Technical Design Document

Florian-Claudiu Gheorghica 100368163

1. Introduction

This document is to outline the features of my solution for the 2nd Systems Programming Assignment, as well as describe the architecture of the implemented code.

2. Major Technical Features of the Solution

The major technical features of the solution are as follows:

- Code provides support for Ring 3 calls.
- Lines in all 8 octants drawn using Bresenham's algorithm, which may also be dotted or dashed.
- The ability to draw any type of polygon. All convex shapes may be filled.
- The use of Flood Fill to fill large shapes, made possible by recursively triangulating the shapes and filling up each triangle individually. The area of each shape is calculated to determine whether triangulation is needed.
- The ability to set custom colours via the SetColour function that modifies the colour palette.

3. Addition Files and Modifications

The additional files created are as follows: *drawing.h*, *drawing.c*, *vgacalls.h* and *vgacalls.c*.

Several provided files were modified for the project: *vgamodes.h*, *vgamodes.c*, *user.h*, *user.c*, *kernel_main.c* and the makefile to include the newly created files.

4. Implemented Methods and Architecture

4.1 *drawing.h* and *drawing.c*

The c file contains the code for all the ring 0 functions that modify video memory, as well as some utility methods. The functions are as follows:

- void ClearScreen():
  - Clears the screen to black.

- void SetPixel(uint16_t **x**, uint16_t **y**, uint8_t **c**):
  - Sets the pixel at coordinates (**x**, **y**) to colour **c**. Makes sure it is within screen bounds.

- void DrawLine(uint16_t * **vertices**, uint8_t **c**):
  - Draws a line of colour **c** in all 8 octants using Bresenham's algorithm. ***vertices** should point to 4 int values containing the x and y values of two points (in the order x_0, y_0, x_1, y_1). Implementation adapted from the code present here: http://tech-algorithm.com/articles/drawing-line-using-bresenham-algorithm/.

- void DrawRectangle(uint16_t * **params**, uint8_t **c**):
  - Draws an axis-aligned rectangle of colour **c**. ***params** should point to the x and y values of the top left point of the rectangle, followed by its width and height.

- void DrawFilledRectangle(uint16_t * **params**, uint8_t **c**):
  - Same as DrawRectangle, but it is filled.

- void DrawCircle(uint16_t * **params**, uint8_t **c**):
  - Draws a circle of colour **c**. ***params** should point to the x and y coordinates of the center followed by the radius size. The implementation uses the Midpoint Circle Algorithm found here: https://en.wikipedia.org/wiki/Midpoint_circle_algorithm.

- void DrawFilledCircle(uint16_t * **params**, uint8_t **c**):
  - Same as DrawCircle, but fills the circle using Flood Fill. This implementation does not accommodate for too large a circle (a

circle with a radius over 62 pixels long might cause the OS to crash). A solution would be using a form of triangulation present in the DrawFilledPolygon method.

- void DrawPolygon(uint16_t * **vertices**, uint16_t **numberOfVertices**, uint8_t **c**):
    - Draw a polygon. ***vertices** should point to an array of ints where even positions are X values and odd positions are Y values. These should be in the same order as shown in the DrawLine method (x_0, y_0, x_1, y_1, etc.) There should always be numberOfVertices * 2 values provided.

- void DrawFilledPolygon(uint16_t * **vertices**, uint16_t **numberOfVertices**, uint8_t **c**):
    - Draw a polygon, then fill it. The area of the polygon is calculated. If the area is low enough, the Flood Fill algorithm is initiated in the centroid of the polygon. If the area is too large, the polygon is triangulated and the method is called again for each triangle.

- void DrawStyledLine(uint16_t * **vertices**, uint8_t **style**, uint8_t **c**):
    - Draw a line of colour **c** using a set style: 0 for dotted lines, 1 for dashed lines.

- void SetColour(uint8_t **id**, uint8_t **r**, uint8_t **g**, uint8_t **b**);
    - Sets the colour at the specified id to have the provided RGB values.

- uint8_t GetPixel(uint16_t **x**, uint16_t **y**):
    - Utility function to return the colour of the pixel at the specified coordinates.

- void KernelDrawLine(uint16_t **x0**, uint16_t **y0**, uint16_t **x1**, uint16_t **y1**, uint8_t **c**):
    - Verbose version of the DrawLine function, used inside any ring 0 function that requires lines to be drawn.

- void Fill(uint16_t **x**, uint16_t **y**, uint8_t **c**);
    - Personal recursive implementation of the Flood Fill algorithm. Fills in 4 directions.

- void Centroid(uint16_t * **vertices**, uint16_t **numberOfVertices**, uint16_t * **center**);
    - Calculates the centroid of a polygon. * **vertices** should point to an array similar to the one in DrawPolygon. * **center** should point to an array where the x and y coordinates of the centroid will be stored.

- int32_t Area(uint16_t * **vertices**, uint16_t **numberOfVertices**):
    - Utility function that calculates the area of the shape enclosed by the provided vertices. Used to determine if polygons can be filled.

4.2 *vgacalls.h* and *vgacalls.c*

The code in here is heavily inspired by the provided sysapi files. Interrupt 0x81 is set here, alongside all user functions inside the InitialiseVGACalls method which is called inside the main in the Initialise method.

4.3 *user.h* and *user.c*

The user header that calls interrupt 0x81 for each method is applied here. Most functions inside *drawing.c* have a correspondent here, although the user functions tend to be more verbose in terms of their inputs for readability and ease of use. These inputs, usually because they are of the same type, are then placed inside an array and passed through embedded assembly code in this form, enabling ring 3 access to ring 0 methods.

4.4 *vgamodes.h* and *vgamodes.c*

Simple modification to allow retrieval of screen height and width via Get methods.

4.5 *kernel_main.c*

Added InitialiseVgaCalls to the Initialise method. In the main itself, some demo code is provided, displaying the functionality implemented.

5. Known Bugs and Future Development

The code would benefit from some bugfixes, such as using a different method for filling large circles as well as providing a stable way to fill concave polygons. In addition, Flood Fill should be improved to account for errors caused by line drawing (two lines intersecting in a point may create isolated pixels between them, which Flood Fill cannot reach).

Some code repetition may be avoided (for example, the three different line drawing algorithms).

Future development would include providing support for different resolutions as well as maybe using keyboard input to control the demonstration.