Universal LLVS Plane File Format

Robert Heller

April 30, 1993

Abstract

This document describes the format of LLVS plane files in a multi-architecture environment. The plane files support different byte orders and different floating point formats. A given machine will write planes using its "native" format and read planes written by other machines,

doing whatever converesions needed to produce an in-core representation of the data in native form.

1 Introduction

This manual describes the format used by the VISIONS group to store image data. Image data is stored as "plane" files, each of which holds one or more image planes. Image planes are two dimensional arrays of picture elements, know as "pixels". There are five types of planes: bit (one bit per pixel), unsigned byte (8 bits per pixel), short integers (16-bits, signed, per pixel), integer (32-bits, signed, per pixel), and floating (32-bit floating point). All of the pixels in a given plane are of the same type. Two data structures are used to hold information about a plane while it is in main memory: a `PLANE` structure (which points at the pixels themselves) and a `PLANE_INFO` structure which describes the plane. These structures are defined as follows:[1]

```
/*------------------------------------------------------
 *
 *  PLANE
 *    This structure describes the format of a plane passed
 *  to C. Each plane is passed to the C routine as a separate
 *  argument of type PLANE.
```

[1] From "llvs_per_plane.h"

```
 *
 *-------------------------------------------------------
 */

typedef struct {
    int plane_base[1];       /* plane data starts here */
    } PLANE;

/*-----------------------------------------------------------
 *
 *  PLANE_INFO
 *   Other information is available for each plane. This
 *  information is passed in an array of PLANE_INFO[MAXPLANE].
 *  The Lisp function BUILD-PLANE-INFO-VECTOR-C may be used to
 *  build the array. The background-value is passed as a C
 *  float value if the plane is a floating point plane, C int
 *  type otherwise.
 *
 *----------------------------------------------------------
```

```
 */

/* plane info struct */
typedef struct {
    long int datatype;      /* data type */
#define LLVS_BIT   0        /* bit type */
#define LLVS_BYTE  1        /* unsigned byte type */
#define LLVS_SHORT 2        /* signed 16 bit type */
#define LLVS_INT   3        /* integer type */
#define LLVS_FLOAT 4        /* float type */
#define FLOAT 4             /* float type */
    long int level;         /* plane level */
    long int row_location;  /* row location */
    long int column_location; /* column location */
    long int row_dimension; /* row dimension */
    long int column_dimension; /* column dimension */
    union {
        long int fixnum;    /* background value */
        float flonum;       /* dito, but as a flonum */
```

```
} background;

} PLANE_INFO;
```

Additional information about planes are stored in a Common LISP association list[2]. The association list is used to store information such as the plane statisics (minimum value, maximum value, etc.) and other user-defined information.

When image planes are written to disk files, this information needs to be preserved in a form that can be read on many different systems, since the one thing all of the various machines have in common is a file system. Planes are written to disk files as four logical records: a plane file header record, an association list record, a plane size record, and a plane data record. The plane file header record is a fixed-length record containing 32 bytes. It defines the plane's type and contains information about how the plane was written (byte sex[3], floating point format used), the plane's size, its location, its relative resolution, etc. The association list record contains the plane's association list as ASCII text. The plane size record defines the plane's dimensions. And

[2] *Common LISP The language*, Guy L. Steele Jr., pp 279-281

[3] byte storage order

finally, the plane data record contains the pixel values themselves.

2 The Plane File Header

This structure is defined in the file `llvs_plane.h`.

```
/* some basic types: */

typedef unsigned char llvs_ubyte;    /* unsigned bytes
                                        (8-bits) */
typedef long int llvs_integer;       /* long integer
                                         (32-bits) */
typedef float llvs_single_float;     /* 32-bit floating
                                        point */
typedef short int llvs_half_integer; /* short integer
                                         (16-bits) */

/* primary header record: LLVS Plane files start with this */

typedef struct {
```

```
    llvs_ubyte ptype;          /* datatype of plane */
#define LLVS_PLF_BIT   0       /* bit type */
#define LLVS_PLF_BYTE  1       /* unsigned byte type */
#define LLVS_PLF_SHORT 2       /* signed 16 bit type */
#define LLVS_PLF_INT   3       /* integer type */
#define LLVS_PLF_FLOAT 4       /* float type */
    llvs_ubyte bsex;           /* byte sex (byte order) */
#define LLVS_LOW_BYTE_FIRST 0  /* low byte first
                                  (VAX, 80x86) */
#define LLVS_HIGH_BYTE_FIRST 1 /* high byte first
                                 (680x0, Spark) */
    llvs_ubyte floatfmt;       /* floating point format */
#define LLVS_DEC_SINGLE_FLOAT 0 /* DEC 32-bit floating point
                                  format (PDP-11, VAX) */
#define LLVS_IEEE_SINGLE_FLOAT 1 /* IEEE 32-bit floating
                                   point format (Sun,
                                   Sequent, Spark, TI
                                   LISPM) */
    llvs_ubyte reserved;       /* reserved byte - must be
```

```
                                ZERO at present */
/* the remaining fields are in the format indicated above */
    llvs_integer pl_level;      /* plane level */
    llvs_integer row_location;  /* row offset of plane */
    llvs_integer col_location;  /* column offset of plane */
    union {
        llvs_integer iback;     /* integer background value
                                  */
        llvs_single_float fback; /* floating point background
                                    value */
        } background;      /* plane background value */
    llvs_integer alist_length;  /* length of alist record */
    llvs_integer data_length;   /* length of data + size
                                   header */
    llvs_integer multi_plane_flag; /* Multi-plane flag - if 0
                                      this is the last/only
                                      plane otherwise,
                                      additional planes are
                                      stored after this one.
```

```
                                                 */
    } LLVS_PLANE_FILE_HEADER;
```

The first three bytes of this header record describe the format of the plane data: its data type, its "byte sex" (byte ordering of multi-byte numerical values), and the floating point number format used for floating point numbers.

3 The Association List Record

Following the plane file header record is a variable-length record containing the plane's association list. This record contains ASCII text which is the printed representation of a Common LISP association list. This record is not bounded - code that reads plane files should make sure that the buffer used is large enough, as indicated by the **alist_length** field on the header record.

4 The Plane Size Record

Following the association list record is the plane size record:

```
typedef struct {
   llvs_integer datatype_again; /* plane data type (redundant) */
   llvs_integer row_dimension;  /* plane row dimension */
   llvs_integer col_dimension;  /* plane col dimension */
   } LLVS_PLANE_SIZE_HEADER;
```

This record describes the size of the plane in *elements*, giving the plane dimensions. The plane data is actually just a two-dimensional array of numerical values, each element of which represents a single picture element (called a "pixel").

5 The Plane Pixels

After the size record, comes the plane data itself. This is also a variable length record. Its size is 12 bytes less than the value of the **data_length** field in the plane file header record. (The **data_length** field in the plane file header record includes the size of the plane size record.) The data is stored in "scan" order: top-to-bottom, left-to-right. The first element is the top-left picture element, the next element is the next picture element to the right, etc., that is the data is stored by rows, starting with the top row of the image.

6 Multi-Plane Files

Some image data consists of a group of related planes, such as the red, green, and blue planes of a color image or the left and right views of a stereo pair, etc. For ease of transport and handling, it is posible to store multiple planes in one file. The **`multi_plane_flag`** field in the plane file header is used to indicate when multiple planes are stored in a given file. If this field is equal to zero, then only one plane is stored in the file. If this field is not equal to zero, then another plane file header record, association list record, plane size record, and plane data record follow after the current plane data record. The last plane has a plane file header record with its **`multi_plane_flag`** field set to zero. The value stored in this field is the number of remaining planes in the file. That is, if the file contains three planes, the value in the first plane's **`multi_plane_flag`** header field is 2, the value in the second planes' **`multi_plane_flag`** header field is 1, and the value in the third (and last) **`multi_plane_flag`** header field is 0.

7 Summary

A LLVS plane file looks like this:

Plane File Header Record	32 bytes
Association List Record	`alist_length` bytes
Plane Size Record	12 bytes
Plane Data Record	$(data\_length - 12)$ bytes

This structure is repeated if the `multi_plane_flag` field in the Plane File Header Record is not equal to zero. The last (or only) Plane File Header Record has its `multi_plane_flag` field set to zero.

A Actual C code to read in a single plane

This is the actual C code[4] to read in a plane file. The function *read_plane* takes as arguments a pointer to a pointer to a **PLANE** structure, a pointer to a pointer to a **PLANE_INFO** structure, a pointer to a string pointer, and a string pointer. This function allocates space (via **malloc()** and **calloc()**) for the **PLANE**, the **PLANE_INFO**, and the associations list and it side-affects its first three arguments, which should be either addresses of variables or array or structure slots. *read_plane* returns **-1** on error and the value of the **multi_plane_flag** on success.

```
#define BLOCKSIZE 512 /* max number of bytes to read at a time */

 /*
  * read_plane(plane,plane_info,associations,filename) - read
  * in a plane file. plane is a pointer to a pointer to a
  * PLANE object, plane_info is a pointer to a pointer to a
  * PLANE_INFO object, associations is a pointer to a pointer
```

[4] From "read_write_plane.c".

```
 * to char, and filename is a pointer to a char.
 * plane, plane_info, and associations are set to malloc'ed
 * space, so should be addresses of cloberable pointers (i.e.
 * addresses of variables or structure fields, etc.).
 */

read_plane(plane,plane_info,associations,filename)
PLANE **plane;
PLANE_INFO **plane_info;
char **associations;
char *filename;
{
```

First we declare local variables:

```
    /* plane file header record structure */
    static LLVS_PLANE_FILE_HEADER header;
    /* plane size header record structure */
    static LLVS_PLANE_SIZE_HEADER size_header;
    int plsize;                 /* # bytes in plane */
```

```
FILE *plfile;                /* plane file */
llvs_ubyte *data_pointer; /* pointer to data buffer */
int rbytes, bytesleft;      /* I/O byte counters */
int need_swap, need_cvt_float; /* flags to indicate if
                                     conversions needed */
```

Now we open the plane file and read in the plane file header record:

```
/* open file.  abort if open failure */
plfile = fopen(filename,"r");
if (plfile == NULL) return(-1);
/* read header record */
if (fread(&header,32,1,plfile) != 1) return(-1);
```

Next we generate the conversion flags. `LLVS_NATIVE_BYTE_SEX` and `LLVS_NATIVE_FLOATFMT` are macros defined in `llvs_plane.h` under control of contitional compilation macros defining the machine type (i.e. VAX, SUN, or SEQUENT, etc.).

```
need_swap = header.bsex != LLVS_NATIVE_BYTE_SEX;
```

```
need_cvt_float = header.floatfmt != LLVS_NATIVE_FLOATFMT;
```

Now we convert the remainder of the header to native internal storage format:

```
if (need_swap) swap_longs(&header.pl_level,7);
if (header.ptype == LLVS_PLF_FLOAT && need_cvt_float)
    cvt_floats(&header.background,1,header.floatfmt,
               LLVS_NATIVE_FLOATFMT);
```

Now we compute the size of the plane in bytes, allocate the `PLANE_INFO` structure, fill in the `PLANE_INFO` structure, and allocate memory for the plane itself.

```
/* compute plane size */
plsize = header.data_length - 12;
/* allocate plane info struct */
*plane_info = (PLANE_INFO *) malloc(sizeof(PLANE_INFO));
if (*plane_info == NULL) return(-1);
/* fill in plane info slots */
```

```
(*plane_info)->datatype = header.ptype;
(*plane_info)->level = header.pl_level;
(*plane_info)->row_location = header.row_location;
(*plane_info)->column_location = header.col_location;
(*plane_info)->background.fixnum =
    header.background.iback;
/* allocate plane it self */
*plane = (PLANE *) malloc(plsize);
if (*plane == NULL) return(-1);
```

Now we allocate space for the associations list and read in the associations list record. This C function does not actually do anything with the associations list. In the Common LISP based system, we pass this string to the Common LISP function READ-FROM-STRING and can then access the association list with Common LISP association list access functions.

```
/* allocate space for association list */
*associations = calloc(header.alist_length+1,
                       sizeof(char));
if (*associations == NULL) return(-1);
```

```
/* read in association list */
data_pointer = (llvs_ubyte *) *associations; /* start of
                                                buffer */
bytesleft = header.alist_length; /* number of bytes to
                                    read */
while (bytesleft > 0) {
    rbytes = bytesleft;
    if (rbytes > BLOCKSIZE) rbytes = BLOCKSIZE;
    if (fread(data_pointer,rbytes,1,plfile) != 1)
        return(-1);
    data_pointer += rbytes;
    bytesleft -= rbytes;
    }
```

Now we read in and convert the size header and fill in the plane dimension info.

```
/* read size header */
if (fread(&size_header,12,1,plfile) != 1) return(-1);
if (need_swap) swap_longs(&size_header,3);
```

```
/* fill in addition slots in plane info */
(*plane_info)->row_dimension = size_header.row_dimension;
(*plane_info)->column_dimension =
    size_header.col_dimension;
```

Finally, we read in the plane data itself, converting the data as we go.

```
/* read in plane */
data_pointer = (unsigned char *) (*plane)->plane_base;
bytesleft = plsize;
while (bytesleft > 0) {
    rbytes = bytesleft;
    if (rbytes > BLOCKSIZE) rbytes = BLOCKSIZE;
    if (fread(data_pointer,rbytes,1,plfile) != 1)
        return(-1);
    if (need_swap && header.ptype == LLVS_PLF_SHORT)
        swap_words(data_pointer,rbytes >> 1);
    if (need_swap && (header.ptype == LLVS_PLF_INT ||
                      header.ptype == LLVS_PLF_FLOAT))
        swap_longs(data_pointer,rbytes >> 2);
```

```
    if (need_cvt_float && header.ptype == LLVS_PLF_FLOAT)
        cvt_floats(data_pointer,rbytes >> 2,
                   header.floatfmt,LLVS_NATIVE_FLOATFMT);
    data_pointer += rbytes;
    bytesleft -= rbytes;
    }
```

When we are done, we close the file. **Read_plane** returns the value of the multi-plane flag. This will be zero if this file contained only one plane or a positive non-zero value indicating the number of additional planes in the file. **Read_plane** returns -1 when it encountes a system error (I/O error or a memory allocation failure).

```
/* close file */
fclose(plfile);
return(header.multi_plane_flag);
}
```

There are 3 conversion functions used by this function: `swap_longs()`, `swap_words()`, and `cvt_floats()`. These three functions are defined in `read_write_plane.c`. The first two functions do byte swaping on long ints and short ints and the third function converts floating point formats. All three functions modify the memory pointed at by their first argument. The second argument is the count of elelemts to convert. `cvt_floats` has two additional arguments: the current format of the floating point numbers and the desired (i.e. native) floating point format.

B Actual C code to write out a single plane

This is the actual C code[5] to write out a plane file. The function *write_plane* takes as arguments, pointers to a `PLANE` structure, a `PLANE_INFO` structure, a string containing the associations list, and a string containing the name of the file to write. If the associations list string is a `NULL` pointer, the string `"NIL"` is written to the file as the associations list record.

```
 /*
  * write_plane(plane,plane_info,associations,filename) -
  * write out a plane.
  */

write_plane(plane,plane_info,associations,filename)
```

[5] From "read_write_plane.c".

```
PLANE *plane;
PLANE_INFO *plane_info;
char *associations;
char *filename;
{
```

First we declare local variables:

```
static LLVS_PLANE_FILE_HEADER header;
/* plane size header record structure */
static LLVS_PLANE_SIZE_HEADER size_header;
int plsize;                  /* # bytes in plane */
FILE *plfile;                /* plane file */
unsigned char *data_pointer; /* pointer to data buffer */
int rbytes, bytesleft;       /* I/O byte counters */
```

Now we open the plane file and setup the plane file header record. The byte sex and float format fields are set to the values "native" to the current machine. *write_plane* does not convert the data to any particular format.

```
    /* open file.  abort if open failure */
#ifdef VMS
    plfile = fopen(filename,"w", "rfm=var");
#else
    plfile = fopen(filename,"w");
#endif
    if (plfile == NULL) return(-1);
    /* fill in plane file header record */
    header.ptype = plane_info->datatype;
    header.bsex = LLVS_NATIVE_BYTE_SEX;
    header.floatfmt = LLVS_NATIVE_FLOATFMT;
    header.reserved = 0;
    header.pl_level = plane_info->level;
    header.row_location = plane_info->row_location;
    header.col_location = plane_info->column_location;
    header.background.iback = plane_info->background.fixnum;
    header.data_length = plane_size(plane_info) + 12;
    if (associations == NULL) associations = "NIL";
    header.alist_length = strlen(associations);
```

```
header.multi_plane_flag = 0;
```

Now write the header record:

```
/* write header record */

if (fwrite(&header,32,1,plfile) != 1) return(-1);

/* compute plane size (for use later) */

plsize = header.data_length - 12;
```

Now write out the associations list:

```
/* get pointer to associaions list */

data_pointer = (llvs_ubyte *) associations;

/* and length */

bytesleft = header.alist_length;

/* write out associations list */

while (bytesleft > 0) {

    rbytes = bytesleft;

    if (rbytes > BLOCKSIZE) rbytes = BLOCKSIZE;

    if (fwrite(data_pointer,rbytes,1,plfile) != 1)
```

```
        return(-1);
    data_pointer += rbytes;
    bytesleft -= rbytes;
    }
```

Now fill in and write out the size header record:

```
/* fill in size header */
size_header.datatype_again = header.ptype;
size_header.row_dimension = plane_info->row_dimension;
size_header.col_dimension = plane_info->column_dimension;


/* write out size header */
if (fwrite(&size_header,12,1,plfile) != 1) return(-1);
```

Finally write out the plane's pixels:

```
/* get pointer & size of plane data */
data_pointer = (unsigned char *) plane->plane_base;
bytesleft = plsize;
/* write plane data */
```

```
while (bytesleft > 0) {

    rbytes = bytesleft;

    if (rbytes > BLOCKSIZE) rbytes = BLOCKSIZE;

    if (fwrite(data_pointer,rbytes,1,plfile) != 1)

        return(-1);

    data_pointer += rbytes;

    bytesleft -= rbytes;

    }
```

And close the file when we are done.

```
/* close file */

fclose(plfile);

return(0);

}
```

This little function is used to compute the size of the plane in bytes.

```
/* compute plane size from plane info */


plane_size(plinfo)
```

```
PLANE_INFO *plinfo;
{
    int elements;              /* element count */

    elements = plinfo->row_dimension *
               plinfo->column_dimension;
    switch (plinfo->datatype) {
        case LLVS_BIT: return( (elements + 7) / 8);
        case LLVS_BYTE:
            return( elements * sizeof(unsigned char));
        case LLVS_SHORT:
            return( elements * sizeof(short int));
        case LLVS_INT:
            return( elements * sizeof(long int));
        case LLVS_FLOAT:
            return( elements * sizeof(float));
        }
    }
```