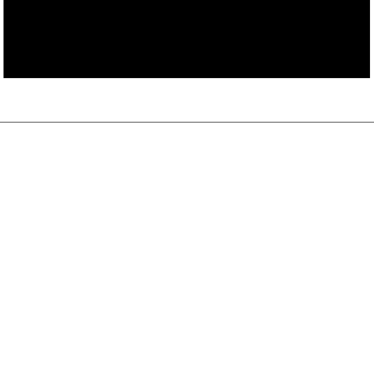
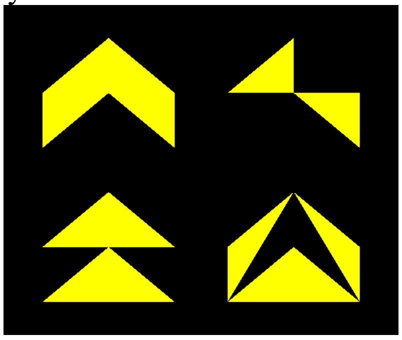
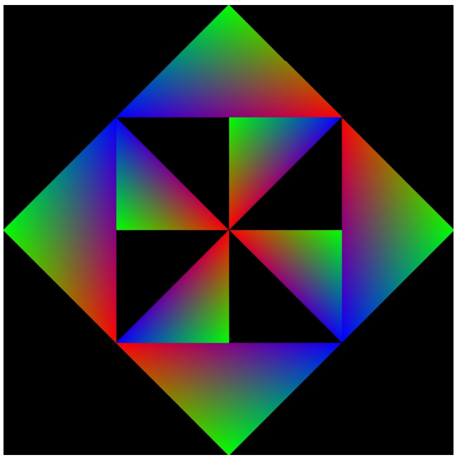
Exercise 02561-01: Primitives and Attributes - Introduction to OpenGL

Reading	ANG: Chapter 2.1 – 2.5, 3.4 - 3.5, 3.8
Purpose	The purpose of this exercise is to give a short introduction to create, edit, link and run C/C++ programs that are based on the graphics library OpenGL, and the utility libraries Angel and GLUT. To keep it as simple as possible we will deal with 2D graphics only. The students will learn to: • Create primitives composed of triangles • Use different primitive modes to render surfaces – this includes triangles, triangle strips and triangle fans. • Use orthographic projection to specify the view in the scene. • Use model transformations (rotate, scale and translate) to transform the rendered objects.(Use functions from `mat.h`)
Startup	In order to create and run the programs you may use the description of starting Visual Studio (on the CampusNet).
Delivery	You deliver all the exercises at the end of the course (both source code, windows executable and a written report). Therefore for each programming exercise make a screenshot of the program and insert it in the final report.
Part 1 Hello triangle world	Run the demonstration program 02561-01-01 (found in Code/Exercise01), which draws a triangle in a window on the screen. This is one of the simplest OpenGL program – a kind of “Hello World” program. a) Try to understand the purpose of each function of the C++ program main-01-01.cpp. Write what each function in the program does. (For now ignore the rest of the files in the project). b) Modify the problem to draw an extra white triangle like this: 

Exercise 02561-01: Primitives and Attributes - Introduction to OpenGL

Part 2 Vertices and transformations	Run the program 02561-01-02 which draws a rectangle. The program has a few more features and can be used as a kind of template for drawing 2D drawings in OpenGL. a) Try to understand the overall structure of the program and the purpose of each function. b) Extend the program to include a triangle with vertices (2, 2), (5, 2) and (3.5, 5). Make that triangle red. c) Translate the triangle with the vector (6,7,0) by modifying the modelView-matrix. Make the color of the vertices red, green and blue respectively. d) Rotate the rectangle (but not the triangle) 45 degree counter-clock-wise around the midpoint of the rectangle.
Part 3 Draw arrays and draw elements using different primitive types	Run the program 02561-01-03 which draws a rectangle. a) Explain the difference between glDrawArrays and glDrawElements. b) Explain the difference between GL_TRIANGLES, GL_TRIANGLE_STRIP and GL_TRIANGLE_FAN. c) Extend the program to draw two other shapes using the VertexArrayBuffer like the screenshot below: 
Part 4 Creating a mesh	Change 02561-01-04 to draw a circle instead of a triangle.

Exercise 02561-01: Primitives and Attributes - Introduction to OpenGL

Part 5 Model transformations	Program 02561-01-05 draws a red/green/blue triangle with the vertex positions (0,0), (1,0), (1,1) using an Ortho2D projection from -20 to 20 in x and y axis. • Modify the display function to get the same result as the image below. This can be achieved by using model transformations (rotate, translate and scale). Do not modify the vertex positions or vertex colors. 
Part 6 Viewport transformations	A window – viewport pair is defined by the coordinates [xw1,xw2,yw1,yw2] and [xv1,xv2,yv1,yv2]. Such a pair defines a transformation (matrix); look at figures 2.5 and 6.4 in the book. a) Write down the transformation matrix which maps a point (xw,yw) to the point (xv,yv). b) The window-viewport transformation can also be defined from concatenation of three simple transformations. Which types of transformations are these? Write down the matrices for these three transformations. Check that the concatenated matrix is equal to the window-viewport transformation matrix. c) How does OpenGL handle window – viewport transformations?
Part 7 - OPTIONAL Procedural generated meshes using plane fractals	OPTIONAL: Program 02561-01-07 shows how the Sierpinski triangle (a plane fractal) can be created (also described in the book chapter 2.9). • Modify the program to render the Sierpinski carpet instead. The Sierpinski carpet is also a plane fractal, but composed of squares. The pictures below show the result for 1, 2 and 3 iterations.