SIENAcollege
Computer Science

Computer Science 010
Introduction to Computer Applications
Siena College
Fall 2010

Topic Notes: File Formats

We have seen how binary representations are used for numbers and for an executable program on a very simple architecture.

All other data used by computers requires a binary representation as well. This data may be in the computer's memory, on a storage device in a file, or be transmitted across a network. But in all cases, it must be binary at the lowest level.

Consider the kinds of files you use on a regular basis. Different programs save and load files of different *file types*.

A file of a given type must follow a strict set of rules for how the data in the file is represented (in binary) within the file. Most of the time, a file's *extension* – the part of the file name after a "dot", gives an indication of its type.

Extension	Application	Data
doc/docx	Word	Formatted Text
xls/xlsx	Excel	Spreadsheet
ppt/pptx	Powerpoint	Presentation
mp3	Audio Players	Sound
jpeg	Image Viewers	Still Image
mpeg	Video Players	Video
html	Web Browsers	Web Pages
exe	Programs	CPU Instructions

The file format for a word processor (such as Microsoft Word or OpenOffice.org's Writer) will certainly contain text encoded using ASCII or Unicode. But it also needs to store all of your formatting information: fonts, indentation, tables, etc. A spreadsheet's file format will have plenty of numbers, but again needs additional information such as formulas and formatting information.

However, things like audio, images, and video are less straightforward. Before we can consider what a file format for these types of data might look like, we need to think more carefully about how those kinds of information might be stored as binary at all.

Analog *vs.* Digital

In the physical world, sound and light exists in an *analog* form. Sound waves are patterns of air pressure changes that we interpret as sound based on the shape of the wave. We see by interpreting the color and brightness of the light waves that reach our eyes. In each of these cases, the data is

continuous. Any pitch or volume can exist in a sound wave. The sound can change at any point in time.

Before computers became commonplace, recording devices for sound and images worked by reproducing the sound and light waves observed by the device. The grooves on a vinyl record allow a needle to vibrate in the pattern corresponding to the recorded sound. Traditional radio signals also allow reproduction of the sound wave.

Computers, however, cannot process analog data directly – the data must be *digital*.

Sound

Taking an analog sound and creating a digital representation is an example of *analog-to-digital conversion (ADC)*. This process involves taking a sound wave (or *signal*) and *sampling* it at fixed intervals. Each of these samples is a number representing the value of the sound wave at a specific point in time.

The sampled, digital signal is an approximation of the original analog signal. The frequency of these samples, the *sampling rate*, determines how accurate an appoximation we have. A higher sampling rate means that samples are closer together, giving a more accurate representation. However, a higher sampling rate means more data is required to store the representation, increasing the cost to store the data on a device or to transmit the data over a network.

See: `http://en.wikipedia.org/wiki/Sampling_rate`

For sound, the process of analog-to-digital conversion requires a fast CPU that can take millions of samples per second and collect the samples into a file stored on a CD or in another audio file format.

While a computer is very good at dealing with these digital representations, such a representation must be converted back to an analog representation to be played back to reproduce the sound waves. This requires a *digital-to-analog conversion (DAC)*. This is done by sending the digital data to a circuit that produces the vibrations that create the sound signal.

CD audio uses a 44.1 kHz sampling rate (44,100 samples per second). Each sample generates 32 bits of data (16 bits on each of 2 channels). This means in one second, 1.411 Mb (megabits) of data is required. For a 4-minute clip, this means a total data size of

$$4\,min \times 60\frac{sec}{min} \times 1.411\ Mb/sec = 338,640Kb$$

By contrast, MP3 audio, which involves a *lossy compression*, generates only 128 Kb/sec, meaning the same 4 minutes of sound would require only

$$4\,min \times 60\frac{sec}{min} \times 128\ Kb/sec = 30,720Kb$$

That's about one tenth of the data for the same audio. But at what price? The lossy compression means that some of the fine detail of the audio signal (which is, hopefully, unimportant) will be

eliminated from the data.

Images

The analog-to-digital conversion of a visual image involves digitization in space rather than in time.

Here, the image field is divided into a grid of *picture elements* (*pixels*).

In each pixel, the color and intensity of the light is encoded as a set of numbers. This is almost always in a *red-green-blue (RGB) color scheme*. These are the primary colors of light, from which the other colors can be created.

The digital-to-analog conversion from a digitized image (the grid of pixel data) involves drawing the image, pixel-by-pixel, on a video screen or printer. If you look closely at a video display (television or computer monitor), you will see the pixels are made up of groups of three lights (or subpixels) – one red, one green, and one blue.

What does this mean for the data size of digitized images? Consider a digital camera that takes 8 megapixel images. 8 megapixels means 8×2^{20} pixels, or around 8 million pixels in the image. In the RAW image format, each color is given a 16-bit value. So a RAW image would require

$$8\ mega \times 16 \times 3 = 384\ Mb$$

However, most images are stored in a compressed format such as JPEG. Like MP3 for audio, JPG is a lossy compression format, so some information is lost. The image is approximated by mathematical formulas, which are applied to retrieve an image similar to the original when the image is displayed.

Video

Digital video requires analog-to-digital conversion in both space and time. The spatial digitization is the same as used for still images. But the time digitization requires a sampling rate – how often do we take another snapshot, or *frame*? To appear as continuous motion to the human eye (and brain), we need at least 24 frames per second.

Here, the size of the file will be determined by the length of the video, the number of frames per second, and the size of each image.

Examining File Formats

Generally, only the applications that created or are intended to be able to read a given file format ever are concerned with the specifics of a file format. But the details are there for all to see, so we will take a bit of a closer look at some common file formats.

We will use a program called “HexDump” to examine the contents of files, as the name suggests, as a sequence of hexadecimal numbers. This is a Windows executable named `hexdump.exe`,

developed by Richard Pasco, and available at `http://www.richpasco.org/utilities/hexdump.html`.

We can run this program by opening a Windows command shell and typing its name followed by the name of the file we wish to examine:

```
hexdump somefile
```

Where `somefile` is the name of the file we wish to examine. Since the output can be quite long, it is often useful to view the result one page at a time. The following command will do this, examining the contents of the `hexdump.exe` executable program itself:

```
hexdump hexdump.exe | more
```

The `|` is called a "pipe." It is a way to string together two commands, taking the output of the first command and sending it as the input of the second command. The `more` command simply displays its input, one screenful at a time, waiting for the space bar to be hit before displaying the next page.

Notice that the HexDump output displays the hexdecmial values and the equivalent ASCII characters where they are printable. Most of these numbers are meaningless in this format, but we can see some readable text. In this case, most of the contents of the file corresponds to machine instructions that can execute on the Intel x86 family of processors. If nothing else, this output should convince you that a hexadecimal display is at least more concise than the equivalent binary representations.

We can look at files in a variety of formats:

- plain text: the file simply contains a sequence of ASCII codes.
- docx-format Word document: we can see some readable text, but not the text contained in our file!
- PDF (portable document format): we see some readable text in the file header.
- JPEG image: we see quite a bit of information in the file header about the specific format being used, the program that created the file, and some information about the image quality.

Let's look at one in a bit more detail: an MP3 audio file.

Looking at the HexDump output for readable text is not helpful here, but we can learn more about this file by looking up the details of the file format. In particular, we will look at the *frame header* of the first segment of the file. We can see the details of this header at `http://www.mp3-tech.org/programmer/frame_header.html`.

The frame header contains a variety of information, and we will see what is contained in this particular file's frame header.