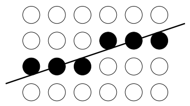
Basic Raster Algorithms for 2D Graphics

- The conversion: primitives → pixmap is scan conversion.



- The most simple scan conversion task, is scan converting a line.
- Theoretically, any scan conversion algorithm should produce a line with constant brightness (independent of orientation and length) and do it quickly.
- It should also be able to cope with lines greater than one pixel in width and be able to display different styles and attributes.
- It is also necessary to minimise the jaggies on the line, by using anti-aliasing methods to set pixel intensities.

Scan converting lines

- Assume that pixels are disjoint circles on an integer, (x, y) grid.
- Line starts and ends at integer coordinates (x_0, y_0) and (x_e, y_e).
- The simplest algorithm to scan convert the line is an incremental one:
  - Compute the slope $m = \Delta y / \Delta x$,
  - Start at the leftmost point and increment x by 1,
  - Calculate $y_i = m x_i + c$,
  - Set the intensity at the pixel value $(x_i, \text{Round}(y_i))$.
- Inefficient, works only for $|m| < 1$.

Scan converting lines

- Note that $y_{i+1} = m x_{i+1} + c = m(x_i + \Delta x) + c = y_i + m\Delta x$.
- $\Delta x = 1$ gives:

$$x_{i+1} = x_i + 1 \,,$$

$$y_{i+1} = y_i + m \,.$$

- More efficient algorithm, which works so long as $|m| < 1$.
- If this is not the case, it is necessary to swap x and y, which will give a slope of $1/m$.
- It is also necessary to check for the special conditions of horizontal, vertical and diagonal lines.

Scan converting lines

```
 void Line ( int x0,  int xe,  int y0,  int ye,  int value)
{           /* Assumes -1 <= m <= 1, x0 < xe */
    int x;
    float y,dx,dy,m;

   dx = xe - x0;
   dy = ye - y0;
   m = dy / dx;
   y = y0;

    for (x = x0; x <= xe; x++){
      WritePixel(x,( int) floor(y + 0.5),value);
      y += m;
   }
}
```

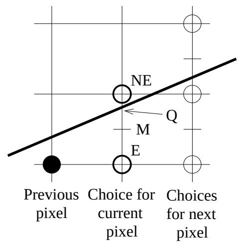
- The algorithm is referred to as the digital differential analyser.
- One potential problem with the method arises because the float m has a finite precision.

Scan converting lines

- A more advanced algorithm is the midpoint line algorithm.
- Does not use floating point arithmetic.
- Generalisation of Bresenham's well known incremental technique.
- Works only for $0 \leq m \leq 1$.
- Other slopes are catered for by reflection about the principal axes of the 2D plane.
- (x_0, y_0) is the lower left endpoint and (x_e, y_e) the upper right endpoint.

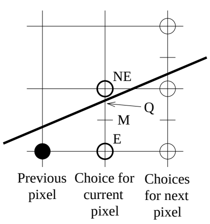
Scan converting lines

- Incremental method:



- Restrictions on the slope of the line implies that if we are at some pixel (x_p, y_p), we now need to choose between only two pixels.

Scan converting lines



- Either the east pixel, E, or the northeast pixel, NE, is chosen depending upon which side of the midpoint, M, the crossing point, Q, lies.
- Write the implicit functional from: $f(x, y) = ax + by + \gamma = 0$.

Scan converting lines

- Noting:

$$y = mx + c = \frac{\Delta y}{\Delta x} x + c \,,$$

gives:

$$f(x, y) = \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot c \,.$$

- Now $a = \Delta y$, $b = -\Delta x$ and $\gamma = \Delta x \cdot c$.
- For any point on the line, $f(x, y)$ is zero, any point above the line, $f(x, y)$ is negative and any point below has $f(x, y)$ positive.
- $d = f(\text{M}) = f(x_p + 1, y_p + 1/2) = a(x_p + 1) + b(y_p + 1/2) + \gamma$. If d is positive we choose NE, otherwise we pick E (including when $d = 0$).

Scan converting lines

- So what happens to the location of the next midpoint, M_{new}? This depends on whether E or NE was chosen. If E is chosen then the new d_{new} will be:

$$\begin{aligned} d_{new} &= f(\mathsf{M}_{new}) = f(x_p + 2, y_p + 1/2) \\ &= a(x_p + 2) + b(y_p + 1/2) + \gamma \,. \end{aligned}$$

- $d_{new} = d_{old} + \Delta_E$, $\Delta_E = a$.
  Similarly $\Delta_{NE} = a + b$.

- First $d = f(x_0 + 1, y_0 + 1/2) = a(x_0 + 1) + b(y_0 + 1/2) + \gamma = f(x_0, y_0) + a + \frac{b}{2}$, since this on the line $d = a + b/2 = \Delta y - \Delta x/2$.

- Using $d = 2f(x, y)$, which will not affect the sign of the decision variable and keep everything integer.

Scan converting lines

```
 void MidpointLine ( int x0,  int xe,
                int y0,  int ye,  int value)
{  /* Assumes 0 <= m <= 1, x0 < xe, y0 < ye */
    int x,y,dx,dy,d,incE,incNE;

   dx = xe - x0;
   dy = ye - y0;
   d = 2*dy - dx;
   incE = 2*dy;
   incNE = 2*(dy-dx);
   x = x0;
   y = y0;
   WritePixel(x,y,value);
    while (x < xe) {
       if (d <= 0) {
         d += incE;
         x++;
      }  else {
         d += incNE;
         x++;
         y++;
      }
      WritePixel(x,y,value);
   }
}
```

Scan converting lines

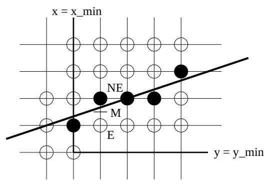
- There are several improvements that could be envisaged to the midpoint algorithm.

- One method involves looking ahead two pixels at a time (so called double-step algorithm).

- Another uses the symmetry about the midpoint of the whole line, which allows both ends to be scan converted simultaneously.

- The midpoint algorithm defines that E is chosen when Q = M so to ensure lines look the same drawn from each end the algorithm should choose SW rather than W in the inverted version.

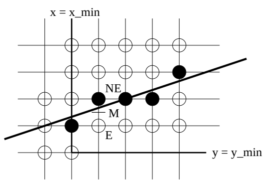
Line clipping



- It is common to clip a line by a bounding rectangle (often the virtual or real screen boundaries).

- Assume the bounding rectangle has coordinates, (x_{min}, y_{min}), (x_{max}, y_{max}).

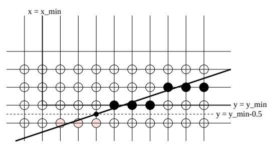
Line clipping



- If the line intersects the left hand vertical edge, $x = x_{min}$ the intersection point of the line with the boundary is $(x_{min}, (m \cdot x_{min} + c))$.

- Start the line from $(x_{min}, \mathrm{Round}(m \cdot x_{min} + c))$.

Line clipping



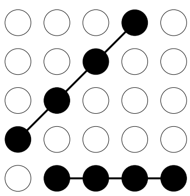
- Assume that any of the lines pixels falling on or inside the clip region are drawn.

- The line does not start at the point $((y_{min} - c)/m, y_{min})$ where the line crosses the bounding line.

- The first pixel is

$$\left(\mathrm{Round}\left(\frac{(y_{min} - 0.5 - c)}{m}\right), y_{min}\right) .$$

Line intensity



- Lines of different slopes will have different intensities on the display, unless care is taken.

- 2 lines, both 4 pixels but the diagonal one is $\sqrt{2}$ times as long as the horizontal line.

- Intensity can be set as a function of the line slope.

Scan converting area primitives

```
 void FillRectangle ( int xmin,  int xmax,
                int ymin,  int ymax,  int value)
{
    int x,y;
    for (y = ymin; y <= ymax; y++) {
       for (x = xmin; x <= xmax; x++) {
         WritePixel(x,y,value);
      }
   }
}
```

- Scan converting objects with area is more complex than scan converting linear objects, due to the boundaries.
  A rule that is commonly used to decide what to do with edge pixels is as follows.

- A boundary pixel is not considered part of the primitive if the half-plane defined by the edge and containing the primitives lies below a non-vertical edge or to the left of a vertical edge.

Filling polygons

- Most algorithms work as follows:
  - find the intersections of the scan line with all polygon edges;
  - sort the intersections;
  - fill those points which are interior.
- The first step involves the use of a scan-line algorithm that takes advantage of edge coherence to produce a data structure called an active-edge table.
- Edge coherence simply means that if an edge is intersected in scan line i, it will probably be intersected in scan line $i+1$.

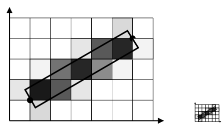
Other issues

- Patterns will typically be defined by some form of pixmap pattern, as in texture mapping.
- In this case the pattern is assumed to fill the entire screen, then `and`ed with the filled region of the primitive, determining where the pattern can 'show through'.
- It is convenient to combine scan conversion with clipping in integer graphics packages, this being called scissoring.
- Floating point graphics are most efficiently implemented by performing analytical clipping in the floating point coordinate system and then scan converting the clipped region.

Scan conversion: OpenGL

- OpenGL performs scan conversion efficiently behind the scenes – typically using hardware on the graphics card.
- However, we can manipulate pixels using OpenGL with `glRasterPos2i(GLint x, GLint y)` and `glDrawPixels(·)` – in the labs you will code your own scan conversion routines.
- Speed is often of the essence in computer graphics, so designing and developing efficient algorithms forms a large part of computer graphics research.

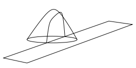
Anti-aliasing



- All raster primitives outlined so far have a common problem, that of jaggies : jaggies are a particular instance of aliasing. The term alias originates from signal processing.
- In the limit, as the pixel size shrinks to an infinitely small dot, these problems will be minimised, thus one solution is to increase the screen resolution.
- Doubling screen resolution will quadruple the memory requirements and the scan conversion time.

Anti-aliasing

- One solution to the problem involves recognising that primitives, such as lines are really areas in the raster world.
- In unweighted area sampling the intensity of the pixel is set according to how much of its area is overlapped by the primitive.
- More complex methods involve weighted area sampling.



- Weighted area sampling assumes a realistic model for pixel intensity. Using a sensible weighting function, such as a cone or Gaussian function, will result in a smoother anti-aliasing, but at the price of even greater computational burden.

Anti-aliasing: OpenGL

- Since anti-aliasing is an expensive operation, and may not always be required OpenGL allows the user to control the level of anti-aliasing.
- Can be turned on using: `glEnable(GL_LINE_SMOOTH)`
- Can also use `glHint(GL_LINE_SMOOTH_HINT,GL_BEST)` to set quality: `GL_BEST`, `GL_FASTEST`, and `GL_DONT_CARE` – hints not always implemented – based on number of samples.
- Works by using the alpha parameter and colour blending. Anti-aliasing of polygons treated in the same way in RGBA mode.

Summary

- Having finished this lecture you should:
  - know what scan conversion means;
  - be able to contrast different appraoches and sketch their application;
  - provide simple solutions to the problems of clipping and aliasing;
  - understand how scan conversion works in OpenGL.
- This completes the graphics part of the module.