Variable Bit Image

A Project by

Cuthbert Allan Y. Guerrero
Dennis Alvin J. Manay
Anela Rosario Z. Ramos

Submitted to

Luisito L. Agustin
Instructor, CE 101

In Partial Fulfillment of the Requirements for the Course
CE 101: Techniques in Signal Processing

Department of Electronics, Computer and Communications Engineering
School of Science and Engineering
Loyola Schools
Ateneo de Manila University
Quezon City, Philippines

February 2011

Abstract

VarBit is an image processing application that allows a user to manipulate the bit lengths of the RGB color channels in an image. Users may select an image which they can then alter by changing the RGB bit lengths in order to change the appearance and size of the image. The application then saves this image using its own custom image file (*.vbi) which can be read by the application. The program can also save images to other formats (*.jpeg, *.png, *.bmp) if no bit manipulation is required.

The wxDevC++ software (build 7.3.1.3) was used in creating the program and graphic user interface (GUI) because it is ideal for image processing applications which requires the handling of different image formats.

Acknowledgments

The group would like to thank:

Dr. Luisito L. Agustin for his guidance and patience in seeing the project to its completion.

Former ELC152 students, specifically the Image Stitching group composed of Mara Duterte, Via Mateo, and Rachel Nayve, whose project aided in the creation and completion of the project.

Fellow CE101 students for their presence and support in class as well as for their input towards the improvement of the project.

Table of Contents

1. Introduction

1.1. Objectives

This project aims to create an application using wxDevC++ which will allow a user to manipulate the RGB bit lengths of an image. This requires the creation of a bitstream class to read and write the select number of bits. The program will then save this onto a custom image file format to allow the different bit lengths to be saved. This custom file format will hold the varied bit data of the image. The main objectives of the group is to then create a custom image file format that allows different bit lengths for the RGB color channels, create a bitstream class that reads and writes the custom image file format with varied bits, and to develop an application which will implement the conversion of these image files.

1.2. Scope and Limitation

The application makes use of various image formats such as JPEG, BMP, and PNG. It also makes use of its own image format (*.vbi) to allow for varied bit lengths. The application allows for file conversions between these formats. It allows the user to increase and decrease the number of bits in the color channels of the selected image.

However, the custom image file format can only be opened using the VarBit application. It cannot be read by other programs and applications. Also, once the bit length of a certain color channel is reduced, it can no longer be restored to its original quality.

1.3. Significance of the Study

The application of variable bit images is useful in data compression due to the ability of the user to manipulate the bit lengths. This allows the user to lessen (or increase) the bit lengths of the color channels which in turn, affects the quality and size of the image.

It also has its uses in image manipulation wherein the user can enhance certain RGB levels as desired. This is applicable in many fields such as digital media and others that require image processing.

At its most basic level, the application can be used as a file converter, converting an image from one file format to another. This is important should image format compatibility be an issue for a certain program.

The success of the group in creating a custom bitstream class is significant because it allowed for the variable bit image to be implemented.

1.4. Introduction

VarBit is an image processing application that allows the manipulation of the bit lengths of the RGB color channels in an image. The application provides the user with the option to vary the bit lengths of each color channel and to save this using a custom image file format (*.vbi). This is done by loading an image and saving it as a *.vbi file format. This opens a dialogue box which allows the user to set the desired RGB values (from 0-8).

Should the user not want to change the RGB values, they have the option of simply saving the image in another file format (*.jpeg, *.bmp, *.png).

2. Theoretical Framework

2.1. Variable Bit Length

Each image has a specific RGB value which corresponds to the “quality” of the colors represented in an image. In most images, the R, G, and the B (Red, Green, and Blue respectively) have an 8 bit value totaling of 24 bits per image. The ability to change the bit length of the image gives the user control over the quality (specifically brightness) of the image. The user is able to control whether what color he/she wants to bring out.

2.1.1 VBI file format

The *.vbi format is a custom image file format, where the RGB channels can be differed from values of 0 to 8 bits. A *.vbi image consists of the uncompressed image file and extra 40 bytes, which “tells” the VarBit program that it is a *.vbi image file. The extra 40 bytes consists of “vbigimage” as the first 9 bytes, a number “26”, 4 bytes each for the following: the width, the height, the red channel, the green channel and the blue channel of the image. This first 33 bytes consists of the “header” of the *.vbi file format. The last 7 bytes consists of “enddata”. The “enddata” is just a checker whether there are no errors in writing the image file to the *.vbi file format.

2.1.2. What is a Bit Stream?

A bit stream is a series of bits when captured and stored in a computer storage medium, a computer file is formed. The group was able to create a custom bit stream class because the need for the writing and reading of custom number of bits used in the custom image file format (*.vbi).

2.1.3. Scaling Formula

The scaling formula used by the group is simply a logical bitwise shift operation, shifting the bits to right to scale down to the desired bit length. In using this method, part of the original data is lost and the original quality of the image can no longer be restored once one of its RGB color channels is scaled down.

3. Methodology

3.1. Background on Bit Stream Class and Custom Header File Format

The group researched on creating a custom Bit Stream Class. Using the knowledge we found inside some C++ forums, the group was able to successfully create a custom Bit Stream class. The group researched on creating a custom header file format for the custom image file format (*.vbi), and the group studied on *i*Stitch, a project of the previous ELC152 class namely Mara Duterte, Rachel Nayve and Via Mateo, using the codes in *i*Stitch for opening and saving image files.

3.2. Program Flow

The application consists of two parts: loading images, saving the image (either *.jpeg, *.png, *.bmp and *.vbi) and if the user chose to save in the *.vbi format a new dialog box will open. In the new dialog box, the user will be able to choose on the bit length of each RGB color (the user can choose between 0 to 8 bit for each color channel).

3.2.1. Main VarBit Interface

Upon opening the application, the program window will display a “Load” button, a “Save” button, and a scrolled window panel. Clicking the “Load” button will open a dialogue box where an image can be selected. This will then be loaded onto the scrolled window of the main application. The user then clicks on the “Save” button which will open another dialogue box where the user can select the desired file format.

3.2.1.1. Loading the Image

In loading an image, the “Load” button (*wxButton1)* is clicked which opens a *wxOpenFileDialog1* which is used to select an image file. The image is then displayed in the *wxScrolledWindow* which can only display static bitmaps. This requires the loaded image to be converted from *WxImage* to *WxBitmap*.

```
void varBitFrm::WxButton1Click(wxCommandEvent& event)
{
```

```
        int success = 1;

        wxFileDialog* WxOpenFileDialog1 = new wxFileDialog(this,wxT("Choose image
    file"),wxEmptyString,wxEmptyString,openWildcards,wxFD_OPEN|
    wxFD_FILE_MUST_EXIST|wxFD_CHANGE_DIR|wxCENTRE);
        if (WxOpenFileDialog1->ShowModal()==wxID_OK)
        {
            loadimage1 = WxOpenFileDialog1->GetFilename();
            if (loadimage1.Matches("*.vbi"))
            {
                success = vBitImage.loadImage(loadimage1);
                if (success)
                {
                    wxBitmap temp_img1(vBitImage.getImage(), -1);
                    image1 = temp_img1;
                }
            }
            else
            {
                wxImage imagetest(loadimage1,wxBITMAP_TYPE_ANY,-1);
                wxBitmap temp_img1(imagetest, -1);
                image1 = temp_img1;
            }
            if (success)
            {
                sb1 = new wxStaticBitmap(WxScrolledWindow1,-1,image1);
                image1Width = image1.GetWidth();
                image1Height = image1.GetHeight();
                WxScrolledWindow1-
>SetScrollbars(10,10,image1Width/10,image1Height/10);
                WxScrolledWindow1->Scroll(1,1);
                sb1->Show(true);
            }
        }
        else
           WxOpenFileDialog1 -> Close();

        temp1 = image1.ConvertToImage();

    }
```

3.2.1.2. Read Header (Custom Image File Format)

The VBitImage::readHeader() function reads the header file of the selected image to be used. If the readHeader reads a “vbitimage” in the header of the image file, this indicates that the image is in the custom file format (*.vbi). At the same time, the

readHeader reads the RGB values of the image file.

```
int VBitImage::readHeader()
{
    char vbit[9];
    char vbitimage[10] = "vbitimage";
    char end[3];
    char endheader[4] = "end";
    int success = 1;

    for (int i = 0; i < 9; i++) {
        vbit[i] = bs->ReadByte();
    }

    success = 1;
    for (int i = 0; (i < 9) && (success == 1); i++) {
        if (vbit[i] != vbitimage[i])
            success = 0;
    }

    if (success) {
        bs->ReadByte();

        _width = bs->ReadBits(32);
        _height = bs->ReadBits(32);
        _countR = bs->ReadBits(32);
        _countG = bs->ReadBits(32);
        _countB = bs->ReadBits(32);

        for (int j = 0; j < 3; j++) {
            end[j] = bs->ReadByte();
        }

        for (int j = 0; (j < 3) && (success == 1); j++) {
            if (end[j] != endheader[j])
                success = 0;
        }
    }
    return success;
}
```

3.2.1.3. Read Data

The VBitImage::readData() function reads each pixel of the selected image and reads the RGB value of the pixel. It then stores this data in variables *_countR, _countG,* and *_countB* which are passed to the BitStream method *ReadBits()*.

```
int VBitImage::readData()
{
    int r, g, b;
    int r2, g2, b2;
    char end[7];
    char enddata[8] = "enddata";
    int success;

    success = 1;
    for (int y = 0; y < _height; y++) {
        for (int x = 0; x < _width; x++) {
            r = bs->ReadBits(_countR);
            g = bs->ReadBits(_countG);
            b = bs->ReadBits(_countB);

            if (_countR < 8)
                r2 = r << (8 - _countR);
            else if (_countR > 8)
                r2 = r >> (_countR - 8);
            else
                r2 = r;
            if (_countG < 8)
                g2 = g << (8 - _countG);
            else if (_countG > 8)
                g2 = g >> (_countG - 8);
            else
                g2 = g;
            if (_countB < 8)
                b2 = b << (8 - _countB);
            else if (_countB > 8)
                b2 = b >> (_countB - 8);
            else
                b2 = b;
            image->SetRGB(x,y,r2,g2,b2);
        }
    }
    for (int i = 0; i < 7; i++) {
        end[i] = bs->ReadByte();
    }

    for (int i = 0; (i < 7) && (success == 1); i++) {
        if (end[i] != enddata[i])
            success = 0;
    }
    bs->stopRead();
    return success;
}
```

3.2.1.4. Write Data

The VBitImage::writeData() function writes the RGB values set by the user (using the Bit Selector Window) onto the image's header file. It writes “enddata” at the end of the data to be written. The function uses the method of the BitStream class *WriteBits() and WriteByte().*

```
void VBitImage::writeData()
{
    int r, g, b;
    int r2, g2, b2;
    for (int y = 0; y < _height; y++) {
        for (int x = 0; x < _width; x++) {
            r = image->GetRed(x,y);
            g = image->GetGreen(x,y);
            b = image->GetBlue(x,y);

            if (_countR < 8)
                r2 = r >> (8 - _countR);
            else if (_countR > 8)
                r2 = r << (_countR - 8);
            else
                r2 = r;
            if (_countG < 8)
                g2 = g >> (8 - _countG);
            else if (_countG > 8)
                g2 = g << (_countG - 8);
            else
                g2 = g;
            if (_countB < 8)
                b2 = b >> (8 - _countB);
            else if (_countB > 8)
                b2 = b << (_countB - 8);
            else
                b2 = b;

            bs->WriteBits(r2, _countR);
            bs->WriteBits(g2, _countG);
            bs->WriteBits(b2, _countB);
        }
    }
    bs->WriteByte('e');
    bs->WriteByte('n');
    bs->WriteByte('d');
    bs->WriteByte('d');
    bs->WriteByte('a');
    bs->WriteByte('t');
```

```
        bs->WriteByte('a');
        bs->stopWrite();
    }
```

3.2.1.5. Saving the Image

A *wxSaveFileDialog1* box will appear to let the user choose where to save the image. The user can save the image as a *.jpeg, *.bmp, *.png, or *.vbi file.

```
void varBitFrm::WxButton2Click(wxCommandEvent& event)
{
    wxFileDialog* WxSaveFileDialog1 = new wxFileDialog(this,wxT("Save as..."),
wxEmptyString,wxEmptyString, saveWildcards,wxSAVE|wxFD_CHANGE_DIR|wxCENTRE);
       if (WxSaveFileDialog1->ShowModal() == wxID_OK)
       {
        wxString filename = WxSaveFileDialog1->GetPath();

        if (filename.Matches("*.vbi")) {
            int R = 8;
            int G = 8;
            int B = 8;
            vBitImage.openImage(temp1);

            bitSelector = new BitSelector(this, -1, wxT("Select number of
bits..."), wxPoint(0,0), wxSize(123,265), wxDEFAULT_DIALOG_STYLE);
            if (bitSelector->ShowModal() == wxID_OK)
            {
                R = bitSelector->getR();
                G = bitSelector->getG();
                B = bitSelector->getB();
                vBitImage.setRGBCount(R,G,B);
                vBitImage.saveImage(filename);
            }

            wxFileOutputStream out1 ("SaveFileDialog.txt");
            wxTextOutputStream debugger (out1);
        }
        else {
            wxInitAllImageHandlers();
            temp1.SaveFile(WxSaveFileDialog1->GetPath());
        }
    }
    else
        WxSaveFileDialog1 -> Close();
}
```

If the chosen file format is *.vbi, a new *wxform* will prompt the user to select variable RGB values.

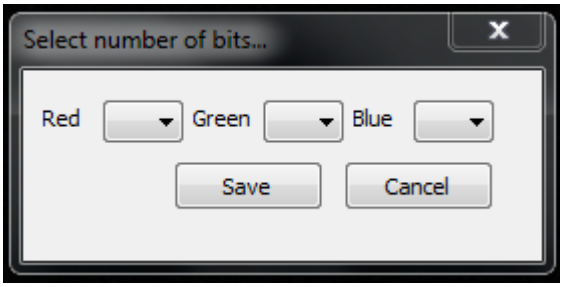

Figure 3.1 Bit Selector Window

3.2.1.6. Write Header

The VBitImage::writeHeader() function writes “vbitimage” onto the image's header file as an identifier for the custom file format of the application. It writes the height and width RGB values of the image and “end” after writing all the information necessary.

```
void VBitImage::writeHeader()
{
    bs->WriteByte('v');
    bs->WriteByte('b');
    bs->WriteByte('i');
    bs->WriteByte('t');
    bs->WriteByte('i');
    bs->WriteByte('m');
    bs->WriteByte('a');
    bs->WriteByte('g');
    bs->WriteByte('e');
    bs->WriteByte(26);

    bs->WriteBits(_width,32);
    bs->WriteBits(_height,32);
    bs->WriteBits(_countR, 32);
    bs->WriteBits(_countG, 32);
    bs->WriteBits(_countB, 32);
    bs->WriteByte('e');
    bs->WriteByte('n');
    bs->WriteByte('d');
}
```

4. Coder's Manual

4.1. Source and Header Files

This application consists of four source files, five header files, and two wxForm files. All of these files must be included in the root folder where the program is to be compiled.

4.1.1. VBitImage.cpp and VBitImage.h

The VBitImage class is used for opening and loading of images with the file type ".vbi". The .vbi custom image format does not have its own wxImageHandler, which necessitates the inclusion of methods that would allow the reconstruction of the image within the class. Essentially, the VBitImage class is just an extension of the wxImage class that allows support for the .vbi image format. Methods for loading and saving of images are included in the class.

Properties:

- int _countR
- int _countG
- int _countB
- int _height
- int _width
- BitStream *bs
- wxImage *imag

Methods:

- writeHeader()

  -This is a private method which allows the writing of the image header onto the file. The properties _countR, _countG, _countB, _height, and _width are stored on the file using the BitStream class. This is used by the saveImage method.

- writeData()

  -This is a private method which is used to store the image data onto the

file. The image data is read from the wxImage image, which is then stored on the file using the BitStream class. This is used by the saveImage method.

- readHeader()

  -This is a private method which reads the header data from the file. The header data is then stored on their respective property fields. The method returns a 1 if the data was successfully read. This is used by the loadImage method.

- readData()

  -This is a private method which reads the image data from the file. This method is called if the readHeader method returns a 1. The image data is then stored on the wxImage image. The method returns a 1 if the data was successfully read and stored on the wxImage. This is used by the loadImage method.

- loadImage

  -The loadImage method is an overloaded method which is used to load a .vbi image. The three variants of the method were designed with flexibility and multi-platform support in mind, by allowing the use of wxString, cstring, and null-terminated strings. This method utilizes the readHeader and readData methods.

- openImage

  -The openImage method allows the loading and storage of common image file formats onto the VBitImage class' wxImage image. This is used in conjunction with saving a file to a .vbi image.

- saveImage

  -The saveImage method is used to save to a .vbi image. This utilizes the writeHeader and writeData methods.

- getImage

  -This method returns the currently stored wxImage image on the VBitImage class. This is used for loading the image onto the screen.

- setRGBCount

  -This method sets the values of _countR, _countG, and _countB.

4.1.2. BitStream.cpp and BitStream.h

The BitStream class allows the reading and writing of data onto a file in values as small as a bit. It utilizes the fstream() both for the input file stream and the output file stream for reading and loading, respectively. The class includes methods for reading and writing in values of a bit, a byte, and n bits (n > 1 and n < 32).

4.1.3. varBitFrm.cpp and varBitFrm.h

The functions for the command buttons seen on the Graphical User Interface (GUI) of the application can be found on the varBitFrm.cpp and varBitFrm.h. The following are the member functions of the class varBitFrm.cpp

- varBitFrm()

  - This function calls the CreateGUIControls() function which has the following parameters:

  - wxWindow *parent
  - wxWindowID id
  - const wxString &title
  - const wxPoint &position
  - const wxSize& size
  - long style

- CreateGUIControls()

  - This function creates the elements of the graphical user interface of the application. These elements include image upload buttons, open file dialog boxes, scrolled windows, panels, and window application title, icon and position. It also includes a function which determines the various image formats that the application can handle.

- OnClose()

- This function basically closes the application by pressing the close button on the upper right-hand corner of the application.

- WxButton1Click()

  - This function is basically the load button. By pressing the load button, the application prompts the user to choose the image file he/she wants to use.

- WxButton2Click()

  - This function is another button used for saving images once the user is finished on the chosen image. Basically, it is the save button. Once pressed, the application will ask the user to give the final filename the user wants for the final image.

4.1.4. BitSelector.cpp and BitSelector.h

The functions for the Bit Selector form are found here, the GUI contains three drop down boxes and two buttons.

- WxButton1Click()

  - This function gets the value for each of the dropdown boxes found in the GUI. It saves the values stored inside the three drop down boxes for the desired bit length values of the three color channel.

- WxButton2Click()

  - This function is basically close/cancel the Bit Selector form.

- getR(), getG(), getB()

  - These functions gets and saves the value inside the drop down menusThe file *rgb.h* contains three structures: *RGBint*, *RGBdouble*, and *RGBpointer*. These structures are used to represent the RGB pixel values as a single variable. This way, instead of using three variables to represent the RGB values of a pixel, only one variable will be used.

4.2. WXFORM

The WXFORM file, contains the elements used in the graphical user interface.

5. User's Guide

5.1. CDROM Contents

- <CD root directory>\
- software\
- DSP pictures\
- Variable Bit\
  - Images\
  - Objects\
  - Output\
  - BitSelector.cpp
  - BitSelector.h
- BitSelector.wxform
  - BitStream.cpp
  - BitStream.h
  - varBit.dev
  - VBitImage.cpp
  - VBitImage.h
  - varBitFrm.cpp
- varBitFrm.h
  - varBitFrm.wxform
- doc\
- documentation.odt – *project documentation (ODT format)*
- documentation.pdf – *project documentation (PDF format)*
- progress_reports\ - *contains progress reports of the developers*
- cuthbert_guerrero\
- alvin_manay\
- anela_ramos\
- proposal\
- project_proposal.odt – *project proposal (ODT format)*

5.2. VarBit User Interface

The VarBit application has two buttons and a scrolled window. The two buttons, labeled as “Load” and “Save” respectively, are found above the scrolled window.

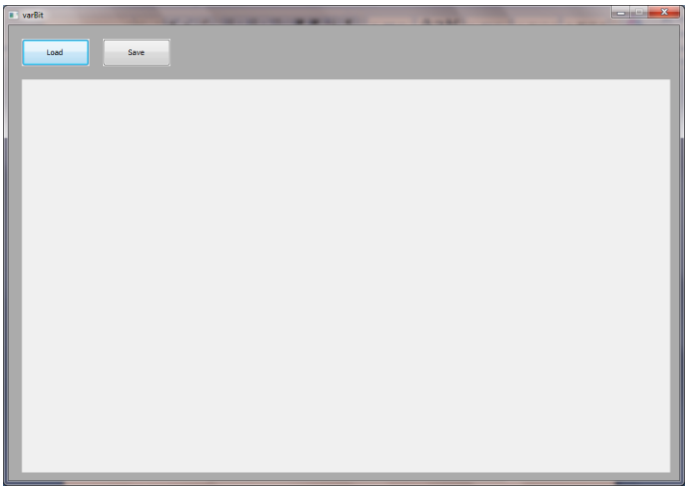

Figure 5.1 Main Window

5.2.1. Load

The “Load” button is used for loading the image selected by the user. The user can select an image file from a certain location to be displayed on the scrolled window.

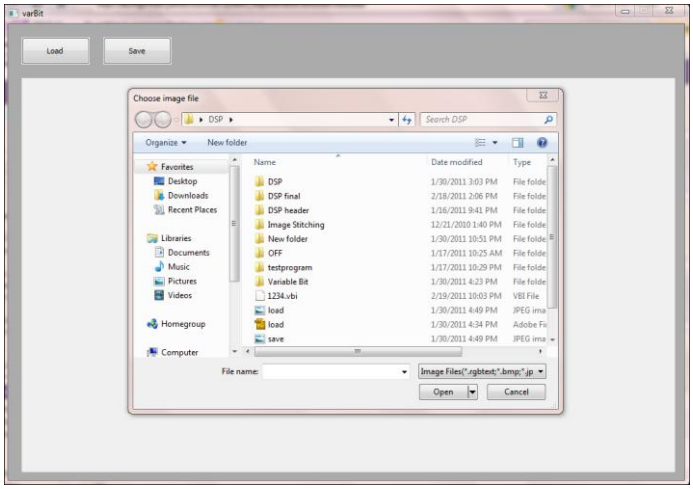

Figure 5.2 Choose Image File Dialogue Box

5.2.2. Display Panel

The scrolled window displays the selected image on the main application window. This allows the user to view the image selected.

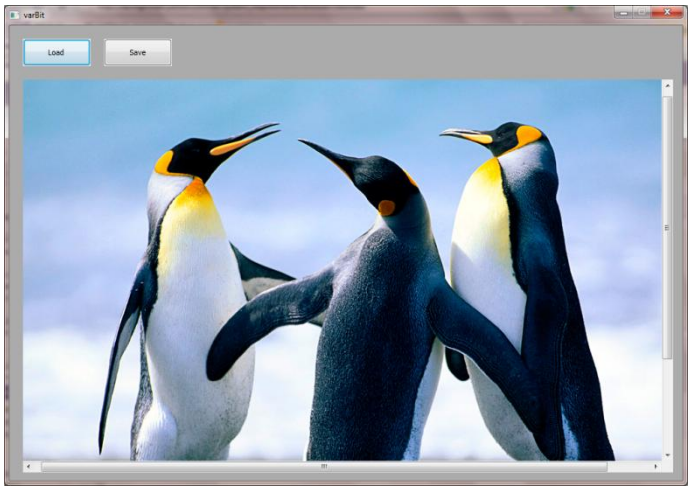

Figure 5.3 Display Image

5.2.3. Save

The “Save” button allows the user to save the image in various file formats. JPEG, PNG, BMP, and the custom file format *.vbi are possible file formats which the user can select from the dialogue box.

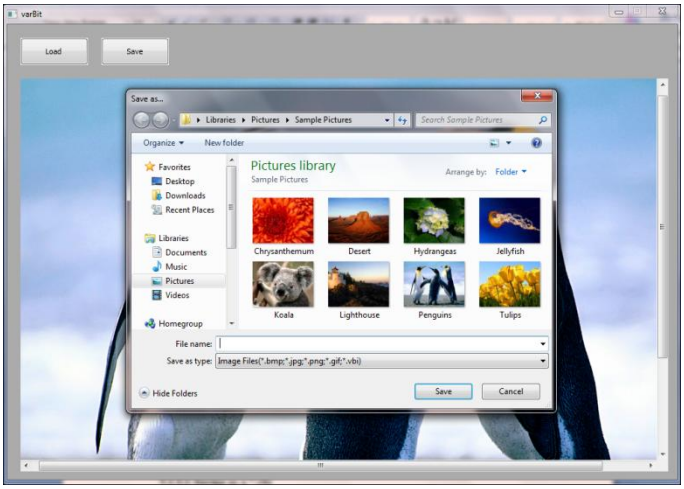

Figure 5.4 Save as Dialogue Box

5.2.3.1. Saving as a JPEG, BMP, and PNG

Saving as either a JPEG, BMP, or PNG image does not allow the user to manipulate the bit lengths of the RGB color channels. It saves an image from one file

format to another.

5.2.3.2. Saving as a *.vbi

Saving the image in the custom image file format (*.vbi) allows the user to vary the RGB values of the image. Upon selecting the *.vbi file format in the save dialogue box, a Bit Selector dialogue box appears wherein the user can select various RGB values.

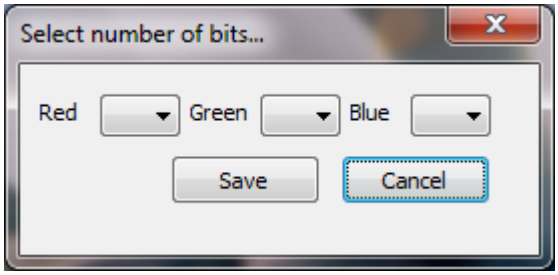

Figure 5.5 Bit Selector

6. Results

6.1. Converting images from JPEG to BMP to PNG

The VarBit application is able to load different image file formats. The application was able to convert an image file name balai.jpeg to various formats namely, BMP and PNG.

6.1.1. JPEG, BMP, and PNG images

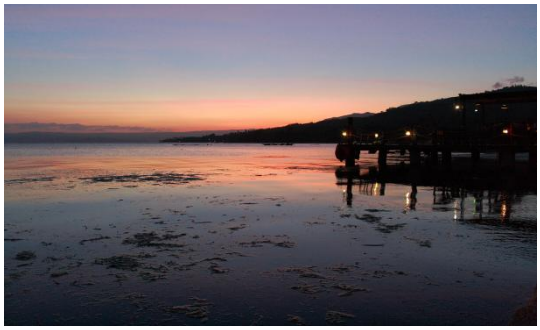

Figure 6.1. balai.jpeg

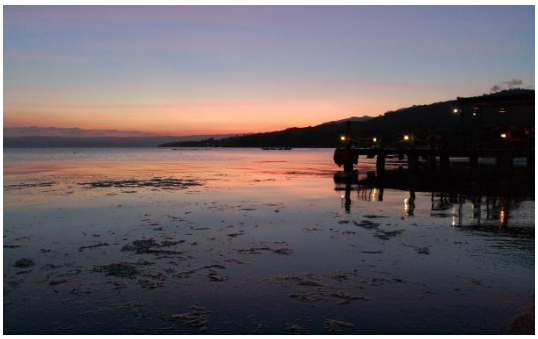

Figure 6.2 balai.bmp

Figure 6.3. balai.png

6.2. Saving as Varied Bit Length Image (*.vbi)

This is the custom image file format where the user can save an image with varying RGB bit lengths. The following are examples of balai.jpeg converted to VBI format.

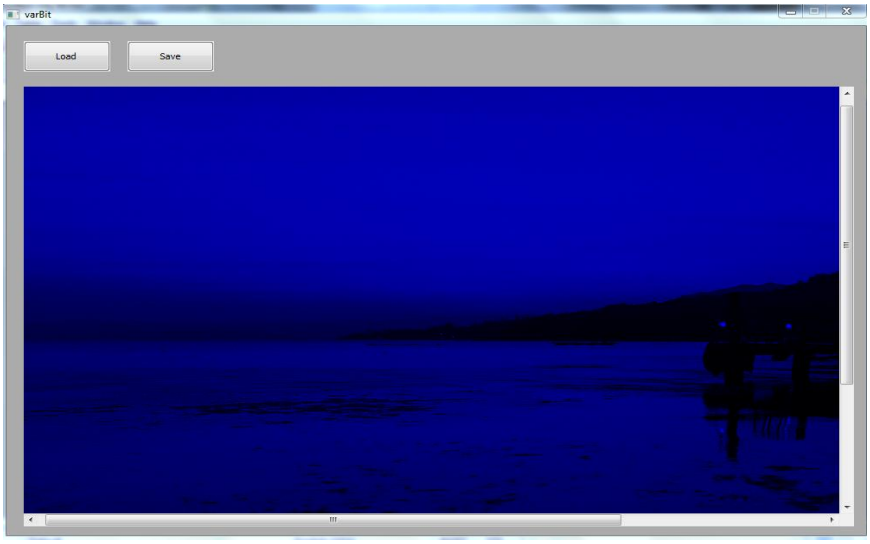

Figure 6.4. balai.vbi with R-0 G-0 B-8

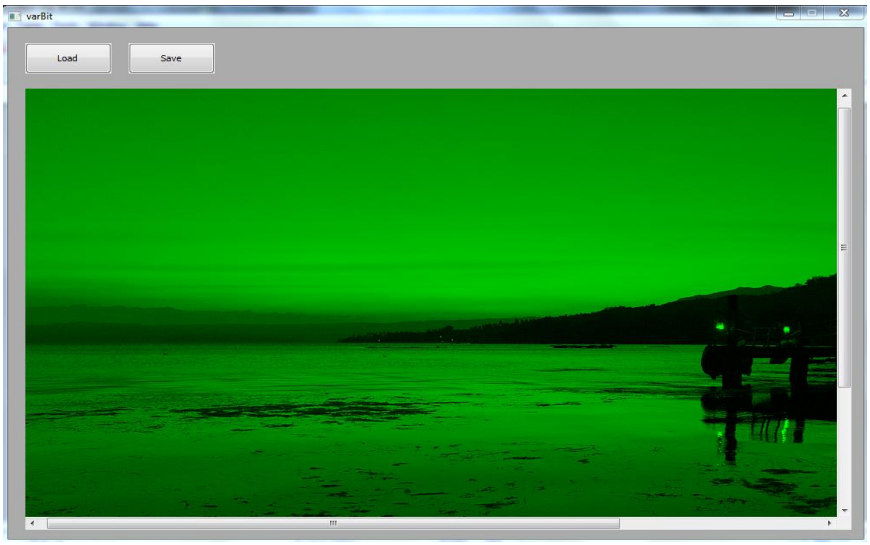

Figure 6.5 balai.vbi R-0 G-8 B-0

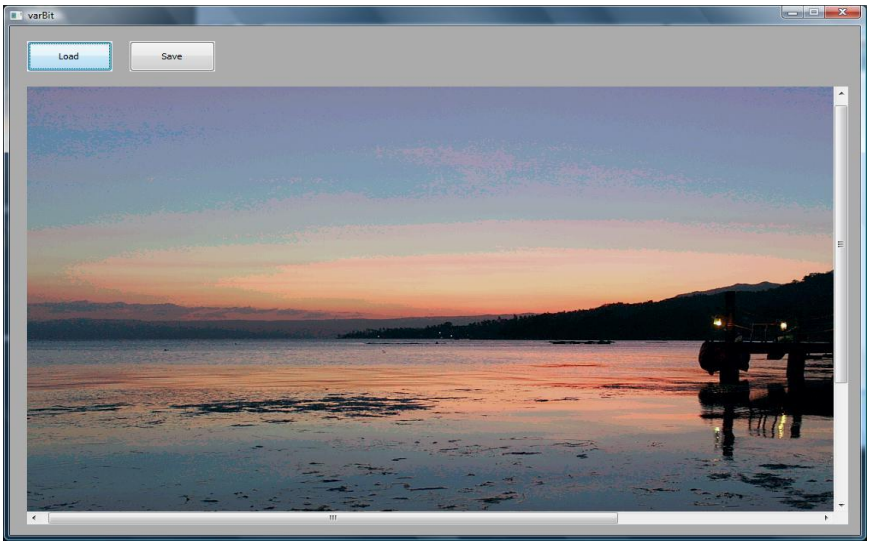

Figure 6.6 balai.vbi R-3 G-5 B-7

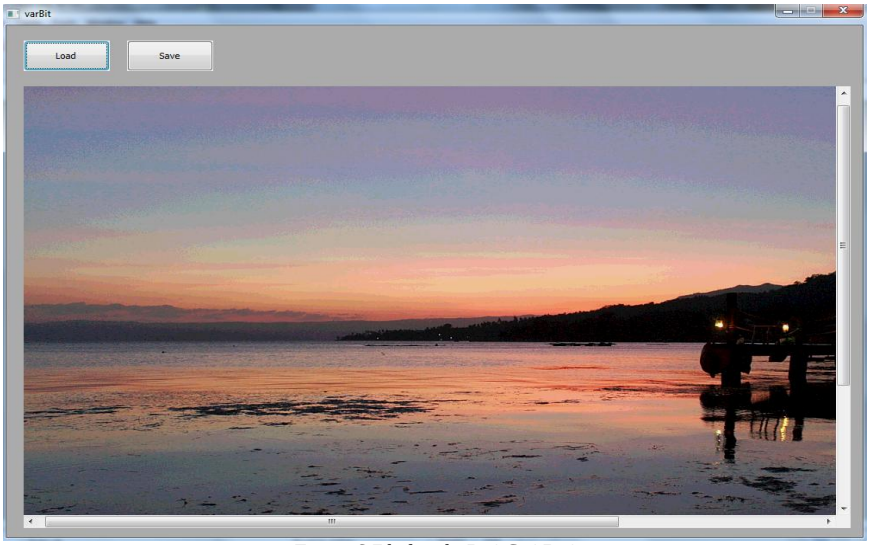

Figure 6.7 balai.vbi R-4 G-4 B-4

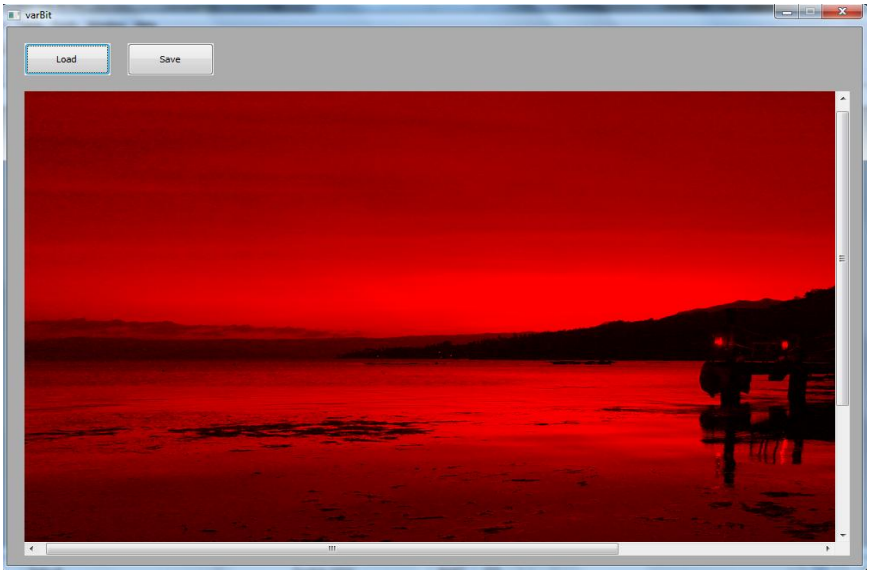

Figure 6.8 balai.vbi R-8 G-0 B-0

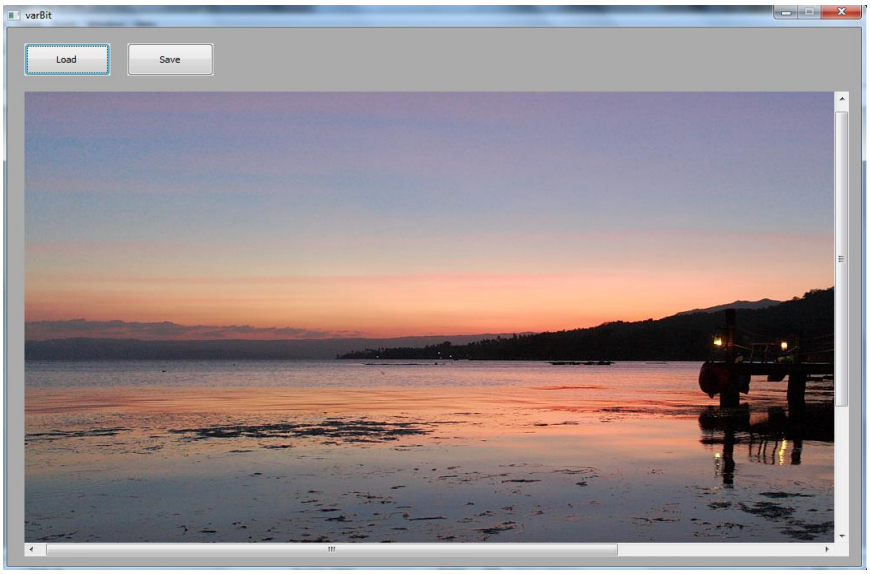

Figure 6.9 balai.vbi R-8 G-8 B-8

6.3. File Size Comparison

File Name: balai

Dimensions: 1280 x 800

Type	Number of Bits	Size
JPEG		1,087,218 bytes
PNG		1,643,543 bytes
BMP		3,072,054 bytes
VBI	R-8 G-8 B-8	3,072,040 bytes
VBI	R-4 G-4 B-4	1,536,040 bytes
VBI	R-0 G-0 B-8	1,024,040 bytes
VBI	R-0 G-8 B-0	1,024,040 bytes
VBI	R-8 G-0 B-0	1,024,040 bytes

Table 6.1 File Size Comparison

7. Conclusion and Recommendations

7.1. Conclusion

The group was able to successfully manipulate the bit lengths of the RGB color channels of an image using the custom header file and custom BitStream class. The application is able to increase and decrease the bit lengths of the color channels.

The applications is able to handle various file formats such as JPEG, PNG, BMP, and the custom file format (*.vbi).

7.2. Recommendations

For the next project, it is recommended that the application provides a preview window alongside the original image in order for the user to view the altered image before saving the file. An improved GUI would also be recommended.

Other recommendations could be the aesthetic of a slider bar instead of a drop-down menu in selecting the bit lengths of the color channels of the image. This gives the user more options as opposed to the set values of the drop-down menu.

Another thing to improve on would be the error notification when a corrupted image file is loaded onto the application. Currently, there are no error handlers for this situation.

Appendix A: Source Code

A.1. varBitFrm.h

```
#ifndef __VARBITFRM_H__
#define __VARBITFRM_H__
#ifdef __BORLANDC__
    #pragma hdrstop
#endif
#ifndef WX_PRECOMP
    #include <wx/wx.h>
    #include <wx/frame.h>
#else
    #include <wx/wxprec.h>
#endif
#include <wx/filedlg.h>
#include <wx/scrolwin.h>
#include <wx/button.h>
#undef varBitFrm_STYLE
#define varBitFrm_STYLE wxCAPTION | wxSYSTEM_MENU | wxMINIMIZE_BOX | wxCLOSE_BOX
#include "VBitImage.h"
#include "BitSelector.h"

class varBitFrm : public wxFrame
{
    private:
        DECLARE_EVENT_TABLE();

    public:
        varBitFrm(wxWindow *parent, wxWindowID id = 1, const wxString &title =
wxT("varBit"), const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, long style = varBitFrm_STYLE);
        virtual ~varBitFrm();
        void WxButton1Click(wxCommandEvent& event);
        void WxButton2Click(wxCommandEvent& event);
    private:
        wxFileDialog *WxOpenFileDialog1;
        wxFileDialog *WxSaveFileDialog1;
        wxButton *WxButton2;
        wxScrolledWindow *WxScrolledWindow1;
        wxButton *WxButton1;
        wxString loadimage1;
        wxStaticBitmap *sb1;
        wxBitmap image1;
        wxImage temp1;
        VBitImage vBitImage;
        int image1Width, image1Height;
        BitSelector *bitSelector;
        wxString openWildcards, saveWildcards;
    private:
        enum
        {
                ID_WXBUTTON2 = 1003,
                ID_WXSCROLLEDWINDOW1 = 1002,
                ID_WXBUTTON1 = 1001,
                ID_DUMMY_VALUE_
        };

    private:
        void OnClose(wxCloseEvent& event);
        void CreateGUIControls();
};

#endif
```

A.2. varBitFrm.cpp

```
#include "varBitFrm.h"
#include <wx/image.h>
#include <stdlib.h>
#include <iostream>
#include <wx/string.h>
#include "BitStream.h"
#include "VBitImage.h"
#include <wx/txtstrm.h>
#include <wx/wfstream.h>

BEGIN_EVENT_TABLE(varBitFrm,wxFrame)

    EVT_CLOSE(varBitFrm::OnClose)
    EVT_BUTTON(ID_WXBUTTON2,varBitFrm::WxButton2Click)
    EVT_BUTTON(ID_WXBUTTON1,varBitFrm::WxButton1Click)
END_EVENT_TABLE()

varBitFrm::varBitFrm(wxWindow *parent, wxWindowID id, const wxString &title, const
wxPoint &position, const wxSize& size, long style)
: wxFrame(parent, id, title, position, size, style)
{
    CreateGUIControls();
    wxInitAllImageHandlers();
}

varBitFrm::~varBitFrm()
{
}

void varBitFrm::CreateGUIControls()
{
openWildcards = _T("Image Files(*.rgbtext;*.bmp;*.jpeg;*.jpeg;*.png;*.gif;*.vbi)")
                    _T("|*.rgbtext;*.bmp;*.jpeg;*.jpeg;*.png;*.gif;*.vbi")
                    _T("|RGBtext files (*.rgbtext)|*.rgbtext")
                    _T("|BMP files (*.bmp)|*.bmp")
                    _T("|JPEG files (*.jpeg;*.jpeg)|*.jpeg;*.jpeg")
                    _T("|PNG files (*.png)|*.png")
                    _T("|GIF files (*.gif)|*.gif")
                    _T("|VBI files (*.vbi)|*.vbi");


    saveWildcards = _T("Image Files(*.bmp;*.jpeg;*.png;*.gif;*.vbi)")
                    _T("|*.bmp;*.jpeg;*.png;*.gif;*.vbi")
                    _T("|BMP files (*.bmp)|*.bmp")
                    _T("|JPEG files (*.jpeg)|*.jpeg")
                    _T("|PNG files (*.png)|*.png")
                    _T("|VBI files (*.vbi)|*.vbi");

    WxScrolledWindow1->SetBackgroundColour(wxColour(3,122,255));

    WxButton1 = new wxButton(this, ID_WXBUTTON1, wxT("Load"), wxPoint(20, 20), wxSize(100,
40), 0, wxDefaultValidator, wxT("Load"));

    WxScrolledWindow1 = new wxScrolledWindow(this, ID_WXSCROLLEDWINDOW1, wxPoint(20, 80),
wxSize(960, 580), wxVSCROLL | wxHSCROLL);

    WxButton2 = new wxButton(this, ID_WXBUTTON2, wxT("Save"), wxPoint(140, 20),
wxSize(100, 40), 0, wxDefaultValidator, wxT("Save"));

    WxOpenFileDialog1 =  new wxFileDialog(this, wxT("Choose a file"), wxT(""), wxT(""),
wxT("*.*"), wxOPEN);

    WxSaveFileDialog1 =  new wxFileDialog(this, wxT("Save as..."), wxT(""), wxT(""),
wxT("*.*"), wxSAVE);

    SetTitle(wxT("varBit"));
    SetIcon(wxNullIcon);
    SetSize(0,0,1000,700);
```

```
    Center();
}

void varBitFrm::OnClose(wxCloseEvent& event)
{
    Destroy();
}

void varBitFrm::WxButton1Click(wxCommandEvent& event)
{
    int success = 1;

    wxFileDialog* WxOpenFileDialog1 = new wxFileDialog(this,wxT("Choose image file"),
wxEmptyString,wxEmptyString,
openWildcards,wxFD_OPEN|wxFD_FILE_MUST_EXIST|wxFD_CHANGE_DIR|wxCENTRE);
    if (WxOpenFileDialog1->ShowModal()==wxID_OK)
    {
        loadimage1 = WxOpenFileDialog1->GetFilename();
        if (loadimage1.Matches("*.vbi"))
        {
            success = vBitImage.loadImage(loadimage1);
            if (success)
            {
                wxBitmap temp_img1(vBitImage.getImage(), -1);
                image1 = temp_img1;
            }
        }
        else
        {
            wxImage imagetest(loadimage1,wxBITMAP_TYPE_ANY,-1);
            wxBitmap temp_img1(imagetest, -1);
            image1 = temp_img1;
        }
        if (success)
        {
            sb1 = new wxStaticBitmap(WxScrolledWindow1,-1,image1);
            image1Width = image1.GetWidth();
            image1Height = image1.GetHeight();
            WxScrolledWindow1->SetScrollbars(10,10,image1Width/10,image1Height/10);
            WxScrolledWindow1->Scroll(1,1);
            sb1->Show(true);
        }
    }
    else
       WxOpenFileDialog1 -> Close();

    temp1 = image1.ConvertToImage();

}

void varBitFrm::WxButton2Click(wxCommandEvent& event)
{
    wxFileDialog* WxSaveFileDialog1 = new wxFileDialog(this,wxT("Save as..."),
wxEmptyString,wxEmptyString, saveWildcards,wxSAVE|wxFD_CHANGE_DIR|wxCENTRE);
    if (WxSaveFileDialog1->ShowModal() == wxID_OK)
    {
        wxString filename = WxSaveFileDialog1->GetPath();

        if (filename.Matches("*.vbi")) {
            int R = 8;
            int G = 8;
            int B = 8;
            vBitImage.openImage(temp1);
            bitSelector = new BitSelector(this, -1, wxT("Select number of bits..."),
wxPoint(0,0), wxSize(123,265), wxDEFAULT_DIALOG_STYLE);
            if (bitSelector->ShowModal() == wxID_OK)
            {
                R = bitSelector->getR();
                G = bitSelector->getG();
                B = bitSelector->getB();
                vBitImage.setRGBCount(R,G,B);
```

```
                vBitImage.saveImage(filename);
            }
        }
        else {
            wxInitAllImageHandlers();
            temp1.SaveFile(WxSaveFileDialog1->GetPath());
        }
    }
    else
        WxSaveFileDialog1 -> Close();
}
```

A.3. VBitImage.h

```
#ifndef VBImage
#define VBImage 1
#include <iostream>
#include <string>
#include <wx/image.h>
#include <wx/string.h>
#include "BitStream.h"

using namespace std;

class VBitImage
{
    private:
        int _countR;
        int _countG;
        int _countB;
        int _height;
        int _width;
        BitStream *bs;
        wxImage *image;
        void writeHeader();
        int readHeader();
        void writeData();
        int readData();

    public:
        VBitImage();
        ~VBitImage();
        int loadImage(wxString filename);
        int loadImage(string filename);
        int loadImage(char *filename);
        void setRGBCount(int r, int g, int b);
        void saveImage(wxString filename);
        void saveImage(char *filename);
        void saveImage(string filename);
        void openImage(wxImage img);
        void openImage(wxString filename);
        wxImage getImage();
};


#endif
```

A.4. VBitImage.cpp

```
#include "VBitImage.h"
#include <iostream>
#include "BitStream.h"
#include <wx/image.h>
#include <wx/string.h>
#include <wx/txtstrm.h>
#include <wx/wfstream.h>

using namespace std;
```

```
VBitImage::VBitImage()
{
    bs = new BitStream();
}

VBitImage::~VBitImage()
{
    delete bs;
}

void VBitImage::writeHeader()
{
    bs->WriteByte('v');
    bs->WriteByte('b');
    bs->WriteByte('i');
    bs->WriteByte('t');
    bs->WriteByte('i');
    bs->WriteByte('m');
    bs->WriteByte('a');
    bs->WriteByte('g');
    bs->WriteByte('e');
    bs->WriteByte(26);
    bs->WriteBits(_width,32);
    bs->WriteBits(_height,32);
    bs->WriteBits(_countR, 32);
    bs->WriteBits(_countG, 32);
    bs->WriteBits(_countB, 32);
    bs->WriteByte('e');
    bs->WriteByte('n');
    bs->WriteByte('d');
}

int VBitImage::readHeader()
{
    char vbit[9];
    char vbitimage[10] = "vbitimage";
    char end[3];
    char endheader[4] = "end";
    int success = 1;

    for (int i = 0; i < 9; i++) {
        vbit[i] = bs->ReadByte();
    }

    success = 1;
    for (int i = 0; (i < 9) && (success == 1); i++) {
        if (vbit[i] != vbitimage[i])
            success = 0;
    }

    if (success) {
        bs->ReadByte();
        _width = bs->ReadBits(32);
        _height = bs->ReadBits(32);
        _countR = bs->ReadBits(32);
        _countG = bs->ReadBits(32);
        _countB = bs->ReadBits(32);

        for (int j = 0; j < 3; j++) {
            end[j] = bs->ReadByte();
        }

        for (int j = 0; (j < 3) && (success == 1); j++) {
            if (end[j] != endheader[j])
                success = 0;
        }
    }
    return success;
}
```

```
void VBitImage::writeData()
{
    int r, g, b;
    int r2, g2, b2;
    for (int y = 0; y < _height; y++) {
        for (int x = 0; x < _width; x++) {
            r = image->GetRed(x,y);
            g = image->GetGreen(x,y);
            b = image->GetBlue(x,y);

            if (_countR < 8)
                r2 = r >> (8 - _countR);
            else if (_countR > 8)
                r2 = r << (_countR - 8);
            else
                r2 = r;
            if (_countG < 8)
                g2 = g >> (8 - _countG);
            else if (_countG > 8)
                g2 = g << (_countG - 8);
            else
                g2 = g;
            if (_countB < 8)
                b2 = b >> (8 - _countB);
            else if (_countB > 8)
                b2 = b << (_countB - 8);
            else
                b2 = b;

            bs->WriteBits(r2, _countR);
            bs->WriteBits(g2, _countG);
            bs->WriteBits(b2, _countB);
        }
    }
    bs->WriteByte('e');
    bs->WriteByte('n');
    bs->WriteByte('d');
    bs->WriteByte('d');
    bs->WriteByte('a');
    bs->WriteByte('t');
    bs->WriteByte('a');
    bs->stopWrite();
}

int VBitImage::readData()
{
    int r, g, b;
    int r2, g2, b2;
    char end[7];
    char enddata[8] = "enddata";
    int success;

    success = 1;
    for (int y = 0; y < _height; y++) {
        for (int x = 0; x < _width; x++) {
            r = bs->ReadBits(_countR);
            g = bs->ReadBits(_countG);
            b = bs->ReadBits(_countB);
            if (_countR < 8)
                r2 = r << (8 - _countR);
            else if (_countR > 8)
                r2 = r >> (_countR - 8);
            else
                r2 = r;
            if (_countG < 8)
                g2 = g << (8 - _countG);
            else if (_countG > 8)
                g2 = g >> (_countG - 8);
            else
                g2 = g;
            if (_countB < 8)
```

```
                b2 = b << (8 - _countB);
            else if (_countB > 8)
                b2 = b >> (_countB - 8);
            else
                b2 = b;
            image->SetRGB(x,y,r2,g2,b2);
        }
    }
    for (int i = 0; i < 7; i++) {
        end[i] = bs->ReadByte();
    }

    for (int i = 0; (i < 7) && (success == 1); i++) {
        if (end[i] != enddata[i])
            success = 0;
    }
    bs->stopRead();
    return success;
}

int VBitImage::loadImage(wxString filename)
{
    return loadImage((char*)filename.char_str());

}

int VBitImage::loadImage(string filename)
{
    char _filename[filename.size()+1];
    strcpy (_filename, filename.c_str());

    return loadImage(_filename);

}

int VBitImage::loadImage(char *filename)
{
    int headerReadResult;
    int dataReadResult = 0;

    bs->Load(filename);

    headerReadResult = readHeader();
    if (headerReadResult) {
        image = new wxImage(_width, _height);
        dataReadResult = readData();
    }
    return dataReadResult;
}

void VBitImage::saveImage(wxString filename)
{
    saveImage ((char*)filename.char_str());
}

void VBitImage::saveImage(string filename)
{
    char _filename[filename.size()+1];
    strcpy (_filename, filename.c_str());

    saveImage (_filename);
}

void VBitImage::saveImage(char *filename)
{
    bs->Save(filename);
    writeHeader();
    writeData();
}

void VBitImage::setRGBCount(int r, int g, int b)
```

```
{
    _countR = r;
    _countG = g;
    _countB = b;
}

void VBitImage::openImage(wxImage img)
{
    image = new wxImage(img);
    _height = image->GetHeight();
    _width = image->GetWidth();
}

void VBitImage::openImage(wxString filename)
{
    image = new wxImage(filename, wxBITMAP_TYPE_ANY, -1);
    _height = image->GetHeight();
    _width = image->GetWidth();
}

wxImage VBitImage::getImage()
{
    return *image;
}
```

A.5. BitStream.h

```
#ifndef MyBitStream
#define MyBitStream 1
#include <iostream>
#include <fstream>

using namespace std;

class BitStream
{
    private:
        int _currentPosition;
        char _storage;
        int _mask;
        void _WriteBit (char value);
        int _ReadBit ();
        fstream *bstream;
    public:
        BitStream ();
        ~BitStream ();
        void WriteBit (char value);
        char ReadBit ();
        void WriteByte (char value);
       char ReadByte ();
        void WriteBits (int value, int length);
        int ReadBits (int length);
        void Save (string filename);
        void Save (char *filename);
        void Load (string filename);
        void Load (char *filename);
        void stopWrite ();
        void stopRead ();
        int eof();
};
#endif
```

A.6. BitStream.cpp

```
#include "BitStream.h"
#include <iostream>
#include <cstring>
#include <string>
#include <fstream>

using namespace std;
```

```
BitStream::BitStream ()
{
    _currentPosition = 0;
    _storage = 0x00;
    _mask = 0x80;
}

BitStream::~BitStream ()
{
    delete bstream;
}

void BitStream::_WriteBit (char value)
{
    _storage |= ((value & 0x01) << (7 - _currentPosition));
    _currentPosition++;
    if (_currentPosition == 8)
    {
        *bstream << _storage;
        _storage = 0x00;
        _currentPosition = 0;
    }
}

int BitStream::_ReadBit ()
{
    int bitVal;
    if (_currentPosition == 0)
        bstream->get(_storage);

    _mask = 0x80 >> _currentPosition;
    bitVal = _storage & _mask;

    _currentPosition++;
    if (_currentPosition == 8)
        _currentPosition = 0;

    bitVal? bitVal = 1: bitVal = 0;
    return bitVal;
}

void BitStream::WriteBit (char value)
{
    _WriteBit(value);
}

char BitStream::ReadBit ()
{
    return _ReadBit();
}

void BitStream::WriteByte (char value)
{
    _mask = 0x80;
    int temp;
    for (int i = 0; i < 8; i++) {
        temp = (value & _mask) >> (7 - i);
        _WriteBit(temp);
        _mask = _mask >> 1;
    }
}

char BitStream::ReadByte ()
{
    int temp = 0x00;
    for (int i = 0; i < 8; i++) {
        temp |= (_ReadBit() << (7 - i));
    }
    return temp;
}
```

```
void BitStream::WriteBits (int value, int length)
{
    if (length >= 1 && length <= 32)
    {
        _mask = 0x01 << (length - 1);
        int temp;
        for (int i = 0; i < length; i++) {
            temp = (value & _mask) >> (length - i - 1);
            _WriteBit(temp);
            _mask = _mask >> 1;
        }
    }
}

int BitStream::ReadBits (int length)
{
    int temp = 0;
    for (int i = 0; i < length; i++) {
        temp |= (_ReadBit() << (length - 1 - i));
    }
    return temp;
}

void BitStream::Save (string filename)
{
    char _filename[filename.size()+1];
    strcpy (_filename, filename.c_str());

    Save (_filename);
}

void BitStream::Save (char *filename)
{
    bstream = new fstream();
    bstream->open (filename, fstream::out | fstream::binary);
}

void BitStream::Load (string filename)
{
    char _filename[filename.size()+1];
    strcpy (_filename, filename.c_str());

    Load(_filename);
}

void BitStream::Load (char *filename)
{
    bstream = new fstream();
    bstream->open (filename, fstream::in | fstream::binary);
}

void BitStream::stopWrite ()
{
    if (_currentPosition != 0) {
        WriteBits(0x00, 8 - _currentPosition);
    }
    bstream->close();
}

void BitStream::stopRead ()
{
    bstream->close();

}

int BitStream::eof()
{
    return bstream->eof();
}
```

A.7. BitSelector.h

```
#ifndef __BITSELECTOR_H__
#define __BITSELECTOR_H__

#ifdef __BORLANDC__
    #pragma hdrstop
#endif

#ifndef WX_PRECOMP
    #include <wx/wx.h>
    #include <wx/dialog.h>
#else
    #include <wx/wxprec.h>
#endif

#include <wx/button.h>
#include <wx/stattext.h>
#include <wx/choice.h>

#undef BitSelector_STYLE
#define BitSelector_STYLE wxCAPTION | wxDIALOG_NO_PARENT | wxCLOSE_BOX


class BitSelector : public wxDialog
{
    private:
        DECLARE_EVENT_TABLE();

    public:
        BitSelector(wxWindow *parent, wxWindowID id = 1, const wxString &title =
wxT("BitSelector"), const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, long style = BitSelector_STYLE);
        virtual ~BitSelector();
        void WxButton1Click(wxCommandEvent& event);
        void WxButton2Click(wxCommandEvent& event);
        void WxChoice1Selected(wxCommandEvent& event );

        int getR();
        int getG();
        int getB();

    private:
        wxButton *WxButton2;
        wxButton *WxButton1;
        wxChoice *WxChoice3;
        wxStaticText *WxStaticText3;
        wxChoice *WxChoice2;
        wxStaticText *WxStaticText2;
        wxStaticText *WxStaticText1;
        wxChoice *WxChoice1;


    private:
        {
                ID_WXBUTTON2 = 1011,
                ID_WXBUTTON1 = 1010,
                ID_WXCHOICE3 = 1009,
                ID_WXSTATICTEXT3 = 1008,
                ID_WXCHOICE2 = 1006,
                ID_WXSTATICTEXT2 = 1005,
                ID_WXSTATICTEXT1 = 1003,
                ID_WXCHOICE1 = 1002,
                ID_DUMMY_VALUE_
        };

    private:
        void OnClose(wxCloseEvent& event);
        void CreateGUIControls();
};
```

```
#endif
```

A.8. BitSelector.cpp

```
#include "BitSelector.h"
#include <wx/txtstrm.h>
#include <wx/wfstream.h>

BEGIN_EVENT_TABLE(BitSelector,wxDialog)

    EVT_CLOSE(BitSelector::OnClose)
    EVT_BUTTON(ID_WXBUTTON2,BitSelector::WxButton2Click)
    EVT_BUTTON(ID_WXBUTTON1,BitSelector::WxButton1Click)
    EVT_CHOICE(ID_WXCHOICE1,BitSelector::WxChoice1Selected)
END_EVENT_TABLE()

BitSelector::BitSelector(wxWindow *parent, wxWindowID id, const wxString &title, const
wxPoint &position, const wxSize& size, long style)
: wxDialog(parent, id, title, position, size, style)
{
    CreateGUIControls();

    SetAffirmativeId(ID_WXBUTTON1);
    SetEscapeId(ID_WXBUTTON2);
}

BitSelector::~BitSelector()
{
}

void BitSelector::CreateGUIControls()
{

    WxButton2 = new wxButton(this, ID_WXBUTTON2, wxT("Cancel"), wxPoint(160, 45),
wxSize(75, 25), 0, wxDefaultValidator, wxT("WxButton2"));

    WxButton1 = new wxButton(this, ID_WXBUTTON1, wxT("Save"), wxPoint(75, 45), wxSize(75,
25), 0, wxDefaultValidator, wxT("WxButton1"));

    wxArrayString arrayStringFor_WxChoice3;
    arrayStringFor_WxChoice3.Add(wxT("0"));
    arrayStringFor_WxChoice3.Add(wxT("1"));
    arrayStringFor_WxChoice3.Add(wxT("2"));
    arrayStringFor_WxChoice3.Add(wxT("3"));
    arrayStringFor_WxChoice3.Add(wxT("4"));
    arrayStringFor_WxChoice3.Add(wxT("5"));
    arrayStringFor_WxChoice3.Add(wxT("6"));
    arrayStringFor_WxChoice3.Add(wxT("7"));
    arrayStringFor_WxChoice3.Add(wxT("8"));
    WxChoice3 = new wxChoice(this, ID_WXCHOICE3, wxPoint(195, 15), wxSize(40, 23),
arrayStringFor_WxChoice3, 0, wxDefaultValidator, wxT("WxChoice3"));
    WxChoice3->SetSelection(-1);

    WxStaticText3 = new wxStaticText(this, ID_WXSTATICTEXT3, wxT("Blue"), wxPoint(165,
17), wxDefaultSize, 0, wxT("WxStaticText3"));

    wxArrayString arrayStringFor_WxChoice2;
    arrayStringFor_WxChoice2.Add(wxT("0"));
    arrayStringFor_WxChoice2.Add(wxT("1"));
    arrayStringFor_WxChoice2.Add(wxT("2"));
    arrayStringFor_WxChoice2.Add(wxT("3"));
    arrayStringFor_WxChoice2.Add(wxT("4"));
    arrayStringFor_WxChoice2.Add(wxT("5"));
    arrayStringFor_WxChoice2.Add(wxT("6"));
    arrayStringFor_WxChoice2.Add(wxT("7"));
    arrayStringFor_WxChoice2.Add(wxT("8"));
    WxChoice2 = new wxChoice(this, ID_WXCHOICE2, wxPoint(120, 15), wxSize(40, 23),
arrayStringFor_WxChoice2, 0, wxDefaultValidator, wxT("WxChoice2"));
    WxChoice2->SetSelection(-1);
```

```
    WxStaticText2 = new wxStaticText(this, ID_WXSTATICTEXT2, wxT("Green"), wxPoint(85,
17), wxDefaultSize, 0, wxT("WxStaticText2"));

    WxStaticText1 = new wxStaticText(this, ID_WXSTATICTEXT1, wxT("Red"), wxPoint(10, 17),
wxDefaultSize, 0, wxT("WxStaticText1"));

    wxArrayString arrayStringFor_WxChoice1;
    arrayStringFor_WxChoice1.Add(wxT("0"));
    arrayStringFor_WxChoice1.Add(wxT("1"));
    arrayStringFor_WxChoice1.Add(wxT("2"));
    arrayStringFor_WxChoice1.Add(wxT("3"));
    arrayStringFor_WxChoice1.Add(wxT("4"));
    arrayStringFor_WxChoice1.Add(wxT("5"));
    arrayStringFor_WxChoice1.Add(wxT("6"));
    arrayStringFor_WxChoice1.Add(wxT("7"));
    arrayStringFor_WxChoice1.Add(wxT("8"));
    WxChoice1 = new wxChoice(this, ID_WXCHOICE1, wxPoint(40, 15), wxSize(40, 23),
arrayStringFor_WxChoice1, 0, wxDefaultValidator, wxT("WxChoice1"));
    WxChoice1->SetSelection(-1);

    SetTitle(wxT("Select number of bits..."));
    SetIcon(wxNullIcon);
    SetSize(8,8,265,123);
    Center();
}

void BitSelector::OnClose(wxCloseEvent& /*event*/)
{
    Destroy();
}

void BitSelector::WxButton1Click(wxCommandEvent& event)
{
        wxFileOutputStream out1 ("BitSelector.txt");
        wxTextOutputStream debugger (out1);
        debugger << getR() << " " << getG() << " " << getB() << endl;
    this->EndModal(wxID_OK);
}

void BitSelector::WxButton2Click(wxCommandEvent& event)
{
    this->EndModal(wxID_CANCEL);
}

void BitSelector::WxChoice1Selected(wxCommandEvent& event )
{

}

int BitSelector::getR()
{
    return WxChoice1->GetSelection();
}

int BitSelector::getG()
{
    return WxChoice2->GetSelection();
}

int BitSelector::getB()
{
    return WxChoice3->GetSelection();
}
```

References

[1] The Code Project. CBitStream - A simple C++ class for reading and writing variable-length data. <http://www.codeproject.com/KB/cpp/CBitStream.aspx>

[4] Smart, J., Roebling, R., Zeitlin, V., Dunn, R., et al. *wxWidgets 2.8.11: A portable C++ and Python GUI toolkit*. <http://docs.wxwidgets.org/2.8/>