ECE291

Lecture 16

Fun with graphics

Lecture outline

- Bresenham's Line Drawing Algorithm
- Bresenham's Circle Drawing Algorithm
- Alpha blending

Drawing Lines Basic Algorithm

- We want to draw line **Y = m * X + b**
  - m = Slope of line
  - b = Displacement
  - X = independent coordinate
  - Choose X so as to plot large number of points
  - Y = Dependent coordinate
  - Plot to nearest Y when fractions occur
- Draw Line from (X1,Y1) to (X2,Y2)
  - m = dy/dx; dy = (Y2-Y1); dx = (X2-X1)
  - b = Y1 – m*X1

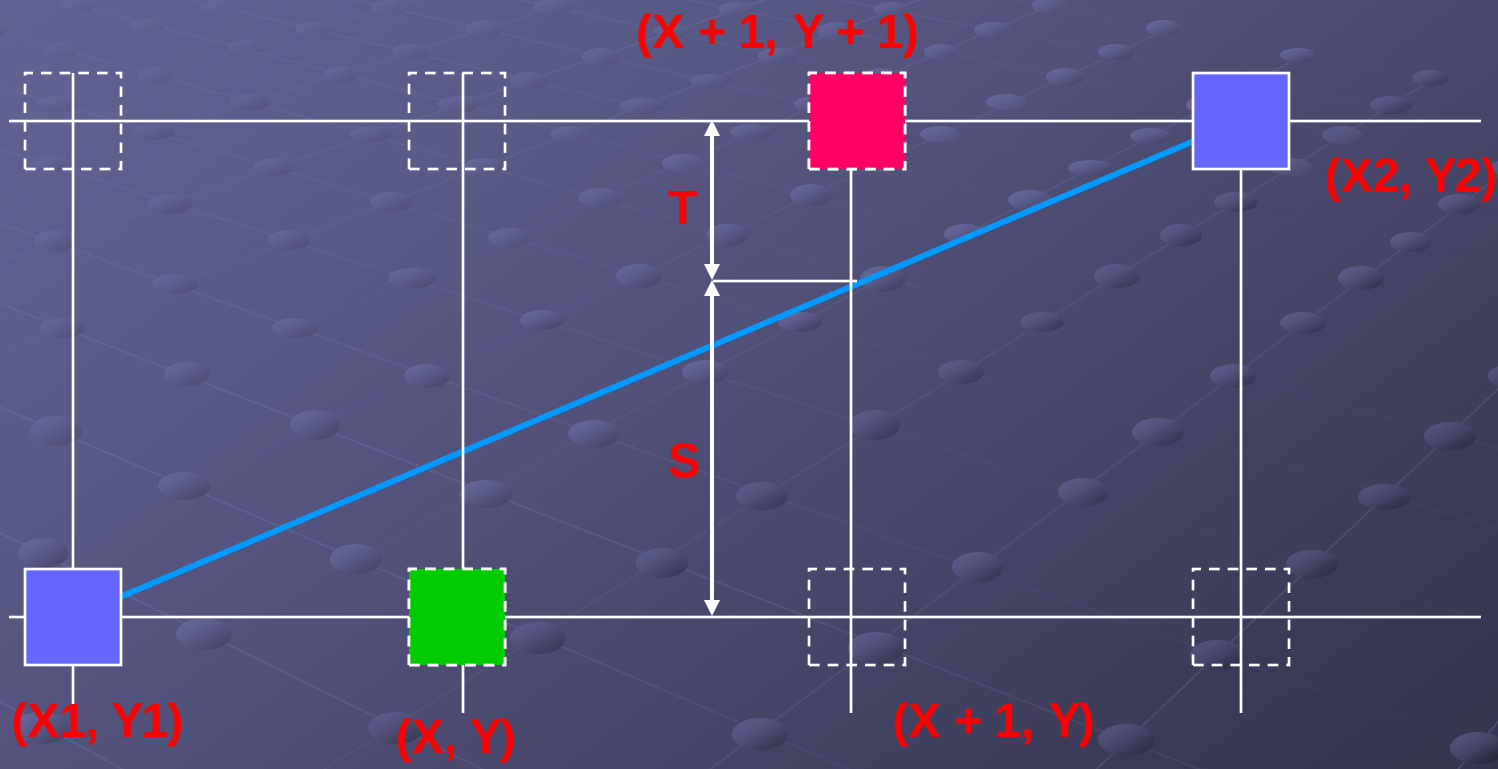
  Plot(X, m*X+b) for X=X1... X2
- This algorithm requires slow floating point math

Derivation of Bresenham's Line Algorithm

- Eliminates floating point calculations
- Given two line endpoints, we are trying to find the "in-between" points on a pixel grid
- The Bresenham's line algorithm determines subsequent points from the *start point* by making a *decision* between the two next available points and selecting one that is closer to the *ideal point*

- Start point *(X1, Y1)*
- End point *(X2, Y2)*
- Slope *m = dy/dx*
- Current Point *(X, Y)*

Derivation of Bresenham's Line Algorithm



Derivation of Bresenham's Line Algorithm

- If we restrict the slope for now to (0 <= slope <= 1) and assume (X1 < X2)
  - step through x one pixel at a time to the right and determine what y value to choose next
  - given current point (X, Y), next ideal point would be between (X + 1, Y) and (X + 1, Y + 1)
  - must choose between (X + 1, Y) or (X + 1, Y + 1)
- How do we decide between these points?
  - must choose the closet point to the ideal

Derivation of Bresenham's Line Algorithm

- Determine S and T

  S = (X + 1) * m - Y

  T = 1 - S = 1 - (X + 1) * m + Y

- Then **S - T = 2 * (X + 1) * m - 2 * Y - 1**
- Select the next point:

  if S < T go horizontal - **next point (X + 1, Y)**

  if S > T go diagonal - **next point (X + 1, Y + 1)**

Derivation of Bresenham's Line Algorithm

- We want to develop a **FAST** algorithm; we create a decision variable (error) that can be used to quickly determine which point to use

  E = (S -T)*dx

  = 2*(X+1)*dy - 2*Y*dx - 1*dx

  = 2*X*dy - 2*Y*dx + 2*dy - dx

NOTE, that m=dy/dx ; the quantity dx is non-zero positive

Derivation of Bresenham's Line Algorithm

Let E be a function of X and Y -- E(X,Y) and assume our line passes through the origin (0,0)

E(0,0) = **2*dy - dx**

E(X+1, Y+1) - E(X,Y) = Additive error for a diagonal move

Diagonal Move: **E(X+1,Y+1) - E(X,Y)** = **2*dy - 2*dx**

E(X+1, Y) - E(X,Y) = Additive error for a horizontal

Horizontal Move: **E(X+1,Y) - E(X,Y)** = **2*dy**

Algorithm for Drawing Lines in the First Octant

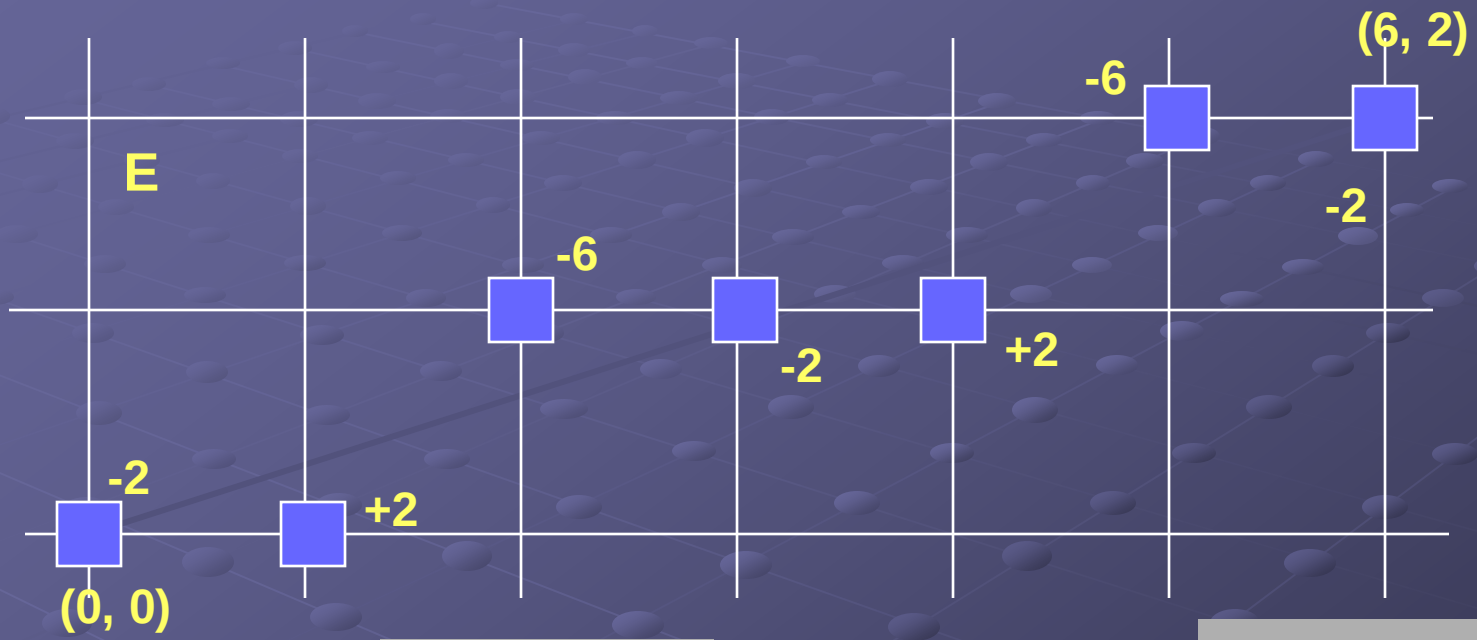
```
Draw_Line (X1, Y1, X2, Y2)

dx = X2 - X1
dy = Y2 - Y1
E = 2dy - dx
HoriMove = 2dy
DiagMove = 2dy - 2dx
Y = Y1

for X from X1 to X2
    set_pixel (X, Y)
    if   E < 0   then    (S < T)
        E = E + HoriMove
    else
        E = E + DiagMove
        Y = Y + 1
    end if
end for

end Draw_Line
```

Algorithm for Drawing Lines in the First Octant

- The code should not use
  - floating point numbers
  - multiplication
  - division
  - comparison to any number other than zero (they are slower

- It should be sufficient to use
  - additions
  - bit-shifts
  - comparison to zero

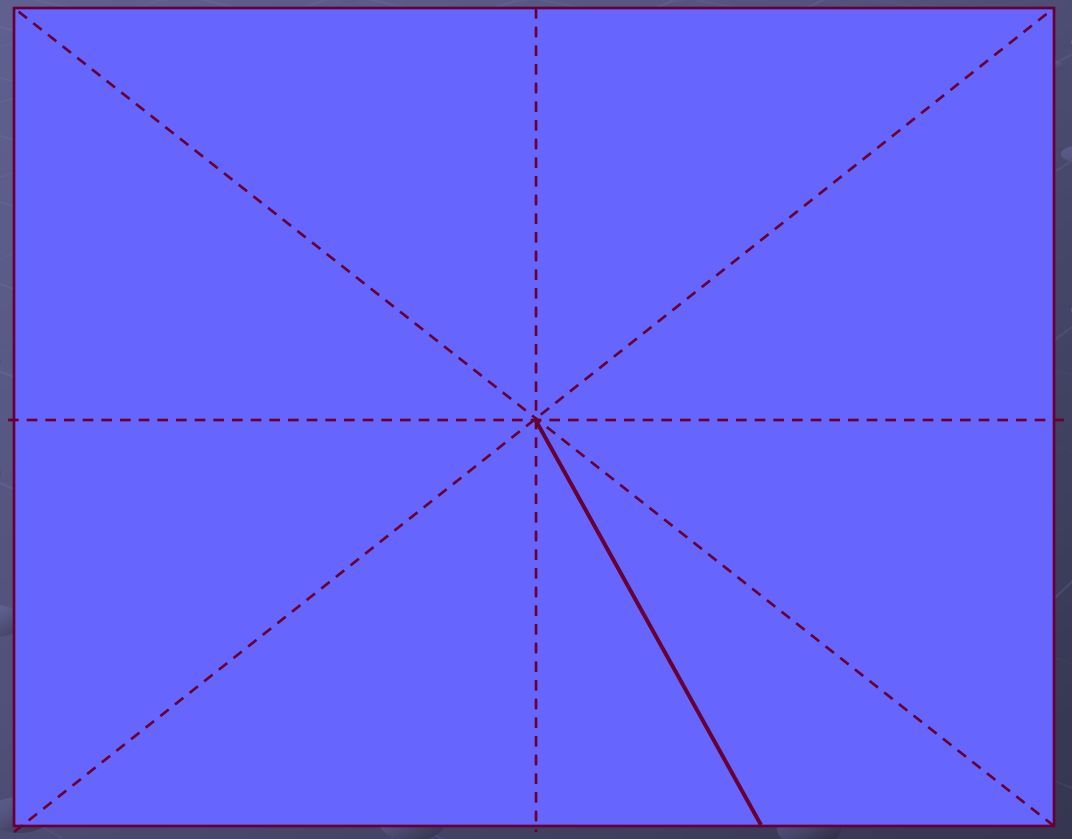
Example (first octant)



(X1,Y1) = (0,0)
(X2,Y2) = (6,2)
dx=6
dy=2
E = 4-6 = -2
HoriMove = 4
DiagMove = -8

X	Y	E
0	0	4-6 = -2
1	0	-2+4 = +2
2	1	+2-8 = -6
3	1	-6+4 = -2
4	1	-2+4 = +2
5	2	+2-8 = -6
6	2	-6+4 = -2

Example (Seventh Octant)



Example (Seventh Octant)

Swap X1, Y1
and X2, Y2

(Y1,X1) = (0,0)
(Y2,X2) = (-6,2)
dx=6
dy=2
HoriMove = 4
DiagMove = -8

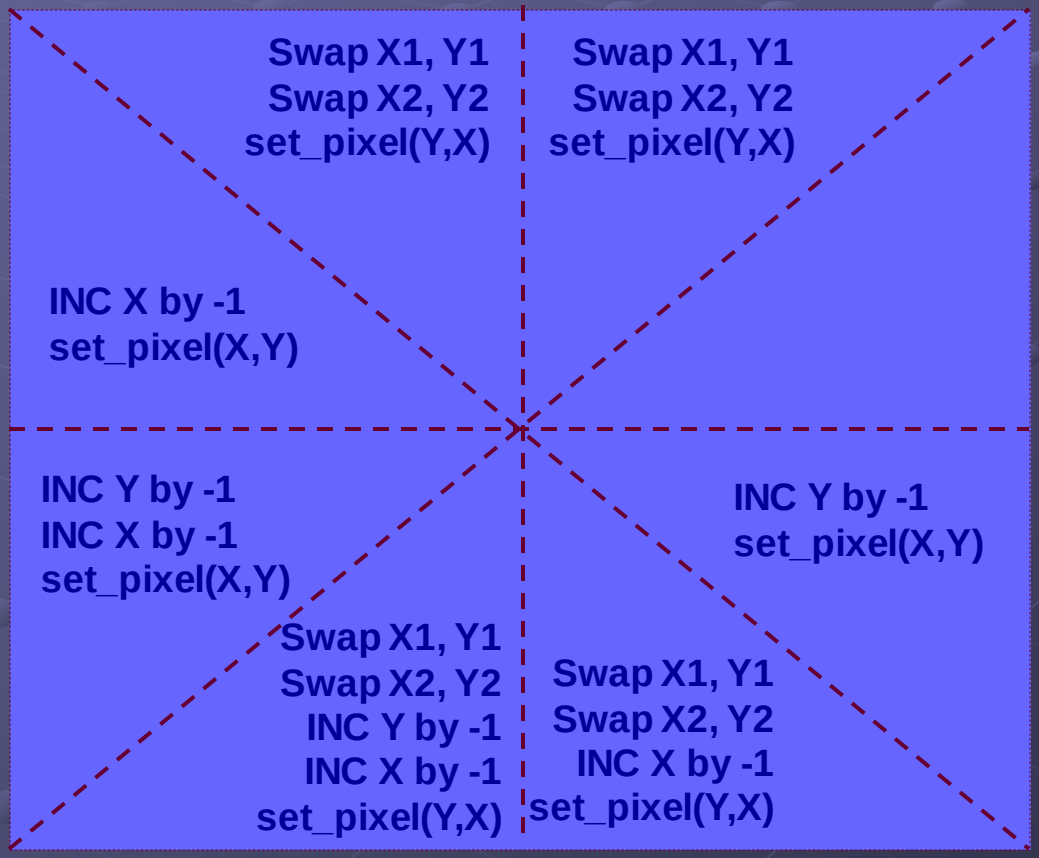
INC X by -1
(X will change
from 0 to -6)

X	Y	E
0	0	4 - 6 = -2
-1	0	-2+4 = +2
-2	1	+2- 8 = -6
-3	1	-6+4 = -2
-4	1	-2+4 = +2
-5	2	+2- 8 = -6
-6	2	-6+4 = -2

Set pixel
(Y, X)

Drawing Lines in All Eight Octants

- There are eight regions (octants) in which the algorithm should work

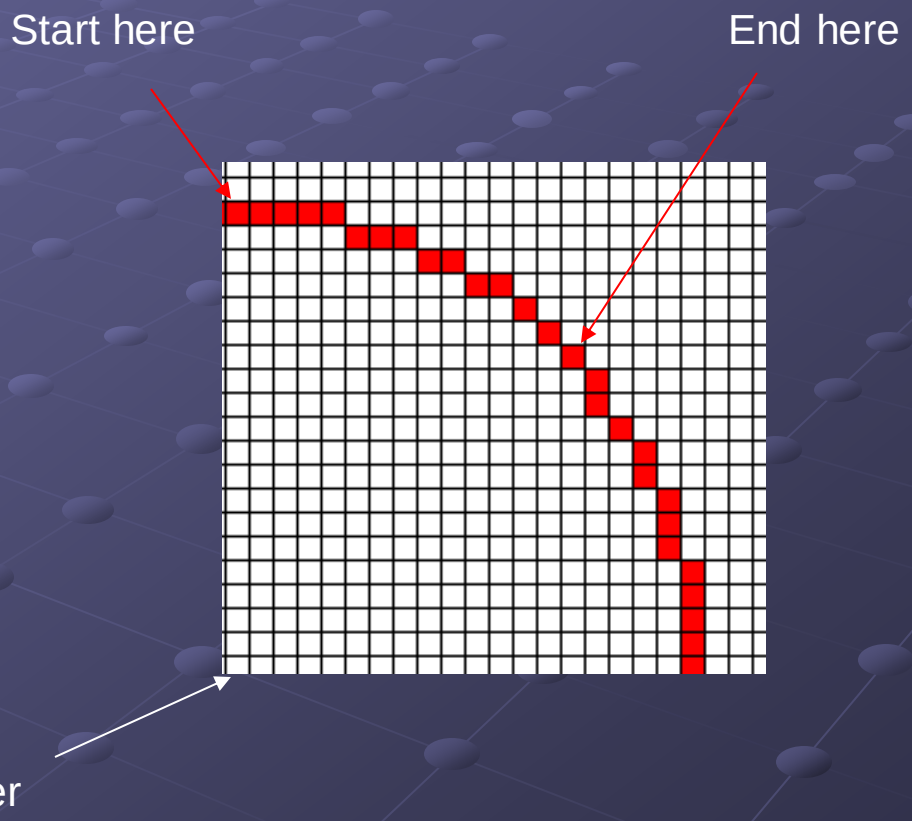


Drawing circles

- Circles are symmetric about the x and y axes, as well as their 45^{o} lines
- We only have to figure out the pixels for one octant of the circle!
- Bresenham has an algorithm for doing this

Drawing circles

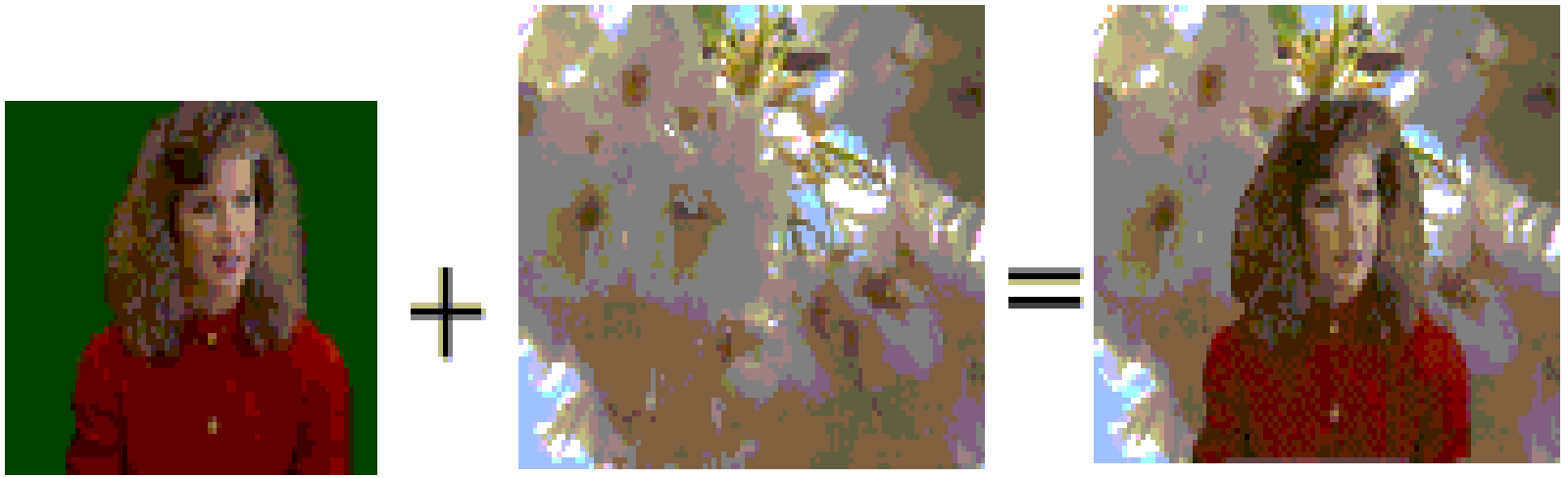
```
e = 3 - (2 * r)
x = 0
y = RADIUS

Do Until x = y
    PutPixel(CenterX + X, Center Y + Y)
    PutPixel(CenterX + X, Center Y - Y)
    PutPixel(CenterX - X, Center Y + Y)
    PutPixel(CenterX - X, Center Y - Y)
    PutPixel(CenterX + Y, Center Y + X)
    PutPixel(CenterX + Y, Center Y - X)
    PutPixel(CenterX - Y, Center Y + X)
    PutPixel(CenterX - Y, Center Y - X)

    if e < 0 then
          e = e + (4 * x) + 6
    else
          e = e + 4 * (x - y) + 10
          y = y - 1
    end
    x = x + 1
End do
```



Drawing circles

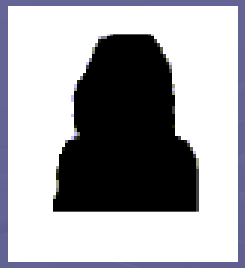
- http://www.gamedev.net/reference/articles/article767.asp
- http://www.rpi.edu/dept/ecse/graphics-f99/Classes/24/

Overlaying images



Overlaying images

Overlay of the Image

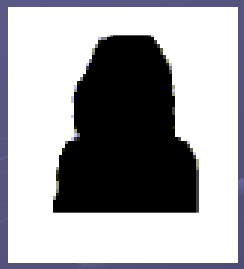


pandn

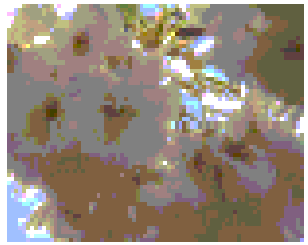
FFFF	0000	FFFF	0000

X1	X2	X3	X4

0000	X2	0000	X4



pand



FFFF	0000	FFFF	0000

Y1	Y2	Y3	Y4

Y1	0000	Y3	0000

por

Y1	X2	Y3	X4

Image Dissolve Using Alpha Blending

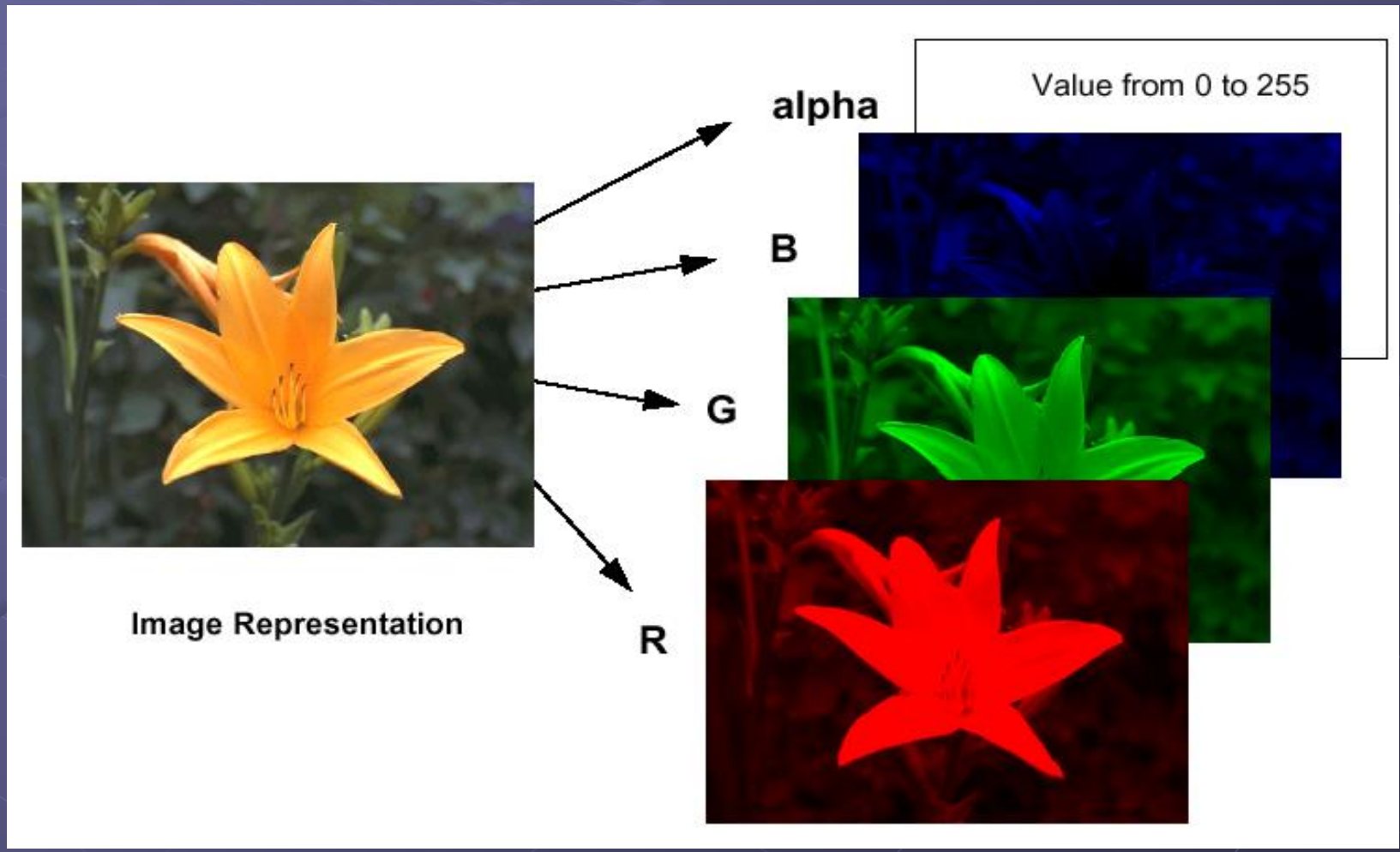


Image Dissolve Using Alpha Blending

- MMX instructions speed up image composition
- A flower will dissolve a swan
- Alpha determines the intensity of the flower
- The full intensity, the flower's 8-bit alpha value is FFh, or 255
- The equation below calculates each pixel:

Result_pixel =Flower_pixel *(alpha/255) + Swan_pixel * [1-(alpha/255)]

For alpha 230, the resulting pixel is 90% flower and 10% swan

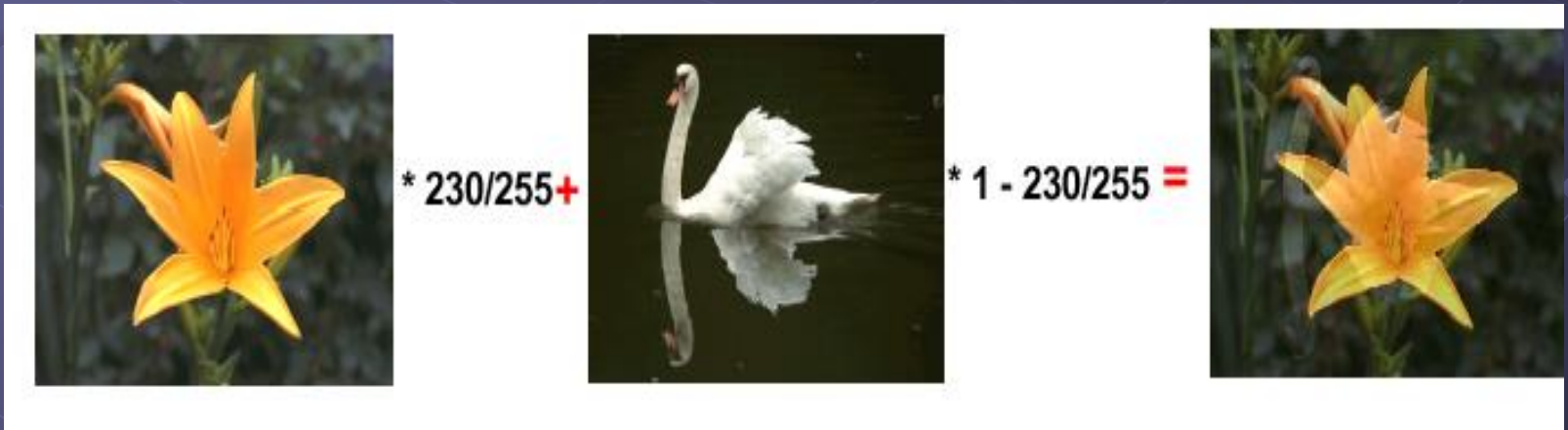
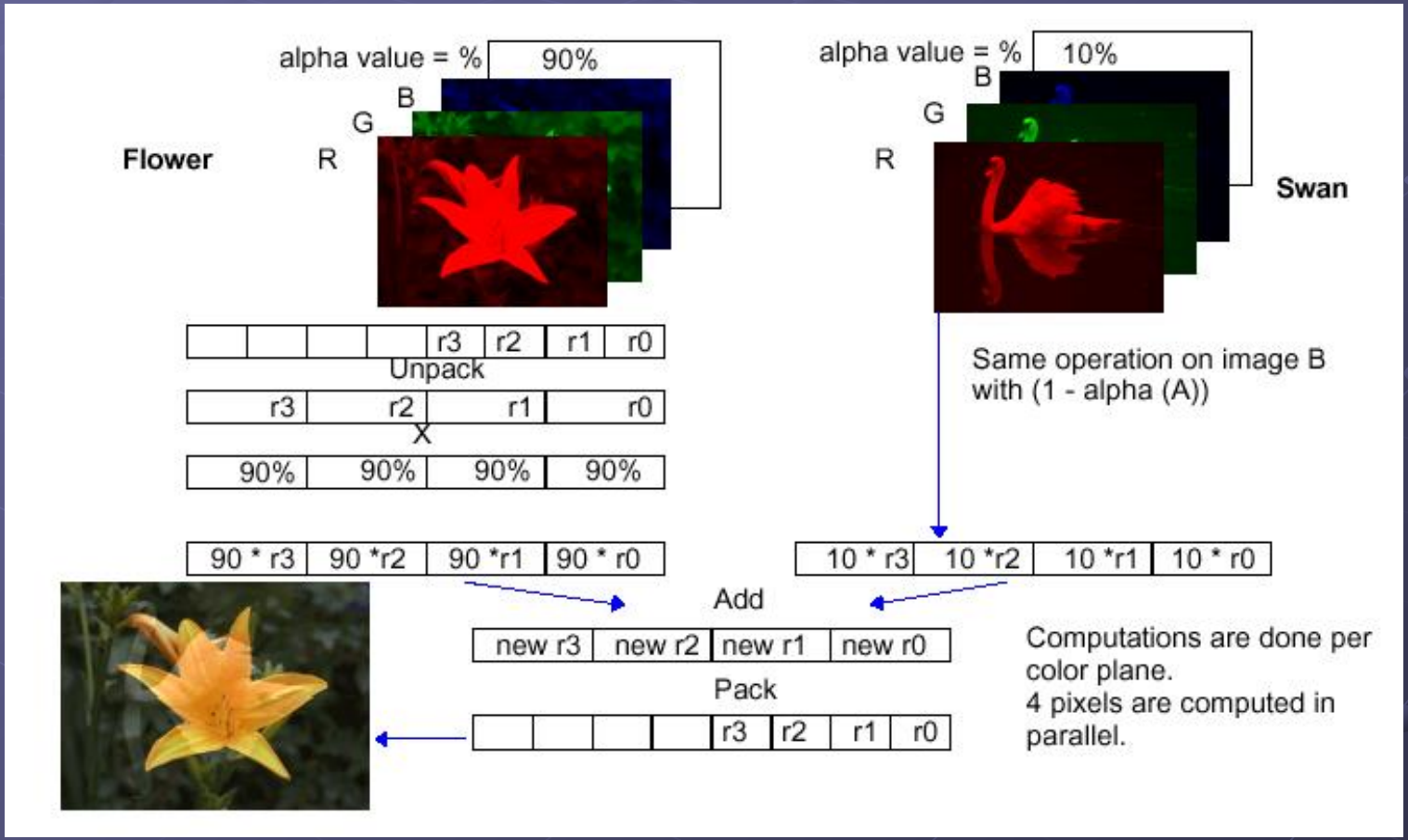


Image Dissolve Using Alpha Blending



Instruction Count with/without MMX Technology

Operation cost for fading from flower to swan by single-stepping the alpha value from 1 to 255

Operation	Calculation without MMX™ Technology	Number of Instructions without MMX Technology	Number of MMX Instructions
Load	(640*480)*255*3*2	470 million	117 million
Unpack	-	-	117 million
Multiply	(640*480)*255*3*2	470 million	117 million
Add	(640*480)*255*3	235 million	58 million
Pack	-	-	58 million
Store	(640*480)*255*3	235 million	58 million
Total		1.4 billion	525 million