原 Tiny Jpeg Decoder （JPEG解码程序） 源代码分析 1：解码文件头

2013年10月11日 15:48:08　阅读数：10057

注：分析Tiny Jpeg Decoder源代码的文章：

Tiny Jpeg Decoder （JPEG解码程序） 源代码分析 1：解码文件头

Tiny Jpeg Decoder （JPEG解码程序） 源代码分析 2：解码数据

==========

Tiny Jpeg Decoder是一个可以用于嵌入式系统的JPEG解码器。也可以在Windows上编译通过。在此分析一下它部分的源代码，辅助学习JPEG解码知识。

通过TinyJpeg可以将JPEG（*.jpg）文件解码为YUV（*.yuv）或者RGB（*.tga）文件。

真正的解码开始于 **convert_one_image()** 函数：

```cpp
/**
 * Load one jpeg image, and decompress it, and save the result.
 */
int convert_one_image(LPVOID lparam,const char *infilename, const char *outfilename, int output_format)
{
  FILE *fp;
  unsigned int length_of_file;
  unsigned int width, height;
  unsigned char *buf;
  struct jdec_private *jdec;
  unsigned char *components[3];

  /* Load the Jpeg into memory */
  fp = fopen(infilename, "rb");
  if (fp == NULL)
    exitmessage("Cannot open filename\n");
  length_of_file = filesize(fp);
  buf = (unsigned char *)malloc(length_of_file + 4);
  if (buf == NULL)
    exitmessage("Not enough memory for loading file\n");
  fread(buf, length_of_file, 1, fp);
  fclose(fp);

  /* Decompress it */
  //分配内存
  jdec = tinyjpeg_init();
  //传入句柄----------·
  jdec->dlg=(CSpecialVIJPGDlg *)lparam;

  if (jdec == NULL)
    exitmessage("Not enough memory to alloc the structure need for decompressing\n");
  //解头部
  if (tinyjpeg_parse_header(jdec, buf, length_of_file)<0)
    exitmessage(tinyjpeg_get_errorstring(jdec));
  /* Get the size of the image */
  //获得图像长宽
  tinyjpeg_get_size(jdec, &width, &height);

  snprintf(error_string, sizeof(error_string),"Decoding JPEG image...\n");
  //解码实际数据
  if (tinyjpeg_decode(jdec, output_format) < 0)
    exitmessage(tinyjpeg_get_errorstring(jdec));
  /*
   * Get address for each plane (not only max 3 planes is supported), and
   * depending of the output mode, only some components will be filled
   * RGB: 1 plane, YUV420P: 3 planes, GREY: 1 plane
   */
  tinyjpeg_get_components(jdec, components);

  /* Save it */
  switch (output_format)
   {
    case TINYJPEG_FMT_RGB24:
    case TINYJPEG_FMT_BGR24:
      write_tga(outfilename, output_format, width, height, components);
      break;
    case TINYJPEG_FMT_YUV420P:
      //开始写入成YUV420P文件
      write_yuv(outfilename, width, height, components);
      break;
    case TINYJPEG_FMT_GREY:
      //开始写入成灰度文件
      write_pgm(outfilename, width, height, components);
      break;
   }

  /* Only called this if the buffers were allocated by tinyjpeg_decode() */
//modify by lei!  tinyjpeg_free(jdec);
  /* else called just free(jdec); */

  free(buf);
  return 0;
}
```

tinyjpeg_init() 用于初始化：

```cpp
/**
 * Allocate a new tinyjpeg decoder object.
 *
 * Before calling any other functions, an object need to be called.
 */
struct jdec_private *tinyjpeg_init(void)
{
  struct jdec_private *priv;

  priv = (struct jdec_private *)calloc(1, sizeof(struct jdec_private));
  if (priv == NULL)
    return NULL;
  priv->DQT_table_num=0;
  return priv;
}
```

tinyjpeg_parse_header() 用于解码JPEG文件头，可见函数前几句主要验证文件是否为JPEG文件：

```cpp
/**
 * Initialize the tinyjpeg object and prepare the decoding of the stream.
 *
 * Check if the jpeg can be decoded with this jpeg decoder.
 * Fill some table used for preprocessing.
 */
int tinyjpeg_parse_header(struct jdec_private *priv, const unsigned char *buf, unsigned int size)
{
  int ret;

  /* Identify the file */
  //0x FF D8
  //是否是JPEG格式文件？
  if ((buf[0] != 0xFF) || (buf[1] != SOI))
    snprintf(error_string, sizeof(error_string),"Not a JPG file ?\n");
  //是
  char temp_str[MAX_URL_LENGTH];
  sprintf(temp_str,"0x %X %X",buf[0],buf[1]);
  //JPEG格式文件固定的文件头
  //begin指针前移2字节
  priv->stream_begin = buf+2;
  priv->stream_length = size-2;
  priv->stream_end = priv->stream_begin + priv->stream_length;
  //开始解析JFIF
  ret = parse_JFIF(priv, priv->stream_begin);
  return ret;
}
```

parse_JFIF() 用于解析各种标签（SOF，SOS，DHT...）：

```cpp
//解各种不同的标签
static int parse_JFIF(struct jdec_private *priv, const unsigned char *stream)
{
  int chuck_len;
  int marker;
  int sos_marker_found = 0;
  int dht_marker_found = 0;
  const unsigned char *next_chunck;

  /* Parse marker */
  //在Start of scan标签之前
  while (!sos_marker_found)
   {
     if (*stream++ != 0xff)
       goto bogus_jpeg_format;
     /* Skip any padding ff byte (this is normal) */
     //跳过0xff字节
     while (*stream == 0xff)
       stream++;
     //marker是跳过0xff字节后1个字节的内容
     marker = *stream++;
     //chunk_len是marker后面2个字节的内容（大端模式需要转换）
     //包含自身，但不包含0xff+marker2字节
     chuck_len = be16_to_cpu(stream);
     //指向下一个chunk的指针
     next_chunck = stream + chuck_len;
     //各种不同的标签
     switch (marker)
      {
        case SOF:
     //开始解析SOF
     if (parse_SOF(priv, stream) < 0)
       return -1;
     break;
```

```
      //Define quantization table
        case DQT:
      //开始解析DQT
      if (parse_DQT(priv, stream) < 0)
        return -1;
      break;
        case SOS:
      //开始解析SOS
      if (parse_SOS(priv, stream) < 0)
        return -1;
      sos_marker_found = 1;
      break;
      //Define Huffman table
        case DHT:
      //开始解析DHT
      if (parse_DHT(priv, stream) < 0)
        return -1;
      dht_marker_found = 1;
      break;
        case DRI:
      //开始解析DRI
      if (parse_DRI(priv, stream) < 0)
        return -1;
      break;
        default:
#if TRACE_PARAM
    fprintf(param_trace,"> Unknown marker %2.2x\n", marker);
    fflush(param_trace);
#endif
     break;
       }
      //解下一个segment
      stream = next_chunck;
   }

  if (!dht_marker_found) {
#if TRACE_PARAM
      fprintf(param_trace,"No Huffman table loaded, using the default one\n");
      fflush(param_trace);
#endif
    build_default_huffman_tables(priv);
  }

#ifdef SANITY_CHECK
  if (   (priv->component_infos[cY].Hfactor < priv->component_infos[cCb].Hfactor)
      || (priv->component_infos[cY].Hfactor < priv->component_infos[cCr].Hfactor))
    snprintf(error_string, sizeof(error_string),"Horizontal sampling factor for Y should be greater than horitontal sampling factor f
 Cb or Cr\n");
  if (   (priv->component_infos[cY].Vfactor < priv->component_infos[cCb].Vfactor)
      || (priv->component_infos[cY].Vfactor < priv->component_infos[cCr].Vfactor))
    snprintf(error_string, sizeof(error_string),"Vertical sampling factor for Y should be greater than vertical sampling factor for C
or Cr\n");
  if (   (priv->component_infos[cCb].Hfactor!=1)
      || (priv->component_infos[cCr].Hfactor!=1)
      || (priv->component_infos[cCb].Vfactor!=1)
      || (priv->component_infos[cCr].Vfactor!=1))
    snprintf(error_string, sizeof(error_string),"Sampling other than 1x1 for Cr and Cb is not supported");
#endif

  return 0;
bogus_jpeg_format:
#if TRACE_PARAM
  fprintf(param_trace,"Bogus jpeg format\n");
  fflush(param_trace);
#endif
  return -1;
}
```

parse_SOF() 用于解析SOF标签：

注意：其中包含了部分自己写的代码，形如：

```cpp
itoa(width,temp_str1,10);
  priv->dlg->AppendBInfo("SOF0","宽",temp_str1,"图像的宽度");
```

这些代码主要用于在解码过程中提取一些信息，比如图像宽，高，颜色分量数等等

```cpp
static int parse_SOF(struct jdec_private *priv, const unsigned char *stream)
{
  int i, width, height, nr_components, cid, sampling_factor;
  int Q_table;
  struct component *c;
#if TRACE_PARAM
  fprintf(param_trace,"> SOF marker\n");
  fflush(param_trace);
#endif
  print_SOF(stream);

  height = be16_to_cpu(stream+3);
  width  = be16_to_cpu(stream+5);
  nr_components = stream[7];
#if SANITY_CHECK
  if (stream[2] != 8)
    snprintf(error_string, sizeof(error_string),"Precision other than 8 is not supported\n");
  if (width>JPEG_MAX_WIDTH || height>JPEG_MAX_HEIGHT)
    snprintf(error_string, sizeof(error_string),"Width and Height (%dx%d) seems suspicious\n", width, height);
  if (nr_components != 3)
    snprintf(error_string, sizeof(error_string),"We only support YUV images\n");
  if (height%16)
    snprintf(error_string, sizeof(error_string),"Height need to be a multiple of 16 (current height is %d)\n", height);
  if (width%16)
    snprintf(error_string, sizeof(error_string),"Width need to be a multiple of 16 (current Width is %d)\n", width);
#endif
  char temp_str1[MAX_URL_LENGTH]={0};
  itoa(width,temp_str1,10);
  priv->dlg->AppendBInfo("SOF0","宽",temp_str1,"图像的宽度");
  itoa(height,temp_str1,10);
  priv->dlg->AppendBInfo("SOF0","高",temp_str1,"图像的高度");
  itoa(nr_components,temp_str1,10);
  priv->dlg->AppendBInfo("SOF0","颜色分量数",temp_str1,"图像的颜色分量数。一个字节，例如03，代表有三个分量，YCrCb");
  itoa(stream[2],temp_str1,10);
  priv->dlg->AppendBInfo("SOF0","精度",temp_str1,"图像的精度。一个字节，例如08，即精度为一个字节。");
  stream += 8;
  for (i=0; i<nr_components; i++) {
     cid = *stream++;
     sampling_factor = *stream++;
     Q_table = *stream++;
     c = &priv->component_infos[i];
#if SANITY_CHECK
     c->cid = cid;
     if (Q_table >= COMPONENTS)
       snprintf(error_string, sizeof(error_string),"Bad Quantization table index (got %d, max allowed %d)\n", Q_table, COMPONENTS-1)
;
#endif
     c->Vfactor = sampling_factor&0xf;
     c->Hfactor = sampling_factor>>4;
     c->Q_table = priv->Q_tables[Q_table];
     //------------
     char temp_str2[MAX_URL_LENGTH]={0};
     sprintf(temp_str2,"垂直采样因子【%d】",i);
     itoa(c->Hfactor,temp_str1,10);
     priv->dlg->AppendBInfo("SOF0",temp_str2,temp_str1,"颜色分量信息：每个分量有三个字节，第一个为分量的ID，01：Y 02：U 03：V；第二个字节高位为水平采样因子，低位为垂直采样因子。");
     sprintf(temp_str2,"水平采样因子【%d】",i);
     itoa(c->Hfactor,temp_str1,10);
     priv->dlg->AppendBInfo("SOF0",temp_str2,temp_str1,"颜色分量信息：每个分量有三个字节，第一个为分量的ID，01：Y 02：U 03：V；第二个字节高位为水平采样因子，低位为垂直采样因子。");
     sprintf(temp_str2,"对应量化表ID【%d】",i);
     itoa((int)Q_table,temp_str1,10);
     priv->dlg->AppendBInfo("SOF0",temp_str2,temp_str1,"颜色分量信息：第三个字节代表这个分量对应的量化表ID，例如，Y对应的量化表ID索引值为00,而UV对应的量化表ID都为01，即它们共用一张量化表。");
    //------------
#if TRACE_PARAM
     fprintf(param_trace,"Component:%d  factor:%dx%d  Quantization table:%d\n",
          cid, c->Hfactor, c->Hfactor, Q_table );
     fflush(param_trace);
#endif

  }


  priv->width = width;
  priv->height = height;
#if TRACE_PARAM
  fprintf(param_trace,"< SOF marker\n");
  fflush(param_trace);
#endif

  return 0;
}
```

parse_DHT() 用于解析DHT标签：

```cpp
//解析DHT表
```

```
//解析DHT表
static int parse_DHT(struct jdec_private *priv, const unsigned char *stream)
{
  unsigned int count, i,j;
  unsigned char huff_bits[17];
  int length, index;
  //----------------------------------------
  char *temp;
  FILE *fp;
  //----------------------------------------
  length = be16_to_cpu(stream) - 2;
  //跳过length字段
  stream += 2;  /* Skip length */
#if TRACE_PARAM
  fprintf(param_trace,"> DHT marker (length=%d)\n", length);
  fflush(param_trace);
#endif

  while (length>0) {
      //跳过第1字节:
      //Huffman 表ID号和类型，高 4 位为表的类型，0：DC 直流；1：AC交流
      //低四位为 Huffman 表 ID。
     index = *stream++;

     /* We need to calculate the number of bytes 'vals' will takes */
     huff_bits[0] = 0;
     count = 0;

     //不同长度 Huffman 的码字数量：固定为 16 个字节，每个字节代表从长度为 1到长度为 16 的码字的个数
     for (i=1; i<17; i++) {
    huff_bits[i] = *stream++;
    //count记录码字的个数
    count += huff_bits[i];
     }
#if SANITY_CHECK
     if (count >= HUFFMAN_BITS_SIZE)
       snprintf(error_string, sizeof(error_string),"No more than %d bytes is allowed to describe a huffman table", HUFFMAN_BITS_SIZE
);
     if ( (index &0xf) >= HUFFMAN_TABLES)
       snprintf(error_string, sizeof(error_string),"No more than %d Huffman tables is supported (got %d)\n", HUFFMAN_TABLES, index&0x
f);
#if TRACE_PARAM
     fprintf(param_trace,"Huffman table %s[%d] length=%d\n", (index&0xf0)?"AC":"DC", index&0xf, count);
     fflush(param_trace);
#endif
#endif

     if (index & 0xf0 ){
         //--------------------
         char temp_str1[MAX_URL_LENGTH]={0};
         char temp_str2[MAX_URL_LENGTH]={0};
         temp=(char *)stream;
         //fp = fopen("DHT.txt", "a+");
         //fwrite(temp, 16, 1, fp);
         for(j=0;j<16;j++){
             //fprintf(fp,"%d ",temp[j]);
             sprintf(temp_str2,"%d ",temp[j]);
             strcat(temp_str1,temp_str2);
         }
         //fprintf(fp,"\n----------------------\n");
         //fclose(fp);
         //----------------------------------------------------
         priv->dlg->AppendBInfo("DHT","定义霍夫曼表【交流系数表】",temp_str1,"Huffman表ID号和类型：1字节，高4位为表的类型，0：DC直流；1：AC交流
可以看出这里是直流表；低四位为Huffman表ID");
         //----------------------------------------------------
    //交流霍夫曼表
       build_huffman_table(huff_bits, stream, &priv->HTAC[index&0xf]);
     }
     else{
         //--------------------
         char temp_str1[MAX_URL_LENGTH]={0};
         char temp_str2[MAX_URL_LENGTH]={0};
         temp=(char *)stream;
         //fp = fopen("DHT.txt", "a+");
         //fwrite(temp, 16, 1, fp);
         for(j=0;j<16;j++){
             //fprintf(fp,"%d ",temp[j]);
             sprintf(temp_str2,"%d ",temp[j]);
             strcat(temp_str1,temp_str2);
         }
         //fprintf(fp,"\n----------------------\n");
         //fclose(fp);
         //----------------------------------------------------
         priv->dlg->AppendBInfo("DHT","定义霍夫曼表【直流系数表】",temp_str1,"Huffman表ID号和类型：1字节，高4位为表的类型，0：DC直流；1：AC交流
可以看出这里是直流表；低四位为Huffman表ID");
         //----------------------------------------------------
         //直流霍夫曼表
       build_huffman_table(huff_bits, stream, &priv->HTDC[index&0xf]);
     }

     length -= 1;
     length -= 16;
```

```
      length -= 16;
      length -= count;
      stream += count;
    }
  #if TRACE_PARAM
    fprintf(param_trace,"< DHT marker\n");
    fflush(param_trace);
  #endif
    return 0;
  }
```

parse_DQT() 用于解析DQT标签：

```cpp
//解析Define quantization table
static int parse_DQT(struct jdec_private *priv, const unsigned char *stream)
{
  int qi;
  float *table;
  const unsigned char *dqt_block_end;

//------------------------------------------------
    int j,k;
    char *temp;
    FILE *fp;
//------------------------------------------------
#if TRACE_PARAM
  fprintf(param_trace,"> DQT marker\n");
  fflush(param_trace);
#endif
  //该Segment末尾
  dqt_block_end = stream + be16_to_cpu(stream);
  //跳过标签length（2字节）
  stream += 2;  /* Skip length */
  //没到末尾
  while (stream < dqt_block_end)
   {
    //跳过该Segment的第1个字节,QT信息
    //precision: 00  (Higher 4 bit)
    //index: 00  (Lower 4 bit)
    //qi取1，第1张量化表（例如，亮度表），取2，第2张量化表（例如，色度表）
     qi = *stream++;
#if SANITY_CHECK
     if (qi>>4)
       snprintf(error_string, sizeof(error_string),"16 bits quantization table is not supported\n");
     if (qi>4)
       snprintf(error_string, sizeof(error_string),"No more 4 quantization table is supported (got %d)\n", qi);
#endif
     //table指向jdec_private的Q_tables数组，为了在其中写入数据
     table = priv->Q_tables[qi];
     //注意：一次搞定整张！写入
     //需要对数值进行变换！cos(k*PI/16) * sqrt(2)
     //这样才能得到离散余弦变换的系数
     build_quantization_table(table, stream);
//---------------------------------------------------------

        temp=(char *)stream;
        //fp = fopen("DQT.txt", "a+");
        //fwrite(temp, 64, 1, fp);
        char temp_str1[MAX_URL_LENGTH]={0};
        char temp_str2[MAX_URL_LENGTH]={0};
        for(j=0;j<64;j++){
            sprintf(temp_str2,"%d ",temp[j]);
            strcat(temp_str1,temp_str2);
            //fprintf(fp,"%d ",temp[j]);
        }
        //计数
        char temp_str3[MAX_URL_LENGTH]={0};
        sprintf(temp_str3,"量化表【%d】",priv->DQT_table_num);
        priv->dlg->AppendBInfo("DQT",temp_str3,temp_str1,"JPEG格式文件的量化表，一般来说第一张是亮度的，后面是色度的");
        priv->DQT_table_num++;
        //fprintf(fp,"\n----------------------\n");
        //fclose(fp);
#if TRACE_PARAM
        for(j=0;j<8;j++){
            for(k=0;k<8;k++){
                fprintf(param_trace,"%d ",temp[j*8+k]);
            }
            fprintf(param_trace,"\n");
        }
        fprintf(fp,"\n----------------------\n");
     fflush(param_trace);
#endif
//---------------------------------------------------------
     //完事了！
     stream += 64;
   }
#if TRACE_PARAM
  fprintf(param_trace,"< DQT marker\n");
  fflush(param_trace);
#endif
  return 0;
}
```

其他标签的解析不一一列举。

待续未完。。。

主页：http://www.saillard.org/programs_and_patches/tinyjpegdecoder/

源代码下载：http://download.csdn.net/detail/leixiaohua1020/6383115

文章标签：jpeg 解码 源代码 tinyjpeg

个人分类：TinyJPEG

所属专栏：开源多媒体项目源代码分析