Graphics Supervision Work 2
http://www.matthew.ath.cx/notes/

Matthew Johnson
Trinity Hall
mjj29-notes@srcf.ucam.org

May 5, 2003

2

P

1

The cicle drawing algorithm for the first octant is:

```
x = x0
y = y0

d = (x + 1)*(x + 1) + (y - 1/2)*(y - 1/2) - r*r

DRAW(x,y)

WHILE x < (x1 - 1/2) DO
   x ++
   IF d < 0 THEN   #move East
      d = d + 2x +3
   ELSE            #move South East
      d = d + 2x - 2y + 5
      y ++
   FI
   DRAW(x,y)
ELIHW
```

2

The reason we have octants is because we must draw some of them by incrementing x every step, and possibly inc- or decrementing y. The change over point is to do with the angle of the line we're drawing. This occurs when the normal to the tangent is at 45 degrees.

3

The midpoint line drawing algorithm has 5 operations, 2 tests and a draw per pixel iteration, but operates on floating point numbers, which is 50 cycles for operations, plus the tests and the draw. Bresenham's line drawing algorithm, with integer end points, does only 4 operations, plus the tests and the draw. Therefore, it will be significantly faster than the midpoint algorithm.

Pixels	Midpoint Cycles	Bresenham Cycles
5	250	20
10	500	40
20	1000	80

both algorithms have the same number of tests and draws.

E

1

(yeah yeah, I only noticed after writing this - is it right though?)

Douglas and Pucker's algrithm works by taking the series of points you are going to draw, and scanning along them to fing the furthest distance points apart. If these are less than one pixel wide, you can reduce the whole system to a single line (or point). Otherwise, the algorithm splits the lines into two groups, and recurses on both sides until you drop below the resolution you are drawing.

Overhauser interpolation? Don't think its been mentioned by name, but possibly its the technique for drawing curves, where you check if the difference between a curve and a straight line approximation of it is less than a pixel. If it is, you use the striaght line approximation, otherwise you recurse on the left and right halves of the curve, until the approximmation error drops below the drawing resolution.

2

Matrices are used to describe transformations, because it is very easy to apply them to points, since it is just a case of a matrix multiplication. You can also combine transformations before applying them to lots of points, by multiplying the matrices representing the transfomations, and then just applying the result.

This means that you only apply once to all the points, rather than once per transformation.

A 2D rotation about an arbitrary point is composed of a translation to the origin, a rotate about the origin, and then a translate back to the point. For a point (x_1, y_1) the initial translation matrix is

$$\begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix}$$

and the final reverse-translation is similar, with x_1 and y_1 negated. a rotation of θ about the origin is

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This combines in the order right to left, to give

$$\begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

3

Here is an algorithm to draw a striaght line using only integer arightmetic:

```
dy = (y1 - y0)
dx = (x1 - x0)
x = x0
yf = 0
y = y0
DRAW(x,y)

WHILE X < X1 DO#
   x = x + 1
   yf = yf + 2dy
   IF (yf > dx) THEN
      Y = y + 1
      yf = yf - 2dx
   FI
   DRAW(x,y)
END WHILE
```

It varies depending which quadrant you are drawing in, since in some cases you draw per-column, and some you draw per-row. It basically consists of moving along horizontally, and drawing the nearest pixel in each row.

For a description of how to draw curves using straight lines, see the answer to 1.

4

The Cohen-Sutherland clipping algorithm uses a 4-bit code to say whether a point is inside of outside a line. Each bit position corresponds to one side of the region you are drawing in, and if the point is outside that bound, then that bit is set. If both ends have a code of zero ($Q_1 = Q_2 = 0$) then it is completely within the bounds and can safely be drawn. If there is a common bit set in the codes ($Q_1 \wedge Q_2 \neq 0$) then it is completely off to one side, and can be ignored, The other case you intersect with one of the lines and start again, but which bit is set tells you which edge to clip against. The last case does require the previous divisions - but this is also the rarer case. If several bits are set, then you have multiple edge crosses, and you have to calculate both clips, and recalculate to find which you should clip against.

In the examples, the first line is entirely inside the area, and hence each end point would have a code of 0000. This means that the line can safely be drawn (since $Q_1 = Q_2 = 0$). The second case, the left-hand bit would be set in both (so $Q_1 \wedge Q_2 \neq 0$), and it is therefore entirely one side of the area, and can be ignored. The third line is more difficult. the end points are both outside the drawing area, so they will be non-zero, however, there are no common bits set in the codes, (eg 1100 and 0011) so $Q_1 \wedge Q_2 = 0$. You draw this by splitting the line into the sections ending at each of the bounds that are crossed, and then apply the algorithm to all of the sections. Only one of those will be in the bounds and drawable.

(insert complicated 2D diagram)

Moving AB to $A'B'$ contains a scale factor of 2, a rotate by 30 degrees, and a translation of $B \to B'$. The matrix representing the entire transformation can be calculated by:

$$\begin{bmatrix} 1 & 0 & B'_x - B_x \\ 0 & 1 & B'_y - B_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 30 & -\sin 30 & 0 \\ \sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

This reduces to the matrix:

$$\begin{bmatrix} 2\sqrt{3} & -2 & 3 \\ 1 & 2\sqrt{3} & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

you then multiply by the co-ordinates of A, in homogenous co-ordinates, $(0, 1, 1)$, to get the co-ordinates of A', which are $(1, 2[1 + \sqrt{3}])$

5

Bezier decided to define the 4 variables as the end points, and 2 other "control" points, that are used to calculate the tangents. This produces the formula:

$P(t) = (i - t)^3 P_0 + 3t(1 - t)^2 P_1 + 3t^2(1 - t)P_2 + t^3 P_3$, which is a sum of Bernstein polynomials.

C_0 continuity is continuity up to the 0-th derivitive, I.E. in the positions of the curve itself. This means that $P_0 = Q_1$ (where P_0 is an end of the first line, and $Q - 1$ is an endpoint of the other line. With C_1 continuity, it is also continuous in the first derivative, so $P'(t) = Q'(t)$. In the second case, the end points must match in position, and the condition on the first derivative corresponds to diametriacally opposite control points at the join, equidistant from the points joined.

A bezier cubic is guarenteed to lie within the convex hull of its control points because... mathematical proof not given afaict.

Notes

There are two ways of drawing lines on a pixel grid. Either we can set every pixel the line passes through, or we can pick the 'best' pixel in each row or column. The second method produces 'better' lines, and both Bresenham's algorithm and midpoint line algorithm do this.

Note: the octants numbered anticlockwise from the right horizontal (+ve x-axis)

Bresenham's

In octant 1, you always move right one pixel, and evaluate some value to decide whether to move up. We calculate the distance the line moves vertically from the last pixel, and if it moves more than 1 pixel, we move up one before drawing. Calculating this variable is complicated, and slow (uses floating point arithmetic).

Midpoint Algorithm

In this case the new decision variable d' can be easily calculated from the previous d. In octant 1 you either go E or NE. If you moved E then $d' = d + a$, and if you moved NE, then its $d' = d + a + b$. If this d' is < 0, then the midpoint is above the line, and need to move E, otherwise it is on or below the line, and you have to move NE.

Midpoint Circle Algorithm

In this case we are drawing in all octants, but we have to draw each one separately with a slightly different algorithm. The decision variable in this case is $d = x^2 + y^2 - r^2$, so it is > 0 outside the circle (have to move in towards the center) and it is < 0 inside the circle, so we have to move out from the center.

Midpoint decision variables

The decision variable is always taken to be the midpoint exactly between the two options for drawing next. Calculating the decision variable from the previous one is always possible, by calculating the (x, y) of the new decision point and the old one, then substituting them into the generic equation, and subtracting the two equations to find the differences. This will give you two options for the new d, depending which square the old value of d caused you to move into.

Bézier Cubics

equation: $\binom{P(t)=\sum +i=0^3 P_i}{3i(1-t)^{3-i}t^i}$. t is a parameter that varies $0 < t <= 1$, and it is a **weighted sum** of the control points, where the weights are always positive an sum to one. This means that the curve always lies within the convex hull of the control points.

When splitting Béziers (P into Q & R), we split at the point $t = 1/2$, and so we can impose the conditions:

$$\begin{aligned} Q(0) = P(0) \Rightarrow Q_0 = P_0 \\ Q(1) = P(1/2) \\ R(0) = P(1/2) \\ R(1) = P(1) \Rightarrow R_3 = P_3 \end{aligned}$$

=we can use those conditions to partly calculate the 4 values for Q and R. We still need more conditions, however, to be able to calculate the control points of Q and R. The join between the two should have C_1 continuity, so both the position, and the first derivative, are continuous between Q and R. This gives us the further conditions:

$$\begin{aligned} Q(1) = R(0) \\ \frac{dQ}{dt}(1) = \frac{dR}{dt}(0) \end{aligned}$$

Solving all six simultaneous equations for Q_{0-3} and R_{0-3} will define the new curves.

Note that a consequence of $Q'(0) = R'(1)$ is that the control vectors at the join are diametrically opposite, and equal in length.