The Dirac Video Codec: A Programmer's Guide

Scott R Ladd

August 13, 2005

In March 2004, BBC R&D released a prototype Open Source video coding algorithm named Dirac. This *codec* (*co*der/*dec*oder) takes a novel approach; while it has many similarities to existing algorithms such and MPEG-2 and H.264/AVC, Dirac is an original invention that uses wavelet transforms, arithmetic encoding, variable-block-size motion compensation, and hierarchical motion estimation.

The design of Dirac is modular, and thus well-suited to implementation with an object-oriented programming language. The reference engine is written in ISO Standard C++, with an eye toward portability and a correspondence between class structures and algorithm concepts. Since Dirac began as a research project, it has not yet been fully optimized for performance—however, its modular design provides many opportunities for improving both compression and algorithm speed.

This document is an overview of the Dirac algorithm, providing developers with the basic information required to understanding how the compression and decompression process works. As such, there is less reliance on mathematical theory and more focus on the conceptual basis of the codec. Prerequisites for understanding this material include practical knowledge of digital signal processing and data compression. A companion document describing the algorithm from a more mathematical perspective is available from `http://www.bbc.co.uk/rd/projects/dirac/documentation.shtml`.

The following web addresses also provide tutorial and educational material that lies outside the scope of this document:

- Digital Signal Processing Guru
  `http://www.dspguru.com/`
- The Scientist and Engineer's Guide to Digital Signal Processing
  `http://www.dspguide.com/`
- Interactive Data Compression Tutor
  `http://www.eee.bham.ac.uk/WoolleySI/All7/main.htm`
- ACM SIGGRAPH Education
  `http://www.siggraph.org/education`

The bibliography lists more resources that delve into the theoretical underpinnings of the algorithm.

1 Software Environment

The core Dirac codec is implemented in platform-neutral ISO Standard C++. The current encoder and decoder run from the command prompt and do not require specific operating system elements. [1]

The Dirac source code is distributed a compressed UNIX tarball, which can be downloaded from the Dirac web site at `http://sourceforge.net/projects/dirac/`. This tarball can be extracted with utilities such as the Unix **tar** command or the Windows WinZip utility.

1.1 GNU/Linux and Unix

Under GNU/Linux, Dirac has been successfully compiled and tested with the GNU g++ compiler (versions 3.3 and 3.4) and Intel's C++ compiler (versions 7.x and 8.0). Dirac has no dependencies on any libraries beyond the standard C and C++ function and class libraries included with your compiler.

For GNU/Linux systems, the Dirac distribution provides a GNU Autotools configure script. To build the codec and command-line utilities, follow the traditional sequence of commands:

```
./configure
make
make install
```

The last command must be executed with root privileges so that Dirac's libraries and programs will be installed to the appropriate directories. By default, the **dirac_encoder** and **dirac_decoder** programs will be installed to **/usr/local/bin**, while the shared and static libraries (described below) will be copied to **/usr/local/lib**. These default directories can be changed by providing options when invoking the **configure** script; the command **./configure –help** will list and explain these options.

1.2 Microsoft Windows

Under Microsoft Windows 2000 and XP Professional, Dirac has been successfully compiled and tested using Visual C++ (versions 12.1 and 13.0) and Intel C++ (versions 7.x and 8.0). You can build the Dirac command-line programs and static libraries using Microsoft **nmake** and the Makefile in the **codec/win**.

[1] Future enhancements to Dirac will included graphical user interfaces and other platform-specific features.

2 Source Code Organisation

Dirac is comprised of four libraries, each maintained in a separate subdirectory of the **codec** base directory. These libraries are:

- **libdirac_encoder**
  Compression classes for sequences, frames, and components.
- **libdirac_decoder**
  Decompression classes for sequences, frames, and components.
- **libdirac_motionest**
  Motion estimation classes used in the encoder.
- **libdirac_common**
  General-purpose classes used by both the encoder and decoder, including wavelet transforms, arithmetic encoding and decoding, and file I/O.

The command-line encoder and decoder also each reside in individual directories, and serve as drivers for the underlying library code.

The source code files contain one or more related classes. For example, **libdirac_common/wavelet_transform.h** and **.cpp** define classes related to the wavelet transform, including `WaveletTransform`, `WaveletTransformParameters`, `Subband`, and `SubbandList`.

3 Codec Concepts

The term *codec* derives from the phrase *co*der/*dec*oder; in essence, a codec translates information to and from an intermediate form. A video codec—such as Dirac—employs a intermediate form that compresses a stream of images for storage or rapid transmission.

Key issues related to video codecs include:

- amount of compression
- image quality
- computational complexity

An effective compression algorithm reflects the nature of the data being encoded. In the case of a text document, "image quality" is vitally important, requiring a lossless algorithm—a document wouldn;'t make much sense if letters or words went missing in the process of compression and decompression. For such *lossless* encoding, a codec uses a compression algorithm (e.g., Huffman encoding, arithmetic coding) that removes statistical redundancy from data, representing common values with short bit codes and uncommon values with longer codes. This style of algorithm works well when the decoded data must be identical to the source data.

A video stream contains considerable subjective redundancy—large areas often contain little or no visible detail, and changes between frames are often imperceptible. A video codec can afford to lose some video information in the compression process, so long as the data being "lost" is not noticed by a human viewer. The goal of a video codec (like Dirac) is to produce the smallest compressed form that can be decompressed into a stream that *appears* for a given loss when compared to the original images.

Video compression algorithms are based on two key techniques: *spatial compression* and *temporal compression*. Spatial compression relies upon removing redundancies within one frame of video or removing redundancies within small areas of a single frame. Temporal compression, on the other hand, relies upon similarity between successive pictures and performs compression utilizing prediction and motion compensation.

Many images or single video frames often contain sizable areas with the same single pixel value. This results in low spatial frequencies. And while high spatial frequency content exists in small detailed areas of the picture, there is typically only a small amount of energy at those frequencies. By transforming the image into the frequency domain, or something similar (such as the wavelet domain) these typical image characteristics can then be taken advantage of to achieve a compression gain. More specifically, in the frequency domain the high-frequency bands can be described with fewer bits because the amplitudes are smaller and because the human eye is less sensitive to noise in high frequencies.

Inter-frame compression takes advantage of the similarities between successive frames in a video stream. Instead of sending complete frames, the inter-

frame coders may only send the difference between the current frame and the previous frame in the form of differential coding.

One way Dirac differs from typical coding algorithms is in how it handles Inter-frame compression and the differences between frames.

A typical Inter-frame coding standard (such as MPEG-2) utilizes three types of frames:

- **I (Intra) frames**
  which are coded without reference to any other pictures
- **P (Predicted) frames**
  which are coded using motion compensation from a previous picture (I or P)
- **B (Bidirectionally predicted) frames**
  which are coded using interpolation from a previous and subsequent I or P picture

Instead of adopting this classification, Dirac has I, Layer 1 (L1) and Layer 2 (L2) frames. Inter frames can be L1 frames, which means that they can be used as a reference for other frames, or L2 frames, which can't. There is no assumption that L1 frames are forward predicted and L2 frames are bidirectionally predicted. Having said that, the current version of the code (0.1.0) implements a traditional GOP structure in which L1 frames are in fact P frames and L2 frames are B frames. However, in future these concepts will diverge.

In an Inter-frame coded video stream, the absolute image data (I-frames) are interleaved with frames containing difference data (inter frames). The I-frame and all the inter frames prior to the next I frame are called a Group Of Pictures (GOP). The larger the number of inter frames in a GOP, the higher is the achieved compression rate. However, a long GOP can cause a long delay before the picture can recover from a transmission error. Also, a long GOP makes video authoring and editing problematic, as the video stream can only be manipulated at I-frames.

In a case of a moving objects, the appearance of two successive frames can be similar, but the fixed sampling grid can cause large differences to be generated between two successive frames. Motion compensation techniques are used to remove this effect of the motion, so that the difference data will only reflect the changes in appearance of the moving object, not the change in location. The change is location is coded as a motion vector. The use of motion vectors can provide a significant additional compression gain.

In essence, encoding reads a byte stream and produces a coded bit-stream; during decoding, the process is reversed, and a coded bit stream is expanded into an output byte stream. This process is lossy, in that compression involves the loss of some information that can not be restored during decompression. The greater the level of compression, the more data that is lost in the coding process; a balance must therefore be struck between image quality and compression levels.

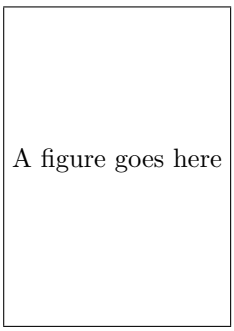

Figure 1: Video encoding and decoding with Dirac

Dirac differs from mainstream codecs that used in the main proprietary or standard video compression systems. Dirac seems to give a two-fold reduction in bit rate over MPEG-2 for high definition video (e.g. 1920x1080 pixels), its original target application. We have begun the process of optimising for Internet streaming resolutions and we aim to be at least broadly competitive with state of the art video codecs.

Figure 1 provides a simplified description of the steps involved in encoding and decoding data, as implemented by Dirac.

4 File Formats

At this time, Dirac supports YUV format streams. YUV is the format native to TV broadcast and composite video signals. It separates the brightness information (Y) from the colour information (U and V). The colour information consists essentially of red and blue colour difference signals; the green component can be reconstructed by subtracting the U and V components from the brightness component. Because the human eye is more sensitive to brightness than it is to colour, the chroma components generally are recorded with fewer samples than is the brightness data.

The supported YUV formats are:

ChromaFormat	*YUV format description*
Yonly	Gray-scale image, Y component only
format422	4:2:2, packed format, $\frac{1}{2}$ horizontal chroma resolution
format444	4:4:4, unpacked format, full chroma resolution
format420	4:2:0, packed format, $\frac{1}{2}$ horizontal/vertical resolution
format411	4:1:1, packed format, $\frac{1}{4}$ horizontal resolution
formatNK	unknown format

At this time, Dirac compresses and decompresses a proprietary BBC video format that is split into two files: a .hdr file containing header information, and a .yuv file with brightness and chroma data; it reads and writes encoded data as a bit stream file with the .drc extension.

Dirac does not support RGB format at this time.

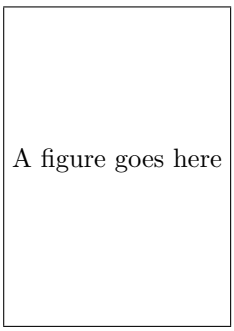

Figure 2: Encoder initialisation

5 Encoding Process

From a given base file name, the encoder creates a PicInput object that opens the .hdr and .yuv files. Based on command-line arguments, the encoder then encapsulates a set of encoding parameters in an EncoderParams object. If the encoder creates these objects successfully, it uses them in constructing a SequenceCompressor for managing the actual compression process.

5.1 SequenceCompressor

SequenceCompressor operates on "Groups of Pictures", or *gops*. Each gop is a sequence of *frames*, each of which has Y, U, and V components. When constructed, a gop creates internal buffers and maps the positions of frames in coded order to their positions in display order. In practice, a SequenceCompressor acts very much like an iterator.

It's important to realize that the order of frames as they are encoded may not be the same order in which they should be displayed. Thus SequenceCompressor::CompressNextFrame compresses the next frame in *coding order* and it returns the next frame in *display order*. It is the responsibility of the calling function to do 'something' with the coded frames after each call to CompressNextFrame—in most cases, this will be to call an output function to write the compressed frames to a file or a network stream. These are provided, for file IO purposes, by the PicInput and PicOutput classes, which should be subclassed for other types of IO.

5.2 Frames and Frame Compression

For each call to CompressNextFrame, a SequenceCompressor creates a FrameCompressor to code the next frame in sequence. The FrameCompressor is provided a reference to a gop and the index of a frame to code; it reads the requested frame from the gop, and applies the following process to it:

1. Motion estimation and motion vector voding

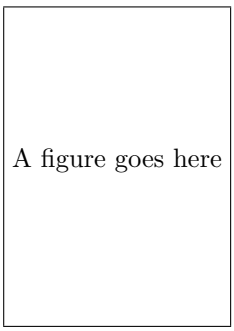

Figure 3: Motion estimation

2. Motion compensation
3. Compression of Y,U and V components, via a CompCompressor object, involving:
   (a) Wavelet transform
   (b) Entropy encoding via arithmetic coding
4. A second reverse motion compensation pass based on compressed components to reconstruct the compressed frame

The Frame object contains Y, U, and V components (unless it is a gray-scale image, in which case it has only the Y component) as separate arrays. When the frame is compressed, these arrays are replaced with compressed versions. It also contains 'upconverted' versions of these components - that is, arrays that are much larger versions of the original arrays, with values interpolated between the original values. These upconverted components are used for doing motion estimation and compensation to an accuracy finer than 1 pixel.

5.2.1 Motion Estimation and Compensation

Successive frames in a video sequence tend to be highly correlated—in other words, adjacent frames are very similar. Thus the difference between successive images is often very small, and a compression algorithm can take advantage of this by encoding differences between successive frames. Objects in motion increase the difference between frames, and decrease an algorithm's ability to compress successive images.

Motion estimation and compensation is the process whereby elements in a frame are correlated to elements in other frames by the estimated amount of change. Or, in simpler terms, motion estimation computes the difference between blocks of successive frames, creating values that describe the amount of change that occurs. The motion vector can then be compressed using the same arithmetic coding technique used for encoding wavelet transformation coefficients.

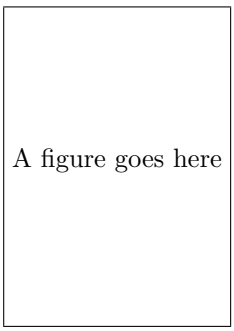

Figure 4: Wavelet transform coefficient packing

Dirac performs motion estimation using a set of Lagrangian parameters defined by the type of video being compressed (e.g., high-definition, standard definition, etc.) A MotionEstimator object computes the motion vectors, which are then compressed using arithmetic coding via an MvDataCodec object. For L1- and L2-frames, motion estimation and compensation occurs before component compression, and another motion compensation pass is performed after component compression to add back in the reference data that was subtracted to produce the motion-compensated difference.

5.2.2 Wavelet Transform

The CompCompressor class uses a wavelet transform (defined by a WaveletTransform object) to map component data from the sample domain into the frequency domain. Component data is stored by a frame in a 2D array; the wavelet transform is an iterated, reversible series of low- and high-pass filters that pack data into the source array, as shown in Figure 3. Each rectangular region represents a set of 2D wavelet coefficients in the same sub-band, and each is encoded separately by the encoder. Metadata for each sub-band, including its coordinates in within the 2D array, is stored in the Subband class structure. The wavelet transform produces a sub-band list containing all sub-bands.

The purpose of the wavelet transform is to 'decorrelate' the input data, reducing the number of coefficients required to describe the image: it provides a more compact representation. This is one way in which Dirac differs from standards such as JPEG and MPEG-2, which use a Discrete Cosine Transform, although JPEG2000 uses wavelet technology. Dirac's wavelet transform is implemented using a 'lifting scheme', which factors the filters into a number of short filters; this technique is faster than filtering in a straight-forward fashion.

5.2.3 Arithmetic Encoding

After application of the wavelet transform, the CompCompressor::Compress function loops through all sub-bands in reverse order, from top-left to bottom

right, compressing each one via arithmetic encoding through a BandCodec object. BandCodec and MvDataCodec derive from ArithCodec, which provides the core arithmetic coding routines.

Before each symbol is fed into the arithmetic coding process, it is approximated, effectively by dividing by some number (a quantisation factor) and throwing away the remainder; this process is called quantisation. This makes the wavelet representation even more compact at the expense of some loss in quality: with luck this loss is imperceptible.

Arithmetic encoding (like the more commonly-known Huffman encoding) is an example of entropy encoding: an algorithm that assigns codes to symbols in such a fashion as to represent the most common symbols with the shortest codes. Motion compensation and wavelet transformation reduce the amount of information in a frame; quantisation reduces it still further, and arithmetic coding then encodes that data in a reversible way.

6 Conclusions

The Dirac codec is a novel approach to high-quality video compression, combining several well-known techniques into a flexible and effective algorithm. The use of C++ provides an object-oriented framework that can be extended with additional features. Future directions include optimisation of the code base, the use of fixed-point math in the wavelet transform, and parallelisation to take advantage of multiprocessor systems for encoding.