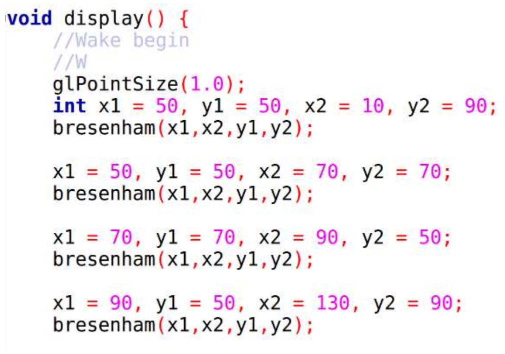
CS460

Homework 1 Report

Sinem Ozden

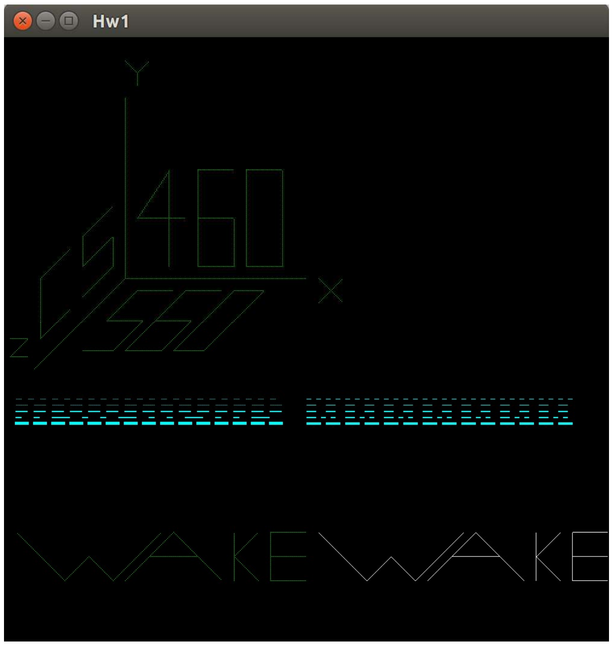
1. Problem Statement:
   1.1. Writing a Bresenham line drawing algorithm. This algorithm will write words WAKE by using only one loop. This algorithm will draw a XYZ coordinate.
   1.2. Writing a Mid-Point line drawing algorithm. This algorithm will draw stipple lines in various thickness.
   1.3. By using OpenGL's line drawing function drawing same shapes as problem 1.1 and 1.2 to compare and verify the algorithms.
   1.4. Writing an algorithm to create lines by using mouse movements. The left click of mouse will create a control point and the right click will create a termite point. After right click f left button is clicked previously created image will be deleted.
2. Algorithm Design:



```
void display() {
    //Wake begin
    //W
    glPointSize(1.0);
    int x1 = 50, y1 = 50, x2 = 10, y2 = 90;
    bresenham(x1,x2,y1,y2);

    x1 = 50, y1 = 50, x2 = 70, y2 = 70;
    bresenham(x1,x2,y1,y2);

    x1 = 70, y1 = 70, x2 = 90, y2 = 50;
    bresenham(x1,x2,y1,y2);

    x1 = 90, y1 = 50, x2 = 130, y2 = 90;
    bresenham(x1,x2,y1,y2);
```

The main function in the program calls the display function. The origin of the screen is set to left bottom corner. The display function automatically creates the shapes for problem 1.1 1.2 and 1.3.

The images that appears in the left side of the screen is created with Bresenham and mid-point algorithms. The right-hand side is created by OpenGL line drawing functions. The series of functions to create the lines separated by the comments.



2.1. For writing the WAKE and the XYZ coordinate the function Bresenham called for each line. In the function bresenham for drawing all octants with one loop all of the other octants should be converted to 1st octant. This conversion done by decide function.

```
int flag = decide(&x1,&y1,&x2,&y2,&dx,&dy);
```

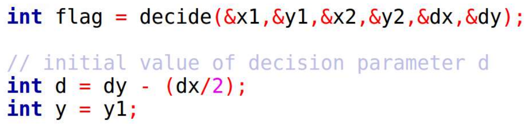
The function convert will do the necessary swapping and reverse the sign of the X, Y dimensions to convert all octants to first octant. The function takes pointers and change the dimensions directly. To decide incoming line's octant, I use dx and dy values. After the conversion of octant is complete the function will return a flag to track in which octant the line was.

```
int decide(int* x1, int* y1,int* x2, int* y2, int* dx, int* dy){
    int flag = 0;
    if(*dx>0 && *dy>=0 && abs(*dx>=*dy)){
        flag = 1;
    }
    else if(*dx>=0 && *dy>0 && abs(*dy>*dx)){
        flag = 2;
        swap(*x1,*y1);
        swap(*x2,*y2);
    }
    else if(*dx<0 && *dy>0 && abs(*dy>*dx)){
        flag = 3;
        *dx = abs(*dx);
        swap(*x1,*y1);
        swap(*x2,*y2);
    }
```

When the points are placed to the screen if necessary, the x and y coordinates are swapped.

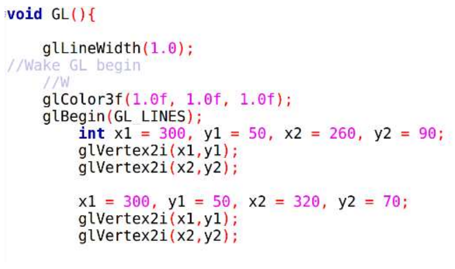
```
if(flag == 2 || flag == 3 || flag == 6 || flag == 7){
    glVertex2i(y,x);
}
else{
    glVertex2i(x,y);
}
```

2.2. The same conversion function is used in the mid-point algorithm. In the display function by using a loop stipple lines created. To hang eth thickens Point width function is used.



```
int flag = decide(&x1,&y1,&x2,&y2,&dx,&dy);

// initial value of decision parameter d
int d = dy - (dx/2);
int y = y1;
```

2.3. The OpenGL functions are called in a function called GL. The GL function called at the end of display function. In GL function by giving to points OpenGL creates a line like the algorithm that I wrote.



```
void GL(){

    glLineWidth(1.0);
//Wake GL begin
    //W
    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_LINES);
        int x1 = 300, y1 = 50, x2 = 260, y2 = 90;
        glVertex2i(x1,y1);
        glVertex2i(x2,y2);

        x1 = 300, y1 = 50, x2 = 320, y2 = 70;
        glVertex2i(x1,y1);
        glVertex2i(x2,y2);
```

To stippled lines stipple should be enabled and the type of stipple is given by the glEnable(GL_LINE_STIPPLE) function.

```
//Stipple Begin

glEnable(GL_LINE_STIPPLE);
x1 = 250.0, y1 = 200.0, x2 = 470.0, y2 = 200.0;
glLineStipple(1, 0x00FF);
glBegin(GL_LINES);
    glVertex2f((x1),(y1));
    glVertex2f((x2),(y2));
glEnd();
```

2.4. This part of the code is created as a different project. To create the mouse clicking algorithm, the glutMouseFunc and glutPassiveMotionFunc functions are used. The glutPassiveMotionFunc function is used because it tracks the mouse even if screen is not clicked. To draw line OpenGL's line drawing function is used. When the left button of the mouse is clicked the initial and ending points for the line is created and the line is pushed to a vector for displaying later. This part also raise the flag of the left button, if the right buttons flag is up the code will clear the vector to clear the screen after the left button is clicked.

```
void mouse(int button, int state, int x, int y) {

    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        Line line;
        if(clickR){
            lineVec.clear();
            glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
            glClear(GL_COLOR_BUFFER_BIT);
            clickR =false;
        }
        line.x1 = x;
        line.y1 = y;
        line.x2 = x;
        line.y2 = y;
        lineVec.push_back(line);
        clickL = true;
    }

    if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
        lineVec.at(lineVec.size()-1).x2 = x;
        lineVec.at(lineVec.size()-1).y2 = y;
        clickR = true;
        clickL =false;
    }
}
```

```
struct Line{
    int x1,y1,x2,y2;
};
vector<Line> lineVec;
bool clickR=false;
bool clickL=false;
```

When the mouse moves, the movement function checks if the left button is clicked. If it was changes the last two points of the last line of the vector.

```
void motion(int x, int y) {
    if(clickL){
        lineVec.at(lineVec.size()-1).x2 = x;
        lineVec.at(lineVec.size()-1).y2 = y;
        glutPostRedisplay();
    }
}
```

The right button disables the left click flag thus the movement function no longer change the end points of the line. This part also raise the flag of the right button is clicked.

```
void display(){
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
        glColor3f(1.0f, 1.0f, 1.0f);
        for(int i=0; i<lineVec.size(); i++){
            glVertex2i(lineVec.at(i).x1/2,(1000-lineVec.at(i).y1)/2);
            glVertex2i(lineVec.at(i).x2/2,(1000-lineVec.at(i).y2)/2);
        }
    glEnd();
    glFlush();
}
```

The display function will display all of the lines that drawn up to that point.

3. To compile the first 3 parts of the assigment run these lines:

   ```
   g++ -Wall -g -I/usr/include -c main.cpp -o main.o
   g++ main.cpp -lglut -lGL -lGLU -lXxf86vm
   ./a.out
   ```

   For mouse click part run:

   ```
   g++ -Wall -g -I/usr/include -c mouse.cpp -o mouse.o
   g++ mouse.cpp -lglut -lGL -lGLU -lXxf86vm
   ./a.out
   ```