This lecture gives an introduction about image processing.

Image processing is a very rich topic. You can find many books on this topic. This lecture provides only a brief introduction of some basic concepts.

First, images may be stored in many different formats, such as bit map, J peg, tiff, and many others.

An image file has two parts: the data and the meta data. What is the data? The data is the pixels’ colors. What is the meta data? Meta data describes the data. For an image, the meta data may include the image’s width, height, color model, camera model, etc.

The meta data is also called the header of the file. This is similar to the Huffman compressed file. The beginning of the file is the header that includes the code tree and the number of characters in the file.

What may be stored in an image’s meta data? This an example. It stores the aperture value, the exposure time, the focal length, the camera model, the time when this image was taken.

In this lecture, we will consider one particular data format called bitmap data format.

This data format is uncompressed. In contrast, the popular J peg data format is compressed.

In the bit map format, each pixel uses three bytes to store red, green, blue. This is called the are gee bee color model. This color model is the most popular color model.

This is the header of the bit map image file. Before the structure, pragma pack one tells compiler not to add space between the structure’s attributes.

Without this pragma, compilers have the freedom to add space between attributes for the purpose of making programs fasters.

With this pragma, compilers are not allowed to add space between attributes.

We need to compiler not to add space because we have to ensure that the header’s size is exactly the same for all images using the bit map format.

The header has many attributes. The beginning U means unsinged. The number 16 means 16 bits or 2 bytes. The number 32 means 32 bits or 4 bytes.

All these attributes together create a new data type called Bee M pee underscore header.

The header is exactly 54 bytes. This table shows how to calculate the size. Each unsigned integer 16 means 2 bytes. Each unsigned integer 32 means 4 bytes.

This is the format for an image file. The structure uses the header structure. In addition, it has an unsigned integer as the file size, the width and height of the image. It specifies the number of bytes per pixel. In this example, we will use three bytes per pixels for red, green, and blue.

At the end of the structure is a pointer to store the data. The amount of data is determined by the width and height. Thus, we need to obtain that header information first because we know how many types to allocate.

How can we read an image file? We need to first read the header. Its size is already known. This is the reason we need to use pragma to ensure that no space is added by the compiler.

After reading the header, we know the number of bytes needed for the data. Then, we need to allocate memory to store the data and read the data from the file.

When we write an image to a file, we need to write the header first and then write the data.

This is the function to read an image file. We need to F open the file and allocate memory for the image.

The first malloc allocates memory for the header but does not allocate memory for the data. This is because in Bee M pee underscore image, the header is not a pointer and the data is a pointer.

The first F read function reads the header. We need to check whether F read is successful or not. This F read function reads only one header. Thus, the returned value should be one. If it is not one, F read fails and the function calls the clean up function.

The next step checks whether the header is expected. The check header function checks whether some attributes’ values are expected.

From the header, we can obtain the size of the entire file. The size of the data is the size of the file minus the size of the header.

From the header, we also know the width and height of this image. We can calculate the number of bytes per pixel. It is expected to be three.

Next, we need to allocate memory for the data based on the information from the header.

After successfully allocating memory for the data, we call F read again to read the pixels’ colors. If F read fails here, that means the file is too small and does not have enough data to read.

Reading the data should consumes everything in the file.

The last F read checks whether the file has any unread byte. If the last F read can read one byte, the file is too large and is also problematic.

If reading the image is successful, the program returns the address of the allocated memory for the image.

What is the check header function? A few attributes’ values are known in advance. We want to check whether the values stored in the file match what is expected. If any of the attribute’s value is different from the expected value, we consider the input file invalid.

Saving an image to a file calls F write twice. The first time, the information in the header is saved. The second F write saves the data.

Next, let’s talk about the color model. This lecture consider the Are, gee, bee, color models. In this color model, colors are composed of three basic components: red, green, and blue.

If a color has both green and blue without red, this color is cyan. If a color has green and red without blue, the color is yellow. If a color has red and blue without green, the color is magenta.

Each color uses one byte and may have a value between 0 and 255. A smaller value means a darker color. If the value is 255, the color is the brightest. When all three colors are 0, it is black.

The three color components uses 3 bytes or 24 bits. These 24 bits can create 16 million different colors.

When all three colors have equal intensity, the color becomes gray.

If each color’s intensity is 255, the combined color is white.

Please notice that this is the additive color model. When two colors are added, the result is brighter.

Another color model is the subtractive model. The model is used in paintings. When you add the pigments of different colors together, the result is a darker color.

In the rest of this lecture, we will use the additive color model.

The left bottom of this slides shows the colors of different intensities. We consider pure green of different intensities, pure blue, and pure red.

We can add different color filters. A color filter means allowing a certain color to pass and blocking the other colors.

For example, if the color filter is blue, then the blue component stays the same and the other two colors, red and green, are set to zero.

If the color filter is red, then the red component stays the same and the other two colors, green and blue, are set to zero.

If the color filter is green, then the green component stays the same and the other two colors, red and blue, are set to zero.

Since gray and white include all three colors, the specific color is shown when a filter is applied.

Another concept is color inversion. We use 255 to subtract the value of each color component.

This creates the complementary color.

If the original color is green, the complementary color is magenta. If the original color is blue, the complementary color is yellow. If the original color is red, the complementary color is cyan.

In the next few slides, we will explain several simple methods to manipulate colors. The first example is adding the green filter. Consider an input image on the left side. After adding the green filter, the image becomes the one at the right side.

How can we add a color filter? Each pixel has three colors, arranged by red, green, and blue. Each color uses one byte. If we want to apply the green filter, then the second argument is one. The green color is unchanged. The other two colors are set to zero.

The next function is color inversion. White becomes black. Magenta becomes green. Yellow becomes blue.

Color inversion is achieved by using 255 to subtract the original color.

The next concept is called color equalization. Consider the image on the left. It is a little too dark. What does that mean? It means most pixels’ colors have low intensities. We want to brighten the image like the one at the right side.

Color equalization may use many different methods. Here we present a simple method. We want to spread the intensity between 0 and 255.

Consider the curve at the left side. This is the histogram of the intensity distribution.

As you can see, most values are small. We want to find the lowest intensity and the highest intensity. Then, we want to stretch the distribution so that the lowest becomes zero and the highest becomes 255.

We need to find a way to assign the intensities.

We can use a linear equation for this purpose. The new intensity is ei times the old intensity plus bee.

If the intensity is the original minimum intensity, the new intensity is zero. Thus, zero equals to ei times min plus bee.

If the intensity is the original maximum intensity, the new intensity is 255. Thus, 255 equals to ei times max plus bee.

This linear equation can be solved by subtracting the first from the second. Thus, ei is 255 divided by the difference of max and min, .

Bee is the negative value of ei times min, .

This equalize function uses the linear method explained in the previous slide.

We first set min to be 255 and max to be 0. Then, we go through each pixel and get the intensity of the three colors.

If the pixel’s intensity is smaller than the currently stored min, min is set to this intensity.

If the pixel’s intensity is greater than the currently stored max, max is set to the intensity.

After going through every pixel, we now have the min and max of each color.

Next, we calculate the values of ei for each color. Then, the intensity of each pixel is adjusted.