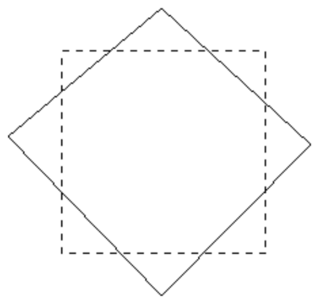
CS460 Assignment 2 Writeup Ryan Shendler

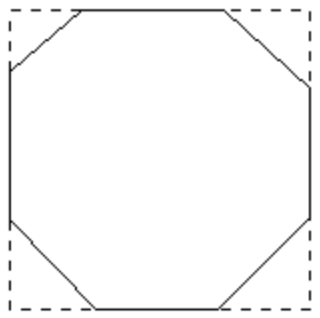
My program has three main functions: draw a polygon and clip it with respect to a window, fill the polygon with the color red and map the clipped polygon from the window to the viewport.

In order to draw a polygon, the user can left click in order to add control points. The program will automatically draw a series of line segments that connect all of the control points. Clicking the middle mouse button will place the final control point and draw the completed polygon. Each time the program registers a left-click, the processMouse function will add the clicked point to the global vector polygon, which contains all of the vertices of the polygon. Clicking the middle mouse button adds the final control point to the polygon vector and then sets the global flag drawPoly, which indicates that all polygon vertices have been specified and the control points can be connected by line segments.



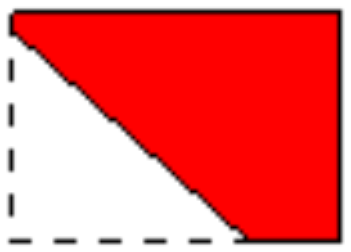
Polygon drawn by mouse input

Right-clicking the mouse will open a menu with three options: Polygon Clipping, Region Filling, and Window-to-Viewport Mapping. If polygon clipping is selected, the program will use the Sutherland-Hodgeman algorithm (implemented in clipPolygon) to clip the polygon with respect to the window (window is the box drawn with dashed lines). The clipPolygon function stores the vertices of the clipped polygon in the global vector clipped.



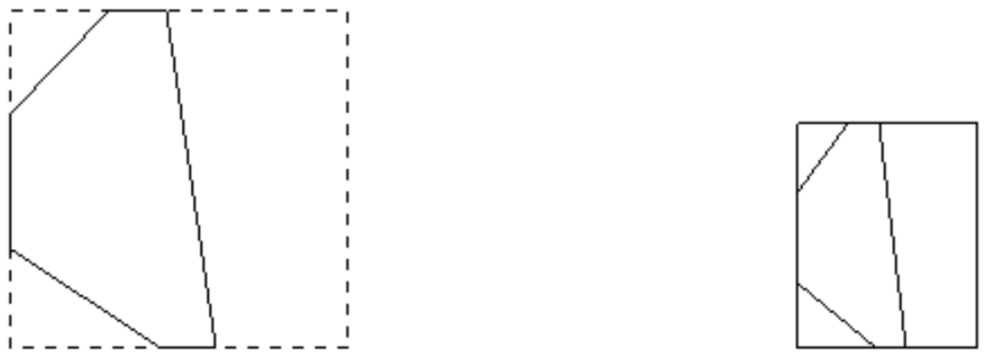
Polygon clipped using Sutherland-Hodgeman

Selecting the region filling menu option will allow you fill the clipped polygon with a red color. If the polygon wasn't already clipped, it will be automatically clipped upon selecting this option. In order to fill the polygon, hover the mouse over any pixel within the clipped polygon and hit the 'f' key. This will use that pixel as the seed pixel for the boundary fill algorithm, which is implemented in boundaryFill, to fill the interior of the polygon with red pixels. Since boundary fill is a recursive algorithm, it can take a significant amount of time and stack size in order to complete. It is likely that the polygon will not fill immediately upon clicking the 'f' key, and you may have to wait a few seconds before the filling completes. I would suggest that you test the boundary filling on a small clipped polygon in order to save time and stack space.



Clipped polygon filled with boundary fill

Selecting the window-to-viewport mapping menu option will display the viewport (viewport is the window drawn with solid black lines) and maps the clipped polygon from the window to the viewport. If the polygon hasn't been clipped, it is automatically clipped before being mapped to the viewport. The window-to-viewport mapping is implemented in function mapToViewport. You can drag the top right corner (by holding left click) of the window in order to change the size of the window. This will cause the mapped polygon in the viewport to zoom-in or zoom-out. You can also drag the top right corner of the viewport in order to change its size and scale the polygon in the viewport. You can also use the arrows to move the window up, down, left or right, which will cause a panning effect in the viewport. Each time the window or viewport change size/location, my program updates the global vectors window and viewport, which store the current vertices of the window and viewport. After updating these vectors, the program clears and redraws the display, thereby making it look like the window/viewport are changing dynamically.



Clipped polygon mapped from window to viewport

My program can be run by opening the .sln file in Visual Studio and building the project. Make sure that the OpenGL libraries are properly installed and that the stack reserve is set to at least “2000000” in order to avoid stack overflows.