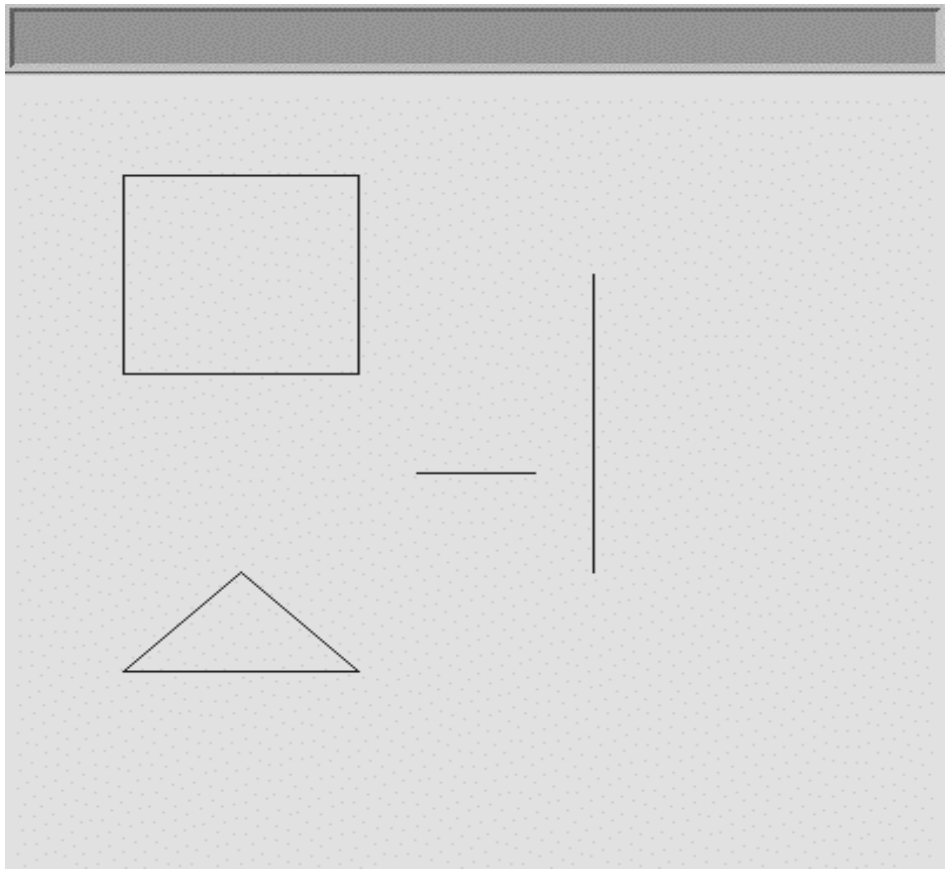
Rendering

- **A simple X program to illustrate rendering**

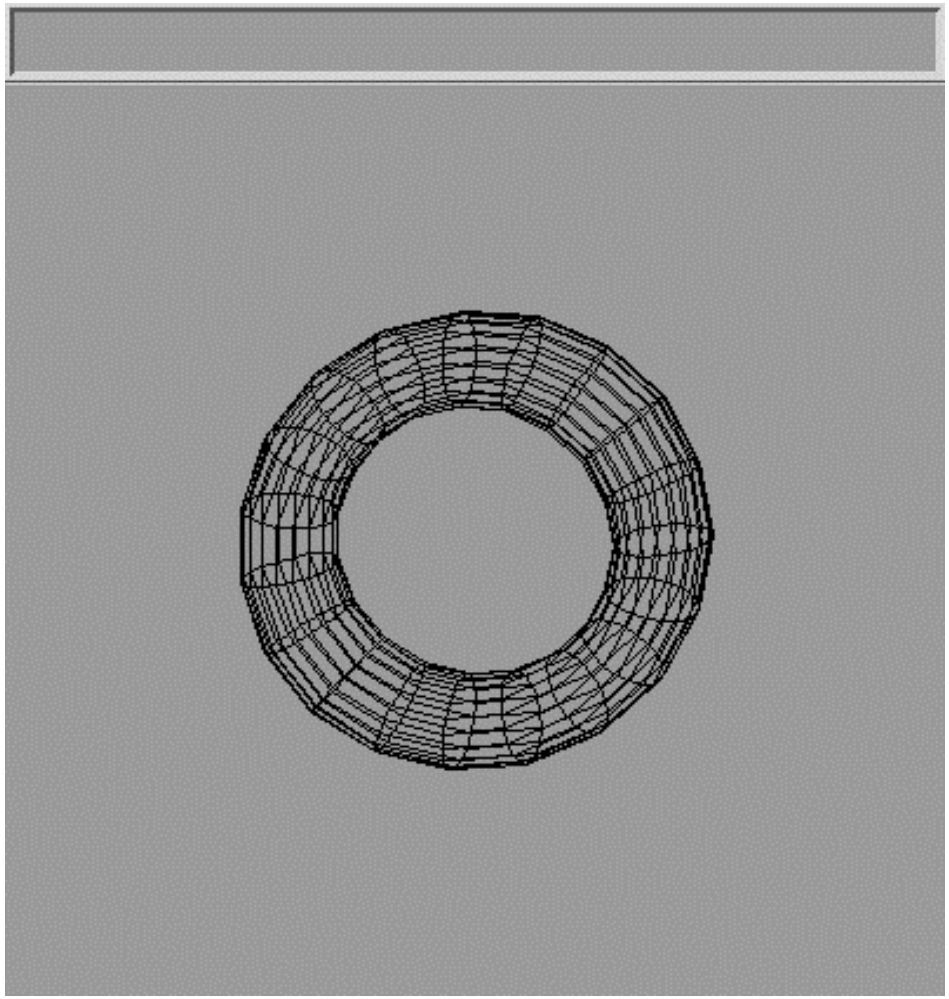
- The programs in this directory provide a simple x based application for us to develop some graphics routines.

- Please notice the following:
  - All points are integers.
  - The line drawing algorithm is very simple and has some definite flaws.

A simple X program



A simple X program



Syntax

- Point
  - point x y where x and y are integers
- Line
  - line x1 y1 x2 y2
  - where x1, x2, y1 and y2 are integers

Point and line drawing routines

- These are the *primitives* in the simple package!!

- Point drawing routines
  - point.c
  - point.h

- Line drawing routines
  - line.c
  - line.h

Other routines

- The main routine
  - main.c
- Include file with the definitions of the data structures
  - mygraph.h
- The routines to Access X functions directly.
  - Xroutines.c
  - Xroutines.h

Line Drawing and Scan Conversion

- A preliminary step to drawing lines is choosing a suitable representation for them.
- There are three possible choices which are potentially useful.
- Explicit: y = f(x)
  - y = m (x - x0) + y0 where m = dy/dx
- Parametric: x = f(t), y = f(t)
  - x = x0 + t(x1 - x0), t in [0,1]
  - y = y0 + t(y1 - y0)

Point and line drawing routines (ctd)

- Implicit: f(x,y) = 0
  - F(x,y) = (x-x0)dy - (y-y0)dx
  - if F(x,y) = 0 then (x,y) is on line
  - F(x,y) > 0 then (x,y) is below line
  - F(x,y) < 0 then (x,y) is above line

Rasterization or Scan Conversion

- Drawing lines on a raster grid implicitly involves approximation.
- The general process : *rasterization* or *scan-conversion*.
- What is the best way to draw a line from the pixel (x1,y1) to (x2,y2)?
- Such a line should ideally have the following properties.
  - straight
  - pass through endpoints
  - smooth
  - independent of endpoint order
  - uniform brightness
  - brightness independent of slope
  - efficient

Line Drawing - Algorithm 1

A Straightforward Implementation

```
Drawline(x1,y1,x2,y2)
int x1,y1,x2,y2;
{
  float y;
  int x;

  for (x=x1; x<=x2; x++) {
    y = y1 +  (x-x1)*(y2-y1)/(x2-x1)
    SetPixel(x, Round(y) );
  }
}
```

Line Drawing - Algorithm 2

A Better Implementation

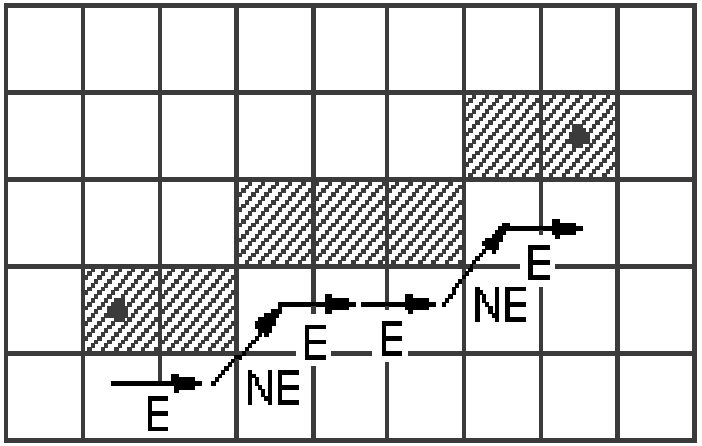
```
DrawLine(x1,y1,x2,y2)
int x1,y1,x2,y2;
{
  float m,y;
  int dx,dy,x;
  dx = x2 - x1;
  dy = y2 - y1;
  m = dy/dx;
  y = y1 + 0.5;
  for (x=x1; x<=x2; x++) {
    SetPixel(x, Floor(y) );
    y = y + m;
  }
}
```

Line Drawing - Algorithm 2

- Advantages over Algorithm 1
  - eliminates multiplication
  - improves speed
- Disadvantages
  - round-off error builds up
  - get pixel drift
  - rounding and floating point arithmetic still time consuming
  - works well only for $|m| < 1$
  - need to loop in y for $|m| > 1$
  - need to handle special cases

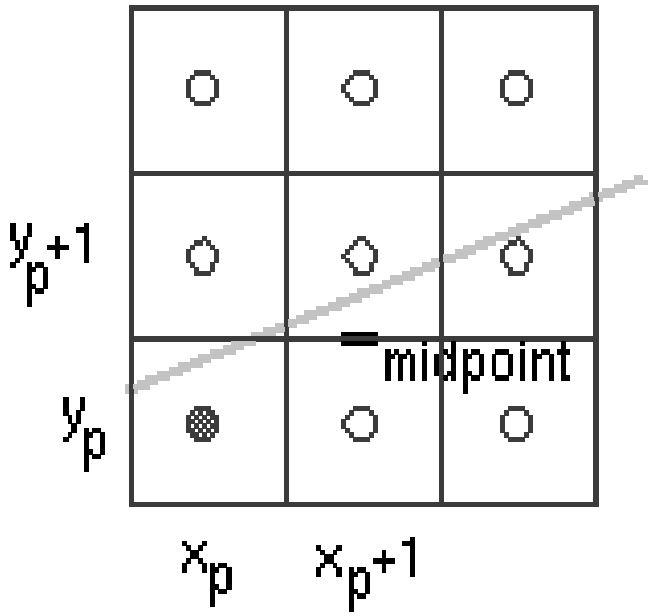
Line Drawing - Midpoint Algorithm

- The Midpoint or Bresenham's Algorithm
  - The midpoint algorithm is even better than the above algorithm in that it uses only integer calculations. It treats line drawing as a sequence of decisions. For each pixel that is drawn the next pixel will be either N or NE, as shown below.



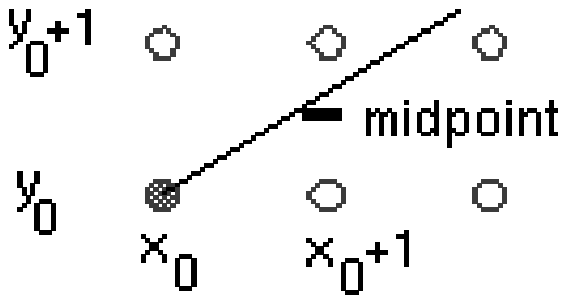
Midpoint Algorithm

- The midpoint algorithm makes use of the the implicit definition of the line, F(x,y) = 0. The N/NE decisions are made as follows.
- d = $F(x_p + 1, y_p + 0.5)$
  - if d < 0 line below midpoint choose E
  - if d > 0 line above midpoint choose NE
- if E is chosen
  - $d_{new} = F(x_p + 2, y_p + 0.5)$
  - $d_{new} - d_{old} = F(x_p + 2, y_p + 0.5) -$
    - $F(x_p + 1, y_p + 0.5)$
  - $\Delta d = d_{new} - d_{old} = dy$



Midpoint Algorithm

- If NE is chosen
  - $d_{new} = F(x_p + 2, y_p + 1.5)$
  - $\Delta d = dy - dx$
- Initialization
  - $d_{start} = F(x_0 + 1, y_0 + 0.5) = (x_0 + 1 - x_0)\, dy - (y_0 + 0.5 - y_0)dx$
    $= dy - dx/2$
- Integer only algorithm
  - $F'(x,y) = 2\, F(x,y)$; $d' = 2d$
  - $d'_{start} = 2dy - dx$
  - $\Delta d' = 2\Delta d$



Midpoint Algorithm for x1 < x2 and slope <= 1

```
drawline(x1, y1, x2, y2, colour)
int x1, y1, x2, y2, colour;
{
  int dx, dy, d, incE, incNE, x, y;

  dx = x2 - x1;
  dy = y2 - y1;
  d = 2*dy - dx;
  incE = 2*dy;
  incNE = 2*(dy - dx);
  y = y1;
  for (x=x1; x<=x2; x++) {
    setpixel(x, y, colour);
    if (d>0) {
      d = d + incNE;
      y = y + 1;
    } else {
      d = d + incE;
    } } }
```