University of Khartoum

Faculty of Engineering - Department of Electrical and Electronic Engineering.

Microprocessor and Assembly Language

Computer Generated Graphics and free Mouse drawing

Prepared by:

Ahmad Mansour	134009
Amr Muhammad Alameen	135058
Omar AbdAlshakour	134064
Sayda Osama Sobahi	134050

Objective:

Switching from the text more to the pixels' mode (VGA mode), and generate simple shapes.

What we intended to do (our objective):

We intended to do is to work with as much hardware as we could and to understand how they can be coordinated to produce a useful output. In order to achieve that we have decided to give the user the ability to switch back and forth between the modes using the keyboard, to choose what shape to view and to use the mouse to draw free shapes.

Software description:

The program is a bootable bare machine program.

The first thing the program does is loading the hard disk sectors into the main memory.

After it is loaded and began to execute, the program asks you to enter an input to choose to view shapes, or to draw with the mouse.

The VGA mode used was the (320*200 pixel, 256 colors) VGA mode.

The shapes in the demo were:

A Circle.

Triangles.

A diagonal line.

Rectangles.

Generating the Graphics

Generating a computer graphics is achieved by setting the color in the appropriate pixel. Setting a pixel is done by writing the pixel's color in the pixel's corresponding address in the video memory.

Video memory resides in **0x000A0000 - 0x000BFFFF**

- Brokenthorn project , graphics tutorial.

We used the Video Mode 0x13 which is a standard IBM VGA BIOS mode number for a 256 color 320x200 resolution. It uses a 256-color **palette** allows access to the **Video Memory** as a **Packed-Pixel Framebuffer**.

Before drawing the shapes the mode of the screen must be switched form the default text mode to the VGA mode.

This was achieved by the software interrupt (0x10)

```
mov ah , 0
mov ax,13h      ; mode = 13h
int 0x10        ; call bios service
```

- We can color a pixel by sending the color to its address(linearly) begging at 0x000A000. However, we used an easier implementation to achieve that, by calling interrupt 0x10 and putting the appropriate data in the interrupt vector. We used the procedure myShapePixel to do that.

```
myShapePixel:
        mov ah , 0Ch
        mov al , byte [Color]
        mov cx ,  [x]
        mov dx ,  [y]
        int 10h
        ret
```

Drawing vertical and horizontal lines is straight forward.

- Regarding drawing lines with slope less than or equal to 1 the famous **Bresenham's** line algorithm was used.

Pseudo code:

```
plotLine(x0,y0, x1,y1)
  dx = x1 - x0
  dy = y1 - y0
  D = 2*dy - dx
  y = y0

  for x from x0 to x1
    plot(x,y)
    if D > 0
       y = y + 1
       D = D - 2*dx
    end if
    D = D + 2*dy
```

Then lines were used to draw the Triangle, the diagonal lines, we didn't use it however to draw the rectangles (no need for it because it adds unnecessary computations in this case).

- Regarding drawing the circle the famous Midpoint algorithm was used.

Simple C code demonstrates the midpoint algorithm:

```
void drawcircle(int x0, int y0, int radius)
{
    int x = radius;
    int y = 0;
    int err = 0;

    while (x >= y)
    {
        putpixel(x0 + x, y0 + y);
        putpixel(x0 + y, y0 + x);
        putpixel(x0 - y, y0 + x);
        putpixel(x0 - x, y0 + y);
        putpixel(x0 - x, y0 - y);
        putpixel(x0 - y, y0 - x);
        putpixel(x0 + y, y0 - x);
        putpixel(x0 + x, y0 - y);

        y += 1;
        if (err <= 0)
        {
            err += 2*y + 1;
        }
        if (err > 0)
        {
            x -= 1;
            err -= 2*x + 1;
        }
    }
}
```

- Bresenham's algorithm and midpoint algorithm were used because they work with integer values for x and y, therefore they are more computationally efficient (no rounding and truncation needed) and much faster than floating point algorithms.
- Also, because they are easier to be implemented in X86 assembly

Mouse Driver Algorithm steps:

1. Turn the screen into video mode.
2. Initialization:
   - Enable PS/2 controller (chip 8042) second connector.
   - Restore default settings.
   - Enable mouse.
   - For this part of the code I had guidance from http://forum.osdev.org/viewtopic.php?t=24277
3. Begin of loop.
4. Check bit 5 in port 64 (status byte) to insure it's from mouse (inner Loop).
5. Depending on the state of the left button clear the screen.
6. Get bit 1 from first byte and save it in a variable called (Left Button).
7. Get second byte (delta x) and added it to a variable called (x _Coordinate).
8. Get third byte (delta y) and subtract it to a variable called (y_Coordinate).
9. Print a pixel (represent the mouse) on the screen in the position defined by the 2 variables above with the color defined in the register "al" using interrupt 10h.
10. Return into begin of loop.

- Everything was working as expected.
- **Features extension we intend to do if we had more time and how to implement them:**
    1. Our program can be improved to become a sophisticated drawing tool, more shapes can be introduced like ellipses.
    2. The program can be more interactive, we can ask the user to enter points as they are the heads for a triangle, center for a circle then we can draw them.
    3. We can enter points with the mouse and use the program to linearly interpolate between them.
    4. The program can be developed more to be used as a functions plotter.
    5. We can make the mouse erase what was drawn by setting the pixel's color black at that point when right click happens.
    6. We could improve the program to curve fit between the points (linear regression).

- **Problems we have faced**
    1. The scope of the project wasn't crystal clear. We intended to implement a lot of stuff and make them work together to produce a nice graphics.
    2. VirtualBox wasn't working on half of the teams' computers, so they couldn't debug what they've written.
    3. Working with on a bare machine is little difficult, in our case debugging was very tedious and time consuming, knowing a register content or a memory variable value at a certain place needed a lot of work.
    4. At the beginning the code was too much to fit in one sector, until the problem was solved by loading the other sectors, we were commenting other parts of the code in order to execute the program.

- **Conclusion**:

We have learned the basics for generating computer graphics and how to uses these basics to generate more complex graphics.

System programming is very interesting, as the programmer knows the underling details of the computer hardware which are hidden normally in application programming. We found it very interesting, enjoyable and tempting to do powerful stuff.

We have chosen this project specifically to have deeper understanding of how the computer deals with I/O.

what we gained is a good understanding of device drivers and how they can be developed, we got a better understanding of the idea of virtual machines, how to deal with secondary storages and how do bootloader work.