Here's my final report for the jpeg analyzer. I assume I'll be continuing work on it through fall quarter, so it is still in a very rough state. I had hoped to have the project completed by now, but I had underestimated the complexity of the jpeg file format. Attached is a text file that documents the classes and their member variables in the project. Since the program is a work-in-progress, the file will be out of date quite soon. There's a brief explanation of the jpeg format at the start of the text file, and it hopefully will provide enough information to make sense of the rest of the file.

Current Status:
The JpegAnalyzer can read jpegs that use chroma subsampling of 1x1. In other words, jpegs that don't have chroma subsampling. This version is stored as JpegAnalyzer.java.old. I am working on the JpegAnalyzer.java file to implement chroma subsampling. In the case of the webcam images we will be working with, the analyzer needs to be able to take two MCUs, one with Cb data and one with Cr data, and write that data to 4 MCUs that are storing Y data. The math needed to keep track of the indexes of the various MCUs is very complicated, since they are all indexed as one-dimensional arrays. To simplify things, I may re-implement the MCUs using 2-d arrays.

At this time, the program has a great deal of output to the screen that I'm using for debugging, and this needs to be removed before the program can be considered finished. Hopefully, I'll be done soon, and I can take out all of the debugging output.

Report: jpegAnalyzer.txt
The jpeg image format stores images as entropy-coded scans. At the highest level, jpegs are made up of frames, which contain components, which are in turn made up of scans. The scans are raw binary data that can be decoded using huffman tables, which are stored in the jpeg. A component stores the data of one visual channel of the jpeg. Since jpegs use the YCbCr color system, each component represents the Y, the Cb, or the Cr. The component data can be read into a structure called an MCU, or Minimum Coded Unit. An MCU stores the data for an 8x8 block of pixels. Typical jpegs will use chroma subsampling to reduce the amount of data stored, so that an MCU hold one Y component, but only a portion of the Cr and Cb components, depending on the ratio. In the webcam images we are planning to use, the ratio is usually 2x2, meaning that the Y channel is stored for each pixel, but the Cr and Cb channels are stored in 2x2 pixel blocks.

The JpegAnalyzer works by taking a jpeg image as a parameter, and reading it into a byte array. The program then reads the data in the array, looking for tags. When tags are encountered, the program reads the information they contain into various data structures. Once the program has read all the tags, the extracted data is converted into an array of RGB pixels.

Classes:

JpegAnalyzer.java

This is the main class of the program. Its members are as follows:

| | |
|---|---|
| int currentIndex | : Stores the index of the next unread byte in the ByteBuffer. |
| int lineCount | : The number of horizontal lines in the jpeg. |
| int samplesPerLine | : The number of vertical lines in the jpeg. |
| byte currentByte | : The byte currently being read. |
| byte readBuff[] | : The buffer which stores the jpeg. |
| boolean logging | : True if the application is writing to a logfile. |
| String logFileName | : The name of the logfile. |
| PrintWriter logFile | : The logfile. |
| Vector<Scan> scans | : Scan tags in the jpeg are read into Scans, and stored in this vector. |
| Vector<Frame> frames | : Frame tags in the jpeg are read into Frames, and stored in this vector. |
| PixelData pixelData | : Stores Mcus for conversion into RGB values. |
| Pixel [][] pixels | : Stores the actual RGB pixels in an image. |

Mcu.java

Mcu is a data structure representing the jpeg MCU, which stands for Minimum Coded Unit. An MCU stores the data for an 8x8 pixel block. The data is stored as  the three spectrum channels that jpeg uses, Y, Cb, and Cr.

| | |
|---|---|
| int [] y | : The Y values. |
| int yDc | : The DC value for the Y channel in this MCU. |
| int [] cr | : The Cr values. |
| int crDc | : The DC value for the Cr channel. |
| int [] cb | : The Cb values. |
| int cbDc | : The DC value for the Cb channel. |
| int yIndex | : The index of the next unwritten Y value |
| int crIndex | : The index of the next unwritten Cb value |
| int cbIndex | : The index of the next unwritten Cr value |
| int xSub | : The chroma subsampling ratio in the x direction. |
| int ySub | : The chroma subsampling ratio in the y direction. |

Frame.java

Frame is a data structure that represents a jpeg frame. A jpeg frame stores components, as well as the number of lines and samples per line, which is to say, the x and y dimensions of the jpeg.

| | |
|---|---|
| int length | : The length of the frame tag. |
| int precision | : The precision of the frame. |
| int lines | : The number of lines in the jpeg. |
| int samplesPerLine | : The number of samples per line. |
| int componentCount | : The number of components stored in this frame. |
| Vector<Component> components | : The components stored in the frame. |

Component

Component is a member class of frame that stores the data about the components in that frame.

| | |
|---|---|
| int id | : The id identifies the component. |
| int horizontalFactor | : The horizontal factor for this component's chroma subsampling. |
| int verticalFactor | : The vertical factor for this component's chroma subsampling. |
| int quantTableDest | : The quantization table that this component has been quantized with. |

HuffmanTable.java

This class stores a huffman table. The table is used to interpret the entropy-coded data in a jpeg.

| | |
|---|---|
| int [] codeLengthCounts | : The huffman table needs to know how many codes of a given length exist, and it is stored here. The index corresponds with the length. |
| Hashtable<CodeLengthKeyPair, Short> codes | : The actual codes are stored in this HashTable. The are keyed using a CodeLengthKeyPair. |
| int tableClass | : The class of the table. 0 is for DC, 1 is for AC. |
| int id | : The table id. 0 is for Y, 1 is for Cr and Cb |
| HuffBinTree codeTree | : HuffBinTree is the binary tree used to create the huffman code. |

CodeLengthKeyPair.java

This class stores a huffman code and its length. It is used as a key to retrieve a value from a huffman table. The length is necessary to distinguish between codes that have similar values, but different lengths (e.g. 110 and 0110).

| | |
|---|---|
| int code | : The code. It's integer representation is meaningless, because codes of different lengths can share the same numerical value. |
| int length | : The length, in bits, of the code. |

HuffBinTree.java

This class is used to create a binary tree of huffman codes. It works by taking an array of hashtable values (i.e. the values which correspond to the entropy encoded codes), an array of code length counts, and a codeTable, and then populating the code table. To create a code of a given length, it creates a binary tree to a depth equal to the length. The root node counts as length 0. The nodes store a possibe code in a short. It finds a code node using in-order traversal. When it finds a node at the correct length, it marks that node as used, and returns the result.

| | |
|---|---|
| Node root | : The root of the binary tree |
| Vector<Short> values | : The values that the codes will correspond to. |
| Hashtable<CodeLengthKeyPair, Short> codeTable | : The code table to be populated. |
| int [] codeLengthCounts | : The counts of codes of each length. The index corresponds to the code length. |

Node

Node is a member class of the HuffBinTree class. It holds a code, as well as a boolean which stores whether or not that code is in use.

| | |
|---|---|
| short code | : The code |
| boolean isCode | : True if the node is already code, false if not. |

| | |
|---|---|
| Node left | : The left child. |
| Node right | : The right child. |

## QuantizationTable.java

Quantization tables are used to quantize the component data in a jpeg, which reduces the size of the coded image.

| | |
|---|---|
| int length | : The length of the quantization table. |
| short precision | : The bit precision of the table. |
| int id | : The id identifies the quantization table for components. |
| Vector<Short> elements | : The elements make up the table. Each one of the 64 values corresponds to the value in an MCU that it quantized. |

## Scan.java

This class represents a scan in a jpeg image. A scan has header information describing the data it contains. After the header, there is a run of entropy coded data. The end of the data is indicated by the start of the next tag.

| | |
|---|---|
| int componentCount | : The number of components in the scan |
| Vector<ScanComponent> components | : Stores the components. |
| int startOfSelection | : Stores the MCU index that the scan starts at. |
| int endOfSelection | : Stores the MCU index that the scan ends at. |
| int bitHigh | : Used for arithmetic encoding. |
| int bitLow | : Used for arithmetic encoding. |

## ScanComponent

This class is a member class of Scan. It stores the information for each scan.

| | |
|---|---|
| int scanSelector | : This variable identifies the scan. |
| int dcTableSelector | : The id of the table that decodes the scan's dc value. |
| int acTableSelector | : The id of the table that decodes the scan's ac values. |

## PixelData.java

This class stores the MCUs of a jpeg. Once the jpeg has been decoded, and all the MCUs have been stored in PixelData, this class can convert the MCUs from 8x8 YCbCr structures to one large array of RGB pixels.

| | |
|---|---|
| Vector<Mcu> mcus | : The MCUs. |
| int currentIndex | : The current MCU index. |
| int xSub; | : The chroma subsampling factor in the horizontal direction. |
| int ySub; | : The chroma subsampling factor in the vertical direction. |
| int xMcus; | : The number of MCUs in the x direction. |
| int yMcus; | : The number of MCUs in the y direction. |