

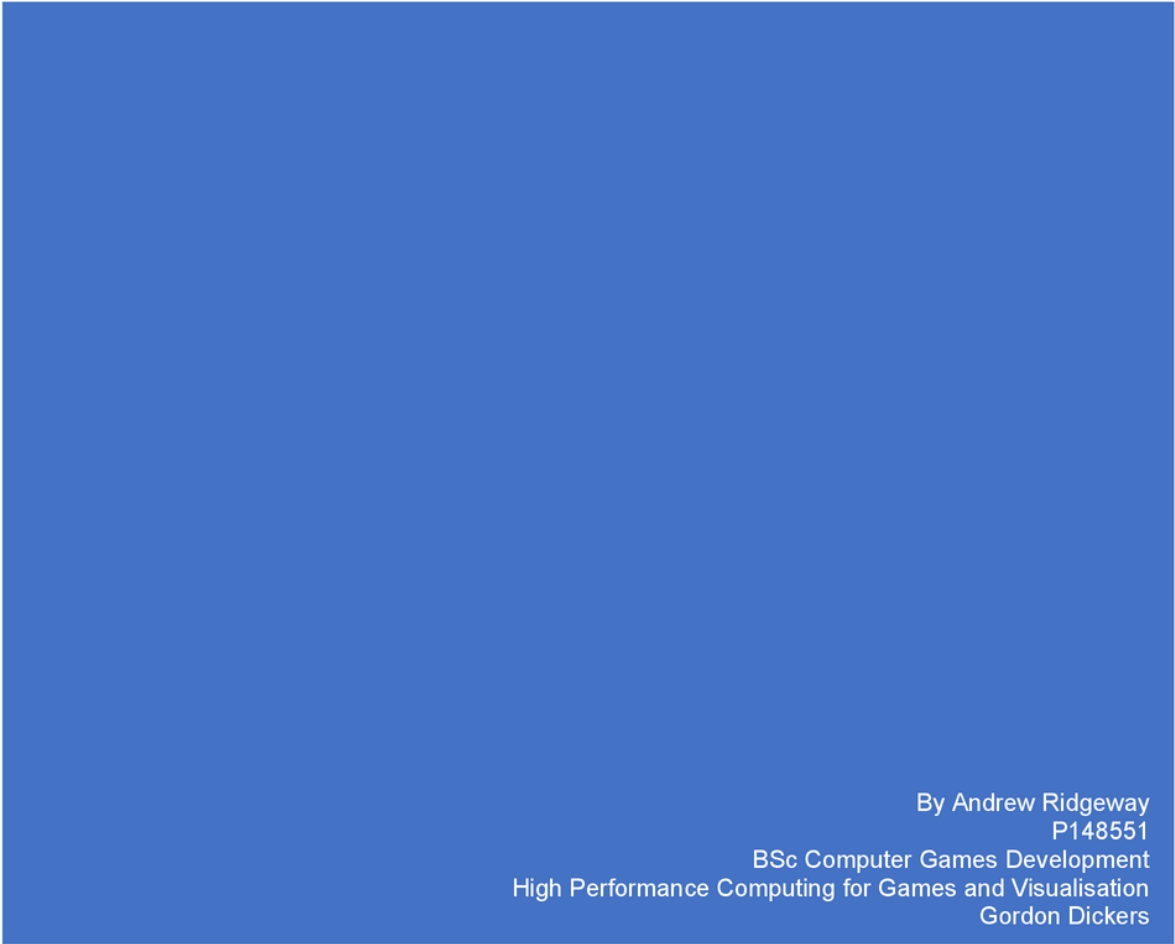
Image Threshold by P148551

by Andrew Ridgeway

FILE	4150_ANDREW_RIDGEWAY_IMAGE_THRESHOLD_BY_P148551_107698_1715366532.DOCX (866.38K)		
TIME SUBMITTED	08-MAY-2019 08:13PM (UTC+0100)	WORD COUNT	5437
SUBMISSION ID	106690101	CHARACTER COUNT	28418



Image Threshold



By Andrew Ridgeway
P148551
BSc Computer Games Development
High Performance Computing for Games and Visualisation
Gordon Dickers

Table of Contents

Abstract.....	2
1 Introduction	2
2 Review	2
2.1 Introduction	2
2.2 Digital Image Processing.....	3
2.3 Image Threshold.....	4
2.4 Image segmentation.....	5
3 Algorithm	6
4 Parallelization of Algorithm.....	7
5 Implementation.....	7
6 Results and Evaluation	8
6.1 Results.....	8
6.2 Evaluation	9
7 Conclusions and Future Work.....	9
References	11

Table of Figures

FIGURE 1 - ANALOGUE CONVERTED TO DIGITAL DIAGRAM[8].....	4
FIGURE 2 - A DIAGRAM OF AN IMAGE BEING GRAYSCALE[4].....	5
FIGURE 3 - HERE IS A DIAGRAM OF IMAGE SEGMENTATION[14].....	6
FIGURE 4 - HERE IS A SCREENSHOT OF THE IMAGE IN COLOUR AND GREYSCALE.	10
FIGURE 5 - HERE IS A SCREENSHOT OF THE CUDA IMAGE IN COLOUR AND GREYSCALE.	10

Abstract

In the review section will be speaking about a brief information on digital image process How to use such as how it uses to enhancing image and Ferris things. Not only will be speaking about image processing it will be speaking about image segmentation and image threshold, there will be a section about algorithm Basically this section will be speaking about how the algorithm is going to run on the CPU, parallelization of algorithm will be speaking about how it's going to be parallelization within the cheap you to see how it's going to be front. Then it'll be implementation section, and this will be used to see which one it's better at rendering the image at the quickest on GPU or CPU.

1 Introduction

In this document will have details explaining about digital image processing, image thresholding, image segmentation, algorithm, parallelization of algorithm, implantation, results, evaluation, conclusion and future work. This information was explained how it is used with images such as digital image processing, image thresholding and image segmentation this will be explained in the document. Algorithm, parallelization of algorithm and implantation is used for the experiment for the images and it will explain with further detail in the document why it is used. Results, evaluation, conclusion and future work is used for information how results went well for this experiment and how it was evaluated such as how well it went or terrible, future work will explain what type of new techniques will be used to improve on this experiment, this will explain with further details in the documents below.

2 Review

2.1 Introduction

In this brief summary will be speaking about digital image process and how it use, such as it's used to enhance the quality or get useful information from the image[1][2]. Also speaking about image threshold how to use, such as it would be used to divide the images into pieces to allow the image into a foreground and background to be change[3][4]. Also speaking about image segmentation how it is used, such as it is used to make it easier and change the reputation for the image[3][5].

2.2 Digital Image Processing

When it comes to convert images to a digital form it would use a technique known as digital image processing and it is used to enhance the image quality or extract information from the image[1][2]. Also, an image is known as a two-dimensional function that would use the x and y coordinates for the spatial/plane and the amplitude of f that would use for any pair of the coordinates for (x, y) also known as the intensity or gray level there is use for the image at that point[6][7]. Since this means is in a two-dimensional array this means the ranges will be between 0 and 255 and here's an example of how this function works, each value could be a range of between 0 and 255 so it could be 140, 210 and 96 so this means each would represent an individual value for the pixels[7]. This is a type of signal processing also known as digital signal processing or DSP, this means it would use the signal processing operations to use a sequence of numbers that represent would continuous values in its domain for its frequency and this can be used with applications, such as digital image processing and other various applications[7][3].

When it comes to using signal processing for digital image processing it would use a technique such as the input would be an image and the output may be an image or that is associated with the image such as characteristics or features[1][2]. These days digital image process is one of currently improving technology and it is used within engineering and computer science to be used for its core research for discipline use, these are the following steps image processing is used such as[1][2][6]:

- The first step of image processing is to Import the image from a device what can take pictures such as cameras, smartphones and other various devices that is capable of taking pictures[1][6].
- The second step of image processing is to analyse and manipulating the image in this step it would include enhancing, data compression, noticing patterns and other various uses there is used with images[1][6].
- The third step of image processing is the output what this means it would get its results, such as altered image or reports that is analysed based on the image (looking at information of the image)[1][3].

Image processing has to types of methods there is used with images such as analogue and digital[1][6]:

- **Analogue image processing:** - This technique is used with hard copies such as printing out an image and use with photographs, these techniques are mostly used with photographers and people who want to print out various images such as posters[1][8].
- **Digital image processing:** - this technique is used with a computer to enhance a digital image or a useful information from it[6][8].



Figure 1 - Analogue converted to digital diagram[8]

In this diagram it shows how an analogue image process is converted to a digital image process that can be used on a computer to either enhance or gather data from it[8].

2.3 Image Threshold

Image processing is used to partitioning for an image, this would mean it would divide into parts for each individual pixel and this would allow the image into a foreground and background[3][4]. Image threshold has an analysing technique that is known as of image segmentation and this technique is used to isolate the objects, such as images converting grayscale and colour into binary and this means with this technique can be used with grayscale and colour images to allow it to create a binary image but image threshold is a lot more effective if it's used with images that has high levels of contrast[3][4][9].



Figure 2 - A diagram of an image being grayscale[4]

In this diagram it shows the colour image is being converted into a black and white image, it's doing it by using the contrast of the Colour image by converting the lighter colours into white and converting the darker colours into black, it's replacing each individual pixels within an image to create the black and white look of the image[4].

Threshold in has methods when it's dealing with an image as such it can calculate automatically the values within a threshold while using an iterative method, threshold can approximate the image to see the midpoint value as what level the threshold and this means it would use the threshold level to pick a midpoint value for the image, threshold can be determined the values by maximizing image or object and background, this means threshold and values can determine the image to maximize to set the values for the background and the objects within the image and there's other methods that thresholding has[10].

2.4 Image segmentation

Image segmentation is used in the process of partitioning this means dividing into pieces and it would put the digital image into multiple segments, this means it would be a set of pixels, but it can also mean super-pixels, the main purpose of image segmentation is to make it easier and change the reputation for the image to make it that manageable to analyse[3][5]. The image segmentation has the capable ability to be used for object, as such recognition is used to identify any different type of objects such as people animals' trees and other, occlusion is used for boundary estimation within the system such as motion or stereo, image compression is used to decrease the size of the digital image such as it would make the image take up less storage and image editing is used to allow the end-user to edit the images[11][12]. The image segmentation would cover the whole image in segments to get the entire image or it would extract from the image such as it would use edge detection, edge detection uses a mathematical method that is used to identify the points within the image and this would be used for brightness, sharpness, changes and discontinuities[13].

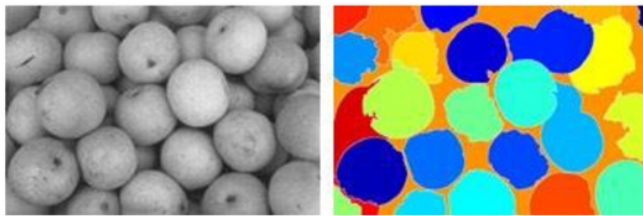


Figure 3 - Here is a diagram of image segmentation[14]

Here is the diagram showing how image segmentation is used with an image to alter it, such as by looking at the diagram on the Left the apples are in more of a greyscale while the image on the left has changed to multiple different colours by using image segmentation[14].

3 Algorithm

For the algorithm that is image threshold then will be used on an image while it's running on the CPU, to see the performance speed how quickly the CPU can convert a colour image to a black and white. This will be done by using Visual Studio and a library called glut, the main reason why Visual Studio will be used because it's a program that is more familiar for coding purposes, and

glut will be used because it's got the necessary library for the image to be shown and it can help with image thresholding[15][16]. This algorithm will be tested on several different computers such as a laptop, a home computer and the universities computers to see there is difference how it performs on different CPUs. The main reason will be tested on different computers to see if it has different results, such as these different computers could have different CPUs to see if it would run terrible or good on different types of computers because they may have different CPUs in them, for example some of these computers may have older or newer generations of CPUs that can affect the performance with the image convert into a black and white.

4 Parallelization of Algorithm

The next step is parallel the image thresholding in by using CUDA, this means it will be able to use the GPU also known as graphics processor unit and this will allow the code by using CUDA to run the thresholding[17][18]. Just like the CPU this algorithm will be tested on different computers as well to see if it gets different results but unlike the CPU it will be tested with different types of graphics card, such as how well it would run on these different types of graphics card to see what performance will differ prepared to the CPU. An example would be how quick the image will convert from its colour format to a black and white and this will compare the CPU to the graphics card to see which one is quick at converting a colour image to a black and white image. The type of graphics card that will be used for this algorithm will be NVIDIA brand graphics card because CUDA was developed by NVIDIA and this means AMD brand graphics card will not work, because since it was developed by NVIDIA and any computer won't uses any AMD graphics card will not work with CUDA[17].

5 Implementation

For the implementation is to see which is better at convert a colour image to a black and white image and this will be done by doing a speed test to see which

one gets the job done the quickest, by doing a speed test will this will also see which one is more efficient when it comes to converting colour to black and white. This algorithm will also be tested on different computers to see what results that will get, such as different computers might have different results with the speed test, for example the CPU may do better than the graphics cards when it comes to convert the image to a black and white or the graphics card may be better then the CPU.

6 Results and Evaluation

6.1 Results

Here is a testing table to see the result of which one would rendering the image the quick, such as preparing the GPU to the CPUs to see which one can render the image at the quickest time.

Gaming Laptop				
Test 1	CPU Speed	2.3 seconds of total execution time	GPU Speed	3.8 seconds of total execution time
Test 2	CPU Speed	3.3 seconds of total execution time	GPU Speed	2.1 seconds of total execution time
Test 3	CPU Speed	3.0 seconds of total execution time	GPU Speed	2.8 seconds of total execution time
Test 4	CPU Speed	2.3 seconds of total execution time	GPU Speed	4.9 seconds of total execution time
Gaming Computer				
Test 1	CPU Speed	2.8 seconds of total execution time	GPU Speed	3.4 seconds of total execution time
Test 2	CPU Speed	2.8 seconds of total execution time	GPU Speed	2.4 seconds of total execution time
Test 3	CPU Speed	2.9 seconds of total execution time	GPU Speed	1.9 seconds of total execution time
Test 4	CPU Speed	2.9 seconds of total execution time	GPU Speed	2.3 seconds of total execution time

By looking at these results comparing the speed of the CPU and GPU there is no difference when it comes to rendering the image, to make sure this test is fair it has the same image for the CPU and GPU to see which results come out quick. This test has been using in using to different devices such as a gaming laptop and a gaming computer, you are the specs of both systems that was used to test the CPU and GPU:

Gaming Laptop

ACER Predator Helios 300 15.6
Intel® Core™ i5-8300H Processor Quad-core 2.3 GHz / 4.0 GHz 8 MB cache
NVIDIA GeForce GTX 1060 6 GB GDDR5
8 GB DDR4 (2133 MHz)

Gaming Computer

Custom Build Gaming PC
Intel® Core™ 7-6700K Processor Quad-core 4.00 GHz / 4.20 GHz 8 MB cache
NVIDIA GeForce GTX 1070 8 GB GDDR5
16 GB DDR4 (2400 MHz)

6.2 Evaluation

While testing this program on 2 different devices just to see if it's any different with The CPU and GPU to see which ones the quickest but The CPU and GPU kept up well with each other without any problems to see which one can rendering the image the quickest, but both run that equally well. So, by doing this experiment it doesn't matter which this program runs on it just runs really well on both.

7 Conclusions and Future Work

7.1 Conclusions

While developing this program it was not enough time to put the image into black and white, but the image still greyscale so by looking at the screenshot below it shows the image on the left side that is in colour and on the right side that is in greyscale.

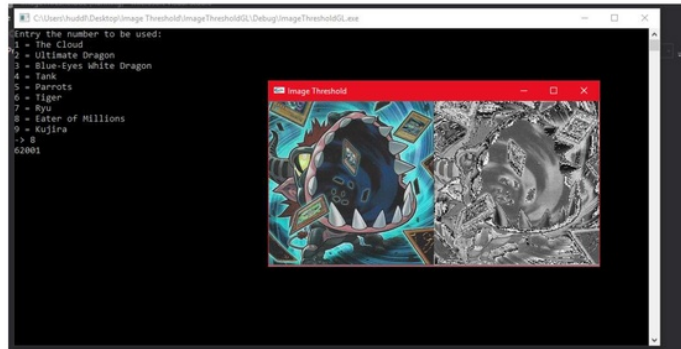


Figure 4 - Here is a screenshot of the image in colour and greyscale.

Another problem was developing this program was using CUDA for example CUDA would run the program but it won't show the image in on the left side in grayscale, but it still runs perfectly fine2 still to test on to get a result for which one can rendering a the image at the quickest time.

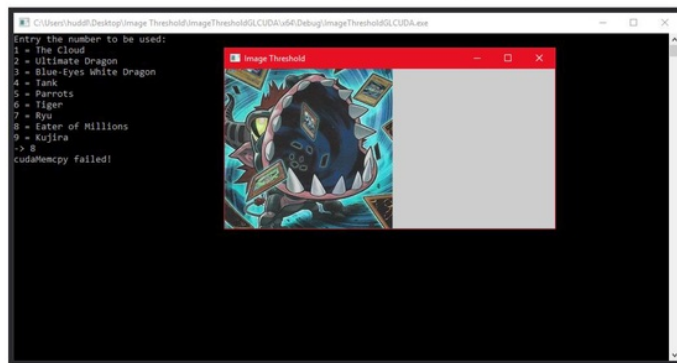


Figure 5 - Here is a screenshot of the CUDA image in colour and greyscale.

If there was a bit more time more work could be done to fix the CUDA problem and have another try print the image in thresholding to see it in black and white.

7.2 Future Work

Improvements will be made in the future such as improving the code to make able to threshold image then just grey scale in it to allow the image to be able to be seen in black and white, another future plan would be try get the image

to be in pop art to be able to mostly show different shades of one colour such as different shades of red. Another plan is once the CPU is done perfectly fine the next step is improving on the crew decide to make it run perfectly smooth on the GPU.

8 References

- [1] D. I. Processing, "Digital Image Processing," *Image Process.*, 2005.
- [2] Engineersgarage, "What is Image Processing : Tutorial with Introduction, Basics, Types & Applications," *EmgineersGarage*, 2012. [Online]. Available: <https://www.engineersgarage.com/articles/image-processing-tutorial-applications>. [Accessed: 20-Dec-2018].
- [3] S. W. Smith, *Digital Signal Processing - Second Edition*. 1999.
- [4] Mathworks.com, "Image Thresholding," *The MathWorks*, 2018. [Online]. Available: <https://www.mathworks.com/discovery/image-thresholding.html>. [Accessed: 26-Dec-2018].
- [5] "Image segmentation," pp. 1–27.
- [6] W. Gonzalez, Rafael C. Paul, *Digital Image Processing 2nd ed.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ©1987, 2002.
- [7] B. Mac Namee, "Digital Image Processing: Introduction," *Digital Image Processing: Introduction*, 2005. [Online]. Available: https://www.tutorialspoint.com/dip/image_processing_introduction.htm. [Accessed: 23-Dec-2018].
- [8] J. R. Levick, "Vascular smooth muscle," in *An Introduction to Cardiovascular Physiology*, no. i, 1991, pp. 171–177.
- [9] R. Gonzalez and R. Woods, "Point Operations - Thresholding." [Online]. Available: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/threshld.htm>. [Accessed: 26-Dec-2018].
- [10] WaveMetrics, "Image Threshold," *WaveMetrics*, 2018. [Online]. Available:

<https://www.wavemetrics.com/products/igorpro/imageprocessing/thresholding>.
[Accessed: 26-Dec-2018].

- [11] J. Fleet, A. D. Jepson D, "Image segmentation," pp. 1–27, 2007.
- [12] M. Rouse, "image recognition," *What/Is.com*, 2018. [Online]. Available: <https://searchenterpriseai.techtarget.com/definition/image-recognition>.
[Accessed: 27-Dec-2018].
- [13] S. Yuheng and Y. Hao, "Image Segmentation Algorithms Overview," 2017.
- [14] MathWorks, "Image Types - MATLAB & Simulink," *MathWorks*, 2018.
[Online]. Available: <https://uk.mathworks.com/discovery/image-segmentation.html>. [Accessed: 27-Dec-2018].
- [15] Visualstudio, "Visual Studio IDE, Code Editor, VSTS, & App Center - Visual Studio," *Microsoft*, 2019. [Online]. Available: <https://visualstudio.microsoft.com/>. [Accessed: 09-Jan-2019].
- [16] T. Opengl, U. Toolkit, O. U. Toolkit, and W. Glut, "GLUT - The OpenGL Utility Toolkit," *Source*, pp. 6–7, 2005.
- [17] G. Gerfin Sr, "Parallel Programming and Debugging with CUDA C."
- [18] N. Corporation, "CUDA Zone," *NVIDIA*, 2018. [Online]. Available: <https://developer.nvidia.com/cuda-zone>. [Accessed: 10-Jan-2019].

9 Appendices

CPU

Libraries.h
<pre>#pragma once #include <iostream> #include <fstream> #include "BimpTexture.h" #include "glut.h" using namespace std;</pre>
BimpTexture.h
<pre>#pragma once class BimpTexture {</pre>

```

private:
    char ch[2];
    int file, set, info, comp, padd, bit, dataSize, width, height, size, w;
    short rese1, rese2, count;
    unsigned char* bitData,* rgbData;

    void grayscaleColour(unsigned char*, int, int);

public:
    BimpTexture() {}
    ~BimpTexture();

    int theScreenX() const;
    int theScreenY() const;
    int theSizeCount() const;
    int theBitCount() const;

    const unsigned char* theData() const;
    const unsigned char* theRGBData() const;

    bool fileRead(const char*);
};

```

BimpTexture.cpp

```

#include "Libraries.h"
// this is used to delete the bit data once it's finished reading it
BimpTexture::~BimpTexture() {
    delete[] bitData;
    delete[] rgbData;
}

// this returns the width of The screen
int BimpTexture::theScreenX() const {
    return width;
}

// this returns the height of The screen
int BimpTexture::theScreenY() const {
    return height;
}

// this returns the data
int BimpTexture::theSizeCount() const {
    return dataSize;
}

// this returns the data
int BimpTexture::theBitCount() const {
    return bit;
}

// this returns the data
const unsigned char* BimpTexture::theData() const {
    return bitData;
}

// this returns the RGB data
const unsigned char* BimpTexture::theRGBData() const {
    return rgbData;
}

// this reads the file

```

```

bool BmpTexture::fileRead(const char* filename) {
    ifstream theFile;
    theFile.open(filename, ios::binary);

    // this is used to read all of the data within the image
    theFile.read(ch, 2);
    theFile.read((char*)&file, 4);
    theFile.read((char*)&rese1, 2);
    theFile.read((char*)&rese2, 2);
    theFile.read((char*)&set, 4);
    theFile.read((char*)&info, 4);
    theFile.read((char*)&width, 4);
    theFile.read((char*)&height, 4);
    theFile.read((char*)&count, 2);
    theFile.read((char*)&bit, 2);
    theFile.read((char*)&comp, 4);
    theFile.read((char*)&padd, 4);

    theFile.seekg(0, ios::end);
    file = theFile.tellg();

    int paddings = (4 - ((width * bit / 8) % 4)) % 4;
    int dataSize = width * height * bit / 8;

    padd = file - set;

    // this is used for the height and width on the
    // screen and used for the data and bits for the image
    this->width = width;
    this->height = height;
    this->bit = bit;
    this->dataSize = dataSize;

    bitData = new unsigned char[padd];
    rgbData = new unsigned char[dataSize];

    if (comp == 0) {
        theFile.seekg(set, ios::beg);
        theFile.read((char*)bitData, padd);
    }
    else if (comp == 1) {
        size = file - set;
        unsigned char *encData = new unsigned char[size];

        theFile.seekg(set, ios::beg);
        theFile.read((char*)encData, size);

        delete[] encData;
    }

    theFile.close();

    if (comp == 0 && paddings > 0) {
        w = width * bit / 8;
        for (int i = 1; i < height; ++i)
            memcpy(&bitData[i*w], &bitData[i*(w + paddings)], w);
    }

    memcpy(rgbData, bitData, dataSize);
    if (bit == 24 || bit == 32)
        granyscaleColour(rgbData, dataSize, bit / 8);
}

```

```

        return true;
    }

    // this is used for grayscale in colour
    void BimpTexture::grayscaleColour(unsigned char* data, int dataSize1, int chan) {
        unsigned char tmp;
        if (!data)
            return;
        if (chan > 3)
            return;
        if (dataSize1 % chan)
            return;

        for (int i = 0; i < dataSize1; i+=chan) {
            tmp = data[i];
            data[i] = data[i + 2];
            data[i + 2] = tmp;
        }
    }
}

```

Main.cpp

```

#include "Libraries.h"

// allow global variable to be called
BimpTexture bl;
GLenum texture, type;

const unsigned char* buf;
unsigned char* image;
int x, y, bit;

// this is use to run the central parts first before
//rest of the code such as display
void display(void), reshape(int, int), init(void);

// this code is used to convert a colour image into a grayscale image
void rgbGray(const unsigned char* in, int width, int height, unsigned char* out) {
    int size = width * height;
    const unsigned char* inPtr = in;
    unsigned char* outPtr = out;

    unsigned int tmp = 0;

    int i = 0;
    float A = 0.0, R = 0.3333, G = 0.3333, B = 0.3333;
    Glubyte pixel[4];

    int x = 0; x = width;
    int y = 0; y = height;

    // set the pixels RGB colours to convert
    int gray = (int)(R*pixel[0] + G*pixel[1] + B*pixel[2] / A*pixel[3]);

    // this is used to try get the values to be black and white
    if (gray > 128)
        gray = 255;
    else
        gray = 0;

    // this is used to convert the colour to grayscale
    while (i++ < size) {
        tmp = *in++ << x;
    }
}

```

```

        tmp += (*in << y) + *in++;
        tmp += *in++;
        *out++ = (unsigned char)(tmp >> gray);
    }

    cout << i << endl;
    in = inPtr;
    out = outPtr;
}

// this is used to help the image to be set up
void loadTex(int image1, char** image2) {
    const char *filename1 = "IMG.bmp", *filename2 = "dragon1.bmp", *filename3 =
"dragon2.bmp", *filename4 = "download.bmp", *filename5 = "parrots.bmp", *filename6 =
"tiger.bmp", *filename7 = "SF.bmp", *filename8 = "millions.bmp", *filename9 =
"dogoran.bmp";
    char choose;
    cout << "Entry the number to be used: " << endl << "1 = The Cloud" << endl <<
"2 = Ultimate Dragon" << endl << "3 = Blue-Eyes White Dragon" << endl << "4 = Tank"
<< endl << "5 = Parrots" << endl << "6 = Tiger" << endl << "7 = Ryu" << endl << "8 =
Eater of Millions" << endl << "9 = Kujira" << endl << "-> ";
    cin >> choose;
    const int num = 256;
    char file1[num];

    // allow the end user to pick an image
    if (choose == '1')
        strcpy(file1, filename1);
    else if (choose == '2')
        strcpy(file1, filename2);
    else if (choose == '3')
        strcpy(file1, filename3);
    else if (choose == '4')
        strcpy(file1, filename4);
    else if (choose == '5')
        strcpy(file1, filename5);
    else if (choose == '6')
        strcpy(file1, filename6);
    else if (choose == '7')
        strcpy(file1, filename7);
    else if (choose == '8')
        strcpy(file1, filename8);
    else if (choose == '9')
        strcpy(file1, filename9);

    if (!bl.fileRead(file1))
        return;

    x = bl.theScreenX();
    y = bl.theScreenY();
    buf = bl.theRGBData();
    // this is used to allow the image to be shown in grayscale
    type = GL_UNSIGNED_BYTE;

    // this is used for the RGB value
    // to allow the image to be shown in colour
    bit = bl.theBitCount();
    if (bit == 24)
        texture = GL_RGB;

    image = new unsigned char[x*y];
    rgbGray(buf, x, y, image);
}

```

```

}

// this is used to allow materials
void init() {
    glClearColor(0.5f, 0.5f, 0.5f, 0.0f);
    glShadeModel(GL_FLAT);
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
}

// display the shape to be shown while it's running
void display(void) {
    glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

    glRasterPos2i(0, y);
    glPixelZoom(1.0f, -1.0f);
    glDrawPixels(x, y, texture, type, buf); // this is used to help to
show the image and Colour
    glRasterPos2i(x, y);
    glDrawPixels(x, y, GL_LUMINANCE, GL_UNSIGNED_BYTE, image); // this is
used to help the image to show in grayscale
    glPixelZoom(1.0f, 1.0f);

    glutSwapBuffers();
}

// this is used to resize the view points for the window
void reshape(int w, int h) {
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, x*2, 0, y, -1, 1); // this is use for displaying the image
    glMatrixMode(GL_MODELVIEW);
}

// call the whole thing to work
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);

    glutInitWindowPosition(100, 100);
    glutInitWindowSize(500, 250);
    glutCreateWindow("Image Threshold");

    loadTex(argc, argv); // this is used to call loadText();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    init();

    glutMainLoop();
    return 0;
}

```

CUDA

Libraries.h

```

#pragma once
#include <iostream>
#include <fstream>
#include "BimpTexture.h"

```

```

#include "glut.h"

using namespace std;

BimpTexture.h

#pragma once
class BimpTexture {
private:
    char ch[2];
    int file, set, info, comp, padd, bit, dataSize, width, height, size, w;
    short rese1, rese2, count;
    unsigned char* bitData,* rgbData;

    void granyscaleColour(unsigned char*, int, int);

public:
    BimpTexture() {}
    ~BimpTexture();

    int theScreenX() const;
    int theScreenY() const;
    int theSizeCount() const;
    int theBitCount() const;

    const unsigned char* theData() const;
    const unsigned char* theRGBData() const;

    bool fileRead(const char*);
};

```

BimpTexture.cpp

```

#include "Libraries.h"
// this is used to delete the bit data once it's finished reading it
BimpTexture::~BimpTexture() {
    delete[] bitData;
    delete[] rgbData;
}

// this returns the width of The screen
int BimpTexture::theScreenX() const {
    return width;
}

// this returns the height of The screen
int BimpTexture::theScreenY() const {
    return height;
}

// this returns the data
int BimpTexture::theSizeCount() const {
    return dataSize;
}

// this returns the data
int BimpTexture::theBitCount() const {
    return bit;
}

// this returns the data
const unsigned char* BimpTexture::theData() const {
    return bitData;
}

```

```

}

// this returns the RGB data
const unsigned char* BimpTexture::theRGBData() const {
    return rgbData;
}

// this reads the file
bool BimpTexture::fileRead(const char* filename) {
    ifstream theFile;
    theFile.open(filename, ios::binary);

    // this is used to read all of the data within the image
    theFile.read(ch, 2);
    theFile.read((char*)&file, 4);
    theFile.read((char*)&rese1, 2);
    theFile.read((char*)&rese2, 2);
    theFile.read((char*)&set, 4);
    theFile.read((char*)&info, 4);
    theFile.read((char*)&width, 4);
    theFile.read((char*)&height, 4);
    theFile.read((char*)&count, 2);
    theFile.read((char*)&bit, 2);
    theFile.read((char*)&comp, 4);
    theFile.read((char*)&padd, 4);

    theFile.seekg(0, ios::end);
    file = theFile.tellg();

    int paddings = (4 - ((width * bit / 8) % 4)) % 4;
    int dataSize = width * height * bit / 8;

    padd = file - set;

    // this is used for the height and width on the
    // screen and used for the data and bits for the image
    this->width = width;
    this->height = height;
    this->bit = bit;
    this->dataSize = dataSize;

    bitData = new unsigned char[padd];
    rgbData = new unsigned char[dataSize];

    if (comp == 0) {
        theFile.seekg(set, ios::beg);
        theFile.read((char*)bitData, padd);
    }
    else if (comp == 1) {
        size = file - set;
        unsigned char *encData = new unsigned char[size];

        theFile.seekg(set, ios::beg);
        theFile.read((char*)encData, size);

        delete[] encData;
    }

    theFile.close();

    if (comp == 0 && paddings > 0) {
        w = width * bit / 8;
    }
}

```

```

        for (int i = 1; i < height; ++i)
            memcpy(&bitData[i*w], &bitData[i*(w + paddings)], w);
    }

    memcpy(rgbData, bitData, dataSize);
    if (bit == 24 || bit == 32)
        grayscaleColour(rgbData, dataSize, bit / 8);

    return true;
}

// this is used for grayscale in colour
void BimpTexture::grayscaleColour(unsigned char* data, int dataSize1, int chan) {
    unsigned char tmp;
    if (!data)
        return;
    if (chan > 3)
        return;
    if (dataSize1 % chan)
        return;

    for (int i = 0; i < dataSize1; i+=chan) {
        tmp = data[i];
        data[i] = data[i + 2];
        data[i + 2] = tmp;
    }
}

```

kernel.cu

```

#include "Libraries.h"
// allow global variable to be called
BimpTexture bl;
GLenum tex, type;

const unsigned char* buf;
unsigned char* image;
int x, y, bit;
const int WIDTH = 500;
const int HEIGHT = 250;

// this is use to run the central parts first before
//rest of the code such as display
void display(void), reshape(int, int), init(void);
cudaError_t addWithCuda(const unsigned char* c, int a, int b, unsigned char* o, int
size);

// this code is used to convert a colour image into a grayscale image
__global__ void rgbGray(const unsigned char* in, int width, int height, unsigned
char* out) {
    int size = width * height;
    const unsigned char* inPtr = in;
    unsigned char* outPtr = out;

    unsigned int tmp = 0;

    int i = 0;
    float A = 0.0, R = 0.3333, G = 0.3333, B = 0.3333;
    GLubyte pixel[4];

    int x = 0; x = width;

```

```

    int y = 0; y = height;

    // set the pixels RGB colours to convert
    int gray = (int)(R*pixel[0] + G*pixel[1] + B*pixel[2] / A*pixel[3]);

    // this is used to try get the values to be black and white
    if (gray > 128)
        gray = 255;
    else
        gray = 0;

    // this is used to convert the colour to grayscale
    while (i++ < size) {
        tmp = *in++ << x;
        tmp += (*in << y) + *in++;
        tmp += *in++;
        *out++ = (unsigned char)(tmp >> gray);
    }

    in = inPtr;
    out = outPtr;
}

// this is used to help the image to be set up
void loadTex(int image1, char** image2) {
    const char *filename1 = "IMG.bmp", *filename2 = "dragon1.bmp", *filename3 =
    "dragon2.bmp", *filename4 = "download.bmp", *filename5 = "parrots.bmp", *filename6 =
    "tiger.bmp", *filename7 = "SF.bmp", *filename8 = "millions.bmp", *filename9 =
    "dogoran.bmp";
    char choose;
    cout << "Entry the number to be used: " << endl << "1 = The Cloud" << endl <<
    "2 = Ultimate Dragon" << endl << "3 = Blue-Eyes White Dragon" << endl << "4 = Tank"
    << endl << "5 = Parrots" << endl << "6 = Tiger" << endl << "7 = Ryu" << endl << "8 =
    Eater of Millions" << endl << "9 = Kujira" << endl << "-> ";
    cin >> choose;
    const int num = 256;
    char file1[num];

    // allow the end user to pick an image
    if (choose == '1')
        strcpy(file1, filename1);
    else if (choose == '2')
        strcpy(file1, filename2);
    else if (choose == '3')
        strcpy(file1, filename3);
    else if (choose == '4')
        strcpy(file1, filename4);
    else if (choose == '5')
        strcpy(file1, filename5);
    else if (choose == '6')
        strcpy(file1, filename6);
    else if (choose == '7')
        strcpy(file1, filename7);
    else if (choose == '8')
        strcpy(file1, filename8);
    else if (choose == '9')
        strcpy(file1, filename9);

    if (!bl.fileRead(file1))
        return;

    x = bl.theScreenX();

```

```

        y = bl.theScreenY();
        buf = bl.theRGBData();
        // this is used to allow the image to be shown in grayscale
        type = GL_UNSIGNED_BYTE;

        // this is used for the RGB value
        // to allow the image to be shown in colour
        bit = bl.theBitCount();
        if (bit == 24)
            tex = GL_RGB;

        const int size = WIDTH * HEIGHT;

        image = new unsigned char[x*y];
        addWithCuda(buf, x, y, image, size);
    }

    // this is used to allow materials
    void init() {
        glClearColor(0.5f, 0.5f, 0.5f, 0.0f);
        glShadeModel(GL_FLAT);
        glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    }

    // display the shape to be shown while it's running
    void display(void) {
        glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

        glRasterPos2i(0, y);
        glPixelZoom(1.0f, -1.0f);
        glDrawPixels(x, y, tex, type, buf); // this is used to help to show the image
and Colour
        glRasterPos2i(x, y);
        glDrawPixels(x, y, GL_LUMINANCE, GL_UNSIGNED_BYTE, image); // this is used to
help the image to show in grayscale
        glPixelZoom(1.0f, 1.0f);

        glutSwapBuffers();
    }

    // this is used to resize the view points for the window
    void reshape(int w, int h) {
        glViewport(0, 0, (GLsizei)w, (GLsizei)h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0, x * 2, 0, y, -1, 1); // this is use for displaying the image
        glMatrixMode(GL_MODELVIEW);
    }

    int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);

        glutInitWindowPosition(100, 100);
        glutInitWindowSize(500, 250);
        glutCreateWindow("Image Threshold");

        loadTex(argc, argv); // this is used to call loadText();
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        init();
    }

```

```

        glutMainLoop();
        return 0;
    }

    // Helper function for using CUDA to add vectors in parallel.
    cudaError_t addWithCuda(const unsigned char* inc, int width, int height, unsigned
    char* out, int size) {
        const unsigned char* in;
        int width;
        int height;
        unsigned char* out;
        cudaError_t cudaStatus;

        // Choose which GPU to run on, change this on a multi-GPU system.
        cudaStatus = cudaSetDevice(0);
        if (cudaStatus != cudaSuccess) {
            fprintf(stderr, "cudaSetDevice failed! Do you have a CUDA-capable GPU
            installed?");
            goto Error;
        }

        // Allocate GPU buffers for three vectors (two input, one output)
        cudaStatus = cudaMalloc((void**)&in, size * sizeof(const unsigned char));
        if (cudaStatus != cudaSuccess) {
            fprintf(stderr, "cudaMalloc failed!");
            goto Error;
        }

        cudaStatus = cudaMalloc((void**)&out, size * sizeof(const unsigned char));
        if (cudaStatus != cudaSuccess) {
            fprintf(stderr, "cudaMalloc failed!");
            goto Error;
        }

        // Copy input vectors from host memory to GPU buffers.
        cudaStatus = cudaMemcpy(&in, inc, size * sizeof(const unsigned char),
        cudaMemcpyHostToDevice);
        if (cudaStatus != cudaSuccess) {
            fprintf(stderr, "cudaMemcpy failed!");
            goto Error;
        }

        cudaStatus = cudaMemcpy(out, out, size * sizeof(const unsigned char),
        cudaMemcpyHostToDevice);
        if (cudaStatus != cudaSuccess) {
            fprintf(stderr, "cudaMemcpy failed!");
            goto Error;
        }

        // Launch a kernel on the GPU with one thread for each element.
        rgbGray<<<32, size>>>(in, WIDTH, HEIGHT, out);

        // Check for any errors launching the kernel
        cudaStatus = cudaGetLastError();
        if (cudaStatus != cudaSuccess) {
            fprintf(stderr, "addKernel launch failed: %s\n",
            cudaGetErrorString(cudaStatus));
            goto Error;
        }

        // cudaDeviceSynchronize waits for the kernel to finish, and returns
        // any errors encountered during the launch.

```

```
    cudaStatus = cudaDeviceSynchronize();
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaDeviceSynchronize returned error code %d after
launching addKernel!\n", cudaStatus);
        goto Error;
    }

Error:
    cudaFree(&in);
    /*cudaFree(WIDTH);
    cudaFree(HEIGHT);*/
    cudaFree(out);

    return cudaStatus;
}
```

Image Threshold by P148551

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

39/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

GRADING FORM: SMU GENERIC_GJ

ANDREW RIDGEWAY

39.25

RESEARCH-INFORMED LITERATURE



Refs included but several are on-line. Thresholding is covered in detail in several good quality text on image processing.

4.75

KNOWLEDGE AND UNDERSTANDING



A lot of space in your lit review is taken up with a detailed discussion of the basics of image processing - you could concentrated more on the detail of relevant thresh-holding algorithms and how your chosen method. Also, as you are not focusing on segmentation here then avoid including a separate section on it.

5.0

ANALYSIS



The choices you made and the arguments for these choices need to be explicitly discussed here. Explain which aspects of the algorithm you chose are suitable for parallelising and describe how you are going to do this on CPU and GPU.

3.5

PRACTICAL APPLICATION AND DEP.



You have produced some timing results for GPU and CPU but your GPU algorithm appears to be flawed (as you point out in your discussion). Your results identify a number of tests but these are not described at all. Evaluation is superficial.

22.75

SKILLS FOR PROFESSIONAL PRAC.



Report is structured, abstract and refs present with table of contents and figures.

3.25