Andrew Lewis - Graphics Revision

Keywords

PIXEL
FRAME BUFFER – memory of the screen
VIDEO -
SCREEN -
WINDOW -
POLYGON – filled area bound by straight lines
SPLINE – bit of wood used by draughtsmen to draw curves / wobbly line

Clipping

Box/ Cube
2D = Viewport
3D = Viewport + near, far clip planes

Perspective

Field of View Y (vertical camera angle)
Aspect (window aspect ration)
Near (near clip plane)
Far (far clip plane)

Hidden Surface Removal

Painters algorithm –

draw what is at the back first and work forward till everything is drawn.
i.e. draw the sky then the grass then a tree then a bird on the tree.
Problems

Inaccurate what if the bird is in the tree what should be drawn first ?

Slow, how should we sort the order things should be drawn in ?

Z- Buffer –

Store the depth for each pixel on the screen and compare stored value with depth of new pixel being drawn.

Each pixel is being drawn anyway therefore there is just 1 new check and 1 new store for each pixel.

Double Buffering

Old graphics programs, single buffer see the individual items appearing.

The idea, effectively 2 banks of memory, to simplify this I will say bank A and bank B.

First bank A is on screen, so you don't see any drawing occurring you draw into bank B, once you finish drawing you tell the screen to use bank B. Now you draw

into bank A and once you finish drawing that you tell the screen to use bank A again etc.

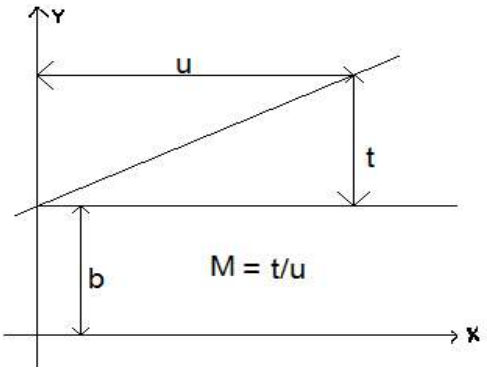
Means there is always a static picture on the screen, prevents flickering as frames change.

Colour –ADD HERE

Line Drawing

Vertical – Straight forward for loop incrementing/decrementing y
Horizontal - Straight forward for loop incrementing/decrementing x

Diagonal



Has a gradient (M)
Has an offset from the origin (b)
If the gradient is less than 1

i.e. y = M * x + b
Step along x working out the y co-ordinate

Can be sped up, points are equally spaced therefore instead of working out the y co-ordinate, after finding the first value adding M will get consecutive points y values.

If the gradient is greater than 1

i.e. y = 2 * x + b
Step along y working out the x co-ordinate

Can be sped up, points are equally spaced therefore instead of working out the x each time can just add 1/M to get consecutive points.

Bresenham Line Drawing

We have our line y = M * x + b
M = change in y / change in x = delta y / delta x = dy/dx

This means our original equation is
y = dy/dx * x + b

Multiplying out gives us

$$dx * y = dy * x + b * dx$$
$$dy * x - dx * y + b * dx = 0$$

a line can be represented like this so

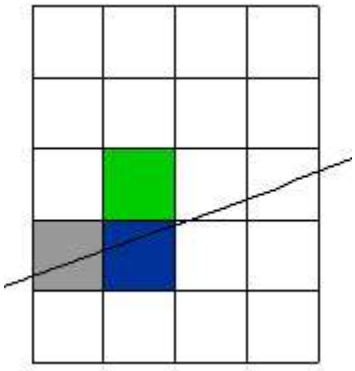
$$a * x + b * y + c = 0$$
$$a = dy$$
$$b = -dx$$
$$c = b * dx$$

Let dy = 0 (a horizontal line)

dy * x – dx * y + b *dx = 0	
0 * x – dx * y + b * dx = 0	(substitute 0 in)
b * dx – y * dx = 0	(throw away first term)
dx * (b – y) = 0	(tidyup)
b – y = 0	(divide by dx)

if (y > b) then its –ve and ABOVE the line
if (y < b) then its +ve and BELOW the line
if (y==b) then its on the line

All the cases can be generalised to vertical / horizontal or this



To see how this work we start drawing on the gray square. We have 2 choices, to blue or to green.
Test roughly where the line will intersect the next square
X+1
Y+0.5
d = a * x + b * y + c
d = a * (x + 1) + b * (y+ 0.5) + c
if (d<0) then X+=1, Y+=1
if (d>0) then X+=1

this can be used to draw the lines BUT

after we move it can be seen that
d2 (the new d)
x2,y2 x, y after moving

if we move east blue
x2 = x+1
y2 = y
d2 = a * (x + 1 + 1) + b * (y + 0.5) + c
d2 = a * x + 2* a + b * y + 0.5 * b + c

now if we subtract the old d
d = a * (x + 1) + b * (y+ 0.5) + c
d = a * x + a + b * y + 0.5 * b + c
d2-d = ~~a * x~~ + ~~2*~~ a + ~~b * y + 0.5 * b + c~~
d2 – d = a
therefore if we move east we just need to add a to d to get the new d

if we move northeast green
x2 = x+1
y2 = y+1
d2 = a * (x + 1 + 1) + b * (y + 0.5 + 1) + c
d2 = a * x + 2* a + b * y + 1.5 * b + c

now if we subtract the old d
d = a * (x + 1) + b * (y+ 0.5) + c
d = a * x + a + b * y + 0.5 * b + c
d2-d = ~~a * x~~ + ~~2*~~ a + ~~b *~~ y + 1~~.5~~ * b~~ + c~~
d2 – d = a + b
therefore if we move east we just need to add a and b to d to get the new d

looking back
a = dy
b = -dx

To get an initial value for d to start this all of we go
d = a * (x + 1) + b * (y+ 0.5) + c

BUT we know
0 = a * x + b * y + c
(as the first point HAS to be on the line)
so we can subtract this from the starting point to get
d = a + 0.5 * b

we can even get rid of the 0.5 * b as all we need to know is d relative to 0 if we just double what we add to it we can do this
d = 2 * a + b

SO SUMMARY
a = dy
b = -dx

Start
d = 2 * dy – dx
east = 2 * dy
northeast = 2 * dy – 2 * dx

```
while not at end
{
        if (d>0)
        {
                d+=east
                x++;
        }
        else if (d<0)
        {
                d+=northeast
                x++;
                y++;
        }
        plot x, y
}
```

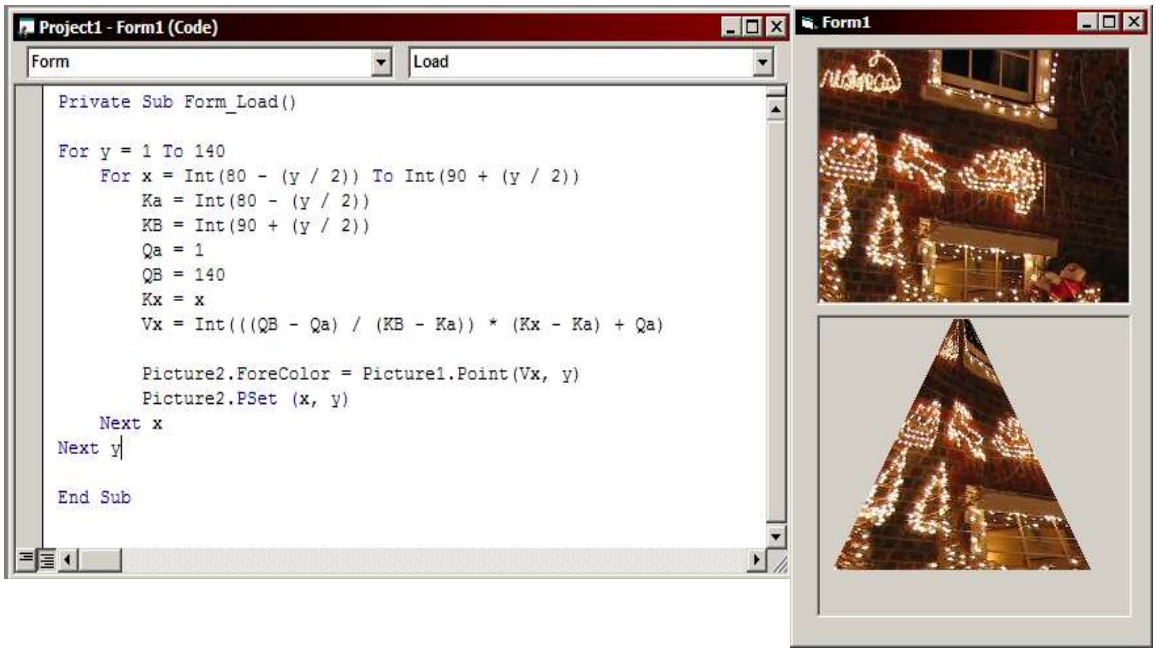
Flood Fill
-Recursive easy

Edge Coherence Polygon Filling
NEED TO FINISH

XY Algorithm
NEED TO FINISH

Converting Polygon to Triangles (See notes)
Find left most vertex min(x) min(y)
Generate triangle from 2 adjacent verticies
if no points inside then cut off as new triangle
else make new triangle with left most point and the leftmost of the inside points.

Bi-Liner Interpolation
Stretches / Skews a texture across a polygon.



How

This is just working in X, just do the same for Y

we have width of texture

T_left, T_right

we have width of polygon at this point

P_left P_right

we have out position on the polygon

P_pos

get the width

Texture_Width = T_right – T_left

Polygon_Width = P_right – P_left

calculate the scaling factor

Scale_Factor = Texture_Width / Polygon_Width

calulate how far we our into the polygon

Position_Offset = P_pos - P_left

Calulate where this is on the texture

Texture_Pos = Scale_Factor * Position_Offset

Finally the left hand side

Position_in_Image = Texture_Pos + T_Left

So putting this togethor

Position_in_Image =
((T_right– T_left) / (P_right – P_left)) * (P_pos – P_left) + T_left

Now from that it can be seen as P_pos increments by 1 each time the only change is multiplying the scale_factor by 1 more. Therefore if we calculate the initial Position we can just add the scale_factor each time after that.

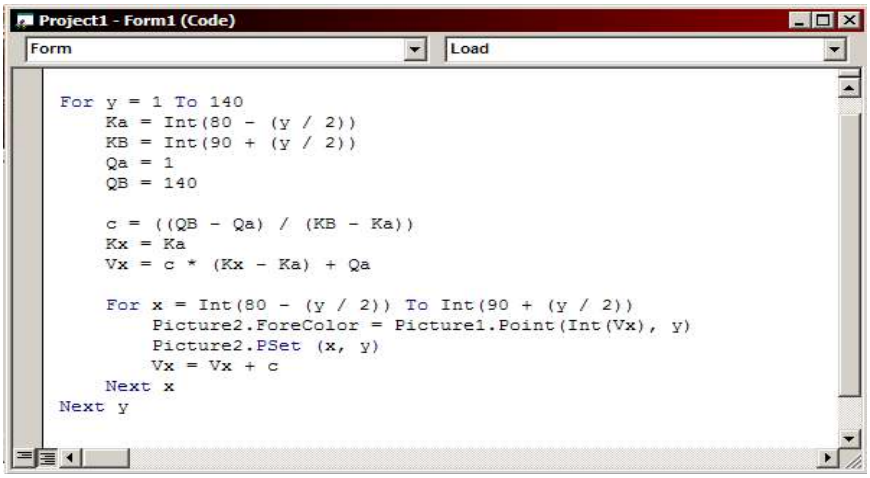


Illustration 1Bi-Linear Filtering Example with Speedup

Clipping

Cohen-Sutherland Line-Clipping

IMPORTANT HE HAS AN ERROR IN THIS SECTION

Compute the outcodes

1001	1000	1010
0001	0000	0010
0101	0100	0110

If OR of the end point outcodes = 0 then line is fully inside
if AND of the end point outcodes > 0 then lines is fully outside

else

calulate an intersection(you know what sides to test) then retest the
end

NOTE line doesn't have to be inside ATALL (he says it means it has to cross)
proof what if a line went from 0100 to 0010 crossing 0110 NOT 0000

Liang-Barsky Algorithm

Choose a vertex on the polygon as a start point find out if its inside or outside the rectangle that is being clipped to.

Loop around the rest of the polygon looking at each segment
4 things we can do

- Inside - Add entire line
- Going out – Add start calculate intersection for new end
- Outside – throw away
- Going in – Calculate intersection for new start and add end

2D/3D Transformations

- Homogeneous Co-ordinates
  - x, y, w
  - proper x = x/w
  - proper y = y/w

  Why ?
  Introduce the concept of infinity VERY useful for maths stuff
  i.e. As w approaches 0 proper x,y approach infinity

- Transformations
  - 2D

    Translate x, y $\begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}$

    Scale x,y $\begin{bmatrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

    Rotate phi $\begin{bmatrix} \cos(\Phi) & -\sin(\Phi) & 0 \\ \sin(\Phi) & \cos(\Phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

  - 3D

    Translate x,y,z $\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

    Scale x,y,z $\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rotate about z $\begin{bmatrix} \cos(\Phi) & -\sin(\Phi) & 0 & 0 \\ \sin(\Phi) & \cos(\Phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rotate about x $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\Phi) & -\sin(\Phi) & 0 \\ 0 & \sin(\Phi) & \cos(\Phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rotate about y $\begin{bmatrix} \cos(\Phi) & 0 & \sin(\Phi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\Phi) & 0 & \cos(\Phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Projections

- Parallel
  - Throw away the z keep everything else $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- Orthographic
- Oblique
- Perspective
  - Not 100% sure about this
  - Method one

    $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$

    then divide though by the w value each time

  - Method two

    $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix}$

How OPENGL Works

- 2 Main Matricies
  - Projection Matrix - this says how the world is viewed by the camera
    - Setup using glFrustrum and glOrtho

Modelview Matrix – this says how the current item is transformed

To render a point multiply it by the modelview then the projection matrix then x,y will be point pos and z will be depth.

BSP Trees

Lighting
- Ambient
- Diffuse

Ray tracing

Splines
- Continuity
  - End of first curve = beginning of next $f1(A) = f2(A)$
- Smoothness
  - Gradient of first curve = gradient of next $f1'(A) = f2'(A)$

- Cubic sections
  - CANT USE $y=ax^3 + bx^2 + cx + d$
  - Parametric form
    - $X(t) = a_x t^3 + b_x t^2 + c_x t + d_x$
    - $Y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$

    3D obvious just have x,y,z

    $a_x, a_y, b_x, b_y, c_x, c_y, d_x, d_y$ are the control variables
    see notes

- Interpolating

- Control points
  - Hermite Curves
    - Ends of curve and the vectors for the gradient at the ends
    - P1,P2 = end points
    - R1,R2 = gradients (vectors from end points)

  - Bezier Curves
    - P1,P4 = end points
    - P2,P3 = control points

    R1 = 3*(P2-P1)

R2= 3*(P4-P3)

Blending Matrix

$$\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

See notes for example of usage

Where the blending matrix comes from probably not needed

Work out the numbers there will be of each co-efficient
M = Blending Matrix
G = Control Points

$X(t) = a_x t^3 + b_x t^2 + c_x t + d_x$
We know the end conditions
t=0
$X(0) = P1_x = (0, 0, 0, 1) * M * G_x$

t=1
$X(0) = P1_x = (1, 1, 1, 1) * M * G_x$

We know the gradients
$dx/dt = 3a_x t^2 + 2b_x t + c_x$
$dx/dt\ (0) = R1_x = (0, 0, 1, 0) * M * G_x$
$dx/dt\ (1) = R2_x = (3, 2, 1, 0) * M * G_x$

therefore we know

$$\begin{bmatrix} P_1 \\ P_4 \\ 3\cdot(P_2-P_1) \\ 3\cdot(P_4-P_3) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot M \cdot \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

Same as

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot M \cdot \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

Post multiply both sides by the inverse of the 3rd one

$$\begin{bmatrix}0&0&0&1\\1&1&1&1\\0&0&1&0\\3&2&1&0\end{bmatrix}^{-1}\begin{bmatrix}1&0&0&0\\0&0&0&1\\-3&3&0&0\\0&0&-3&3\end{bmatrix}.\begin{bmatrix}P_1\\P_2\\P_3\\P_4\end{bmatrix}=\begin{bmatrix}0&0&0&1\\1&1&1&1\\0&0&1&0\\3&2&1&0\end{bmatrix}^{-1}\begin{bmatrix}0&0&0&1\\1&1&1&1\\0&0&1&0\\3&2&1&0\end{bmatrix}.M.\begin{bmatrix}P_1\\P_2\\P_3\\P_4\end{bmatrix}$$

Simple observation shows

$$\begin{bmatrix}0&0&0&1\\1&1&1&1\\0&0&1&0\\3&2&1&0\end{bmatrix}^{-1}\begin{bmatrix}1&0&0&0\\0&0&0&1\\-3&3&0&0\\0&0&-3&3\end{bmatrix}.\begin{bmatrix}P_1\\P_2\\P_3\\P_4\end{bmatrix}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}.M.\begin{bmatrix}P_1\\P_2\\P_3\\P_4\end{bmatrix}$$

And so

$$\begin{bmatrix}0&0&0&1\\1&1&1&1\\0&0&1&0\\3&2&1&0\end{bmatrix}^{-1}\begin{bmatrix}1&0&0&0\\0&0&0&1\\-3&3&0&0\\0&0&-3&3\end{bmatrix}.\begin{bmatrix}P_1\\P_2\\P_3\\P_4\end{bmatrix}=M.\begin{bmatrix}P_1\\P_2\\P_3\\P_4\end{bmatrix}$$

More calculation shows, then

$$\begin{bmatrix}-1&3&-3&1\\3&-6&3&0\\-3&3&0&0\\1&0&0&0\end{bmatrix}.\begin{bmatrix}P_1\\P_2\\P_3\\P_4\end{bmatrix}=M.\begin{bmatrix}P_1\\P_2\\P_3\\P_4\end{bmatrix}\qquad\begin{bmatrix}-1&3&-3&1\\3&-6&3&0\\-3&3&0&0\\1&0&0&0\end{bmatrix}=M$$

How to Draw them

Bezier curve can easily be split into 2

Original Curve P1,P2,P3,P4

new Curves

h = (P2 + P3)/2

L1 = P1
L2 = (P1 + P2) / 2
L3 = (h + L2)/2
L4 = R1 = (L3 + R2) / 2
R2 = (h + R3)/2
R3 = (P3 + P4)/2
R4 = P4

Stop when

All control points same value ie its a point
All control points in a line

And another ???

Bezier surfaces

same principals as Bezier Curve but in 3D