of a certain slope that moves slowly over a pixel, we see that the subpixels are touched in a certain order. We would get the same sequence, if we observed the order, in which the subpixel centers are covered, or the order in which the subpixels are covered totally (e.g. Fig. 4 c) or d)). Normally it starts with a subpixel at one corner. The next in the row will be the neighbors of the first one and so on until the opposite corner is reached. Now we can show, that there is only a certain number of different orders that are possible. In Fig. 4 we see, that all edges that have a slope between that of E_1 and E_2 cover the subpixels in the same order. The total number of possible sequences is only 32 for a subpixel mask of 4×4 subpixels. Now we can assign one of these sequences to each slope. At the borders, where two sequences would be possible, we choose one and make sure that an edge of the opposite orientation gets the opposite sequence. Such we can ensure that two adjacent objects that cover the whole pixel cause all subpixels to be set in any case. In Fig. 4 a) the numbering scheme for the subpixels is shown. Fig. 4 b) shows the different

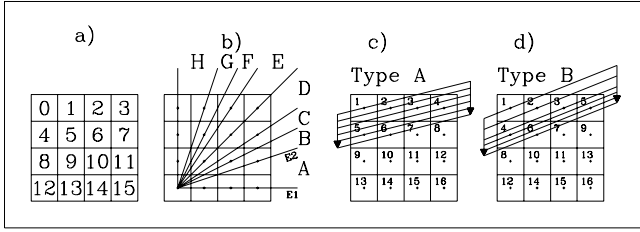


Figure 4: Numbering scheme and possible sequences for subpixel coverage. (See Section 3 for explanation)

sequences that are possible for edges between 0^0 and 45^0 . As examples, lines of type A and B are drawn in Fig. 4c) and d) resp. The figures show how the subpixel centers are touched by the edges in different sequences.

4 Exact Area Subpixel Algorithm for $n \times n$ subpixels

In the following, we have to make use of an arbitrary representation of the polygons. The representation with edge functions, chosen for this paper is explained in Appendix A. For the $n \times n$ subpixel mask, the approach is the same as in the 4×4 case. First we calculate the number of subpixels we want to set (n_{os}) with the general formula²:

$$n_{os} = \begin{cases} \text{floor}(A * n^2 + 0.5) & : 0 \leq \text{slope} < 180^\circ \\ \text{ceil}(A * n^2 - 0.5) & : 180^\circ \leq \text{slope} \end{cases}$$

A denotes the area of the pixel that is covered by the half plane defined by the polygon edge. (The two cases have to be distinguished in order to ensure that two adjacent polygons always complement each other, i.e. that we don't get a total coverage of more than $n \times n$ subpixels.)

The next step is a sort of the subpixels $s_1 \dots s_{n \times n}$. They are sorted in a way that the most covered one is the first, the least covered one the last in the list (falling edge function). If there exist groups of subpixels with equal coverage (equal edge function), the subpixels of these groups are sorted using a function that results from the original edge function through a rotation by $+90^\circ$.

Now we take the first n_{os} subpixels from the list and set them to 1, all others to 0.

In a final step, the resulting subpixel masks for all edges of the (convex) polygon are ANDed together.

5 Hardware Implementation

Several problems had to be solved in order to implement the coverage unit in hardware. The function of

² $\text{floor}(x)$ means the largest integer not greater than x , $\text{ceil}(x)$ means the smallest integer not less than x .

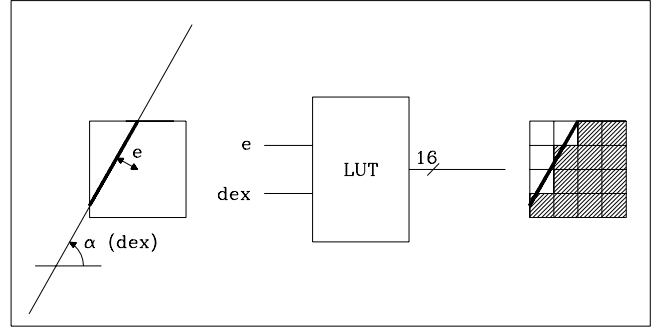


Figure 5: Lookup table for the subpixel mask.

	ES	E18	E17	E16	E15	E14	E13	E12	E11	E10	Result with trunc.	Result wanted
D	0	1.00	0	:	0	:	:	:	:	:	D	D
C	0	0.11	1	:	0	:	:	:	:	:	D	D
B	0	0.11	0	:	0	:	:	:	:	:	C	C
A	0	0.10	1	:	0	:	:	:	:	:	B	B
A'	0	0.01	1	:	0	:	:	:	:	:	A	A
B'	0	0.01	0	:	0	:	:	:	:	:	A'	A'
C'	1	1.11	1	:	0	:	:	:	:	:	B'	B'
D'	1	1.11	0	:	1	:	:	:	:	:	C'	C'
	1	1.10	1	:	1	:	:	:	:	:	D'	D'
	1	1.10	0	:	1	:	:	:	:	:	D'	D'
	1	1.01	1	:	0	:	:	:	:	:		
	1	1.01	0	:	0	:	:	:	:	:		
	1	1.00	1	:	0	:	:	:	:	:		
	1	1.00	0	:	0	:	:	:	:	:		

Figure 6: Problems with truncating numbers represented in the two's complement.

the unit is simple. We can think of it as a single lookup table like the one shown in Fig. 5. Input parameters are the distance or error term e and the x-increment of the edge function de_x , which serves as a measure for the slope of the edge. In addition to that we need only the sign of the y-increment de_y .

The first question was the required precision of the input parameters. Under the condition that the result of the quantization error has to be smaller than one subpixel we get required resolutions for the distance or error term e of 5 bits and for the slope measure de_x of 4 bits. The condition for the slope measure with regard to the sequence leads to the same resolution of 5 bits for de_x . In order to cover all directions, the signs of de_x and de_y have to be considered also. As a result, we get a total of 11 input bits which correspond to a lookup table size of $2k \times 16$ bits.

But if we look into the details, we still have to in-

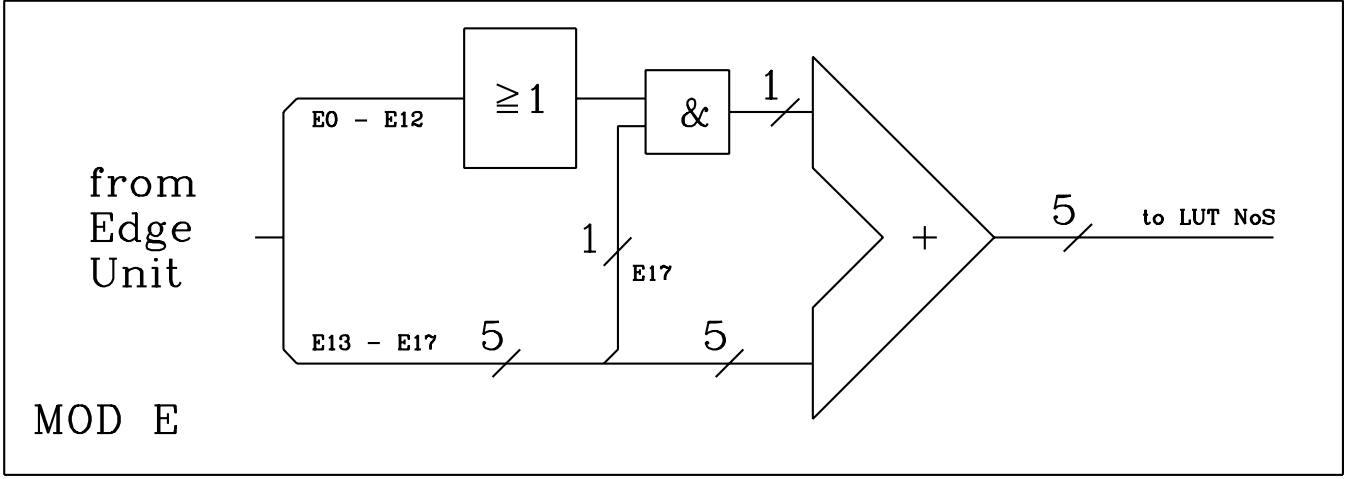


Figure 7: Modification unit for distance e .

crease the size. The reason for this is that our input parameters are coded in the two's complement form, meaning, that if we only truncate the numbers to the required bit count, we always round down to the next smaller number (Fig. 6; the bits are numbered E0 (LSB) to E18 (MSB), ES is the sign bit). However, we want the two adjacent polygons to complement each other exactly. This means that the sum of subpixels for a given error term and slope adds up to 16 subpixels, if the error terms add up to 0 ($e_1 = -e_2$). But with the property of the two's complement, that truncation always rounds down, this cannot be fulfilled. An example can clarify this. If we truncate as shown in Fig. 6, we want to obtain the same result for an input of 0/0.100 and 0/0.101. Now consider the negative value for both of these numbers: for 0/0.100 we get 1/1.100, for 0/0.101 1/1.011. If we truncate these negative numbers, we do not get the same result anymore: 1/1.10 and 1/1.01 are not equal. This simple example shows that one more input bit for the lookup table is required for each input parameter. This bit has to indicate whether a value was changed by truncation or if we already had a flat number. This bit can be generated by ORing all truncated (lower) bits together. A single stage lookup table would thus have a size of 8k by 16 bits.

5.1 Multi Stage Implementation

The size of the lookup table can be reduced by a factor of more than 8, if it is implemented in several stages. The easiest approach, then, is to replicate the logical structure of the algorithm into the hardware structure. Thus the unit consists of three subunits, which perform the following functions: 1. Determination of the subpixel count, 2. Determination of the angle index (sequence), 3. Determination of the final subpixel mask. A more explicit examination leads to a three stage design (Fig. 8).

In the first stage, the input data coming from the three units that calculate the edge functions (EU1 – EU3) is modified in order to eliminate the problems with the two's complement. The blocks are labelled MOD E and MOD DEX. The stage MOD E is used to modify the error term e , in the stage MOD DEX the slope parameter including the sign bits is modified. The need to modify the sign bits comes out of similar observations to those about the two's complement. This modification is performed with a PLA that is also used to determine one bit of the angle index. A schematic of the unit MOD E (Fig. 7) shows how the modification is performed, however the adder was replaced by a PLA with the same function in the final design. The second stage consists of the lookup table for the subpixel count and the lookup table for the remaining two bits of the angle index. In the third stage, those two parameters are used to lookup the final subpixel mask. Three special cases are also handled in this stage:

1. The edge unit is disabled – all 16 subpixel bits are set to 1 (Line DE).
2. The error term is greater or equal to 0.5 – all 16 subpixel bits are set to 1 (Line E_GE_OP5).
3. The error term is smaller than -0.5 – all 16 subpixel bits are set to 0 (Line E_LT_M0P5).

The whole coverage unit consists of three of the described multi-stage lookup tables (one for each edge). The final subpixel mask is the result of ANDing together on a bit-by-bit basis the three outputs for each edge. Fig. 9 shows the result for the slim line and the sharp triangle of Fig. 3. In Figures 10 – 12, simulation results for the exact area subpixel algorithm compared with supersampling and no antialiasing at all are shown.

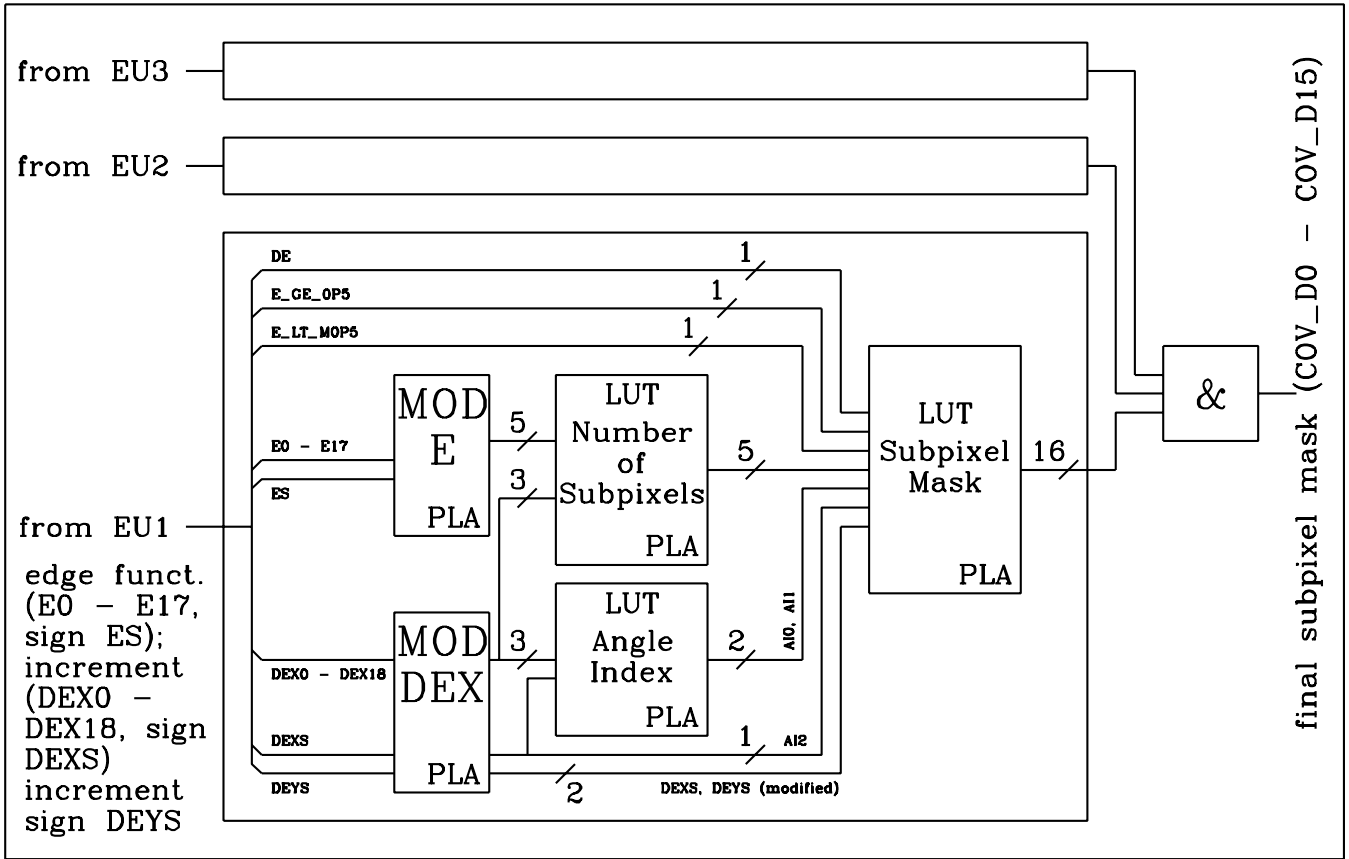


Figure 8: Coverage Unit.

6 Conclusion

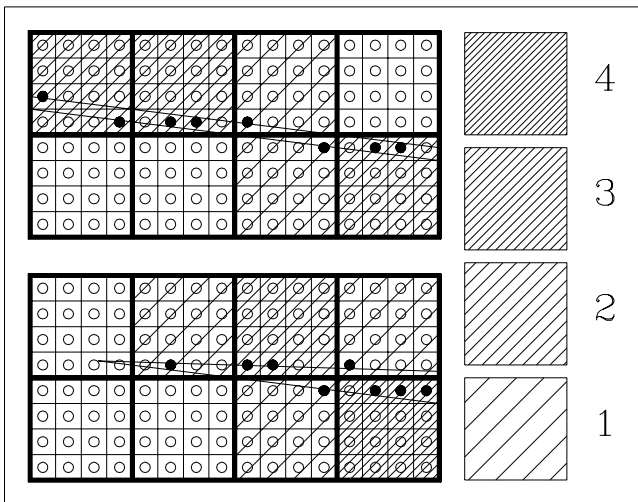


Figure 9: Thin line and sharp triangle treated correctly. The density of the hatching indicates the final pixel brightness.

An algorithm for antialiasing and its realization in hardware has been presented that obtains better results than other methods with comparable simplicity, e.g. regular supersampling (see Figures 10 – 12). Existing systems that already use subpixel mask lookup, can be modified to apply this algorithm simply by changing the contents of the lookup tables at nearly no costs.

7 Acknowledgement

I thank Paul Munsch of Caption for the idea with the "square distance" representation. Claudia Romanova provided me with *a lot of* useful information and literature on the topic of antialiasing. I also appreciate the guidance of Wolfgang Straßer and the cooperation with my colleagues at the Graphics Department of the University of Tübingen.

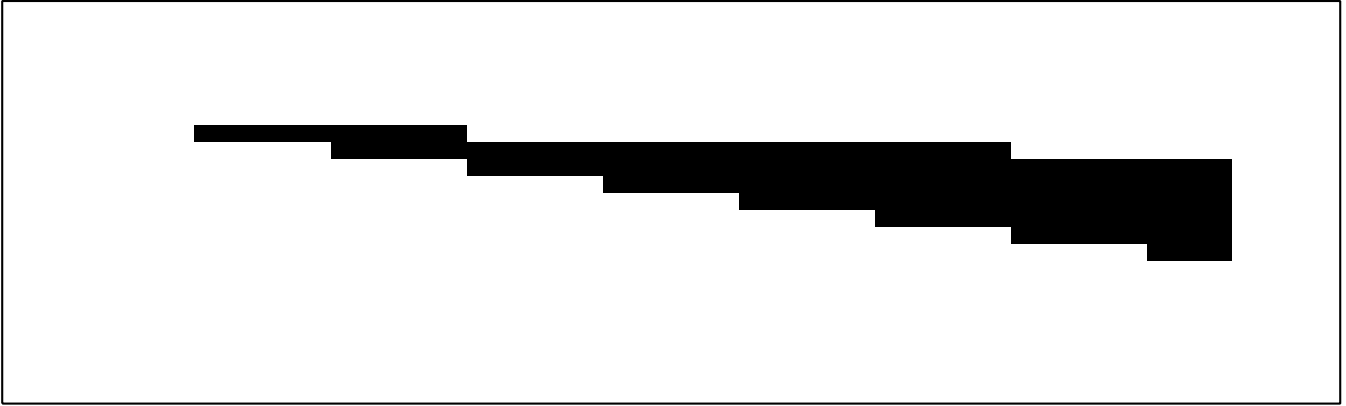


Figure 10: Sharp triangle of Figures. 9 and 3 without antialiasing.



Figure 11: Sharp triangle of Figures. 9 and 3. Problems with conventional supersampling.

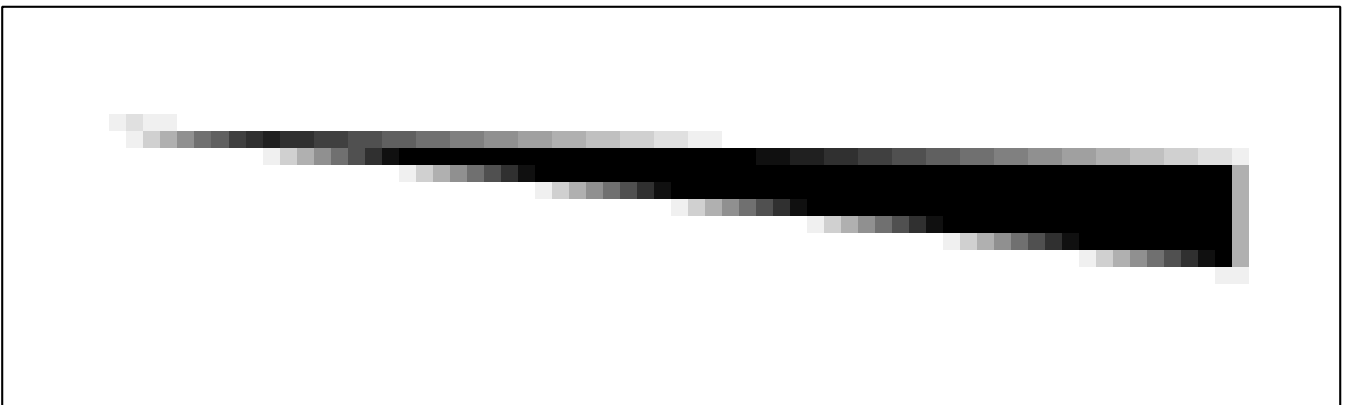


Figure 12: Sharp triangle of Figures. 9 and 3. Correct treatment with Exact Area Subpixel Algorithm.

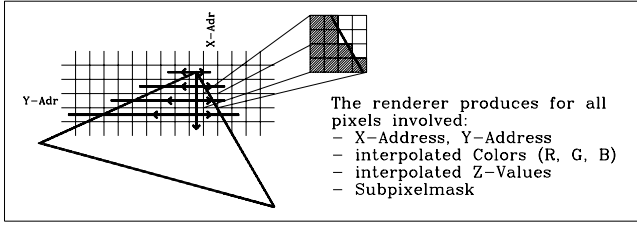


Figure 13: Rendering machine, functionality.

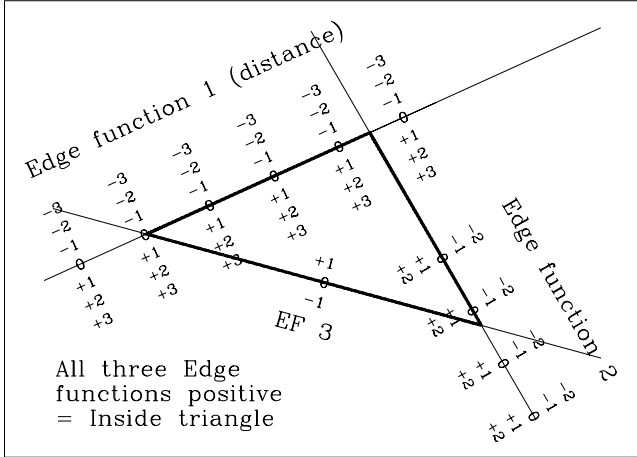


Figure 14: Example of Edge Functions for Rendering.

A Appendix

A.1 Representation of polygon edges

Polygons can be represented by edge functions, that are negative on one, and positive on the other side of the edge. This representation is used in rendering hardware like PROOF [12], Pixel Planes [8] or in software algorithms like the one described by Pineda [11]. In Fig. 13, the rendering with the Pineda algorithm is illustrated and the desired results of the rendering process are shown.

The process of rendering consists of deciding whether or not pixels belong to a given polygon. For a rendering algorithms like the above mentioned, we need an edge function that behaves like the ones shown in Fig. 14. It is positive on one side and negative on the other side of the edge. With three units that can calculate such edge functions, we can now decide whether a certain point lies inside the triangle or outside. If all three edge functions are positive, the point is inside, otherwise it is outside.

A.1.1 How can we get such a function?

The easiest way is to choose a linear function

$$E(x, y) = (x - X)de_x + (y - Y)de_y$$

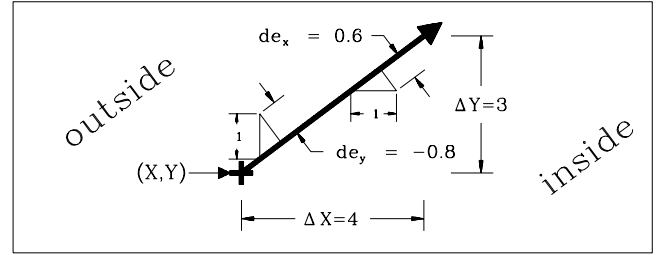


Figure 15: X and Y Increments of the Edge Function.

with the condition:

$$de_x\Delta X + de_y\Delta Y = 0$$

If we use

$$de_x = \Delta Y$$

as x increment and

$$de_y = -\Delta X$$

as increment in y direction, we get the formula suggested by Pineda with the advantage that the calculation is very simple. Now the edge units can be built of only adders, without multipliers, as we only have to add the increments proceeding from one pixel to its neighbor.

We can scale the above formula by an arbitrary factor. So if we need the euclidean distance, we can normalize the increments de_x and de_y by dividing the values by the euclidean length of the vector (L_2 norm). We then get the following increments:

$$de_x = \frac{\Delta Y}{\sqrt{\Delta X^2 + \Delta Y^2}}$$

and

$$de_y = -\frac{\Delta X}{\sqrt{\Delta X^2 + \Delta Y^2}}$$

and we can still use the same edge units, because the distance is a linear function in x and y (see example edge in Fig. 15).

A.1.2 Why do we need this distance?

Until now, we used only the sign of the edge function for the decision if we are in or out. So the value of the distance is of no interest. But if we want to calculate subpixel information for later antialiasing (which is in fact the scope of this paper), we need exact data about the edge. In this case, the normalization is essential [7]. The distance, together with the slope information is enough to look up the subpixel mask, i.e. the information, which part of the pixel is covered (see Fig. 13). One little detail has to be noticed. We now have to consider not only pixels with their center inside the

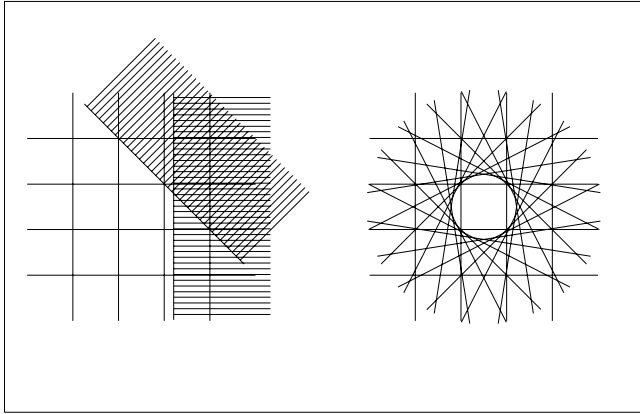


Figure 16: "Circular" Distance.

polygon (positive edge function), but also pixels that are covered less than half (edge function between 0 and $-0.\text{something}$). We cannot give a fixed distance, because it is different for edges with different slopes ($1/\sqrt{2}$ for edges with a slope of 45° , $1/2$ for vertical or horizontal edges). So if we take all pixels not more than $1/\sqrt{2}$ away from the edge into consideration, we will get too many pixels, but that is better than losing pixels that we wanted to get. Fig. 16 shows, that all edges that have a given distance from the pixel center form a circle.

Now there is a formula for the increments that solves several problems at one time³. If we look at the most demanding part of the increment calculation above, we see the square root in that formula. Now the simplest solution is to omit the root and take the sum of the absolute values of ΔX and ΔY instead. Speaking in mathematical terms, we divide by the L_1 norm or Manhattan distance instead of the L_2 norm. The new increments are:

$$de_x = \frac{\Delta Y}{|\Delta X| + |\Delta Y|}$$

and

$$de_y = -\frac{\Delta X}{|\Delta X| + |\Delta Y|}$$

The result is a change in the distance function. It is not independent of the angle anymore, which was the case for the euclidean distance. But if we do not want to do a more complex filtering (like e.g. a convolution with $\text{sinc}(\text{dist})$), then we need something like a rectangular box filter. A circular filter never would result in a homogeneous coverage of the screen. So this formula which is easier to calculate is not an approximation for

³I got this formula from Paul Munsch, who developed several distance formulas, to be used as an approximation for the euclidean distance, among them an "octagonal distance" and the "square distance", described here.

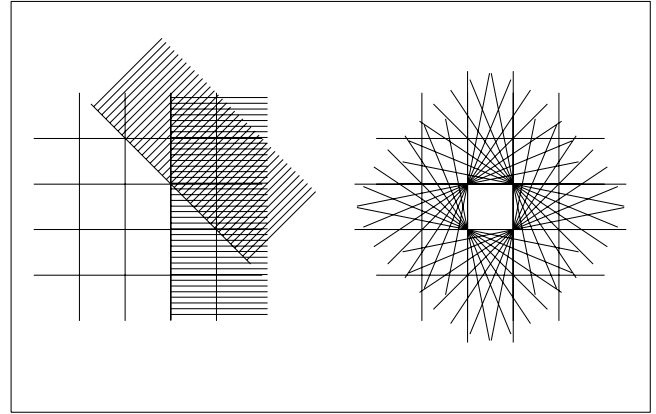


Figure 17: "Square" Distance.

something else, we originally wanted to get, it gives in fact a more desired result. Fig. 17 shows, that all edges that have a given distance from the pixel center form a square.

For the calculation of the subpixel mask, the "square" distance is as useful as the euclidean distance, because all information about the edge is contained in the distance and the increments (for the slope of the edge).

References

- [1] ABRAM, G., WESTOVER, L., AND WHITTED, T. Efficient alias-free rendering using bit-masks and look-up tables. *Computer Graphics* 19, 3 (July 1985), 53–59.
- [2] BLINN, J. F. What we need around here is more aliasing. *IEEE Computer Graphics & Applications* (Jan. 1989), 75–79.
- [3] CARPENTER, L. The a-buffer, an antialiased hidden surface method. *Computer Graphics* 18, 3 (July 1984), 103–108.
- [4] COOK, R. L. Stochastic sampling in computer graphics. *ACM Transactions on Graphics* 5, 1 (January 1986), 51–72.
- [5] DIPPÉ, M. A. Z., AND WOLD, E. H. Antialiasing through stochastic sampling. *Computer Graphics* 19, 3 (July 1985), 69–78.
- [6] FIUME, E., FOURNIER, A., AND RUDOLPH, L. A parallel scan conversion algorithm with anti-aliasing for a general-purpose ultracomputer. *Computer Graphics* 17, 3 (July 1983), 141–150.
- [7] FUCHS, H., GOLDFEATHER, J., HULTQUIST, J. P., SPACH, S., AUSTIN, J. D., BROOKS, F. P., EYLES, J. G., AND POULTON, J. Fast spheres, shadows, textures,

transparencies, and image enhancements in pixel-planes. *Computer Graphics* 19, 3 (July 1985), 111–120.

- [8] FUCHS, H., POULTON, J., EYLES, J., GREER, T., GOLDFEATHER, J., ELLSWORTH, D., MOLNAR, S., TURK, G., TEBBS, B., AND ISRAEL, L. Pixel-planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. *Computer Graphics* 23, 3 (July 1989), 79–88.
- [9] HOFFERT, E. M., AND BISHOP, G. Exact and efficient area sampling techniques for spatial antialiasing. Technical Memorandum, AT & T Bell Laboratories, December 1985.
- [10] MUNSCH, P. Private communication. On the occasion of a meeting in Rennes, France, Dec. 1989.
- [11] PINEDA, J. A parallel algorithm for polygon rasterization. *Computer Graphics* 22, 4 (Aug. 1988), 17–20.
- [12] SCHNEIDER, B.-O. *Eine objektorientierte Architektur für Hochleistungs-Display-Prozessoren*. PhD thesis, Eberhard-Karls-Universität Tübingen, 1990.