# CSE 3200 Micro-Computer Graphics
# 2D/3D Geometry

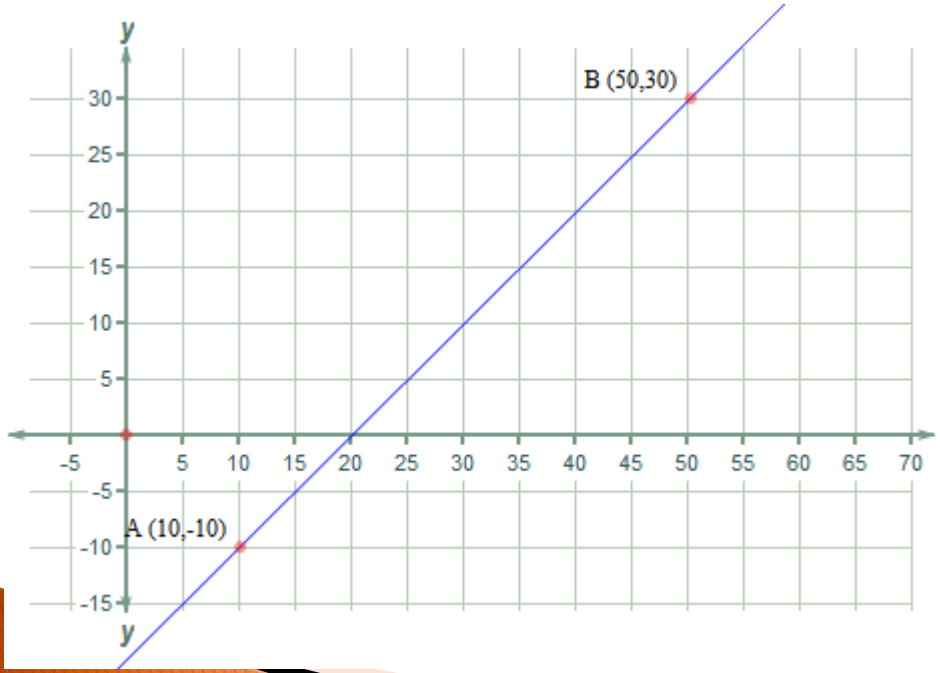
Presenter: Girendra Persaud
University of Guyana

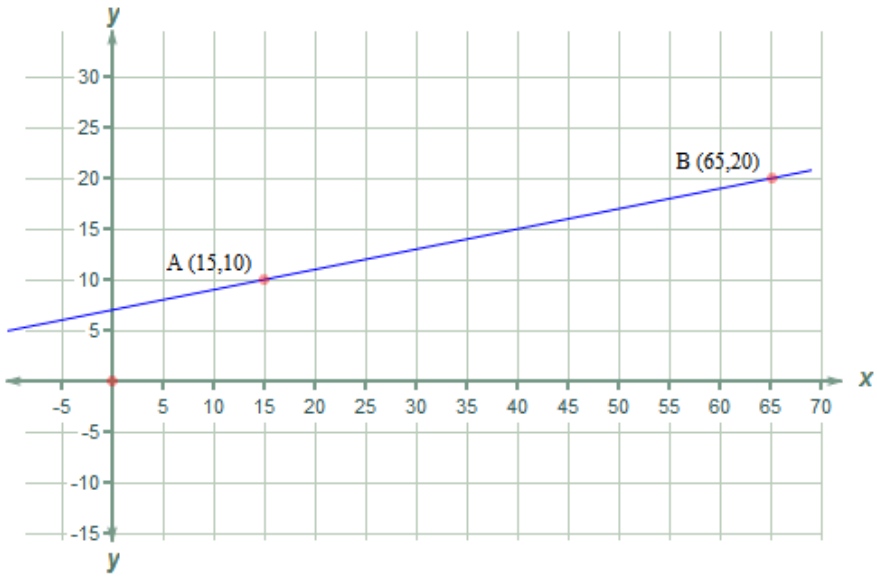
# Outline

- Definition of a Line (segment)
  - Length of a line
  - Point on a line
  - Line Drawing Algorithms
- Definition of a plane
  - Point on a plane
  - Calculating a plane from 3 points
- Definition of a circle
- Questions?

# Definition of a line

- A geometrical object that is straight, infinitely long and infinitely thin. Its location is defined by two or more points on the *line* whose coordinates are known.



# Definition of a line

- Define by its equation
  - "slope-intercept" form:
    - $y = mx + b$ *(x increment) where* $(x_{start} < x < x_{stop})$
    - $X = (y - b)/m$ *(y increment) where* $(y_{start} < y < y_{stop})$
- Length of line:
  - $d=\sqrt{((dx)^2+(dy)^2)}$
- Point on a Line:
  - Will satisfy line equation

# Line Drawing

- Line drawing is an area of *scan conversion*.
- Scan Conversion: rasterization – conversion of vector data to pixels on a rater display

# Lab Task 1

- Draw a line segment (-5500, 900, 5500, -5000) using the formula y=mx+c, where m is the  (Change in X/Change in Y) – save your project as “line1”
- Time: 15 Minutes

# Algorithm 1

- Drawline(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) );
  - }
- }

- Pros: easy to understand
- Cons: slow – heavy processing (multiplication & Division for each pixel in the line)

# Algorithm 2 – DDA

- **Digital Differential Analyzer**
  - A mechanical device use for numerical solutions of differential equations
  - DDA method is to take unit steps along one coordinate and compute the corresponding coordinate values
  - The unit steps are always along the coordinate of greatest change

# DDA Algorithm

- DDADraw(x1, x2, y1, y2) (all int)
  - int: dx, dy, steps;
  - float: x_inc, y_inc, x, y;
  - dx = x2-x1; dy = y2-y1;
  - if (dx > dy) steps = dx; else steps = dy;
  - x_inc = dx/steps; y_inc = dy/steps;
  - x = x1; y = y1;
  - set_pixel(round(x), round(y));
  - for (i=1, I < steps, i++) {
    - x+=x_inc;
    - y+=y_inc;
    - set_pixel(round(x), round(y));
  - }

# Lab Task 2

- Implement the line
  (-5500, 900, 5500, -5000)
  using the DDA Algorithm – save your project as “DDA”
- Time: 25 Minutes

# DDA Algorithm

- Pros over Algorithm 1
  - Faster
  - Eliminates multiplications

- Cons – generally
  - Floating point arithmetic
  - Rounding is time consuming
  - Round-off error build up
  - Ok only for $|m| < 1$

# Bresenham's Line Algorithm

- uses only integer addition, subtraction and bit shifting all of which are very cheap operations in standard computer architectures

# Bresenham's Line Algorithm

```
void line(int x0, int y0, int x1, int y1) {
    int Dx = x1 - x0;   int Dy = y1 - y0;
    int steep = (abs(Dy) >= abs(Dx));
    if (steep) { SWAP(x0, y0);  SWAP(x1, y1); Dx = x1 - x0; Dy = y1 - y0;   }
    int xstep = 1;
    if (Dx < 0) { xstep = -1; Dx = -Dx;   }
    int ystep = 1;
    if (Dy < 0) { ystep = -1;  Dy = -Dy;   }
    int TwoDy = 2*Dy;  int TwoDyTwoDx = TwoDy - 2*Dx;
    int E = TwoDy - Dx;
    int y = y0;  int xDraw, yDraw;
    for (int x = x0; x != x1; x += xstep) {
        if (steep) { xDraw = y;  yDraw = x;  } else { xDraw = x; yDraw = y; }
        plot(xDraw, yDraw);
        if (E > 0) { E += TwoDyTwoDx;  y = y + ystep; } else {E += TwoDy;}
    }
}
```

# Lab Task 3

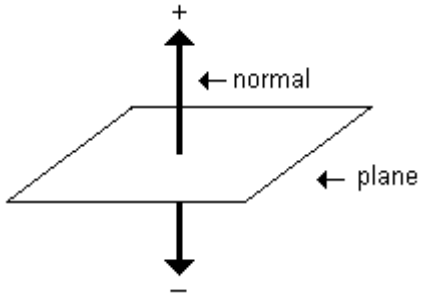
- Implement the line segment

(-5500, 900, 5500, -5000)

using the Bresenham's Line Algorithm – save your work as "Bresenhams"

- Time: 25 Minutes

# Definition of a plane



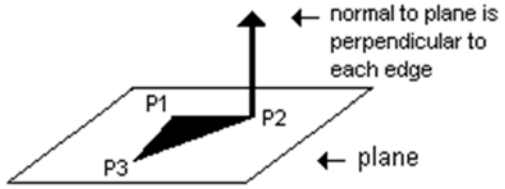

- A plane is an infinitely wide flat surface
  - defined by a normal $(n_x,n_y,n_z)$ and a scalar value k
  - The normal can be thought of as representing the direction that the surface of the plane is facing

# Point on a plane

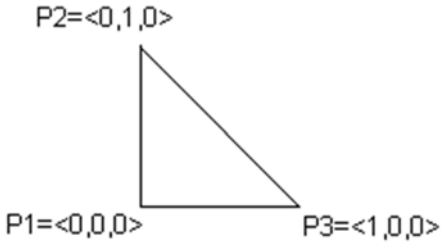
- All points (x,y,z) that lie on the plane will satisfy this equation.
  - $(x,y,z) \cdot (n_x,n_y,n_z) = k$
- **If $(x,y,z) \cdot (n_x,n_y,n_z) \neq k$ (Point is on one side of plane)**
- The vector $(n_x,n_y,n_z)$ and scalar k are unique to every plane
- These equations are helpful in performing back-face culling. Substitute the view point into the equation, if the value comes out less than k then you know that you are facing the "back" side of the polygon and thus don't need to draw it.

# Calculating a plane from 3 points

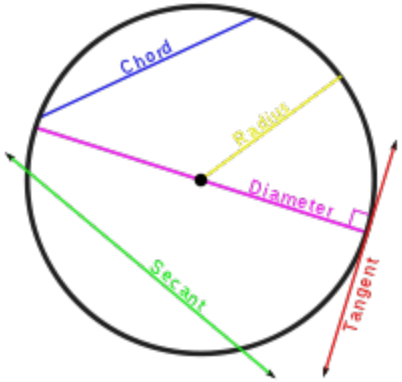
- For points p1,p2 and p3 we get:
  - normal = (p1–p2) X (p3–p2)
  - k = normal . P1

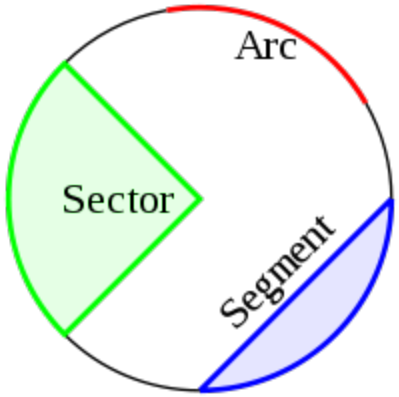



- Example:
  - normal = (p1–p2) X (p3–p2)
    - = (0,–1,0) X (1,–1,0)
    - = ((–1)*0 – 0*(–1), 0*1 – 0*0, 0*(–1) – (–1)*1)
    - = (0,0,1)
  - K = (0,0,1) . (0,0,0)
    - = 0
    - (plane is through the origin, point is on the plane)



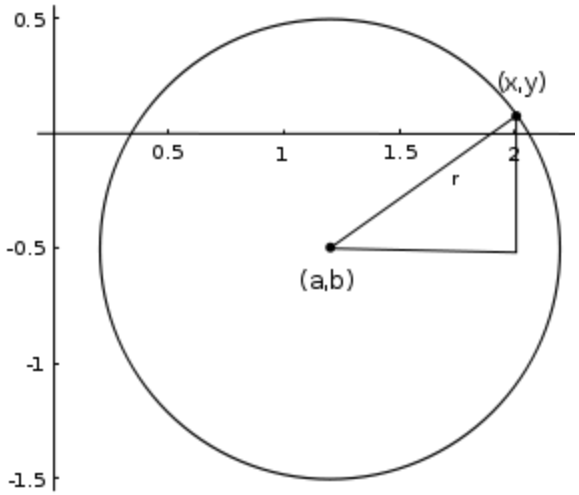
# Definition of a circle




- $x^2 + y^2 = r^2$ (at origin)
  - $x = \sqrt{(r^2 - y^2)}$

- $(x-a)^2 + (y-b)^2 = r^2$



# Lab Task 4

- Draw a circle using the formula

$x^2 + y^2 = r^2$, where the centre is at the origin and $r = 4300$;

  - Time: 20 Minutes

# Circle Drawing Algorithm

- The approach for the **Circle Variant** of the Bresenham's Algorithm starts with circle equation
- $x^2+y^2=r^2$
- draw a curve which starts at point (r,0)
- proceed to the top left, up to reaching the angle of 45° and reflect in all remaining octants

# Summary

- Definition of a Line (segment)
  - Length of a line
  - Point on a line
  - Line Drawing Algorithms
- Definition of a plane
  - Point on a plane
  - Calculating a plane from 3 points
- Definition of a circle
  - Circle Drawing Algorithm

# Questions?