CS460

Homework 2 Report

Sinem Ozden

1. Problem Statement:
   1.1. Draw a clipping rectangular window.

   1.2. Draw a polygon around the clipping window through the mouse input.
   1.3. When choose the menu “polygon clipping”, perform the polygon clipping using Sutherland-Hodgeman algorithm.
   1.4. When menu “region filling” is clicked, fill the clipped polygon using a polygon filling algorithm with boundary-filling to fill a purple color.
   1.5. Define a viewport and map the clipped polygon in the window to the viewport.

   1.6. Change the viewport size by using the mouse-click to drag the top-right corner of the viewport and display the scaling effect of the clipped polygon simultaneously.
   1.7. Change the window size and map the clipped polygon from the window to viewport dynamically, achieving the *zoom-in* and *zoom-out* effect.
   1.8. Move the window horizontally (or vertically) and map the clipped polygon from the window to viewport, achieving the *panning* effect.
2. Algorithm Design:
   2.1. I used GL_LINES_LOOP to draw the window rectangle

```
glEnable(GL_LINE_STIPPLE);
glLineStipple(1, 0x00FF);
glBegin(GL_LINE_LOOP);
    glColor3f(1.0, 0.0, 0.0);
    glVertex2f(xwmin,ywmin);
    glVertex2f(xwmax,ywmin);
    glVertex2f(xwmax,ywmax);
    glVertex2f(xwmin,ywmax);
glEnd();
glDisable(GL_LINE_STIPPLE);
```

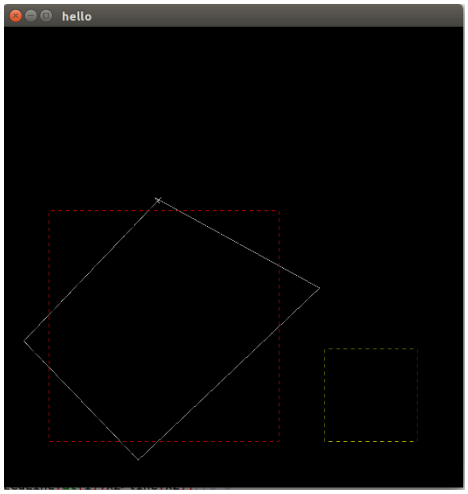
   2.2. To create the mouse clicking algorithm, the glutMouseFunc and glutPassiveMotionFunc functions are used. The glutPassiveMotionFunc function is used because it tracks the mouse even if screen is not clicked. To draw line OpenGL’s line drawing function is used. When the left button of the mouse is 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.

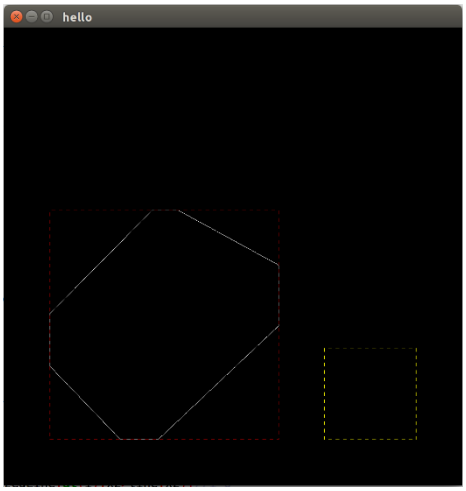
```
else{
    glutDetachMenu(GLUT_RIGHT_BUTTON);
    Line line;
    if(clickR){
        lineVec.clear();
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);
        clickR =false;
    }
    line.x1 = x;
    line.y1 = (1000-y);
    line.x2 = x;
    line.y2 = (1000-y);
    lineVec.push_back(line);
    clickL = true;
    clickV = false;
    clickW = false;
    moveW = 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.
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.
The display function will display all of the lines that drawn up to that point.

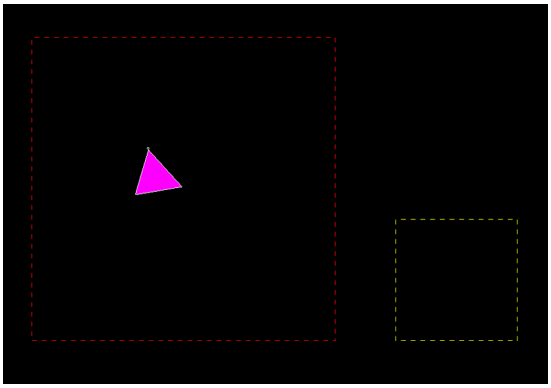
```
void motion(int x, int y) {
    if(clickL){
        lineVec.at(lineVec.size()-1).x2 = x;
        lineVec.at(lineVec.size()-1).y2 = (1000-y);
        glutPostRedisplay();
        glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Se
    }
    seedx = x, seedy = 1000-y;
}
```

2.3. For polygon clipping I call the function SuHo in the menu. The SuHo functions variable differs vether if its called for clipping or for window to viewport. The SuHo function then calls clipping function 4 times. The clipping function first cut from left then right finally bottom and up and return a point vector back to SuHo function. Then by using this point vector the SuHo function creates a line vector. Then this line vector shown by the display function.





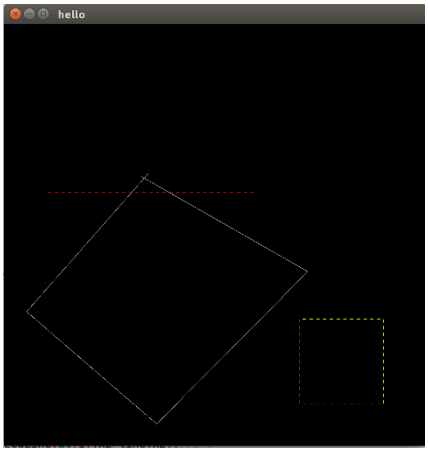
2.4. To make sure region fill works correctly when you right click for opening the menu please right click into the region that you want to fill. The mouse movement will get that point as seed. When the boundary fill function is called it uses these points as seed and start filling form these points. The boundary fill is a 4-way boundary fill. So, boundary fill function calls boundary4 function. The boundary4 is a recursive function. It calls itself until readPixels see a boundary or previously filled area.

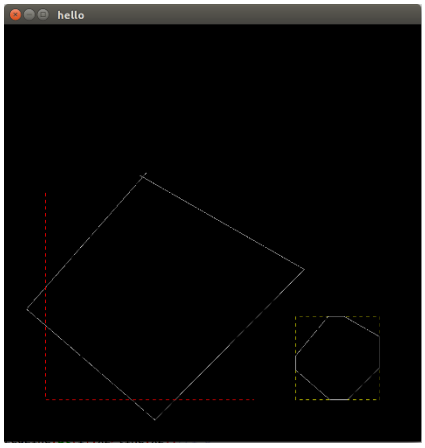


2.5. To convert the polygon in the window to viewport I used a mathematical conversion formula. The menu calls for the win_view function. This function does conversion for very line of the polygon. For this conversion resize function is called. After conversion done the

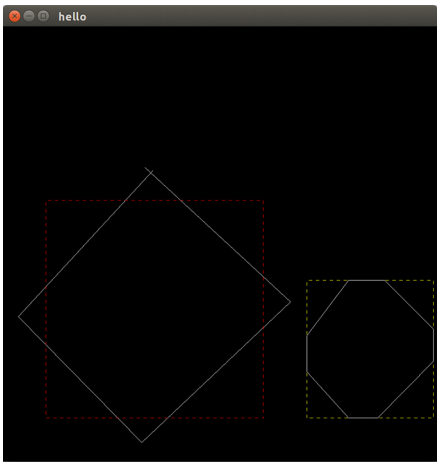
win_view function calls the SuHo function to clip the polygon. Finally, the new lines displayed in the display function.

```
void resize(Line line){
    Line newline;
    newline.x1 = (line.x1-xwmin)*(xvmax -xvmin)/(xwmax - xwmin) +xvmin;
    newline.x2 = (line.x2-xwmin)*(xvmax -xvmin)/(xwmax - xwmin) +xvmin;
    newline.y1 = (line.y1-ywmin)*(yvmax -yvmin)/(ywmax - ywmin) +yvmin;
    newline.y2 = (line.y2-ywmin)*(yvmax -yvmin)/(ywmax - ywmin) +yvmin;
    resizedLine.push_back(newline);
}
```

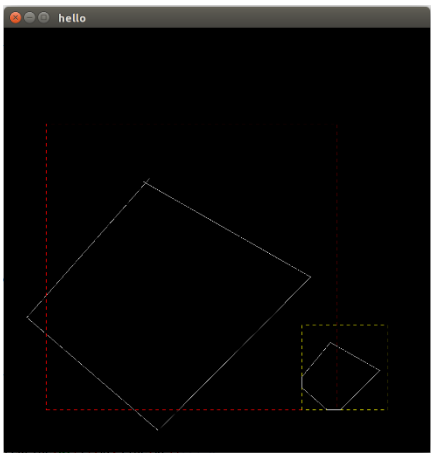




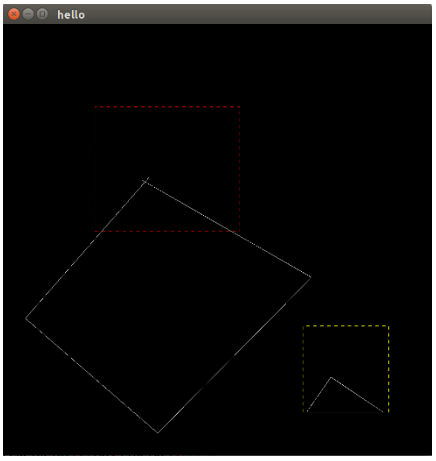
2.6. Viewports right top corner (corner and 50 radius pixel) is used to make viewport size bigger and do the necessary conversions accordingly. When the region around right corner is left clicked a flag activated. And when you drag and drop the motion 2 function is activated. It changes the top right coordinates simultaneously and re draws the rectangle. Then call the win-view function to resize the polygon.



2.7. When the window size is changed the similar flow happens to 2.6.



2.8. When you want to move the window the top edge of the window should be grabbed. When it grabbed the flag is activated, the old coordinates of grab is saved. As the mouse moved without releasing the left button the window moves. Every time mouse moves the motion 2 function is called. The coordinates of window changed, and the line vector of the window updated. The changes in the viewport is done.



3. To compile the first 3 parts of the assignment run these lines:
   g++ -Wall -g -I/usr/include -c main.cpp -o main.o
   g++ main.cpp -lglut -lGL -lGLU -lXxf86vm
   ./a.out