Easy Theora Programming With Etheora.

Ribamar Santarosa, ribamar@gmail.com

October 2007

Etheora is a C library aimed to provide a straightforward API for programming applications that encodes or decode theora videos, using ogg as container. Etheora users don't need to know nothing about the libtheora or libogg (and derivatives) API.

*Etheora **doesn't** have audio/speech support yet. Although with etheora it's possible to get the video data from a video that contains audio/speech, audio/speech data will be unavailable.*

1 Installing.

No etheora installation. Just:

1. download the three files (`etheora.c`, `etheora.h`, `etheora-int.h`),
2. put the `.h` files in an include-findable directory (or tell your compiler to search the download directory, e.g, `-Idownload_dir` in gcc)),
3. and add `etheora.c` to your build project (e.g. add `download_dir/etheora.c` to gcc command-line).

But libtheora (development version, if your system can tell) is required to be present in the system. Etheora will work in the systems where libtheora works.

2 Encoding.

Encoding steps:

1. Declare an etheora context structure,

   ```
   etheora_ctx ec;
   ```

2. Configure the encoder with `etheora_enc_setup()`, e.g,

   ```
   etheora_enc_setup(&ec, 640, 480, ETHEORA_ASPECT_NORMAL,
                     12, 1, fout, finfo);
   ```

   will setup to encode a 640:480, 12/1 frames per second video, with 4:3 aspect ratio. You may change to `ETHEORA_ASPECT_WIDE_SCREEN` to get a 16:9 aspect ratio or `ETHEORA_ASPECT_PRESERVE` to preserve the width:height ratio. In this case, $640/480 = 4/3$, the latter wouldn't produce difference.

   The encoded video will be written to the file descriptor `FILE* fout`. Debug info will be written to `FILE* finfo`. You may want set this as your system's null device file (e.g. `/dev/null`) as a large amount of information can be printed.

3. The unnexperienced theora user may jump this step. The experienced libtheora user has now a last chance to change `theora_info` and `theora_comment` values before encoding:

   ```
   char *vendor = "Vendor name";
   ec.ti.target_bitrate = 200000;
   ec.tc.vendor = vendor;
   /*etc*/
   ```

   Or just maintain the default values assigned by etheora when configuring.

4. Start the encoder engine:

   ```
   etheora_enc_start(&ec);
   ```

5. Draw a frame. For e.g, to draw in YUV colorspace,

   ```
   etheora_enc_yuv_draw(&ec, i, j, y_value, u_value, v_value);
   ```

   will draw the pixel in the frame coordinate (i, j).

   Although libtheora deals with frames in YUV, etheora supports transparent drawing in RGB with another function:

   ```
   etheora_enc_rgb_draw(&ec, i, j, r_value, g_value, b_value);
   ```

   Other colorspace may be available in the future.

   For the experienced theora user: etheora supports `yuv_buffer`s of the kinds `OC_PF_420`, `OC_PF_422` and `OC_PF_444` in despite of your libtheora version supporting them or not. Access to these frame buffers is transparent, using the functions above. `etheora_enc_setup()` configures the encoder to use `OC_PF_420` which is the only one known to be supported by any libtheora version. If you're dealing with different chromas, you may be interested in the function `etheora_resample(yuv_buffer *source, yuv_buffer *dest)` for converting them.

6. If the frame you've just drawn *isn't* the last one you want to encode, submit it to encoding by calling:

   ```
   etheora_enc_nextframe(&ec);
   ```

   Go back to draw more frames!

7. If the frame you've just drawn *is* the last one, you may submit it by finishing the decoding process:

   ```
   etheora_enc_finish(&ec);
   ```

And we're done.

3 Decoding.

Decoding steps:

1. Declare an etheora context structure,

   ```
   etheora_ctx ec;
   ```

2. Configure the decoder with `etheora_dec_setup()`, e.g,

```
etheora_dec_setup(&ec, fin, finfo);
```

The video will be read from the file descriptor `FILE* fin`. Debug info will be written to `FILE* finfo`. You may want set this as your system's null device file (e.g. `/dev/null`) as a large amount of information can be printed.

3. Start the encoder engine:

```
etheora_dec_start(&ec);
```

4. Video info can now be known and shown:

```
printf("video dimensions: %u:%u\n", etheora_get_width(&ec),
                                    etheora_get_height(&ec));
printf("frame rate : %u:%u fps\n", etheora_get_fps_numerator(&ec),
                                   etheora_get_fps_denominator(&ec));
printf("aspect ratio: %u:%u\n", etheora_get_aspect_numerator(&ec),
                                etheora_get_aspect_denominator(&ec));
```

The experienced libtheora user can read `theora_info` and `theora_comment` values:

```
printf("video target bitrate: %i\n", ec.ti.target_bitrate);
printf("vendor name: %s\n", ec.tc.vendor);
/*etc*/
```

5. Loop to obtain all frames. When `etheora_dec_nextframe()` returns *not zero* it can't read more data.

```
while(!etheora_dec_nextframe(&ec)){
            ...
}
```

6. After decoding a frame with `etheora_dec_nextframe()`, frame data is available to be read. To read using YUV colorspace:

```
etheora_dec_yuv_read(&ec, i, j, y_value, u_value, v_value);
```

will draw the pixel in the frame coordinate (i, j).

Although libtheora deals with frames in YUV, etheora supports transparent reading in RGB with another function:

```
etheora_enc_rgb_read(&ec, i, j, r_value, g_value, b_value);
```

Other colorspace may be available in the future.

For the experienced user, the same note shown when `etheora_enc_yuv_draw()` and `etheora_enc_rgb_draw()` were presented, in the section **Encoding**, is valid.

7. When the decoding loop finishes, you may free the structures with

```
etheora_dec_finish(&ec);
```

And we're done.

4 Examples/Building Examples.

3 simple examples can be downloaded: `encoder-example.c`, `decoder-example.c` and `decoder-example-opencv.c`

4.1 encoder-example.c

A very simple encoder that allows you to generate videos with mathematical functions.

Using gcc, you can build the encoder-example with this command:

```
gcc encoder-example.c etheora.c  -I. -ltheora -o encoder-example
```

Assuming etheora files are in the same directory as `encoder-example.c`.

Then, this command:

```
./decoder-example output-video.ogg
```

will generate a video and write it to the `output-video.ogg`.

You can open `encoder-example.c` and play with the numeric parameters of `etheora_enc_rgb_draw()`, or use `etheora_enc_yuv_draw()` instead. Recompile the encoder and get different videos.

4.2 decoder-example.c

A very simple decoder that does nothing but decoding video. It allows you to read the pixels of decoded frames.

Using gcc, you can build the decoder-example with this command:

```
gcc decoder-example.c etheora.c  -I. -ltheora -o decoder-example
```

Assuming etheora files are in the same directory as `decoder-example.c`.

Then, this command:

```
./decoder-example input-video.ogg
```

will read a video from `input-video.ogg`.

4.3 decoder-example-opencv.c

A very simple decoder that decodes frames and draws them in a opencv/highgui window. It doesn't have a good performance, it's only to provide a visible result of the decoding process. Also it doesn't deal with the correct video frame rate, it just waits for $100\,ms$ after drawing a frame. Requires, of course, opencv/highgui present in the system.

Using gcc, you can build the encoder-example with this command:

```
gcc -Wall decoder-example-opencv.c etheora.c  \
  -lhighgui -lstdc++  -I.  -ltheora -o decoder-example-opencv
```

Assuming etheora files are in the same directory as `decoder-example-opencv.c`.

Then, this command:

```
./decoder-example-opencv input-video.ogg
```

will read a video from `input-video.ogg` and draw in a opencv/highgui window:

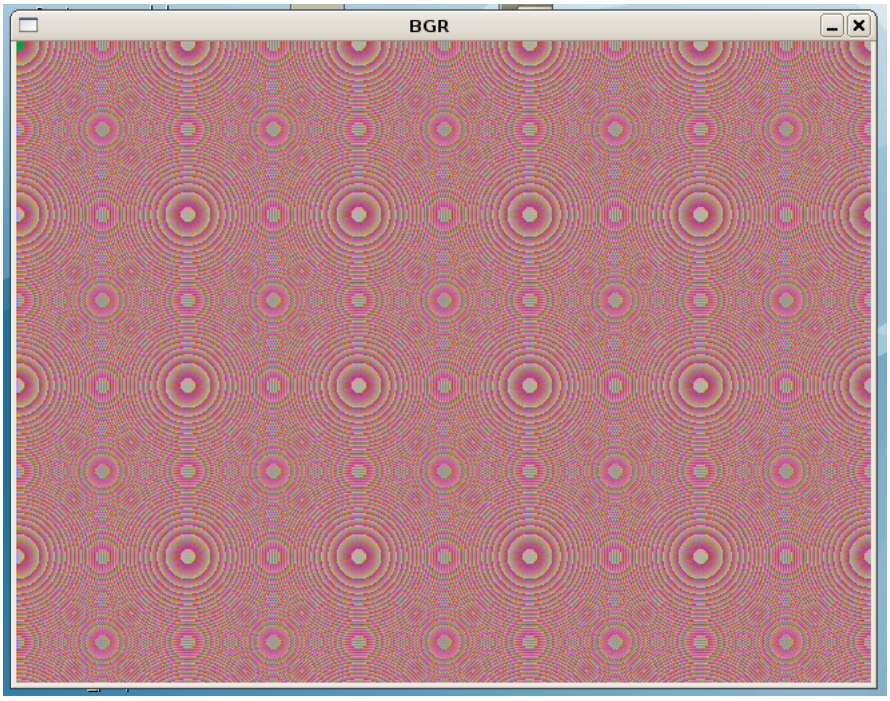

Figure 1: OpenCV/Highgui window for decoding theora videos.