# Texturing and Conical Sections in the Pixel-planes Graphics Engine

John G. Eyles

May 16, 1984

This report describes an algorithm for the Pixel-planes graphics engine, which may be used to implement tectures by imposing regular geometric patterns upon an existing shape. This algorithm may also be used to rapidly load a segment of memory with arbitrary functions of the form $Rx^2 + Sxy + Ty^2 + Ux + Vy$. The main advantage of this algorithm is that the Q-buffer used in the circle drawing algorithm may be loaded "on the fly"; this opens the possibilities of (1) using the large section of memory, ordinarily occupied by the Q-buffer, for other purposes, and (2) drawing ellipses and other conic sections by loading the Q-buffer with other functions besides $x^2 + y^2$.

In the Pixel-planes graphics engine, there is an Enable register and a word of memory (64 bits in V.4) associated with each pixel. The standard mode of operation is to define a region in the screen space by (1) setting all of the Enable registers to 1, and then (2) selectively disabling certain pixels by clearing their Enable registers to 0. A subset of pixels whose Enable registers contain a 1 is thereby defined; this set of pixels defines a region in the screen space which may then be painted or smooth shaded in some fashion.

When a set of coefficients (A, B, and C) is shifted into the mutliplier associated with each pixel, the product $Ax + By + C$ is computed. For a given straight line in the screen space, A, B, and C may be defined in such a way that the sign-bit of $Ax + By + C$ defines which side of the line the pixel $x, y$ lies upon. If this sign-bit is AND'ed with the Enable register and the result stored back into the Enable register, the half-plane of pixels lying to one side of the edge is disabled. By defining 3 or more such edges, successively disabling half-planes of pixels, arbitrary convex polygons may be defined.

F. P. Brooks has suggested an algorithm for defining disks as polygons with one edge, which allows disk shapes to be defined very rapidly, in the amount of time required to process one set of coefficients. This algorithm requires that a section of the memory word associated with each pixel be reserved to store the value of $x^2 + y^2$ for that pixel. For a standard 512 by 512 system, this "Q-buffer" must be 19 bits in length. When the coefficients (A, B, and C) are properly defined and the contents of the Q-buffer are subtracted from the tree result $Ax + By + C$, the result is non-negative for pixels lying within or on the border of a given disk. (For a disk with center $x_c, y_c$ and radius $r$, the coefficients are defined as $2x_c, 2y_c$, and $r^2 - \{x_c^2 + y_c^2\}$, respectively). It has been supposed that this Q-buffer would be loaded with $x^2 + y^2$ values at system initialization time, in a fairly lengthly process involving the processing of approximately 1000 sets of coefficients.

The signals which actually produce images in the Pixel-planes memory chips are generated by the Image Generation Controller (IGC). The IGC is based on a microcode RAM which is loaded at system initialization time (and may be reloaded later). "IGC commands" are implemented as a sequence of microcode words in this microcode store. "Basic commands" for the Pixel-planes graphic engine consists of a specification of (1) the starting microcode address for the IGC command which implements the basic command, (2) addresses for source and destination Pixel-planes memory locations, (3) a count field, which may be used in different ways by different IGC commands, (4) the A, B, and C coefficients for the command (used to form the tree result $Ax + By + C$), and (5) information concerning how the coefficients are to be truncated. Thus, one IGC command, defined in microcode, may be used to implement a number of basic commands by specifying different Pixel-planes memory addresses and counts. This is discussed in detail in the document "Specification of Image Generation Controller for Pixel-planes Graphics Engine".

For this discussion several basic commands must be defined. The *setenabs* command sets the Enable registers for all pixels to 1. The *Qload* command loads the *whole* (integer) part of the tree result $Ax + By + C$ into the portion of Pixel-planes memory defined as the Q-buffer. The *addtoQ* command adds the *whole* part of the tree result to the contents of the Q-buffer (modulo-$2^Q$, where Q is the length of the Q-buffer). The new approaches described in this report all are implemented by defining a new "basic command" for the Pixel-planes graphics engine, *tect*; for the *tect* command, the A, B, and C coefficients are shifted into the multiplier/tree structures as usual, and when the least significant bit of the *whole* part of the tree result appears at the ALU, this LSB is AND'ed with

the Enable register and the result stored back into the Enable register. Thus the Enable register is cleared for any pixel for which the integer part of the tree result is even.

The use of the *tect* command to implement texturing can best be demonstrated by an example. Suppose the Enable registers all are set to 1 (using the *setenabs* command). Then the *tect* command is issued with A, B, and C coefficients of 1, 1, and 0 respectively. The tree result is therefore $x + y$. The resulting pattern of the Enable registers forms a checkerboard pattern, with each square consisting of 1 pixel. If the coefficients are set to 0.5, 0.5, and 0, the result is a checkerboard pattern with squares 2 pixels by 2 pixels. If the coefficients are set to 1, 0.5, and 0, a pattern with rectangles 2 pixels by 1 pixel is produced. Thus different patterns may be implented by varying the A and B coefficients, and more complex patterns may be obtained by processing several sets of coefficients in this fashion. The patterns of Enable bits produced by multiple sets of coefficients may be logically AND'ed, or they may be logically OR'ed by using one of the regular memory bits as an "auxiliary enable register".

Using the *tect* command, a polygon may be defined and shaded, and then a texture introduced. This could also be used to introduce transparency effects, by defining a checkerboard pattern before the polygon is shaded. Thus the $x + y$ *even* pixels would show the color of the underlying image while the $x + y$ *odd* pixels would be shaded according to the definition of the new polygon.

This approach may also be used to load the Q-buffer with $x^2 + y^2$ values very rapidly. This method is best illustrated by example; consider an 8 pixel by 8 pixel system. First, the Q-buffer would be cleared, loaded with 0's. Next set the Enable registers; then issue the *tect* command with A, B, and C coefficients of 1, 0, and 0, respectively. This will leave those columns with odd $x$ indices enabled. Then issue the *addtoQ* command with coefficients 1, 0, 0. The Q-buffer for pixels in the odd-numbered columns, 1, 3, 5, and 7, now contain $x$, while those in the even-numbered columns still contain 0. Then reset the Enable registers, and issue the *tect* command with coefficients 0.5, 0, 0. Columns 2, 3, 6, and 7 will remain enabled, since the whole part of the tree result is odd for pixels in these columns. Then issue the *addtoQ* command again, with coefficients 2, 0, 0; this adds $2x$ to the Enable'd columns. Now columns 0 and 4 contain 0, columns 1 and 5 contain $x$, columns 2 and 6 contain $2x$, and columns 3 and 7 contain $3x$. Finally reset the Enable registers, issue the *tect* command with coefficients 0.25, 0, 0, leaving columns 5, 6, 7, and 8 enabled, and issue the *addtoQ* command with coefficients 4, 0, 0, adding $4x$ to columns 5, 6, 7, and 8. The Q-buffer for each pixel will now contain $x^2$. The process is then repeated, with the A and B coefficients interchanged, thus adding $y^2$ to these $x^2$ values.

Stated formally: to load the Q-buffer with the term $Rx^2 + Sxy + Ty^2 + Ux + Vy$, issue the following sequence of instructions (N is the $log_2$ of the display resolution, i.e.: N=9 for a 512 by 512 system):

| COMMAND | A | B | C |
|---|---|---|---|
| setenabs | | | |
| Qload | 0 | 0 | 0 |
| tect | 1 | 0 | 0 |
| addtoQ | R | S | U |
| setenabs | | | |
| tect | 1/2 | 0 | 0 |
| addtoQ | 2R | 2S | 2U |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| setenabs | | | |
| tect | $1/2^{N-1}$ | 0 | 0 |
| addtoQ | $2^{N-1}R$ | $2^{N-1}S$ | $2^{N-1}U$ |
| setenabs | | | |
| tect | 0 | 1 | 0 |
| addtoQ | 0 | T | V |
| setenabs | | | |
| tect | 0 | 1/2 | 0 |
| addtoQ | 0 | 2T | 2V |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| setenabs | | | |
| tect | 0 | $1/2^{N-1}$ | 0 |
| addtoQ | 0 | $2^{N-1}T$ | $2^{N-1}V$ |

Thus the Q-buffer for a full 512 by 512 system could be loaded (with $x^2 + y^2$) by processing 37 sets of coefficients, the time required to process several polygons, on the order of a few hundred microseconds for the proposed system clock rates of 5-10 Mhz.

The capability of rapidly loading the Q-buffer opens the possibility of loading it with other functions in order to define more complex shapes. As another example, for the 8 by 8 system of the previous example, consider the following sequence of commands:

| COMMAND | A | B | C |
|---|---|---|---|
| setenabs | | | |
| Qload | 0 | 0 | 0 |
| tect | 1 | 0 | 0 |
| addtoQ | 1 | 1 | 0 |
| setenabs | | | |
| tect | 1/2 | 0 | 0 |
| addtoQ | 2 | 2 | 0 |
| setenabs | | | |
| tect | 1/4 | 0 | 0 |
| addtoQ | 4 | 4 | 0 |

It is easy to see that the Q-buffer would contain $x^2 + xy$. Similarly, arbitrary functions of the form $Rx^2 + Sxy + Ty^2 + Ux + Vy$ may be loaded into the Q-buffer, using just $4N + 1$ sets of coefficients. This general quadratic term could be used to define ellipses, parabolas, and hyperbolas. However, for the general quadratic term, the Q-buffer would probably have to be larger than the $2N + 1$ memory bits required when just $x^2 + y^2$ is loaded for the disk drawing algorithm.