Assignment Self Evaluation Sheet

Mathematics for Computer Graphics Assignment 2

Student's Name: Maciej Legas 4922675

This self-evaluation sheet is marked only on completeness (i.e. please be honest!). The purpose is to help you reflect on your performance and to help identify features of your work.

	Yes	**No**
Did I complete the minimum requirements for the assignment?	**X**	
Did I add any extensions (i.e. more advanced features) to the assignment?		**X**
Did I read up on the subject beyond lecture / lab contact?	**X**	
Did I spend enough time on the assignment?	**X**	

	Very happy	Satisfied	Disappointed	Ashamed
How happy am I with what I submitted?	**X**			

References for sources and tutorials I used:	Academo, not dated. *2D Rotation about a point* [online]. Academo. Available from: https://academo.org/demos/rotation-about-point/ [Accessed 4 May 2018]. Beej, 2017. *The Mandelbrot Set* [online]. Beej.us. Available from: http://beej.us/blog/data/mandelbrot-set/ [Accessed 4 May 2018]. damix911, 2016. *How to use glm::project to get the coordinates of a point in world space?* [online]. Stack Overflow. Available from: https://stackoverflow.com/a/35269684 [Accessed 4 May 2018]. Dr. Dilts, 2017. *The Cantor Set* [online]. Infinity Plus One. Available from: https://infinityplusonemath.wordpress.com/2017/10/21/the-cantor-set/ [Accessed 4 May 2018]. de Vries, J., not dated. *Camera* [online]. Learn OpenGL. Available from: https://learnopengl.com/Getting-started/Camera [Accessed 4 May 2018]. Duke University, not dated. *Lab 7: Sierpinski Fractals and Recursion* [online]. Duke University. Available from: https://www2.cs.duke.edu/courses/cps001/summer05/labs/lab7.html [Accessed 4 May 2018]. Duke University, not dated. *Lab 7: Sierpinski Fractals and Recursion* [online]. Duke University. Available from: https://www2.cs.duke.edu/courses/fall01/cps001/labs/lab7.html [Accessed 4 May 2018]. MathWarehouse, not dated. *The Distance Formula* [online]. MathWarehouse. Available from:

	http://www.mathwarehouse.com/algebra/distance_formula/index.php [Accessed 4 May 2018]. Hardmath123, 2015. *Coding the Mandelbrot Set - Comfortably Numbered* [online]. Hardmath123.github.io. Available from: https://hardmath123.github.io/scratch-mandelbrot.html [Accessed 4 May 2018]. karatedog, 2015. *How would you generate a Sierpinski Triangle in C (recursively)* [online]. Stack Overflow. Available from: https://stackoverflow.com/a/33166953 [Accessed 4 May 2018]. Khan Academy, 2015. *1. Weighted average of two points* [online]. Khan Academy. Available from: https://www.khanacademy.org/partner-content/pixar/environment-modeling-2/mathematics-of-parabolas2-ver2/v/weighted-average-two-points [Accessed 4 May 2018]. LukeP, 2015. *How can I set up an intuitive perspective projection/view matrix combination in OpenGL, using GLM?* [online]. Game Development Stack Exchange. Available from: https://gamedev.stackexchange.com/questions/98226/how-can-i-set-up-an-intuitive-perspective-projection-view-matrix-combination-in [Accessed 4 May 2018]. Opengl-tutorials, not dated. *Tutorial 3: Matrices* [online]. Opengl-tutorials. Available from: http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/ [Accessed 4 May 2018]. Schiffer, J., 2015. *CODE GENIUS - Drawing Lines with Bresenham's Line Algorithm by Jenn Schiffer* [online]. YouTube. Available from: https://www.youtube.com/watch?v=zytBpLlSHms&t=272s [Accessed 4 May 2018].
Main features of my project:	The user, using a text menu, can draw: - Basic 2D shapes - Filled 2D shapes - A showcase of matrix transformations on a 2D square - A rotating 2D square (animated) - Bezier curves - Fractal sets: Cantor, Sierpinski and Mandelbrot - A rotating 3D pyramid (animated)
The best part of my performance was:	Drawing the fractal sets and the overall documentation of the code. I am really satisfied with the implementation of the fractals, as well as the detailed comments of almost everything in the code.
The worst part of my performance was:	The projection of the 3D shape. For some, unknown to me reason, the projection does not work fully except for the aspect ratio and the angle of the camera, as it still draws the sides that are should not be possible to be seen.
One way in which I could improve my submission is:	Improve the interpolation of colours and draw a Gouraud triangle, with each corner being the epicentre of one of the main RGB colours.

One thing I will do to improve my next submission is:	Increase the amount of time reading the documentation of the provided libraries, so that my knowledge of using the proper functions will adequately help me while coding the next assignment.

Report Template

Introduction

The main task of this assignment focused on drawing a line and at least the most basic primitive shapes: a circle, square, and a triangle. The analysis in this work will be about the research background behind drawing these shapes.

Previous Work

Drawing shapes requires the use of points, which are most often represented as two dimensional vectors (Rogers and Adams 1990, p. 61). For drawing two out of three of the required shapes, we need to be able to draw their sides, which means we need to draw a line between two points. Without using a graphics library function for it, the most basic approach would be using the Cartesian slope-intercept equation for a straight line, which is (Hearn and Baker 2004, p. 93):

$$y = m * x + b$$

If the equation values are unknown except for the points, we can determine the slope m and the y intercept b with the use of following calculations (Hearn and Baker 2004, p. 93):

$$m = \frac{y_{end} - y_0}{x_{end} - x_0}$$
$$b = y_0 - m * \ x_0$$

Implementing the equation into the code to draw a line has its troubles though. A vertical line will cause the program to fail, as it will try to divide by zero when calculating the slope. This can be handled by an if statement, but it will draw two points at best – the starting point and the end point. Slopes of a high, absolute value (above 1) will impact the lines to have wide gaps between pixels. The only lines that will look properly will be horizontal ones. Also, accidentally switching the end points with the starting points will cause the lines to be wrongly drawn, but this could of course be fixed with if statements.

Therefore, more evolved line drawing algorithms arose. The most popular line drawing algorithms are Digital Differential Analyzer and Bresenham.

The Digital Differential Analyzer, DDA for short, is a quite similar approach to the line equation one. The k value is being incremented with each iteration, until we have reached the end point with our pixel position. We add the calculated slope of the line to each point y, and round the end value to an integer, which should create a smooth line (Hearn and Baker 2004, p. 94).

$$y_{k+1} = y_k + m$$

If the slope is above 1, we switch y with x, and add an inverse of the slope in the equation. That means we divide the rate of change of the x value over y (Hearn and Baker 2004, p. 94):

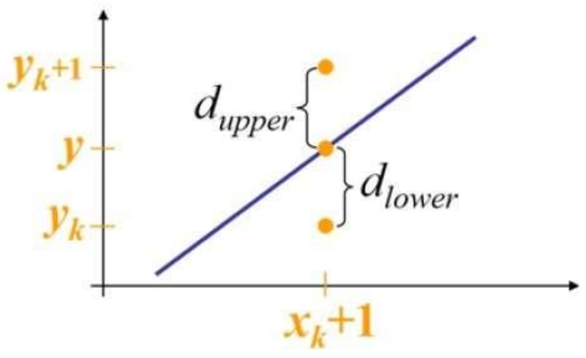
$$x_{k+1} = x_k + \frac{1}{m}$$

The DDA line algorithm is an improvement to the implementation of the standard line equation, as switching the endpoints with starting points will just cause the equations to subtract the slope instead of adding it (Hearn and Baker 2004, p. 94):

$$y_{k+1} = y_k - m$$
$$x_{k+1} = x_k - \frac{1}{m}$$

Overall, the DDA line algorithm is quicker than using the standard line equation, as it switches the multiplication of the slope with adding it to the x and y values. Unfortunately, it has disadvantages as well. The DDA algorithm bases greatly on floating point values, since the slope is always between 0 and 1, which may cause the line to gain an offset from the slope for longer line length due to rounding the float values to integers, as well as being slower than algorithms which use integers only.

The Bresenham line algorithm uses a decision parameter to decide which pixel position to draw when iterating.



1 How the decision parameter works in the Bresenham line algorithm (Tutorials Point n.d.).

The decision parameter is calculated by this equation (Hearn and Baker 2004, p. 97):

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} + y_k)$$

As the rate of change of the value of y and x are integers, this version of the algorithm allows for quick calculations while maintaining precision when drawing the line.

What about the circle? The most common approach is to use this equation for drawing the pixels (Boreskov and Shikin 2014, p. 90):

$$F(x, y) = x^2 + y^2 - R^2$$

This function will return 0 when the points are on the circle, a negative value when the points are inside of the circle and a positive value if they are outside of it.

Analysis

I used a version of the Bresenham's line algorithm which is using a floating-point value for the slope and pitch (offset). In comparison to the algorithm referenced in the previous work, it will possibly draw a slightly more precise line with a small efficiency cost of calculations.

Though it takes more lines to code than the DDA approach, the lines are more precise with only a small loss of execution time, and that is what is important. What surely could be improved in the algorithm is the implementation of anti-aliasing to soften the sharp edges of the drawn lines.

As for the circle drawing, instead of scanning the entire screen I could use the Bresenham's circle drawing algorithm to improve efficiency. The edge of the circle could be also drawn by rotating a point by 360 degrees on the end of the radius.

References

Boreskov, A. and Shikin, E., 2014. *Computer Graphics: From Pixels to Programmable Graphics Hardware*. Boca Raton, FL: CRC Press.

Hearn, D. and Baker, M., 2004. *Computer graphics with OpenGL*. Upper Saddle River, NJ: Pearson Prentice Hall.

Rogers, D. and Adams, J., 1990. *Mathematical elements for computer graphics*. New York: McGraw-Hill.

Tutorials Point, n.d. *How the decision parameter works in the Bresenham line algorithm.* [online]. image. Available from: https://www.tutorialspoint.com/computer_graphics/line_generation_algorithm.htm [Accessed 5 May 2018].