

# *ECE390*

## *Computer Engineering II*

### *Lecture 11*



**Dr. Zbigniew Kalbarczyk**  
**University of Illinois at Urbana- Champaign**

### *Lecture outline*



- Color theory
- Mode 13h Graphics
- Final projects

## *Color theory*

---

- TV's, Computer Monitors, Projectors...  
It's all **Red**, **Green**, **Blue**
- Why RGB? These are the primary colors for visible light. You can get all the colors by mixing them.
- **Red** + **Green** = **Yellow**
- **Green** + **Blue** = **Cyan**
- **Blue** + **Red** = **Purple**

## *Color Theory - Biology*

---

- How can mixing a **Low Frequency** with High Frequency make **Really High Frequency**?
- Our eyes have red, green, and blue sensors. These are the colors we can see best (i.e., the frequencies for which we have color receptors)
- The people who invented color photography and color TV used this fact.
- If three-eyed aliens with different color receptors visited us, they would “see” the same three frequencies coming from the monitor, but they wouldn't be able to see any pictures.

## *Full Color Images*

---

- Each pixel has a Red, Green, and Blue value.
- 24-bit color means **1 red byte**, **1 green byte**, 1 blue byte. Each determines the color intensity (0 = none, 255 = full) of their respective color components.
- 16-bit color means **5 bits red**, **5 bits green**, 5 bits blue. The extra bit is sometimes a green bit depending on the system.
- Full color is when we use a given number of bits to specify the intensity of each color component.

## *24-bit Full Color Images*

---

- 24-bit color allows 256 different intensities for each color component. This allows for  $2^{24}$  or 16,777,216 different colors.
- That's way more than our eyes can distinguish.
- This 24-bit color is sometimes called **Full** or **True** color.
- True color requires 3 bytes for every pixel.

## *16-bit Full Color Images*

---

- 16-bit color allows 32 different intensities for each color component. This allows for  $2^{15}$  or 32,768 different colors (assuming 5 bits green)
- Not quite as good as True color, 16-bit color is also referred to as High color.
- High color requires 2 bytes for every pixel.

## *Full Color Images*

---

- Positives of high and true color images
  - More colors means higher quality pictures
  - Easier to do image manipulation (math)
    - Make everything a little redder
    - Blur an image by averaging RGB values of adjacent pixels
- Negatives of high and true color images
  - Take up a lot more memory

Ex: A 320x200 pixel image with 24-bit color needs  
 $320 \times 200 \times 3 = 192,000$  bytes of memory

## *Color Palettes*



- Want to do everything in full color, in real mode we need to reduce the amount of memory our images use.
- The **color palette** contains the 3-byte RGB values of 256 different colors.
- The image data is then just a bunch of indexes into this palette. It is a look-up table!

## *Color Palettes*



- Positives of palettes
  - Less memory and bandwidth needed to store and move pictures
  - You can do some cool stuff with palettes
    - Invisible pixels – assign one palette entry to be “clear” and when you’re copying one image over another, don’t copy the “clear” pixels.
    - You can fade out a screen just by subtracting from each entry in the palette table until they’re all zero

## *Color Palettes*

---

- Negatives of palettes
  - Fewer colors (at most 256) mean lower quality pictures
  - Harder to do manipulation
    - Average blurring is harder because new color isn't in palette
  - All “real” images come in full color and have to be converted
    - This is hard because you have to choose a good palette
    - For each “full color” pixel, find closest palette entry
  - When you put images together, they share same palette

## *Mode 13h*

---

- **Mode 13h** is a 320 x 200 x 256-color graphics display mode that was very popular for DOS programming due to its relatively high number of colors (256 vs 16 for other VGA modes) and simple addressing.
- 320 x 200 = 64000 pixels. With 1 byte per pixel this easily fits in a single real-mode segment.
- It's linear just like text mode. The pixels increase in memory as you go from left to right, while each row immediately follows the previous one.

$$\text{Offset} = \text{row} * 320 + \text{col}$$

## *Mode 13h - How to get there*

- The following instructions get you into Mode 13h video

```
mov ah, 0           ; int 10h's set mode subfunction
mov al, 13h        ; The aptly named Mode 13h
int 10h            ; Call the Video BIOS interrupt
```

## *Mode 13h How to draw a pixel*

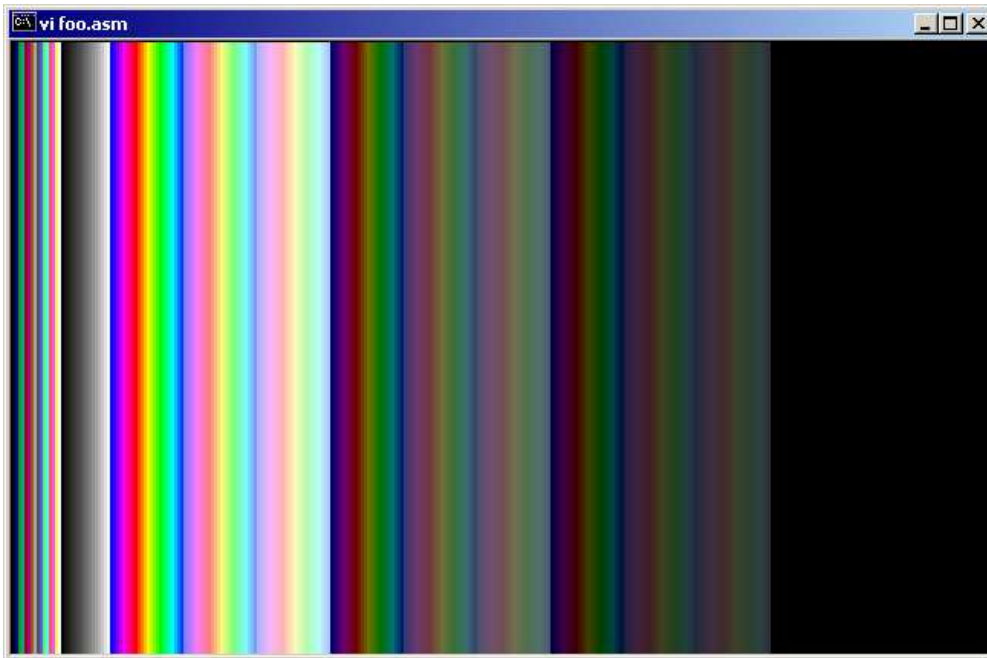
- The following instructions draw a pixel on the screen

```
VidGrSeg equ      0A000h
Location dw       (200/2)*320+(320/2) ; Center
Color db         98 ; Some random palette index
```

```
mov ax, VidGrSeg
mov es, ax ; Use extra segment to address VRAM
mov bx, [Location] ; Offset to center of screen
mov al, [Color] ; Palette Entry Number
mov [es:bx], al ; Write it to the screen
```

## *Mode 13h*

### *The Default Palette*



Z. Kalbarczyk

ECE390

## *Mode 13h*

### *How to change the palette*

- Change just one palette entry using VBIOS call
  - BX = Palette Number (0 – 255)
  - CH = Green (0 – 63)
  - CL = Blue (0 – 63)
  - DH = Red (0 – 63)
  - AX = 1010h ; Set VGA Palette Registers command
  - INT 10h ; vBIOS call

Z. Kalbarczyk

ECE390

## Mode 13h How to change the palette

; Change a palette entry to light purple

```
Red    db    39
Green  db    0
Blue   db    63

mov    bx, 0    ; changing palette entry 0
mov    ch, [Green]
mov    cl, [Blue]
mov    dh, [Red]
mov    ax, 1010h
int    10h
```

## Mode 13h Code Example

```
BITS    16
VidGrSEG equ    0A000h

EXTERN  kbdt, dspout, dspmsg, dosxit

SEGMENT stkseg STACK                ; *** STACK SEGMENT ***
resb    64*8
stacktop:
resb    0                            ; work around NASM bug

;Define code segment =====
SEGMENT code                        ; *** CODE SEGMENT ***

; Declare variables for main procedure =====
Location dw    (200/2)*320+(320/2)
Color    db    98
Red      db    39
Green    db    0
Blue     db    63
```

# Mode 13h Code Example

```
; Program initialization =====
..start:
    mov     ax, cs                ; Initialize Default Segment register
    mov     ds, ax
    mov     ax, stkseg           ; Initialize Stack Segment register
    mov     ss, ax
    mov     sp, stacktop         ; Initialize Stack Pointer register
Main procedure =====

main:
    ; Go into Mode 13h
    mov ah, 0
    mov al, 13h
    int 10h

    ; Draw a Pixel
    mov ax, VidGrSEG
    mov es, ax
    mov bx, [Location]
    mov al, [Color]
    mov [es:bx], al
    call  kbdtone    ; Pause
```

Z. Kalbarczyk

ECE390

# Mode 13h Code Example

```
; Draw a bunch of pixels
    mov cx, 320*30

.DrawLoop:
    add bx, 1
    add al, 1
    mov [es:bx], al
    loop .DrawLoop
    call  kbdtone    ; Pause

    ; Change the Palette
    mov bh, 0
    mov bl, [Color]
    mov ch, [Green]
    mov cl, [Blue]
    mov dh, [Red]
    mov ax, 1010h
    int 10h
    call  kbdtone    ; Pause

; Change a bunch of Palette entries
    mov di, 256

.PaletteLoop:
    add bl, 1
    add ch, 1
    add cl, 2
    mov dh, 3
    mov ax, 1010h
    int 10h
    dec di
    call  kbdtone    ; Pause
    jnz .PaletteLoop

    ; Go back into text mode (Mode 3h)
    mov ah, 0
    mov al, 3h
    int 10h
    call  dosxit     ; Exit to DOS
```

Z. Kalbarczyk

ECE390

## *Graphics File Formats*

---

- Raster Formats
  - Arrays of pixels (what we've been talking about)
  - Raw pixels (.bmp)
  - Run-Length encoding (.pcx)
  - LZW encoding (.gif)
  - Transform encoding (.jpg)
  - Deflate (zip) compression (.png)

## *Graphics File Formats*

---

- Included in image files...
  - Header: Most file formats start out with some kind of info about the image. For graphics files this info could be:
    - Size (horizontal and vertical pixels)
    - Color depth (bits per pixel)
    - Version (makes life difficult...GIF89A, GIF97A, ...)
    - Compression Technique
  - Palette: If you're not using true color, you must include a palette for the image in the image file
  - Data: Raw or compressed, raster or vector information.

## *Milestones and deadlines in selecting final projects*

Actions	Students	Instructor & TAs
Posting preliminary proposals for final projects on the ECE390 web page  <i>final projects à final project options à create a final project</i>	Whoever has an idea can provide a short description of a project  <u>Due:</u> <b>Tuesday, March 7, 5pm</b>	
Posting (on ECE390 web page) a list of potential projects		<u>Due:</u> <b>Thursday, March 9</b>
Selecting projects	Fill in forms to indicate preferred project(s) (three choices)  <u>Due</u> (for submitting forms): <b>Tuesday, March 14, 10:30 am in class</b>	
Posting team and project assignments		<u>Due:</u> <b>Thursday, March 16</b>