AN ASIC FOR OUTLINE CHARACTER GENERATION

Marc Morgan, Roger D. Hersch
Swiss Federal Institute of Technology, Lausanne, Switzerland

Abstract

This paper presents the design and implementation of an ASIC for real-time rasterization of characters described by their outline based on vertical scan-conversion and flag fill algorithms. The chip is a coprocessor which rasterizes outline fonts given by Bézier splines and straight line segments. It generates high quality fonts 30 times faster than the equivalent assembly language code on a 16 MHz M68020.

Introduction

For several years, engineers and typographers have been working on a new generation of fonts. The old bitmap description of a character is giving way to characters defined by their outlines.

Research undertaken at the Peripheral Systems Laboratory (LSP) at the Swiss Federal Institute of Technology (EPFL) has led to an algorithm for the rasterization and filling of outline characters. This algorithm is the base of an intelligent font rasterization program [Hersch89]. The fonts are given by outlines described by cubic Bézier splines and by line segments. The quality of the resulting black and white bitmaps depends on two important aspects of the program: the sub-pixel precision used during the rasterization and an appropriate control of the phase of the contour relative to the pixel grid.

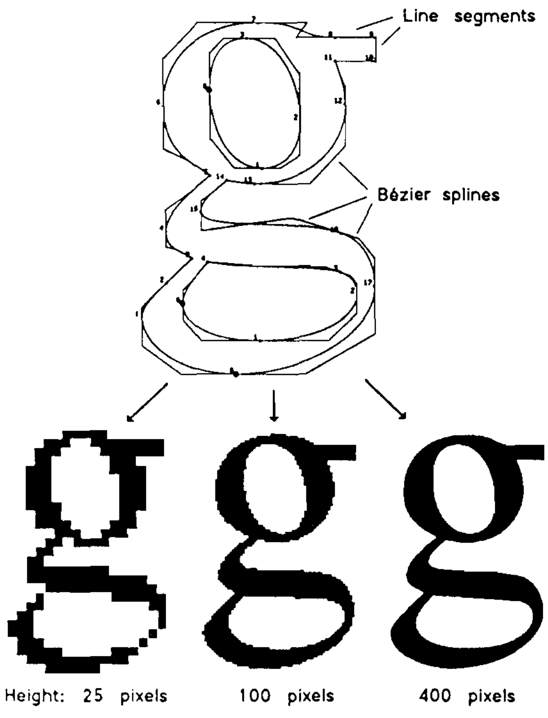


Fig. 1 Automatic rasterization of outline fonts

Motivation

The generation of a whole font (ASCII codes 32 to 126) by the intelligent character rasterization program [Hersch90] at a resolution of 300 dpi has been evaluated on a 16MHz M68020 microprocessor. Figure 2 shows that grid outline adaptation (phase control) is independent of the size of the characters. However, the scan-conversion and filling times become dominant for large characters. Characters are generated at an average rate of 50 characters per second at 600 dpi. This is clearly not sufficient for a high performance raster image processor. Since time-consuming parts of the program are already written in assembly language, further performance improvements can be obtained by designing an application specific IC (ASIC).

Given the complexity of intelligent character rasterization, it is not possible to implement all its functionalities into one VLSI chip. It was chosen to assign the ASIC the task of the highly repetitive part of the algorithm leaving the more complex work to the main processor. In other words, only scan-conversion and filling were integrated on the ASIC. Phase control is much more complex and requires the programming capabilities of a general-purpose processing unit.

This ASIC is mainly intended for high-resolution printers (600 dpi). At this resolution, only simplified phase control (hinting) is required. This ASIC can also be used for average-resolution printers (300 dpi) where good quality results can only be achieved with complete phase control.

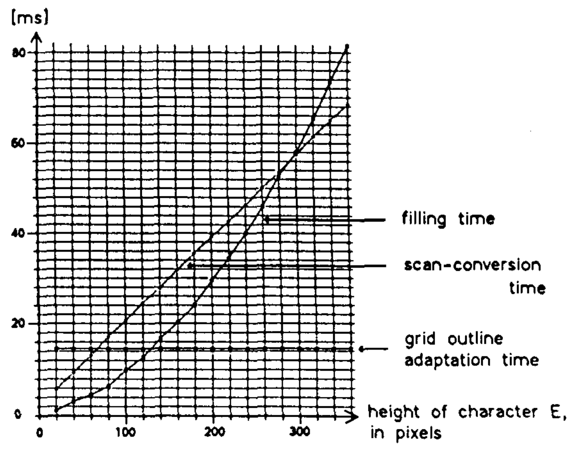


Fig. 2 Average generation time of one character by software.

The Environment

The ASIC has been designed as a simple coprocessor. It has a circular input buffer to which the main processor writes the character outline data. Once the coprocessor is ready to return the bitmap character, it sends an interrupt signal to the main processor which can then request the bitmap words sequentially from the ASIC. Synchronisation between the

Proceedings of the Society for Information Display, International Conference Anaheim, May 1991, Published in SID'91 Digest, Vol XXII, pp. 201-204.

processor and the ASIC is achieved by standard interrupt request and handshake signals (CircularBufferEmpty, Full, and HalfFull).

The first prototype of the circuit has been designed to generate raster characters up to 64 pixels high and wide. This prototype was realised to demonstrate the feasibility of such a circuit. The final version of this circuit will generate larger characters (256 x 256 pixels).

The principal task of the ASIC is to scan-convert the outline of the character which it will then fill before sending the resulting bitmap back to the main processor.

The Flag Fill Algorithm

The flag fill algorithm used by the ASIC is one [Ackland81] which was improved for the purpose of accurate shape filling [Hersch90]. The basics of the algorithm will be explained briefly here.

A pixel is considered to be inside a shape if more than 50% of its surface lies inside the shape. Since characters have relatively flat curves, the section of the outline which crosses a pixel can be assumed to be a line segment. Therefore, a pixel is considered to be an interior pixel if its center is inside the shape.

The bitmap which will be generated by the flag fill algorithm can be considered as a set of black horizontal spans for the inside of the shape and white horizontal spans for the outside. The first pixel of each span is marked by a flag. Once all the flags corresponding to an outline have been set, the flag fill algorithm scans the flag image memory from left to right. Each flag encountered indicates the start of a new horizontal interior or exterior span.

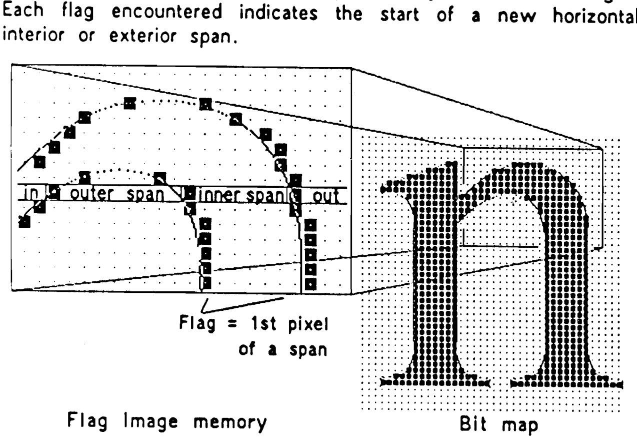


Fig. 3 Flag fill algorithm applied to a character

The Vertical Scan-Conversion Algorithm

The Bézier splines and line segments which make up an outline have to be converted into flags for the filling algorithm. Two strategies can be adopted to scan-convert a Bézier spline: recursive subdivision and forward differencing [Newman79]. Both strategies have been developed in order to reduce the number of required operations without reducing the precision of the scan conversion.

Ordinary forward differencing has one main drawback: the incremental step of the parameter used to describe the curve is a constant. Adaptive forward differencing (AFD) corrects this problem [Lien87]. AFD ensures that most of the points which are generated will be used to trace the curve. Integer AFD further improves the algorithm by using fixed point arithmetic instead of floating point arithmetic [Lien89] [Gonczarowski89]. The resulting algorithm is yet faster.

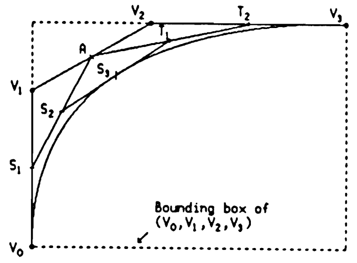
Recursive subdivision has also been optimized [Hersch90]. It presents several advantages over forward differencing. First, computation errors are not amplified as they are in AFD. In order to get the same result, recursive subdivision requires a significantly smaller number of bits of precision than AFD. Second, recursive subdivision can be carried out with the control points of a Bézier curve which provide easy monitoring of subdivision depth.

Scan-conversion by recursive subdivision

Recursive subdivision of Bézier splines is based on DeCasteljou's theorem. As figure 4 shows, a Bézier spline

$$P(u) = V_0\cdot(1-u)^3 + V_1\cdot 3u(1-u)^2 + V_2\cdot 3u^2(1-u) + V_3\cdot u^3$$
$$\text{with } u \in [0,1]$$

represented by its control polygon (V_0,V_1,V_2,V_3) can be subdivided into two smaller Bézier splines, (V_0,S_1,S_2,S_3) and (S_3,T_1,T_2,V_3). The smaller splines will have their control polygons closer to the spline. Therefore, if a spline is subdivided enough times, the resulting control polygons become a good approximation to the spline. One of the delicate points of the algorithm is the criterion for stopping subdivision. It is based on the convex hull property of Bézier curves: a Bézier curve always lies within the convex hull formed by its control polygon.



Subdivision of (V_0,V_1,V_2,V_3) into (V_0,S_1,S_2,S_3) and (S_0,T_1,T_2,V_3):

$$S_1 := (V_0 + V_1) / 2$$
$$T_2 := (V_3 + V_2) / 2$$
$$A := (V_1 + V_2) / 2$$
$$S_2 := (S_1 + A) / 2$$
$$T_1 := (T_2 + A) / 2$$
$$S_3 := (S_2 + T_1) / 2$$

Fig. 4 DeCasteljou's subdivision of Bézier splines

Repeated subdivision of Bézier splines can result in three types of Bézier splines (cf figure 5):

- splines which do not intersect any scan line, and which can be discarded since they won't generate any flag,
- splines which do not intersect any vertical grid lines, and can be considered as vertical line segments,
- splines which still intersect both a scan line and a vertical grid line.

The third case poses the problem of determining when to stop subdividing a spline and how to choose the position of the corresponding flag from the point at which the subdivision left off. The choice of stopping a subdivision has to take into account the maximum tolerable error and the precision used in the computations. The criterion we have chosen is to have the spline's bounding box smaller than the largest tolerable fractional error, *MinFracLength*, along both the x and y axes. When this criterion is met, the small spline is replaced by the three line segments which make up its control polygon. Experience shows that the choice of 1/8 pixel for *MinFracLength* is a good trade-off.

The ASIC implementation of the subdivision-stopping criteria is slightly simplified by an additional hypothesis: all the splines which make up an outline are monotonic along both x and y. This hypothesis implies that the font production software will have to subdivide any non-monotonic splines.

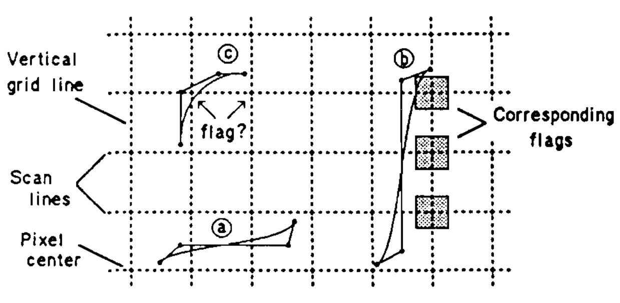


Fig. 5 The three possible final outcomes of a Bézier subdivision

Comparison of recursive subdivision and forward differencing

The same criteria which we developed for recursive subdivision could be applied to adaptive forward differencing (AFD) [Lien89]. This however would require precomputing the next point on the curve before deciding whether the step size is correct or needs adjusting. This corresponds to an extra addition for each adjust up or down operation. Even so, when *MinFracLength* has been reached, the curve would have to be replaced by a line segment, which is less accurate than the control polygon given by recursive subdivision.

Recursive subdivision requires 3 adds and 3 shifts per sub-spline (figure 4). Integer AFD requires 3 adds, 2 shifts and an increment per step and per adjust up or adjust down operation. Both algorithms therefore require approximately the same number of arithmetic operations. However, integer AFD requires dynamic shift operations which are avoided by recursive subdivision. This tends to make AFD slower than subdivision. On the other hand, the recursive aspect of subdivision has to be implemented with a stack. Stack access will slow down subdivision. This problem can be partially eliminated by accessing the stack in parallel with other operations. This is true for the ASIC presented here which writes into the stack at the same time as it computes other control points or as it tests the locations of spline control points.

Another important difference between the two methods comes from the number of bits which are worked on. For example, for a maximum curve length of 64 pixels and with *MinFracLength* = 1/8 pixel, recursive subdivision requires 14-bit operators. Ordinary forward differencing would require 35 bits to get the same precision [Lien89] while AFD would need 20 bits. AFD therefore would work on 43% more bits than recursive subdivision. Having more bits slows down arithmetic operations and requires more place on the integrated circuit.

The Architecture

The ASIC has been designed to optimise the scan-conversion of Bézier splines. Line segments are processed by the same subdivision hardware as the splines.

The last step in the processing of a character consists of returning the filled bitmap to the main processor. The ASIC reads the flag image memory word by word and fills the outline on the fly. At the same time, it clears the flag image memory for the next character.

The architecture of the ASIC is fairly simple (figure 6). It can be decomposed into the following blocks:

- an input interface,
- a circular input buffer to store incoming bytes until they can be sent to the subdividers,
- two nearly identical subdividers for the X and Y coordinates,
- a stack to store intermediate results,
- the sequencer which is based on a small ROM,
- a RAM for the flag image memory,
- the flag set/fill unit,
- an output interface.

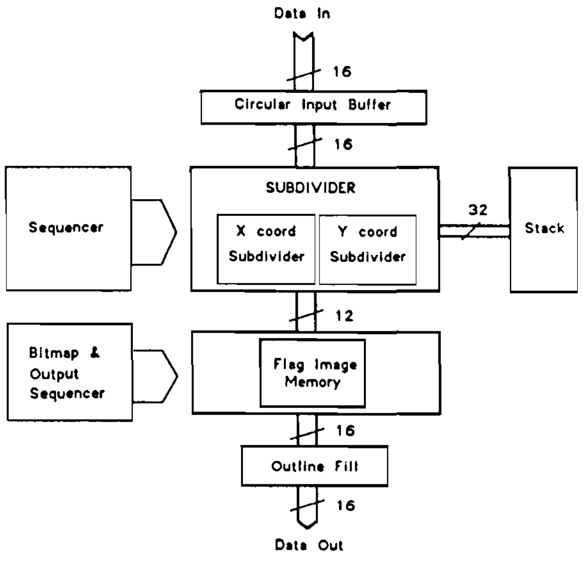


Fig. 6 General architecture of the ASIC

The Circular Input Buffer

The circular input buffer is used to store each curve sent by the processor. It can store up to 512 words which corresponds, for instance, to 64 Bézier splines; each spline is described by 4 control points requiring two 14-bit coordinates each.

The Subdivision Units

The architecture of the subdivision units has been carefully optimized for Bézier subdivision. The data processing unit (figure 7) is nearly identical for the x and y coordinates. It is made up of 7 registers, one adder/subtractor, one adder, four busses, and a condition code tester.

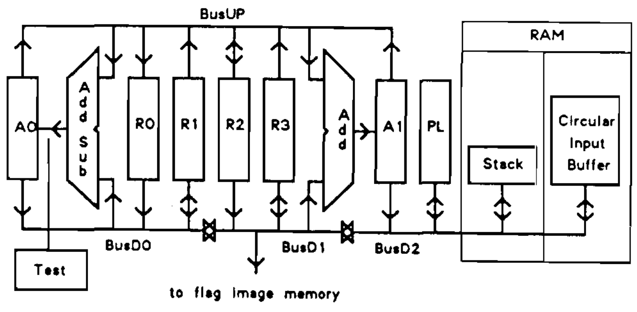


Fig. 7 Architecture of a data processing unit

The four registers (R0, R1, R2, R3) in the center of a DPU contain the coordinates of the four control points of a Bézier spline. The results of the average operations are stored in the two outer registers (A0, A1). The seventh register (PL) latches the last value pushed onto the stack as a temporary buffer. An average operation is generated by an adder followed by a hard wired shift right. The subtractor is used to test a curve to determine whether to keep subdividing it or not.

Each register contains 14 bits: 6 integer bits and 8 fractional bits. No extra sign or overflow bits are required since averaging is the only operation performed. The 6 integer bits allow for characters up to 64 pixels high. The 8 fractional bits are sufficient in order to keep the required precision.

The bus configuration has been designed to enable parallel access to both adders and to limit the capacity (pF) of each bus. The busses also give access to the stack and the circular input buffer. The stack is used to save curve parts from one level of subdivision to the next.

The Stack

The stack contains the sequence of x and y coordinates of points and the type of curve they belong to (Bézier spline or line segment). The height of the stack is given by the range of accepted coordinates (0.00 to H´3F.FF in our case) and the chosen value for *MinFracLength* (1/8). In our case, a maximum of $\log_2(H'40) + |\log_2(1/8)| = 9$ levels of recursive subdivisions can occur. Since each subdivision requires that 4 control points ($4 * 2$ coordinates) of a new spline be stored on the stack, the stack has to contain at least $9 * 4 * 2 = 72$ words. Note that no new curve is processed before the last one has been completely scan-converted; this keeps memory requirements for the stack to a strict minimum.

The Results

Simulation work has been completed, although the circuit has not yet been manufactured. Character generation rate is around 2500 characters per second for fonts with 32-pixel high capitals. This character generation speed includes the time required to load the character outline into the circuit and to fetch the bitmap from it.

The choice of the technology used for the ASIC was based on ease of conception and former experience. The solution adopted for the first prototype was VTI's 1.6 um CMOS (CMN16) standard cell technology. The size of the ASIC will be under 20 mm^2.

Conclusions

This paper has presented a hard-wired algorithm for fast rasterization of characters described by their outlines. A comparison between recursive subdivision and forward differencing shows that recursive subdivision is easier to implement in an application specific integrated circuit. The proposed architecture guarantees the correct rasterization of outline characters due to precise criteria for stopping recursive subdivision.

A simple architecture is proposed for the implementation of the rasterization algorithm. Simulations show that the ASIC generates an average of 2500 characters/second. This number, however, depends on the character size and the font being rasterized. This ASIC can be used as the core of a high performance raster image processor.

References

[Ackland81]	B.D. Ackland, N.H. Weste, "The Edge Flag Algorithm - A Fill Method for Raster Scan Displays", IEEE Trans. on Computers, Vol. 30, No. 1, pp. 41-48 (1981)
[Gonczarowski89]	J. Gonczarowski, "Fast Generation of Unfilled and Filled Outline Characters", Raster Imaging and Digital Typography, Cambridge University Press, pp. 97-110 (1989)
[Hersch89]	R.D. Hersch, "Introduction to Font Rasterization", in J. André, R.D. Hersch (eds.), Raster Imaging and Digital Typography, Cambridge University Press, pp. 1-13 (1989)
[Hersch90]	R.D. Hersch, "Efficient Rendering of Outline Characters", 1990 SID Symposium Digest of Technical papers, Vol. 21, Society for Information Display, pp. 392-394 (1990)
[Lien87]	S.L. Lien, M. Schantz, V. Pratt, "Adaptive Forward Differencing for Rendering Curves and Surfaces", Proceedings SIGGRAPH'87, ACM Computer Graphics, Vol. 21, No. 4, pp. 111-118 (1987)
[Lien89]	S.L. Lien, M. Schantz, R. Rocchetti, "Rendering Cubic Curves and Surfaces with Integer Adaptive Forward Differencing", Proceedings SIGGRAPH'89, ACM Computer Graphics, Vol. 23, No. 3, pp. 157-166 (1989)
[Newman79]	W.M. Newman, R.F. Sproull, Principles of Interactive Graphics, McGraw-Hill. (1979)