PRIME OPTICS

17 Crescent Road
EUMUNDI Q 4562
AUSTRALIA

Using ZEMAX® "Binary" Files

2022 May 31

Rev 2022 June 08, 2022 December 20

Telephone :	+61-7-5442 8813	*Local Time :*	UT + 10
SMS preferred :	+61-429-02 8831	*EMail :*	damien@primeoptics.com.au
Contact :	**Damien Jones**	*Web :*	www.primeoptics.com.au

Background

This note describes one particular programming technique to generate “Binary IMA files” for ZEMAX®/OpticStudio®.

It also describes a method for reading BIM files for subsequent custom analysis.

An example is given.

Binary IMA File Structure

This is a direct quote from the ZEMAX® user manual:

“The binary IMA file format is more complicated than the text format, and binary IMA files cannot be edited with a text editor. However, the binary IMA files are dramatically more powerful. Each pixel in the binary IMA file is represented by an unsigned byte, which means there are 256 "grey-scale" levels of intensity. Furthermore, each wavelength can be assigned a separate pixel map. Therefore, very realistic photograph like extended sources can be modelled.

The binary IMA file format requires 3 16-bit header values. The first 16-bit value is a signed integer that must be equal to zero. The second 16-bit signed integer is the width of the pixel map in pixels, which can be any number from 1 to 8000. The third 16-bit signed integer is the number of pixel maps, which correspond to the number of colours (or wavelengths) represented in the file.

For example, a 3-color binary pixel map of a 50 by 50 image would have 6 bytes of header (0, 50, and 3), followed by 2500 bytes for colour 1, then 2500 bytes for colour 2, then 2500 bytes for colour 3, for a total of 7506 bytes. The data for each colour is stored by columns for each row (the column index changes faster than the row index).”

Of course, all files are binary, and the proper term here should be “untyped IMA files” to distinguish them from the IMA files which are read as TEXT.

Binary IMA File Data Structure

In PASCAL/DELPHI one implementation of the data block looks like:

```
TYPE
  SeeingBlockData = RECORD
    SF, Width, NumMaps: Smallint; // 16 bit signed integers
    sData: ARRAY[-9..9,-9..9] OF BYTE;  //1 byte = 8 bits
  END;
```

Note that the ordering of the variables follows the required schema.

Or in C (watch for machine dependent definitions of “short” and remembering to cast from “int” to “char” or to convert from “float” to “int” to “char”):

```
struct  SeeingBlockData  {
  short SF, Width, NumMaps;
  unsigned char sData[19][19];  //8 bit implementation of “char”
};
```

C is my “second programming language” so use the above data types with caution! Note that arrays in C are zero-indexed, unlike the PASCAL/DELPHI declaration.

This data type example is for 19 x 19 simulation of a Moffat Function for atmospheric seeing.

Example: Seeing Simulation

The Moffat function used here looks like:

$$I(r) = I_0\left(1 + \frac{r^2}{\alpha^2}\right)^{-\beta}$$

$$where:$$

$$\alpha = \frac{fwhm}{2\sqrt{2^{\frac{1}{\beta}} - 1}}$$

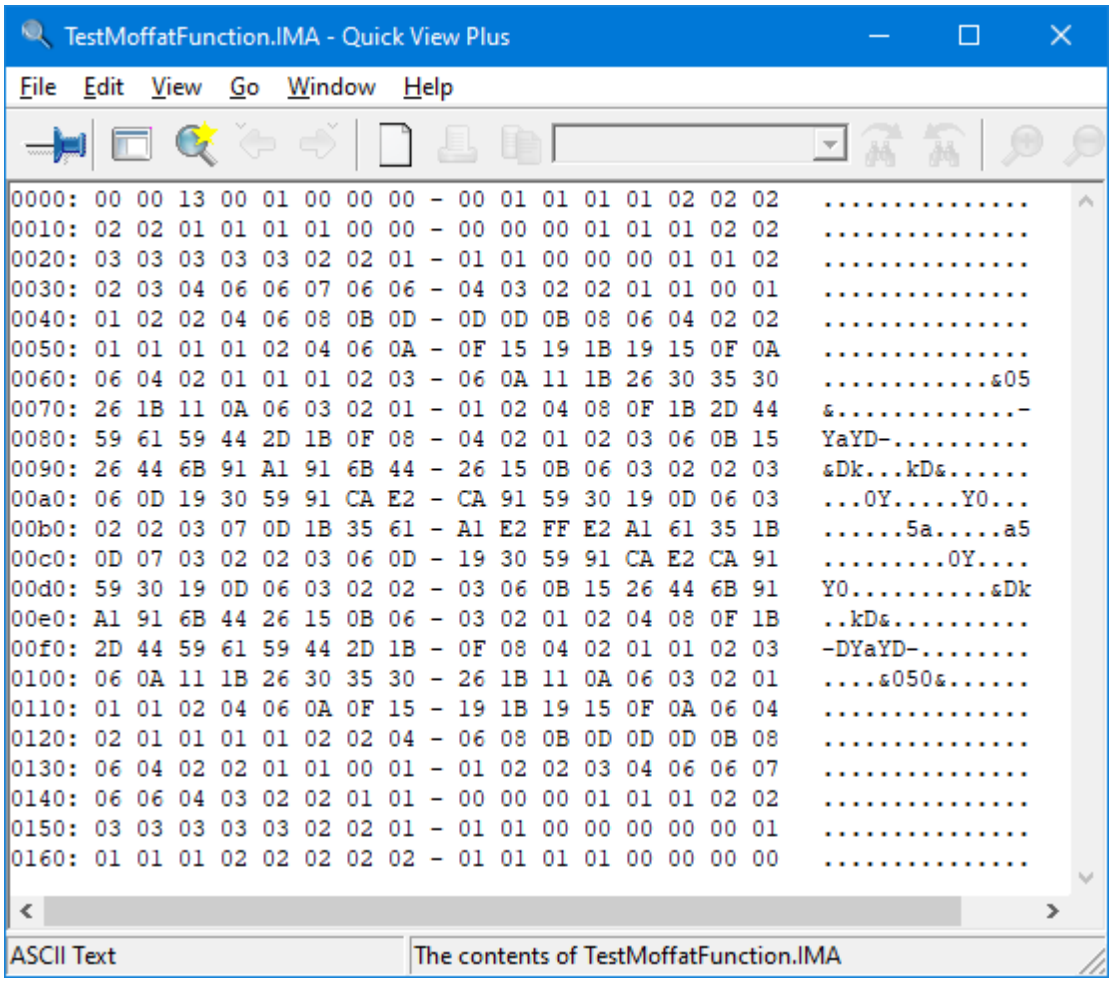
This is a modified Lorentzian function sometimes described as a “softened Gaussian”.

The site dependent parameters used in this example are:

$$fwhm = 5;\ \beta = 4;\ I_0 = 255$$

Of course, this will be a floating-point operation so the computed values will need to be rounded to the nearest positive 8-bit integer (0 – 255).

The resultant hexadecimal values in the file are:



The first 2 bytes represent zero, as required. The next 2 bytes, 13 00, represent the square array dimension of 19 (in little-endian order), as required. The next 2 bytes, 01 00 represent 1, for 1 map. 361 data values follow; one can see the central value FF (255) just to the right of centre in line 00b0; and the overall symmetrical morphology.

Example: code snippet

A PASCAL/DELPHI code snippet demonstrates how the data is written using a "File Stream".

This essentially writes the data structure block as an untyped stream of bytes, having pre-calculated and filled the data structure.

```
FS := TFileStream.Create(fn_bin, fmCreate);
FS.WriteBuffer(SBD,SizeOf(SBD));   //write the data block, SBD
FS.Free;
```

The "BIM" File Structure

Again, a quote from the OpticStudio® manual:

"*The drawback to the IMA format is that a maximum of 256 grey scale levels are supported. The BIM format is a binary double precision floating point file format which effectively makes the number of grey scales many trillions. The BIM format consists of the following binary values:*

1 32 bit integer representing the number of x pixels, nx.
1 32 bit integer representing the number of y pixels, ny.

*followed by n*n 64 bit double precision floating point values representing the relative intensity. The first pixel is the bottom left corner, and the remaining pixels are listed by rows along x.*

Currently, the nx and ny values must be identical or an error message will be issued."

The BIM File Data Format

As before, the data block looks something like:

```
TYPE
  BIM_data_block = RECORD
    n_x, n_y: INTEGER; // 4 bytes or 32 bits each
    BIM_data: ARRAY OF ARRAY OF DOUBLE; // 64 bit/8 bytes float
  END;
```

Note that a dynamic array is required as the size of the floating-point data block is unknown until the 2 integers are read. The dynamic array is then accessed in DELPHI code via reference (pointer) to the array.

A "TMemoryStream" object is used to load the BIM file. Quite large files (~1GB, for instance a model of a 10K x 10K detector ~ 10^8 by 8 bytes ~ 800MB) can be loaded in a second or two with a modern machine.

Once in memory, the dynamic array is initialised even faster.

By contrast, initialising the ".CSV" output text file is an order of magnitude slower, but can be speeded up somewhat by using a "TStringList" object, rather than conventional text writing routines.

The DELPHI code looks like:

```
    MS := TMemoryStream.Create;
    MS.LoadFromFile(fn_BIM);
// Read in the 2 leading integers
    s_n := 2*SizeOf(BDB.n_x);
    MS.Read(BDB,s_n);

    WITH BDB DO BEGIN
      SetLength(BIM_data,n_y,n_x);
      s_bdb := SizeOf(BIM_data[0,0]);

      rj_max := n_y - 1;
      ri_max := n_x - 1;
// dynamic array elements accessed sequentially
      for rj := 0 to rj_max do
        for ri := 0 to ri_max do
          MS.Read(BIM_data[rj,ri],s_bdb);

      MS.Free;
    END;
```

The array values are oriented correctly in a conventional Cartesian sense, as per the manual.

If one needs a spreadsheet layout to have the same orientation, then the “rj” index needs to be written in reverse order. Typically, one would use a “.CSV” format.

The “.CSV” format is handy for analysing sparse monochromatic slit images from a spectrograph. A filter and an auxiliary file can be used to list the spreadsheet coordinates where non-zero data can be found, otherwise it is like looking for a needle in a haystack!

References (at the time of writing):

https://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/PSFsAndSampling.pdf

ZEMAX OpticStudio® 19.8 User Manual, October 2019, pp1089 - 1090