Conversion of RGB images to Intensity images

Niels de Waal (1698041), Jasper Smienk(1700502)

April 15, 2018

Contents

1 Target

The assignment of TICT-V2VISN1-13 at the HU, which we have chosen to undertake consists of two different components. The first one consists of writing a image shell which has to ability to extract or modify data from a given image. The second assignment is to convert an RGB image to an intensity/grey-scale image.

We will be mostly focusing on the second assignment. This is because we are interested in comparing a few different approaches to this problem.

For the second assignment we will be doing two measuring tests. The first will cover speed across multiple methods of grey-scaling. The second test will revolve around parallel execution of the grey-scaling process.

Both can be of interest. This could be because testing multiple methods of grey-scaling, including testing the speed at which this can be done could have effects when grey-scaling becomes an integral part of a system. This could happen in a real-time face recognition system, where grey-scaling has to be done every frame.

2 Methods

In this section we will discuss several methods of grey-scaling an image.

Let us refer to pixel i as part of image P, where $i \in \{0, \ldots, (P_{width} * P_{height}) - 1\}$. Every i has a set of (R, G, B) where R, G and B have a range $\{0, \ldots, 255\}$.

2.1 Methods for grey-scaling

In this sections a few methods for grey-scaling will be discussed. There are far too many methods to compare them all, thus in this article we will discuss these three:

- Averaging
- Luma
- Decomposition

2.1.1 Averaging

This will be considered as the naive approach to grey-scaling. Here we take the RGB values and using the average of them to calculate the grey-scale value. With the values: $\forall i \in P$ the grey-scale is $(i_R + i_G + i_B) \div 3$.

This technique has the advantage that it is very easy to implement and cheap to run. It has the disadvantage that it is not a very good at giving a good representation of a grey-scale image for human eyes.

2.1.2 Luma

The luma method fixes the problem that averaging has regarding the correctness for the human eye. Luma gives different priorities to different channels. The values for the different channels are described in [1]. With the values: $\forall i \in P$ the grey-scale is $i_R * 0.2126 + i_G * 0.7152 + i_B * 0.0722$.

With this method we could lose speed in comparison to averaging, because of the multiple floating point operations that have to be done each pixel. The upside is that this method delivers better images, because they are suited for the human eye.

2.1.3 Decomposition

Decomposition is the simplest method of grey-scaling. This method of grey-scaling has the advantage that it is very cheap to implement and run. The disadvantage is that, depending on the version used, it can result in a vastly lighter or darker image. With the values: $\forall i \in P$ the grey-scale is $\max(i_R, i_G, i_B)$ or $\min(i_R, i_G, i_B)$.

This method is not adjusted for the human eye.

2.2 Parallel execution of grey-scaling

Grey-scaling an image is often used in computer vision because it makes it easier for the algorithms to recognize edges. When this step has to be done in a real-time system, the amount of processing needed could add up to have a noticeable effect. With this section we explain the 2 possible methods of decreasing the workload caused by grey-scaling. Namely we chose to investigate these two methods:

- SIMD
- OpenCL

2.2.1 SIMD

SIMD or "Single Instruction, Multiple Data" describes the action of performing the same action on multiple data points. It can add parallelism to data by performing one instruction on multiple data points at once.

SIMD was introduced in the early 1970s by vector supercomputers, e.g [2]. Intel was the first to bring SIMD to consumer processors with MMX[3] on the x86 architecture.

The biggest advantage of SIMD comes when a pieces of data need to have the same operation applied to them using some constant and aren't depended on the outcome of the other data points. For example when 4 points on a 2D plane need to be shifted 4 positions, this can be efficiently done using SIMD.

SIMD could be of interest for grey-scaling because of it's parallelism on data points. This is especially of interest to us because all of the above mentioned methods for grey-scaling use only data from the same pixel. This means that, by using the XMM registers, we can fit multiple pixels in the same register.

2.2.2 OpenCL

OpenCL[4] or "Open computing language" is a framework for developing cross-platform applications that utilize more efficient resources on different devices. OpenCL gives us the ability to do a lot of the same things that SIMD can do, but do this on the GPU. The challenge with this approach lies in the fact that the data has to be transported to the GPU, and this operation could take more time than it is worth.

With OpenCL we can write a special kernel that will run on the GPU to do the computation of the grey-scale algorithms. This kernel can handle many datapoints at once, far more than SIMD can. The trade-off OpenCL has with SIMD has to do the transfer to the computation device. The transfer of memory form the host device (e.g the main memory) to the GPU will be considered the the main bottleneck of this approach and will be tested.

3 Choice

In this section we will shortly discuss why we chose the afore mentioned methods for grey-scaling and parallel execution.

3.1 Grey-scaling

There are any number of different methods for grey-scaling. Far too many to be explored in this assignment. That is why we will examine the three methods mentioned above. We chose these methods of grey-scaling for the following reasons:

- Simple
- All methods operate on a per pixel basis
- Because of the per pixel operations, the methods have a high chance of being run in parallel
- Although the operations are similar they still differ enough that a difference should me measurable

3.2 Parallel execution

There are multiple techniques available other than the afore mentioned two. For example we could have chosen to also include multi-threading in our examination, we didn't chose this particular example because from a certain point multi-threading isn't actually true parallel any more. SIMD has this same problem (e.g when all the registers are filled) but this ceiling is higher.

4 Implementation

In this section we present the implementations of the different methods.

4.1 Grey-scaling

What follows are the different methods of grey-scaling, implemented in the framework required for this assignment.

The average method:

```
IntensityImage * StudentPreProcessing::stepToIntensityImage(const
    RGBImage &image) const {
       auto* intensityImage = new
           IntensityImageStudent(image.getWidth(), image.getHeight());

       for (int i = 0; i < (image.getWidth() * image.getHeight()); i++)
           {

              //Average
              intensityImage->setPixel(i, ((image.getPixel(i).r +
                  image.getPixel(i).g + image.getPixel(i).b) / 3));

       }
       return intensityImage;
}
```

The luma method:

```
IntensityImage * StudentPreProcessing::stepToIntensityImage(const
    RGBImage &image) const {
       auto* intensityImage = new
           IntensityImageStudent(image.getWidth(), image.getHeight());

       for (int i = 0; i < (image.getWidth() * image.getHeight()); i++)
           {

              //Luma
              intensityImage->setPixel(i, (image.getPixel(i).r * 0.2126
                  + image.getPixel(i).g * 0.7152 + image.getPixel(i).b
                  * 0.0722));


       }
       return intensityImage;
}
```

The decomposition method:

```
IntensityImage * StudentPreProcessing::stepToIntensityImage(const
    RGBImage &image) const {
       auto* intensityImage = new
           IntensityImageStudent(image.getWidth(), image.getHeight());

       for (int i = 0; i < (image.getWidth() * image.getHeight()); i++)
           {

              //Decomposition; max
              //intensityImage->setPixel(i,
                  std::max({image.getPixel(i).r, image.getPixel(i).g,
                  image.getPixel(i).b}));

              //Decomposition; min
              intensityImage->setPixel(i,
                  std::min({image.getPixel(i).r, image.getPixel(i).g,
                  image.getPixel(i).b}));


       }
       return intensityImage;
}
```

5 Evaluation

References

[1] ITU-R, Recommendation ITU-R BT.709-6, http://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.709-6-201506-I!!PDF-E.pdf

[2] Texas Instruments, https://en.wikipedia.org/wiki/TI_Advanced_Scientific_Computer

[3] Intel, Pentium processor with MMX, https://www.intel.com/content/www/us/en/intelligent-systems/previous-generation/embedded-pentium-mmx.html

[4] khronos, OpenCL home page, https://www.khronos.org/opencl/