INF-1100: Introduction to programming and computer behavior

Assignment 3 - Teapot

Kristian Harvey Olsen and Eskil Bjørnbakk Heines

05.Oktober.2022

1 Introduction

In this assignment we are tasked to make a teapot out of triangles and fill the triangles with color. We were given some pre code which we were tasked to complete. The implementation is divided as follows:

1. Implementation of translating the triangles to the center of our screen
2. Make an algorithm to draw lines through all the triangles to fill up the teapot.
3. Calculation of the bounding boxes for each triangle.
4. With the use of two different algorithms, fill the triangles with color.

2 Technical Background

The pre-code uses the Simple Directmedia Layer (SDL)[1] to open a window with a given size where we can see what is being drawn and the changes made to it. From the pre-code we use Makefiles[2] to simplify the compiling process of all the files. When solving the problems, we use if/else test, for loops, and functions that are already defined in the pre-code.

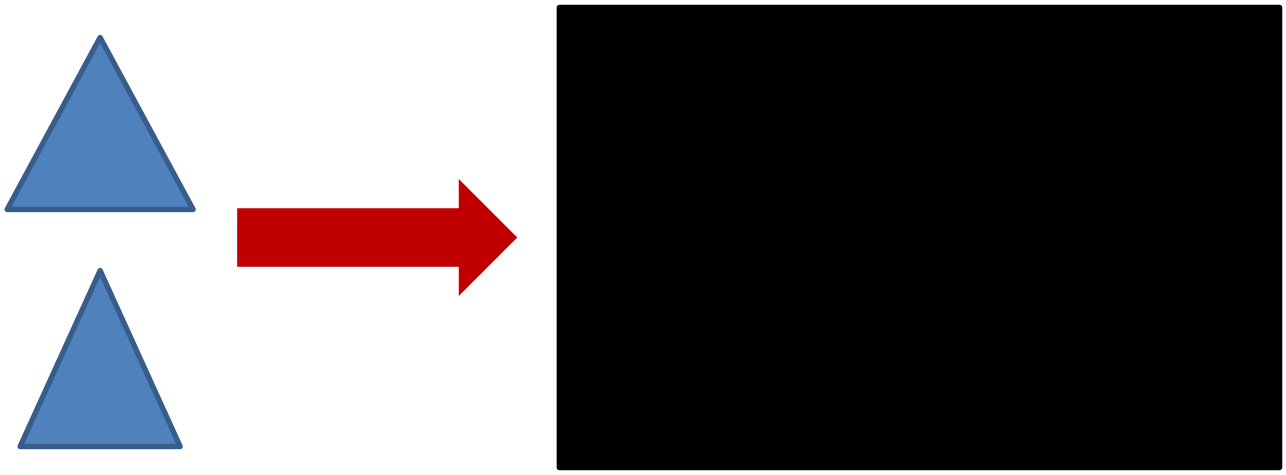
3 Design, Implementation and evaluation

To solve the tasks, several algorithms have been chosen. The algorithms aim to translate the triangles, calculate the bounding boxes and at last fill them with color. We used the “example triangles” as a reference while we implemented different algorithms to find a solution, and at last implement it to the actual teapot. The algorithms work according to the following process:

3.1 Triangle translation

We were given all the points which make up the teapot in our pre code, but it doesn't know where our screen coordinates are. So, to make it centered to the screen, a translation is needed.

The translation is implemented by giving variables the total size of the screen and dividing it by two. With the use of our variables, we can calculate new on-screen coordinates by adding the variable to our existing triangle coordinates.



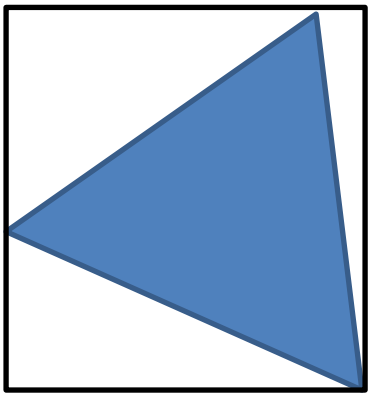
3.2 Algorithm for drawing the triangles

The pre code includes an algorithm to make the function that draw lines translate to the screen. So, what we need to implement is a conditional function to draw a line through the array of coordinates given in the pre-code.

With the use of a for-loop through the total amount of coordinates and draw a line between all of them. The for-loop goes through the whole array of coordinates and draws lines between all of them to make up all the triangles in the teapot.

3.3 Calculating the bounding boxes

To calculate the bounding boxes of all the triangles, we implement simple geometry to our code. The bounding box should be a rectangle that exactly hits all the corners of the triangles. Here is an example of how this may look:

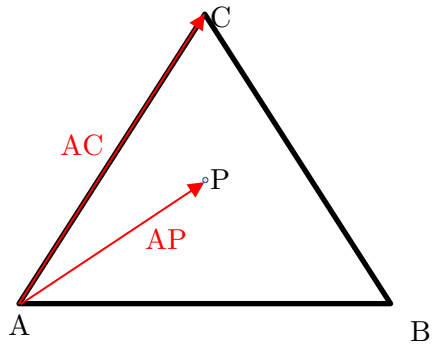


The corners of the bounding box will have the same values as highest and lowest x and y values in the triangle. We find the highest and the lowest x and y values in the triangle by using the fmax and fmin functions. Fmax will compare two values that are implemented and give us the highest of the two values. Vice versa for fmin, only we are given the lowest value of the two. We implement the functions with the value of the coordinates in the array given to us. Now we can define these values so we can draw the bounding boxes, but also so we can use them further down in the code. We have defined these values as XMIN, YMIN, XMAX ang YMAX.

3.4 The first algorithm to fill the triangles

In the first algorithm for filling the triangles with color, we solve it using the cross product of two vectors to check if a point is inside or outside of the triangle. If the point is inside the triangle, we paint it in the chosen color of the triangle given in the same array where we find out coordinates.

The algorithm consists of a nested for-loop that checks all the points in the bounding box of the triangle. For each point we go through we check if the point is inside the triangle by finding the cross product of the vector between two corners and the vector between one of the corners and the point as shown below:



AC x AP = |AB| * |AP| * sin(angle) * n-vector

If AC x AP = 1, point P is in the triangle

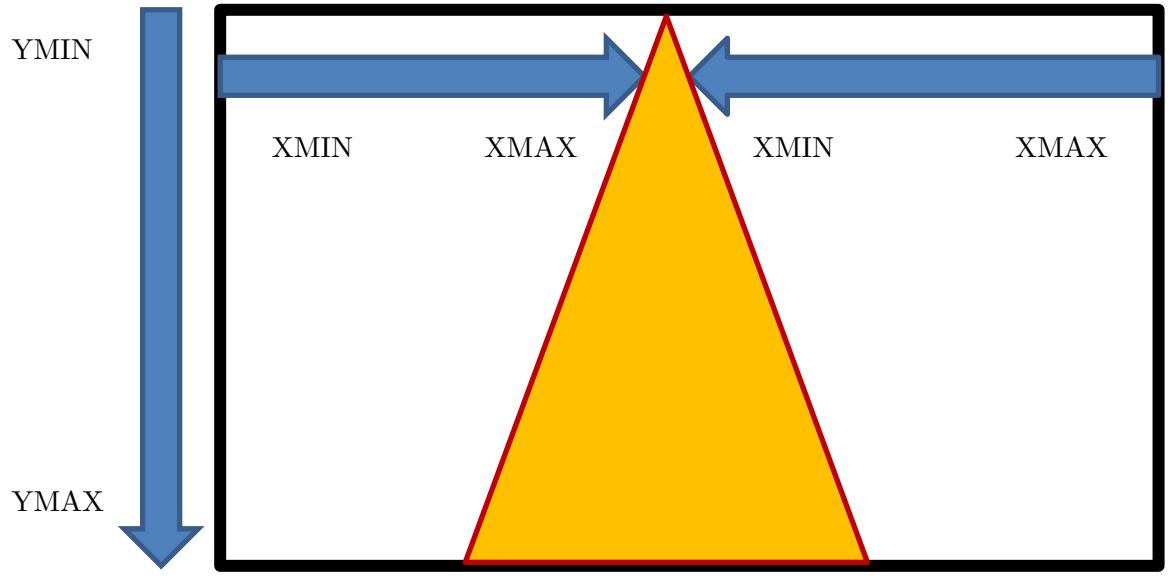
If AC x AP = 0, point P is not in the triangle

By using an if test, we condition that the vector between two corners and a corner and the point must be over 0 for the function to draw the pixel in the point to the chosen color. This loop checks all the points and paints those who match with the condition.

3.5 The second algorithm to fill the triangles

In the second algorithm for filling the triangles with color, it is implemented with the use of conditional algorithms. When it is inside a bounding box and it does not see the color of the triangles, the condition should continue till the triangles are filled with color.

To implement this algorithm, we have used triple nested for-loops which start from YMIN to YMAX, and while it is inside those boundaries, then it should increment from XMIN to XMAX. And finally, to check from the other side of the bounding box we start from XMAX and go to XMIN while decreasing in each iteration. In each for-loop we also use if tests to break the for-loop if the color is equal to the color, we have painted the triangles in the teapot. We find two x values, that are the borders of the triangle, and draw a line between them in the color given by the array.



4 Discussion (optional)

In the first algorithm we find all the points that are inside the triangles, but not the ones that are on the triangle. The algorithm does not include if the cross product of the vectors is equal to 0. We know that the consequence of this is that when we draw the lines of the triangles using TRIANGLEPENCOLOR, we can see the red lines between the triangles. That's why we draw all the triangles using triangle->fillcolor.

5 Conclusion

We were tasked to implement the translation of the triangles to make them centered to our SDL screen, make an algorithm to draw the lines through all the triangles to fill up the teapot, calculate the bounding boxes of the triangles, and to find two different algorithms to fill the triangles with color.

We implemented algorithms and calculations that solved all the problems we were tasked to solve. The algorithms did exactly what we expected them to do and resulted in both drawing the triangles and filling them with the chosen color.

Acknowledgement (optional)

We have discussed the assignment with Aslak Bjordal (INF 3th year) and Vetle Hofsøy Woie (INF 5th year).

Solving the first algorithm for filling the triangles we found a bit of code [4] on the internet that helped us a lot understanding how to write the code for the cross product of the vectors. We have not directly copied it, mainly because the code did not work seeing that it had implemented the wrong values.

References

[1] Simple Directmedia Layer, Simple Directmedia Project,
https://www.libsdl.org/
[2] Makefiles, GNU Make,
https://www.gnu.org/software/make/
[3] Triangle Interior, Mathematical Algorithm to fill triangle
https://mathworld.wolfram.com/TriangleInterior.html?fbclid=IwAR2XAKhIAmElTAThU4i9JkruaNL6U-Dl9nkwHPE6oom-SZNvEu664Pr0YWo
[3] Points in triangle Mathematical Algorithm to fill triangle,
https://blackpawn.com/texts/pointinpoly/?fbclid=IwAR0UlXA-Scg0axyLY0VTbNQCOAGr_O7yjDfRI-1VLnbal5xEnrkCNeXpAA4
[4] Stackoverflow fill triangle with color, Matematical Algorithm to fill triangle
https://stackoverflow.com/questions/58332985/incorrectly-colored-triangles-with-sdl2-based-triangle-rasterizer