Working with shaders allows more results than just rendering a scene. Sometimes special techniques are implemented for altering a rendered image with an additional effect or visualization style that is not possible to obtain through the regular rendering process. Such kind of techniques (e.g., blur, edge detection, sharpening…) are classified under the names Post-Production or Post-Processing and often are variants of 2D Image Processing.

Such methods are usually applied to the image obtained after the rendering process. In our case such techniques will be implemented in a fragment shader after the image from the 3D scene is obtained. In OpenGL, the image obtained after drawing the scene is saved in a buffer (i.e., the framebuffer) can be saved and/or treated as an image manipulated by a fragment shader. Depth information (for 3D scenes) can also be saved.

Objective:
The objective of the assignment is to implement well-known image post-processing techniques once a scene is rendered. The techniques to be implemented are the following:

- Sepia tone color
- Motion blur
- Swirl distortion
- Sketch filter
- Oil painting (optional)

Note: The framework for this assignment already contains a basic blur implementation. Remember that post-processing is implemented in a fragment shader, then the rendering logic must be translated from the vertex arrays approach to the vertex and fragment shaders style.

Objects will be loaded in Wavefront .OBJ format and a parser for that is provided – the format is quite trivial. An image reading function is also provided for convenience. An example cube.obj is provided.
Specifics:

(20%) Sepia Tone Color
Manipulate the color of the texture in order to obtain a sepia tonality in the rendered images. One simple conversion is

\[
\begin{align*}
\text{outputRed} &= (\text{inputRed} \times 0.393) + (\text{inputGreen} \times 0.769) + (\text{inputBlue} \times 0.189) \\
\text{outputGreen} &= (\text{inputRed} \times 0.349) + (\text{inputGreen} \times 0.686) + (\text{inputBlue} \times 0.168) \\
\text{outputBlue} &= (\text{inputRed} \times 0.272) + (\text{inputGreen} \times 0.534) + (\text{inputBlue} \times 0.131)
\end{align*}
\]

The effect is toggled when the user presses the ‘t’ key.

(20%) Motion Blur
Implement the horizontal motion blur. A blur means to compute a weighted average of myself with my neighboring pixels. Assuming a horizontal motion blur for this assignment; however the blur should be dependent on camera to object distance. Use the depth buffer to blur closer objects more than farther objects, as is intuitive. Motion blur is toggled when the user presses the ‘m’ key.

(30%) Swirl Distortion
Implement the swirl distortion effect on the texture. A swirl means pixels within some radius of a center rotate by some amount about the center. You choose the center, angle and radius values for the effect. They could be defaults or random values. The effect is toggled when the user presses the ‘d’ key.

(30%) Sketch filter
Implement a sketch-like effect on the rendered image – use your creativity to create a “sketch like effect”. One basic approach is to find edges and render them as line segments and the rest of
the scene very washed out. Two traditional mechanisms for edge detection are global threshold and adaptive threshold. In global threshold the same range comparison is applied to the entire image while on adaptive threshold the range comparison depends on the values of the neighborhood of the current pixel. You choose which mechanism to use. Intensity, thresholds and edge detection methods are also up to you. For example, you can use [-1 -1 -1; -1 8 -1; -1 -1 -1] as the edge detection kernel (see https://en.wikipedia.org/wiki/Kernel_(image_processing)).

Further, you could use the depth buffer to alter the sketch effect in some way; for example, far objects can be more washed out and with softer edges – near objects should be sharp and with thick edges. You may use additional or alternative depth-based sketching logic but the result should be intuitive. The effect is toggled when the user presses the ‘s’ key.

NOTE: only one technique needs to enabled at a time, but you can support one technique on top of the other if you like, but the program should not crash in any case.

Extra credits 1: Mouse interaction (+5 points if applied successfully)
Use the mouse location as the center of the effects in the Swirl distortion. Also, use the mouse location for defining the motion vector in the Motion blur. The motion vector is defined by the origin of the texture (bottom left corner) and the mouse location.

Extra credits 2: Oil Painting (+10 points if applied successfully)
Manipulate the colors of the texture and generate an oil painting effect on all rendered images. You choose the radius of the painting strokes. The effect is selected when the user press the ‘o’ key.

Grading:
Your program will be tested against the aforementioned functionality and your code will be inspected – PLEASE DO NOT COPY CODE FROM ANY SOURCE!

NOTE: you must turn-in a precompiled executable of your program!

Extra credit evaluations will be done by inspecting the execution of the program and the amount of credit you get depends on the clarity and effectiveness of your extra credit. If you implement the extra credit, please ensure instructions are given on how to use it --- put such in the GUI or have clear instructions printout – do not assume we will read your code and decipher how to use your extra credit.

If you have more questions, please see myself or the TA.
Good luck, but most important have fun!