Program 1 - Non-photorealistic Rendering

You must work individually on this assignment. To receive credit, turn in all required materials by February 25th.

Summary

Non-photorealistic rendering is an area of computer graphics that involves creating stylized digital art. It ranges from cartoon rendering to making painting-like imagery. In this assignment you will be writing a program that takes an image as input and allows the user to turn the image into a painterly version of it. The user will do so by clicking and dragging on the window which will draw OpenGL primitives which inherit the color of the corresponding pixel in the image. Here are some examples of photos and what your program may be able to generate from them.

Required Materials

Your program1 directory must include:

- All source code for the completed program
- A photo to be used as input to the program
- A screenshot of your program displaying a non-photorealistic version of your photo
- A readme text file with any necessary instructions for using your program, along with a write up on why you chose the provided image

Detailed Requirements

Your program must

Be an original program written by you. You may use code from labs as a starting point. You may talk with other students about the program, but looking at their code is not allowed.

Accept an image as input to the program. At a minimum, your program must accept an image path as a command line argument. Feel free to input an image in other ways (a file browser dialog for example). Be sure to document any other input methods in your readme file.

Respond to mouse click and drag events. When the mouse is clicked or dragged your program must place a shape made up of OpenGL primitives at that location on the screen.

Be able to draw different kinds of shapes. Think of the drawn shape as a brush stroke. Different brushes will draw different shapes and all prior strokes must stay the same. You must develop suitable data structures to store and display these shapes. At a minimum, you must be able to draw squares and circles. You are encouraged to include other shapes and to implement interesting features such as orienting the brush to the direction of the mouse stroke or creating dynamically generated shapes. Responding to keyboard events is probably the easiest way to allow the user to change what shape is being drawn (pressing ‘s’ to start drawing squares or ‘c’ to start drawing circles for example, or cycling through different available shapes by pressing ‘s’). You may use UI elements such as buttons or menus as well. It is up to you to decide how the user is able to change the shape, but be sure to document how it is done in your readme file.

Be able to change the size of shapes. Changing the size of the current brush can’t affect any shapes that have already been drawn. At a minimum, you must implement at least two different sizes: small and large. You are encouraged to implement finer granularity to your brush sizes (perhaps with a slider or by responding to mouse wheel events). You may also implement a way to adjust the size of previously drawn shapes, but there must be a way to only change the size of new shapes. Be sure to document how the user is able to change shape sizes in your readme file.

Be able to assign colors to the shapes. At a minimum, your program must be able to inherit the color of the corresponding pixel in the image. You are encouraged to implement a way to allow the user to specify a custom color. Be sure to document any extra color features you implement in your readme file.

Have a fill action. When the fill action is invoked, brush strokes are drawn regularly distributed across the entire image using the current size and shape options. You are encouraged to add variations to the fill action so it looks more interesting. Document how to invoke the fill action in your readme.

Allow resizing of the window. When the window is resized the drawn shapes must maintain the correct aspect ratio (no non-uniform scaling). You can decide what coordinate space you want to store your shapes in, but you must use an orthographic projection matrix to ensure proper scaling. You are encouraged to allow the shapes to be scaled up as the window size is increased. Think about how the aspect ratio of the image as well as the aspect ratio of the window will play a role in how you set up your orthographic projections (you'll likely have different rules depending on the relationship between the different aspect ratios).

Include a readme. The readme must include any special instructions on how to use your program. It must also include a writeup (about a paragraph or two long) about the image you chose to turn in as well as the painterly version of it. Why did you choose it? What about it is interesting? What do you like about it? You are encouraged to choose a photo that you composed yourself (whether you moved to capture a certain angle, or you actually placed things in certain positions), which will help when answering the questions above.

Point breakdown

35 - Correct position (20) and color behavior (15)
35 - Correct size (10), shape (10) and fill functionality (15)
10 - Correct resize functionality
20 - General (code style, execution, creativity, self expression and readme)
---
100 total points