

MASTER

LA-UR-80-3642

TITLE: A METAFILE FOR EFFICIENT SEQUENTIAL AND RANDOM DISPLAY OF GRAPHICS

AUTHOR(S): Theodore Niles Reed

SUBMITTED TO: SIGGRAPH'81
Dallas, Texas
August 3 to 7, 1981

DISCLAIMER

By acceptance of this article, the publisher recognizes that the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes.

The Los Alamos Scientific Laboratory requests that the publisher identify this article as work performed under the auspices of the Department of Energy.


los alamos
scientific laboratory
of the University of California
LOS ALAMOS, NEW MEXICO 87548

An Affirmative Action/Equal Opportunity Employer

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED



A METAFILE FOR EFFICIENT SEQUENTIAL AND RANDOM DISPLAY OF GRAPHICS

by

Theodore N. Reed

ABSTRACT

Graphics metafiles have been in use at the Los Alamos National Laboratory (LANL) since early 1977. The first metafile format was defined in 1976 and has been revised several times to provide efficient graphics support in the LANL computing environment. Objectives and current applications of the Common Graphics System (CGS) Metafile are given. Details of the format and the random access techniques incorporated in the CGS Metafile are described.

DEFINITION

A graphics metafile can be defined as follows: "A graphics metafile is a device-independent description of a picture intended for subsequent display on a graphics output device." Two concepts are implicit in this definition. First, the metafile is a device-independent description of a picture that is capable of being displayed on a wide variety of graphics devices. Second, the metafile is intended for subsequent display; thus, it is output-only.

HISTORY

In 1976, the Graphics Group at the Los Alamos National Laboratory (LANL) defined a graphics metafile for the Common Graphics System (CGS) for use at LANL [1,2]. It had 14 bits of resolution based on 8-bit bytes. We defined this format after performing detailed analysis on a variety of LANL-generated plots and looking at existing graphics metafiles [3,4]. In 1978, we modified this metafile to include 15 bits of resolution based on a 16-bit word and restructured it to include the ACM/SIGGRAPH Graphic Standards Planning Committee "CORE" capabilities [5]. This revision was proposed to an interlaboratory task group consisting of LANL, Sandia National Laboratories, and the Air Force Weapons Laboratory [6]. After some modification, it was adopted and called the Basic Graphics Package (BGP) Metafile format. The current CGS Metafile format is the BGP format with extensions to allow efficient random access and error recovery.

In 1979, the ACM/SIGGRAPH Graphic Standards Planning Committee published its status report, which included a proposal for a graphics metafile [7]. This metafile was based on a study of four existing metafile formats. One of these was the BGP format from which much of the command structure and format was derived [8]. Since then, the ANSI X3H3 Technical Committee on Computer Graphics Programming Language has formed the X3H33 Virtual Device Interface Task Group to standardize a graphics metafile and a graphics device driver interface [9,10].

OBJECTIVES

A variety of objectives that a graphics metafile at LANL should satisfy has been established. These objectives include:

- o A metafile that maintains the smallest file size possible while supporting necessary resolution requirements.
- o A metafile that is efficient to process.
- o A metafile that can be moved across computers of different word lengths without conversion.
- o A metafile that can be moved across different operating systems without conversion.
- o A metafile mechanism that allows a particular frame to be displayed without sequentially processing preceding frames.
- o A metafile format that is extensible.

APPLICATION

The CGS Metafile is currently used in a variety of ways at LANL (Fig. 1).

- o A CGS Metafile can be directed to a graphics terminal by using one of several graphics postprocessors. These postprocessors read the CGS Metafile and use the appropriate CGS Device Driver to display the picture at the user's terminal.
- o A graphics postprocessor that allows display of selected frames at the user's terminal can be used to preview or select frames before sending the output to a particular graphics output device.
- o A CGS Metafile can be directed to the Print and Graphics Express Station (PAGES) in the Central Computing Facility, where either paper or film output is produced. PAGES

functions as a graphics device driver and translates the metafile format to that required by the particular device selected.

- o The CGS Metafile is used in conjunction with an interactive session at a graphics terminal to record selected pictures for later output on a hard-copy device.

METAFILE FORMAT

The CGS Metafile design has provided a compact format that can be efficiently processed. This format is simple, yet easily extensible, to allow later enhancement. The format is based on multiples of 16-bit words to facilitate processing on most mini- and microprocessors, as well as many of the larger processors. This also facilitates the processing of ASCII characters, which are packed two per word.

Coordinate Positioning Command Format

In many graphics applications, most graphics data consist of coordinate positioning information. Each coordinate is contained within one 16-bit word to preserve as much resolution as possible and still keep the total size reasonably small. Each coordinate positioning command consists of an x, y, and optional z coordinate (Fig. 2). Analysis of "typical" graphics output at LANL indicates that no significant metafile size reduction results when either the x or the y coordinate is eliminated when unchanged from the previous position. In fact, because of the extra control bits necessary to identify whether a coordinate consists of x or y or both instead of a "simple" x, y coordinate pair, either the total size must increase or the resolution must decrease.

The coordinate positioning commands maintain 15 bits of resolution. This is sufficient for most existing graphics devices. These coordinates are transformed and clipped normalized device coordinates. After completion of a coordinate command, the current position is at the specified x, y coordinates. Either two- or three-dimensional coordinates can be output, allowing support of three-dimensional devices. For three-dimensional devices, a mode can be set and the coordinate command consists of an x, y, and z coordinate.

Op-Code Command Format

The op-code command format accommodates the remainder of the device-independent commands (Fig. 3). Although this format provides most of the possible CGS Metafile capabilities, it comprises a very small portion of the total graphics data in the CGS Metafile. Since this is the case, attention has been given

to providing a simple, uniform op-code command format that can be efficiently processed. Each op-code contains a count of the words associated with it. This simplifies searching the file for particular op-code commands and allows unsupported op-code commands to be easily skipped since the number of words associated with a particular op-code command is immediately available.

The 7-bit op-code is divided into a 3-bit class and a 4-bit subclass. This allows the various op-codes to be defined in an organized fashion with 8 major classes, each consisting of up to 16 subclasses. By dividing the op-code command in this fashion, jump tables can provide efficient processing of the class and subclass operations.

This format is written on disk as a "bit-stream"; that is, there is no explicit record or file structure. This allows the metafile to be moved between computers of different word lengths or different operating systems without conversion. All that is required to process the metafile is simple I/O, allowing transfer of a specified amount of data to or from a particular location on disk. To avoid word-boundary conflicts, the end of each metafile is padded with a "no-operation" command so that the file is a multiple of 60 16-bit words. This forces the file to align on word boundaries for all of the LANL computers (64-, 60-, 32-, and 16-bit words). A multiple of 180 16-bit words would be necessary to include computers with word lengths of 36- or 18-bit words.

RANDOM ACCESS EXTENSIONS

The CGS Metafile format allows efficient sequential processing of the CGS Metafile. The following additions allow efficient random access with minimal impact on the size of the CGS Metafile or the efficiency of sequential processing.

We added an escape function and modified two existing commands (end-of-data and new-frame) (Fig. 4). The index-block escape command contains the 16-bit word disk addresses of the preceding 28 frames. Each address or pointer consists of 32 bits. Word 0 indicates the escape function and that 59 words follow. Word 1 indicates the index-block escape command. Words 2 and 3 contain the 16-bit word disk address of the preceding index-block entry. Words 4 through 59 contain the 16-bit-word disk address of the preceding 28 frames. An address of zero indicates the end of this linked list of index-block escape commands.

The end-of-data and new-frame commands each consist of seven words. Word 0 gives the command and word count. Words 1-3 are a synchronization pattern that force the first 64 bits to a bit pattern unique to the end-of-data and new-frame commands. Words 4 and 5 are the 16-bit word disk address of the previous index-

block escape command. Word 6 is the current frame number for the new-frame command or the total number of frames for the end-of-data command.

As the file is generated, the disk address of each frame is saved. This disk address is a 16-bit-word address independent of the word length of the machine that is generating the CGS Metafile. When 28 frames have been generated, the index block is written to disk and its disk address is saved. Each new-frame command written to disk contains the disk address of the previous index block. When 28 more frames have been generated, the index block with the disk address of the previous index block is written to disk. When the job is complete, the last partial index block is written to disk and is followed by an end-of-data command containing the disk address of the last index block.

When the file is sequentially processed, the index-block escape command is ignored. However, when the file is randomly processed, a table of frame addresses can be quickly constructed (Fig. 5). The end-of-data command is located at the end of the file. It is read and the total frame count is used to give the table entry for the last frame. The disk address of the last index block is also obtained from the end-of-data command. This index block is read and the disk address of each frame is stored in the frame address table. The address of the previous index block is obtained; it is read, and disk addresses of those frames are stored in the frame address table. This process continues until a complete table of frame addresses has been constructed. Each frame can now be accessed immediately by disk address. The frame address table can be constructed at a fraction of the cost of sequential reading of each frame since only one disk access is made for every 28 frames. To further increase efficiency, the index-block escape command could be increased in size in multiples of 60 16-bit words.

To simplify processing, the end of each frame is padded to a multiple of 60 16-bit words so that each new-frame command and index-block escape command start at the beginning of a word (independent of computer word length). This ensures that the addresses will be at the beginning of a word on disk when the 16-bit word disk addresses are converted to an actual disk address for a machine of a particular word length.

Randomly Accessing a Frame

To randomly access a frame and get a correct picture, it is necessary that the current attributes be associated with each frame. After the new-frame command is generated, all of the current attributes are written. These will be ignored when processing the file sequentially, but are necessary to establish the environment when randomly processing frames. The current CGS Metafile does not contain segments or color/font definitions. A

method to support both sequential and random access of a metafile containing segments and color/font definitions is discussed under Future Extensions.

ERROR RECOVERY

When a job aborts, the metafile may not have been properly terminated. When this happens, the last index-block escape command and end-of-data command have not been written to disk. The frame address table is constructed by searching the file from the end, looking for the new-frame command. Once found, the pointer to the previous index block can be obtained, and the frame address table constructed. The file can be searched forward from this index block, looking for new-frame commands to complete the last few entries in the frame address table.

FUTURE EXTENSIONS

Segmentation

When segmentation is supported in the CGS Metafile, the following scheme will be adopted to allow efficient sequential and random processing while maintaining a small file size. A new escape function will be defined, called the frame environment block. This will contain all current attributes and disk addresses (in 16-bit words) of all segments. This escape command will be ignored when processing the file sequentially, but will be used to provide a correct picture when processing the file randomly. Little additional file space would be required in the "typical" case where there are relatively few large segments. This same scheme would be used for color definitions and font definitions if added to the CGS Metafile.

Random Processing by Key Identifier

An extension that would facilitate random processing is the addition of a user-specified identifier to the CGS new-frame subroutine call. This identifier would be written as part of the CGS Metafile new-frame command. When the frame address table is constructed, this identifier would be associated with the frame number and disk address. This identifier would then be used as a key to randomly access a particular frame, thus allowing the user to specify a logical identifier rather than a frame number.

CONCLUSIONS

The CGS Metafile is efficient in file size and processing time for both sequential and random display. We accomplished this by avoiding an explicit record or file structure and

incorporating an escape function containing frame pointers in a linked list. By avoiding an explicit record structure, we achieved portability of the metafile across different computers and operating systems without conversion. These techniques are extensible and will be used when supporting additional features in the CGS Metafile.

REFERENCES

1. R. G. Keller, T. N. Reed, and A. V. Solem, "An Implementation of the ACM/SIGGRAPH Proposed Graphics Standard in a Multisystem Environment," Computer Graphics, Vol. 12, No. 3, August 1978, pp 308-312.
2. T. N. Reed, "The Common Graphics System," Computer Sciences and Services Division Technology Review, Los Alamos National Laboratory, report LASL-79-16 (April 1979), pp 17-20.
3. T. Wright "A Schizophrenic System Plot Package," Computer Graphics, Vol. 9, No. 1, Spring 1975, pp 252-255.
4. D. Groot, "GPGS 16 Bits Device Independent Picture Code," T. H. Delft and Informatica, Faculty of Science, University of Nijmegen, The Netherlands, October 1975.
5. "Status Report of the Graphic Standards Planning Committee of ACM/SIGGRAPH", Computer Graphics, Vol. 11, Number 3, Fall 1977.
6. T. N. Reed, "The Common Graphics System - An Implementation of the ACM/SIGGRAPH Proposed Graphics System," VIM-28, April 1978, pp 138-140.
7. "Status Report of the Graphic Standards Planning Committee of ACM/SIGGRAPH", Computer Graphics, Vol. 13, Number 3, August 1979, Part IV.
8. J. R. Warner, "Device Independent Intermediate Display Files," Computer Graphics, Vol. 13, No. 1, March 1979, pp 78-109.
9. "X3H33 SD-3 Proposal for an ANSI X3 Standards Project for the Computer Graphics Virtual Device Metafile," CBEMA, 1828 L Street NW, Washington, DC 20036, November 1980.
10. "X3H33 SD-3 Proposal for an ANSI X3 Standards Project for the Computer Graphics Virtual Device Interface," CBEMA, 1828 L Street NW, Washington, DC 20036, September 1980.

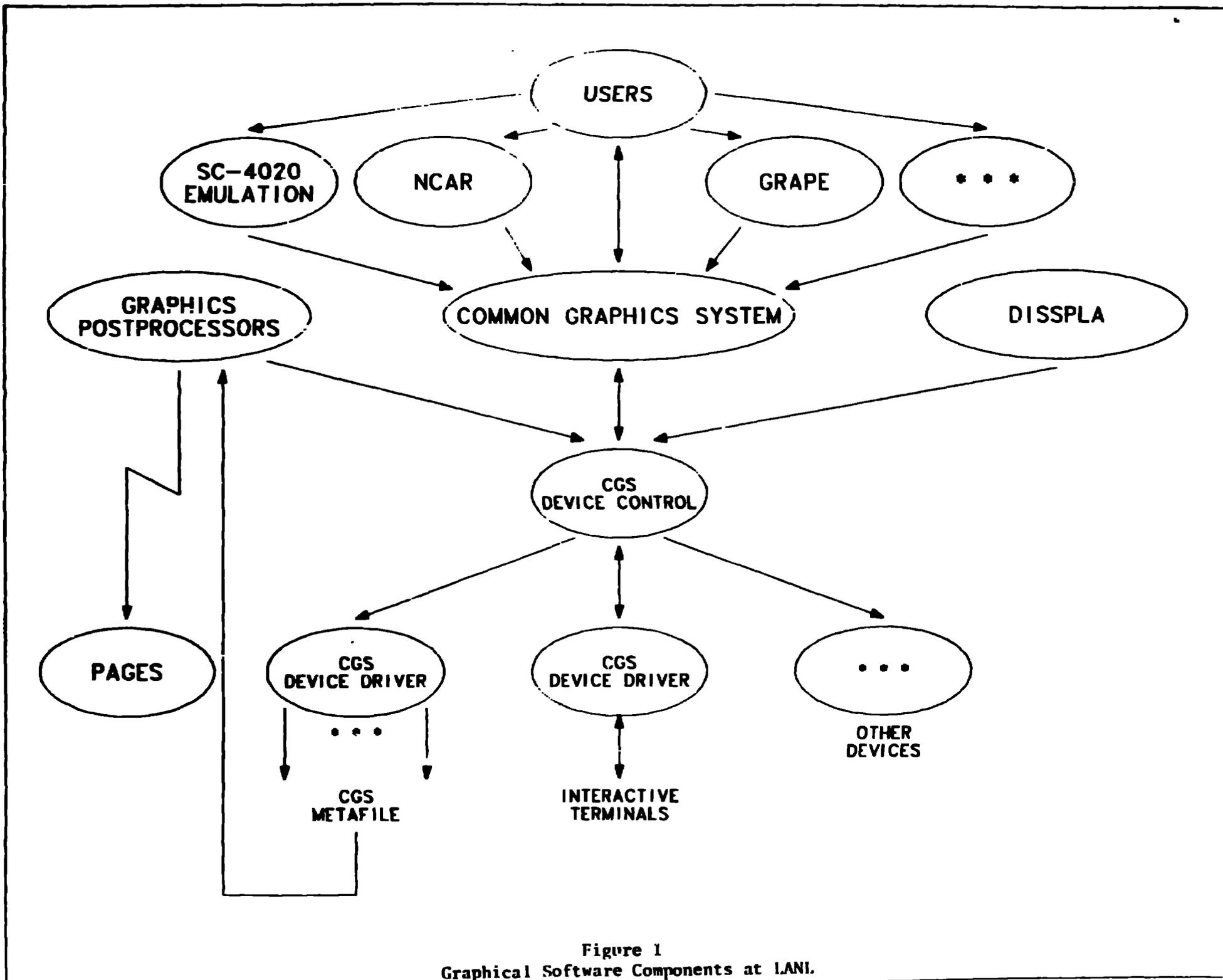


Figure 1
Graphical Software Components at LANL.

ESCAPE	59
INDEX -- BLOCK (IB)	
MSB INDEX-BLOCK POINTER	
LSB INDEX-BLOCK POINTER	
MSB FRAME POINTER	
LSB FRAME POINTER	
⋮	
MSB FRAME POINTER	
LSB FRAME POINTER	

NEW-FRAME	6
SYNC PATTERN	
SYNC PATTERN	
SYNC PATTERN	
MSB INDEX-BLOCK POINTER	
LSB INDEX-BLOCK POINTER	
FRAME NUMBER	

END-OF-DATA	6
SYNC PATTERN	
SYNC PATTERN	
SYNC PATTERN	
MSB INDEX-BLOCK POINTER	
LSB INDEX-BLOCK POINTER	
TOTAL FRAMES	

Figure 4
Index-Block Escape, New-Frame, and End-of-Data Commands

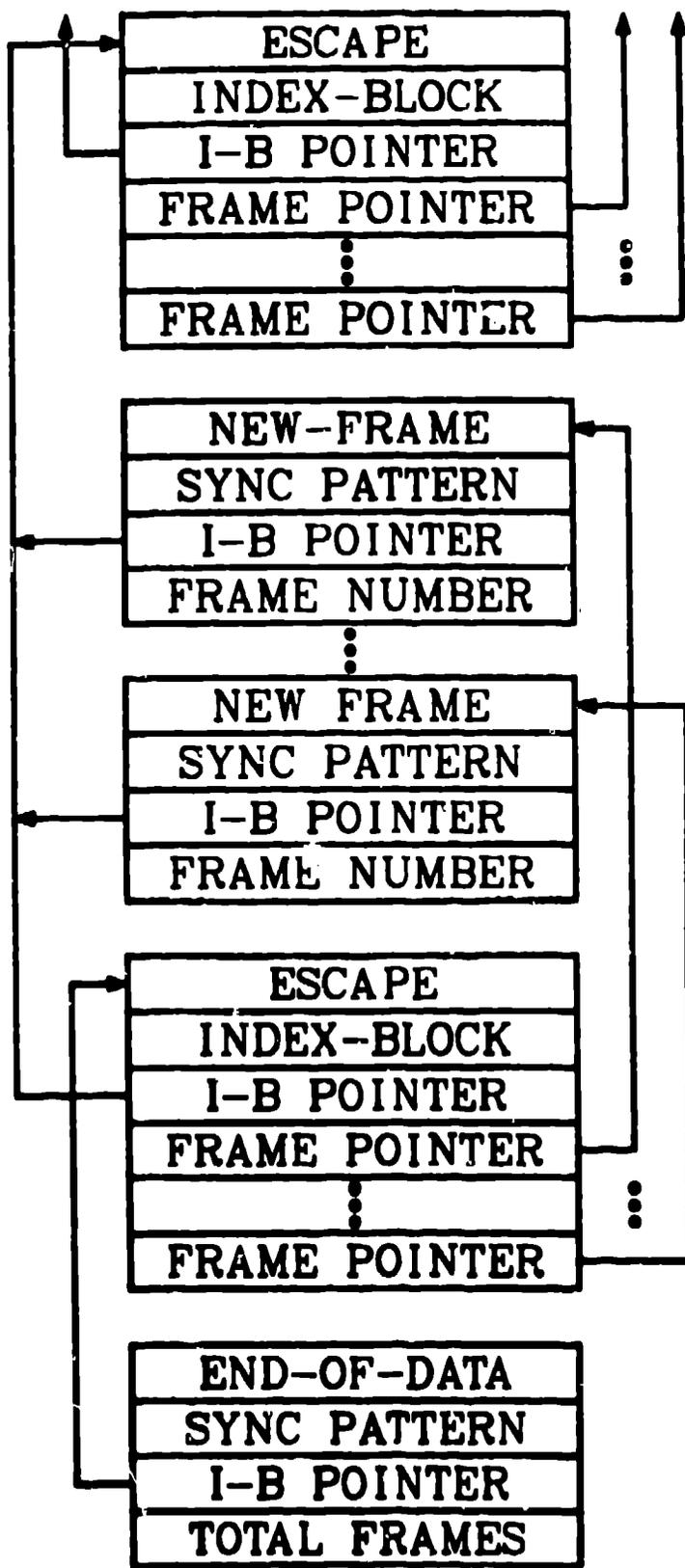


Figure 5
CGS Metafile Random Access Linked List Structure