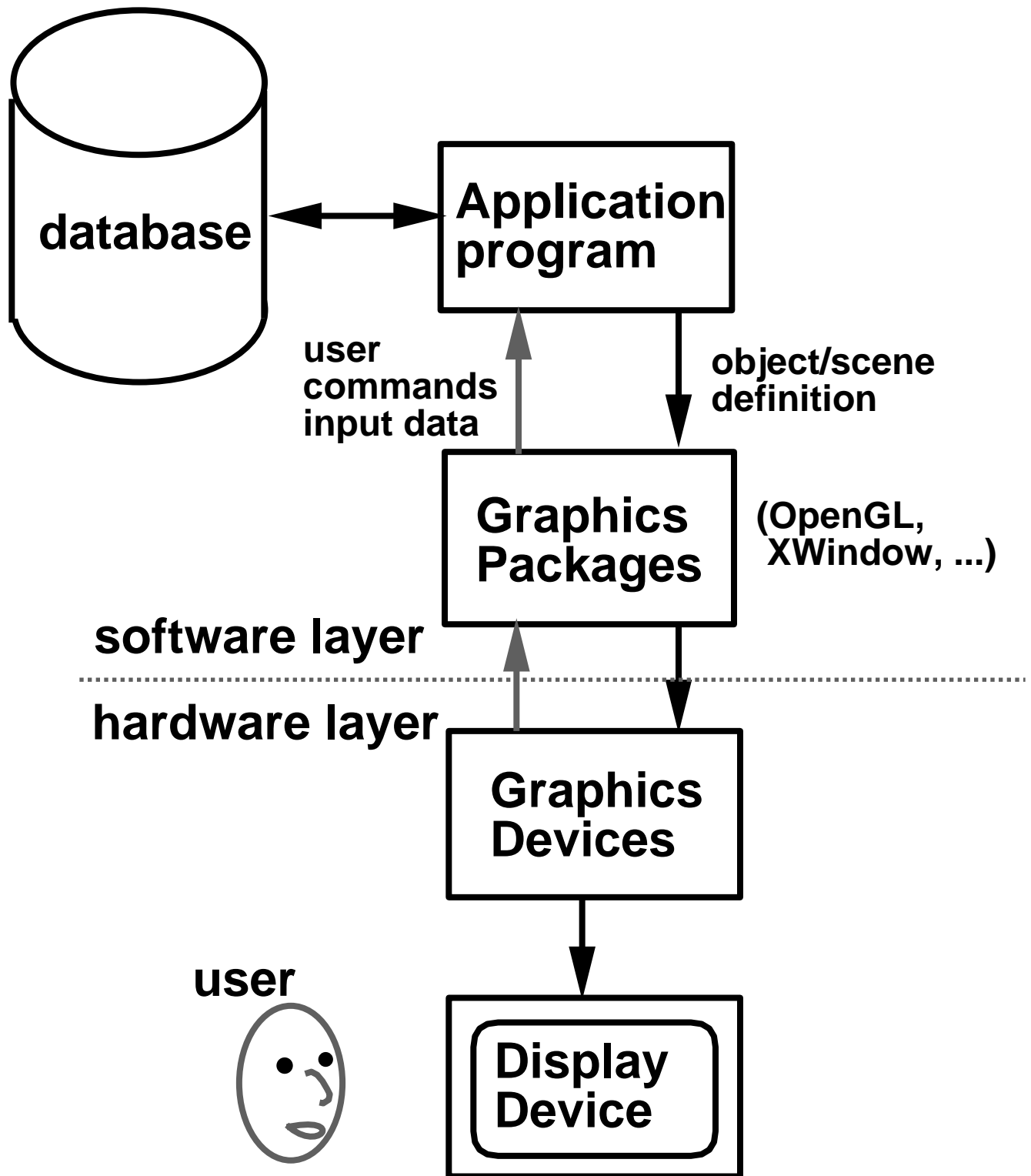


A Computer Graphics Application



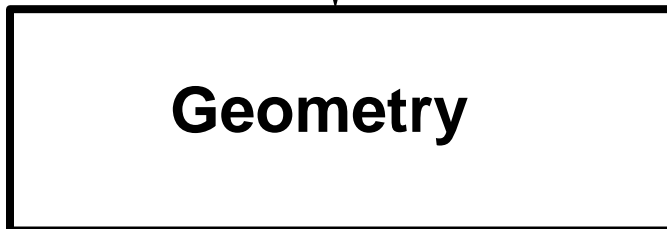
Raster Graphics Pipeline

Model



**polygonal approximation
decimation
collision detection
user interface
animation, ...**

**software
implementation**



**Per-polygon operations
(per-vertex operations)**

**hardware
implementation**



Per-pixel operations



images

Geometry Stage

graphics primitives



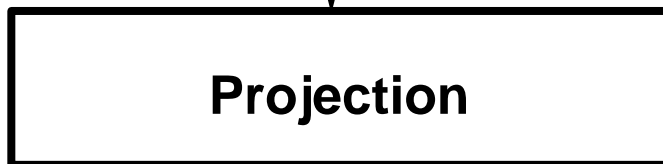
Model & View Transformation

**Object to world coordinates
World to view coordinates**



Lighting

**How light sources change
the color of each vertex**



Projection

**Map the volume of the modeled
world to a canonical volume
(e.g., a unit cube) to facilitate
subsequent calculations (map
view coordinates to normalized
3d screen/image coordinates)**



Clipping

**Avoid rasterizing parts of objects
not in the current viewport**



Hidden-surface removal

**Remove obscured parts,
drop z values, pass 2d
screen/device coordinates**



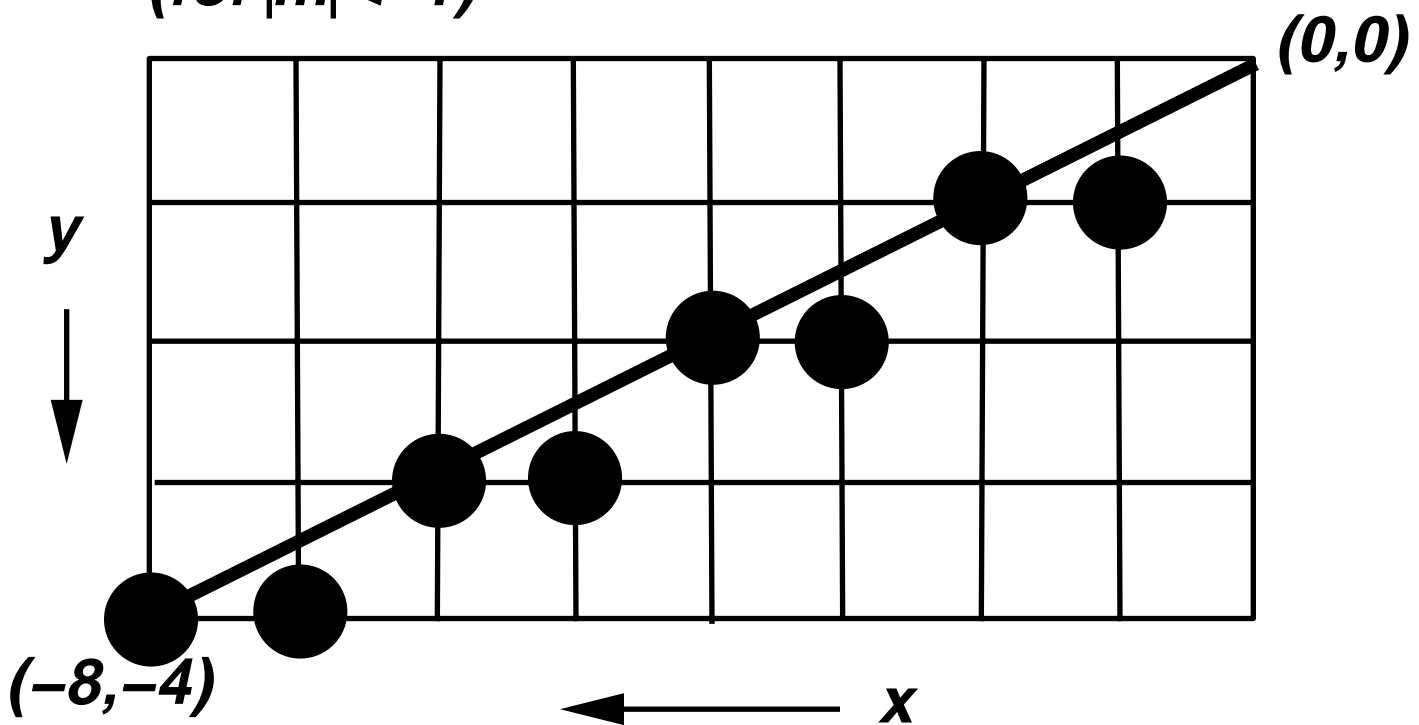
Rasterization

**Determine the color of each
pixel**

DDA – A Simple, Incremental Line Drawing Algorithm

Compute the line slope m as $\Delta y/\Delta x$

Increment x by 1 and the y increment is m
(for $|m| < 1$)



Problems:

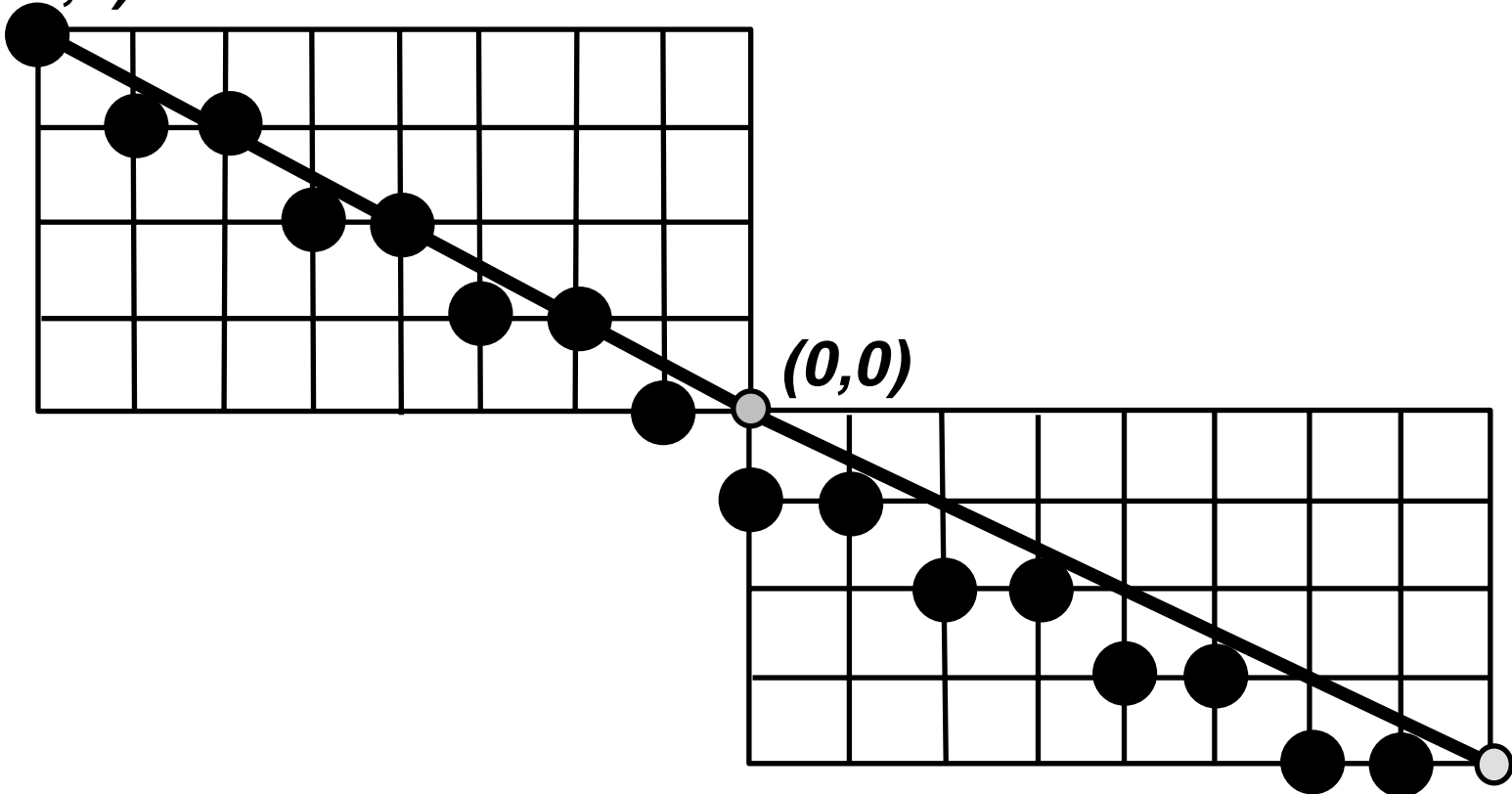
1. Floating point calculations
2. Rounding calculations
3. Rounding errors
4. Orientation dependent

Rounding Errors

```
Line (integer x0, y0, x1, y1) {  
  
    integer x;  
    float dx, dy, y, m;  
  
    dx = y1 - y0;  
    dy = x1 - x0;  
  
    m = dy/dx;  
  
    y = y0;  
  
    for x = x0 to x1 {  
  
        writePixel(x, Round(y));  
        y = y + m;  
  
    }  
}
```

Orientation Dependent Problem

$(-8,4)$

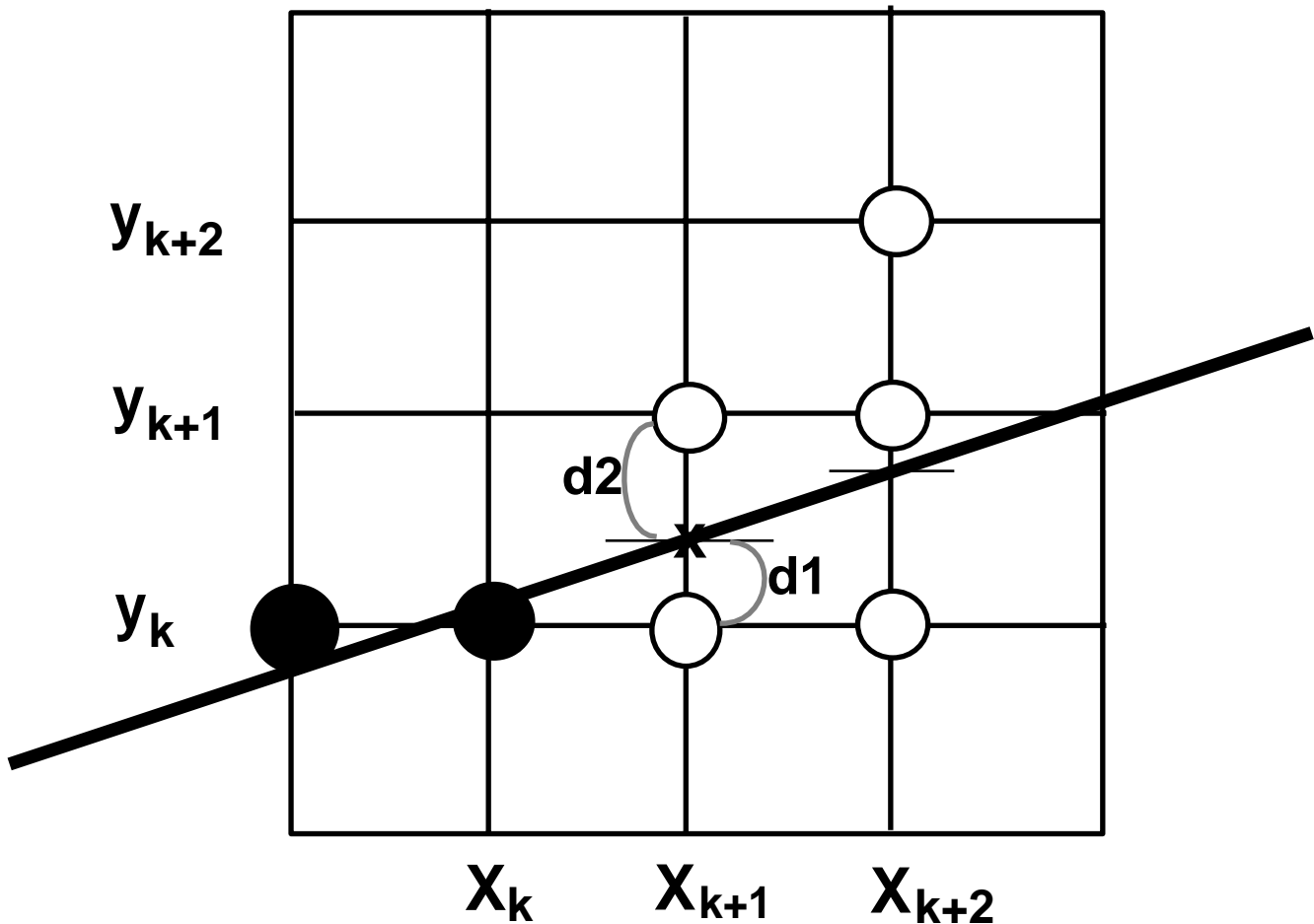


$(0,0)$

$(8,-4)$

Bresenham's Line Algorithm

$(0 < m < 1)$



$d2 > d1$ plot y_k
else Plot $y_k + 1$

Is it possible to compute and compare $d1$ and $d2$ using only integer operations?

$$y = m(x_k + 1) + b$$

$$d2 = (y_k + 1) - y = y_k + 1 - m(x_k + 1) - b$$

$$d1 = y - y_k = m(x_k + 1) + b - y_k$$

$$d1 - d2 = 2m(x_k + 1) - 2y_k + 2b - 1$$

$$d1 - d2 = 2 \frac{dy}{dx} (x_k + 1) - 2y_k + 2b - 1$$

$$dx(d1 - d2) = 2 dy (x_k + 1) - 2dx y_k + dx(2b - 1)$$

$$\text{let } 2dy + dx(2b - 1) = c$$

We have:

$$dx(d1 - d2) = 2dy \cdot x_k - 2dx y_k + c$$

since $0 < m < 1$, $dx > 0$

$dx(d1 - d2)$ has the same sign as $(d1 - d2)$, so

if $(d1 - d2) < 0$, then $y_{k+1} = y_k$

else $y_{k+1} = y_k + 1$

$$\text{Let } P_k = (d1-d2) = 2dy \cdot x_k - 2dx y_k + c$$

$$\text{where } c = 2dy + dx(2b-1)$$

$$P_{k+1} = 2dy \cdot x_{k+1} - 2dx y_{k+1} + c$$

To derive P_{k+1} from P_k :

$$\begin{aligned} P_{k+1} - P_k &= 2 dy (x_{k+1} - x_k) - 2dx (y_{k+1} - y_k) \\ &= 2 dy - 2dx (y_{k+1} - y_k) \end{aligned}$$

$$P_{k+1} = P_k + 2dy - 2 dx (y_{k+1} - y_k)$$

$$\begin{aligned} P_0 &= 2 dy \cdot x_0 - 2dx \cdot y_0 + 2dy + \\ &\quad dx (2y_0 - 2(dy/dx) x_0 - 1) \\ &= 2dy - dx \end{aligned}$$

Bresenham's Line Algorithm

Given end points (x_0, y_0) (x_1, y_1)

$dx = x_1 - x_0$, $dy = y_1 - y_0$

Starting with an end point (x_0, y_0) :

1. Compute $P_0 = 2dy - dx$

2. For each k , starting with $k=0$

if $(P_k < 0)$

the next point is (X_{k+1}, Y_k)

$P_{k+1} = P_k + 2 dy$

else

the next point is (X_{k+1}, Y_{k+1})

$P_{k+1} = P_k + 2dy - 2dx$

3. Repeat step 2 $x_1 - x_0$ times