Computer Graphics Assignment #2
Lighting and shading
Triangle fill and z-buffer

Part 1 of the second assignment focuses on painting full triangles using either the scanline or edge walking approaches, and implementing the z-buffer algorithm. Note that for this assignment, while there are no performance requirement, an inefficient implementation you will make you spend too much time on producing results. You can choose which algorithm to use for filling a triangle: Scanline or Edge walking. The advantages and disadvantages were discussed in class. Since Scanline is a bit more difficult to implement, a bonus point will the added if you choose to implement it. For this part please create a markdown file 'Assignment2Report/Assignment2Report_part1.md', and do the following:

1. For debugging purposes, change the DrawTriangle function you implement in Assignment 1 such that it will draw the bounding rectangle for each triangle in screen space. The color of a rectangle should be related to the depth of the triangle, and you are free to choose exactly how. Explain, and show an image in the report. Allow the user to toggle this feature on and off.

2. If you choose to implement the edge walking approach you can skip this part. With the scanline approach, the triangle edges are drawn first, and based on their pixels, the *pixels* inside the triangle can be identified. However, this is only possible if no other triangle edges are already drawn on top of the current triangle. To overcome this, in the DrawTriangle function, create a 2D boolean buffer with the same dimensions as the bounding rectangle and initialize its values to false. This array will be used to determine which pixels belong to the triangle independently of other triangles that were perhaps already drawn. For every pixel you put when you call drawLine, change the corresponding entry in the array to true. Then use this buffer when you run the scanline procedure.

3. Change the DrawTriangle function such that it fills the entire triangle. Place a model in the scene such that it is clearly visible. Pick a random color for every triangle and place a screenshot of the result in your report. Note that you will see triangles overlapping; triangles in the back might be occluding triangles in the front.

4. Implement the z-buffer algorithm. Create a buffer the same size as the color buffer but with one floating point value per pixel. Update the z-buffer in the DrawTriangle function. The issue mentioned above should not occur anymore. The z-buffer can be visualized as a grey scale image (as in the lecture). Show side-by-side pictures of the color buffer and z-buffer, for several models or viewpoints.