Report, Project, TDT4195

Kim Rune Solstad, Bjørn Bråthen

November 18, 2013

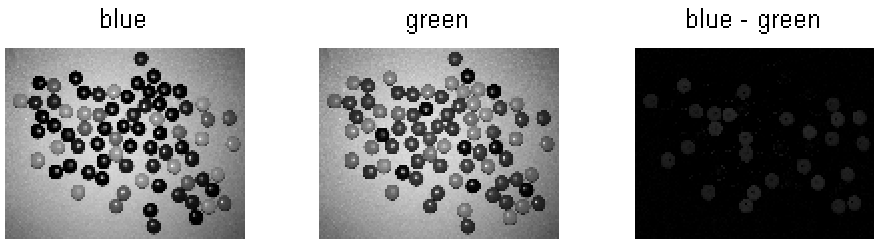
1 Abstract

This is the report for the main project in the course TDT4195. The assignment was to graphically represent a image with different coloured circular objects using computer-graphics and image-processing techniques. The task was solved in to parts, one image-processing part using Matlab, and one computer-graphics part using OpenGL. The Matlab-script writes a .txt file with coordinates, radius and color of the objects found. The OpenGL program then reads that file and represents the content as different coloured cubes in a three dimensional coordinate system. The programs are further explained in the sections below.

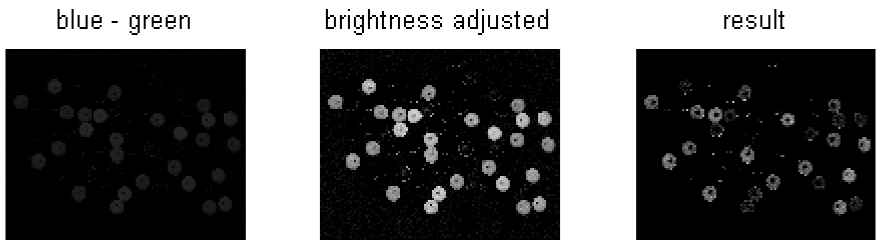
2 Matlab

First the SeparateColors.m function is used to filter out the different colors. It takes in a multi-color-image matrix, and returns six images. Each image contains only the non-stops of a certain color. The images are generated by subtracting the unwanted colors, and increasing the values of the wanted colors.

For example: The blue non-stops are separated by subtracting the green-image from the blue-image, thus removing green nonstops.



As the green and the blue images look quite alike, the resulting image is mostly black. To be able to further subtract colors from the image, the max values are increased. Red colors are then subtracted from the image.



Below is the code that generated the resulting image.

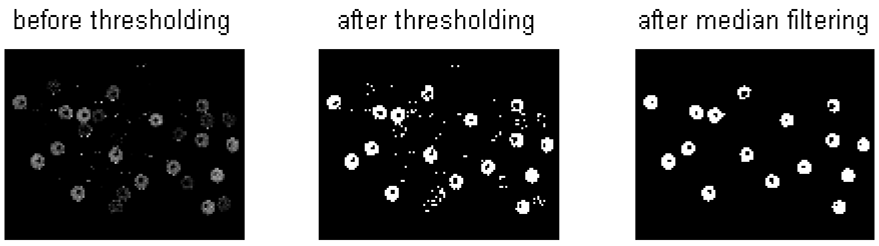
```
only_green = only_green.*(1/max_green);

only_blue = blueImg;
only_blue = only_blue - greenImg;
[max_blue, ~] = max(only_blue(:));
only_blue = only_blue.*(1/max_blue);
only_blue = only_blue - redImg;
```

The image created so far has the blue objects represented as the brightest circles. To remove the unwanted representation of other objects, thrsholding is used.

```
only_yellow = only_yellow.*(1/max_yellow);
```

To remove noise from the images, the images are median filtered. This is done by the function Median filter.

```
function [ output_args ] = medianFilter( img, filterSize )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
    [m, n] = size(img);
    output_args = zeros(m, n);
    l = floor(filterSize / 2);
    for x = 1 + l: n - l
        for y = 1 + l: m - l
            set = img(y - l:y + l, x - l: x + l);
            output_args(y, x) = median(set(:));
        end
    end
end
```



2.1 Improvements

A more adjustable but computational heavier approach to separate colors in the image, is by filtering. Each matrix would then have to be filtered, to extract the r, g and b values. An appropriate range would have to be set, for this method to work. For example: The blue non-stops in a image would be separated by filtering the values 0 to 50 from the red matrix, 0 to 50 from the green matrix and 205 to 255 from the blue image. A new matrix is then generated. The overlapping values are represented as 1.0 in the new matrix. The remaining values are set to 0.0.

3 OpenGL

The program created for this part of the assignment, is made by modifying code given to us for the graphics part of the course. The color-buffers and the vertex-buffer is created in the initialization phase. For each frame, RenderScene is called. The txt-file is read once for each object everytime RenderScene is called. The text-file is read using ReadFile.cpp. ReadFile uses the windows library conio.h to read the file. The read function used to get the values, requires an integer representing a line in the .txt as input. Parameters requiered to represent the object is extracted from that line, and placed in a array. A pointer to the first parameter is then returned.
By reading the txt-file each time the scene is rendered, the program is able to change the scene while running. This is done by modifying the txt-file. Each cube needs four parameters to be represented. One for color, one for radius, one for the x-coordinate and one for the y-coordinate. The format in the exchange file looks like this:

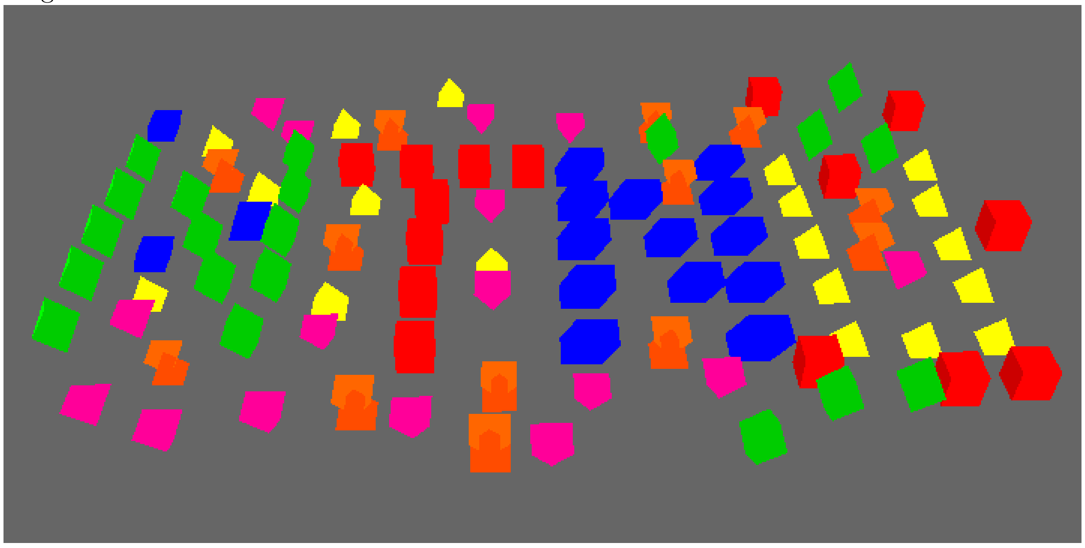
```
11103000076
11102360159
```

The fist line represents a shape with the following parameters:
color: red(1), radius=11, coord-x=300, coord-y=76.
After setting the values, a model-matrix for the shape is made. The model-matrix consists of an identity-matrix multiplyed with a translation-matrix, mul-

tiplyed with a scale-matrix. The model-matrix for the cube represents the radius and position of the object.

```
M = glm::mat4(1.0f);

M = glm::translate(M,glm::vec3(
        d[0 + X] / 100
        , 0.0,
        d[0 + Y] / 100));

M = glm::scale(M,glm::vec3(
        d[0 + S] * 3/ 100,
        d[0 + S] * 3/ 100,
        d[0 + S] * 3/ 100));
```

After multiplying the model-matrix with the mvp-matrix, a color-buffer and a vertex-buffer is bound before drawing the shape. The process is repeated until all objects are represented. below is a screenshot taken from the program running.



3.1 Improvements

Today, the cubes are given color by static pre-made color-buffers. To support all colors, one buffer for each different color given in the input-file should be made. A solution like that was considered, but not pursued. Lack of experience programming in the C-language, resulted in our program not being able to support dynamically made colorbuffers.