(NASA-CR-174348) RASTER GRAPHICS EXTENSIONS TO THE CORE SYSTEM Final Report, 1 May 1981 - 15 Nov. 1984 (George Washington Univ.) 7 p HC A02/MF A01 CSCL 09B

N85-16590

Unclas G3/61 01407

RASTER GRAPHICS EXTENSIONS

TO THE CORE SYSTEM

Final Report to NASA Langley Research Center

James D. Foley
Principal Investigator
Department of Electrical Engineering and Computer Science
The George Washington University
Washington, D.C. 20052

NASA Research Grant NAG-1-185
May 1, 1981 to November 15, 1984

Significant Accomplishments
under
NASA Grant NAG-1-185
to
The George Washington University
J. Foley, Principal Investigator
1983-84

During the first year of the grant we developed a conceptual model of raster graphics systems. The model integrates Core-like graphics package concepts with contemporary raster display architectures.

The model captures the wide range ofraster system capabilities. Output primitives are passed through a window to viewport transformation into one of several normalized device coordinate (NDC) spaces. Each primitive in NDC space is then scan converted into a pixel matrix. The application program can also read/write arbitrary parts of pixel matrixes. RasterOps may be performed to transfer parts of one pixel matrix to another, or to a different area in the same pixel matrix. A view maps selected pixels from a pixel matrix window to a screen viewport on the viewsurface, where they are combined according to a composition rule and displayed.

The concepts of a window on a world coordinate space and a viewport in a normalized device coordinate space are exactly the same as in the conceptual model of vector graphics. The raster model also includes all the functions of our initial raster extensions with one minor modification: whereas in the current Core System output primitives that have passed through the viewing pipeline are scan-converted and displayed directly; they are now scan-converted into an intermediate pixel matrix. A pixel matrix is a two-dimensional array of pixel values. Height, width and bits/pixel are static attributes of a pixel matrix, defined when it is created and remaining unchanged until it is deleted. A distinct pseudo-display file as in the Core System is associated with each pixel matrix and contains the output primitives, arranged by segments that are scan converted into the pixel matrix.

Associated with each pixel matrix is an index table which specifies the color or intensity corresponding to each possible pixel value in that pixel matrix. Thus the same pixel value in different pixel matrixes can represent a different color in each of them. In addition, each pixel matrix has dynamic attributes which specify individual transparency values, and erasability. The transparency of a pixel value indicates that pixels with that value are not to be displayed, and thus do not obscure other pixels they overlap. Any number of pixel values may be transparent. The erasability attribute is used to allow a pixel matrix to be unaffected by a newframe.

'RasterOps' are defined as operations on pixel matrixes. A RasterOp moves an arbitrary rectangular array of pixels from a specified location in a source pixel matrix to a location in a destination pixel matrix. The source and destination may be the same pixel matrix. Rotation and scaling transformations may be applied as part of RasterOp and scaling transformations may be applied as part of RasterOp. An application program may transfer pixels directly between a pixel matrix and an appropriately formatted 2-D array of pixel values, thus entirely bypassing the viewing process and segmented pseudo-display file of the vector graphics systems. Pixels are rendered in temporal priority and may overwrite images already in the pixel matrix.

A window defined on the world and an associated viewport defined on NDC space specify a viewing transformation through which real objects are mapped into NDC space. In the same way, a pixel matrix window and screen viewport define another transformation called a view which maps pixel matrix images onto the viewsurface. All parts of a pixel matrix appearing in the pixel matrix window are mapped into the screen viewport. Any number of views may be defined on a pixel matrix. Because a view is defined as a transformation from one raster coordinate space (the pixel matrix) to another (the viewsurface), the pixel matrix window and screen viewport are defined in logical raster coordinates, as are the pixel matrixes and viewsurface. The resolution of the viewsurface in logical raster coordinates is set by the application program at initialization time, and the mapping to physical device coordinates is performed, if needed; a uniform integer replication to use the maximum screen area. The use of logical device coordinates for both pixel matrixes and viewsurfaces all a programmer to work entirely with raster graphics concepts, if the Core System capabilities themselves are unneeded.

The final visible picture on the viewsurface is a combination of some or all of the views and is created by a process called view composition. A composition is specified by a composition frame and a composition expression. The composition frame is a rectangular region defined on the view-surface. Only those views or parts of views that fall within a composition frame are composed using its composition expression. The composition expression defines how several overlapping images from different views are to be rendered on the viewsurface. The composition expression includes conditional, boolean and arithmetic operators on any number of pixel matrixes and is applied to all non-transparent pixel values. The full set of 16 boolean operations is supported. Thus, for example, the final picture may by the sum of all images in a composition frame, or it may be an arrangement with higher-priority images obscuring lower-priority ones. Any number of composition frames may be in effect

at a time. Where composition frames overlap, the visible images are in strict temporal priority of composition frame creation-the most recently created takes precedence. Similarly, although the aspect ratios of composition frames or their composition-rules may be changed arbitrarily, the more recently modified compositions will have priority over earlier ones.

In summary, the conceptual model of raster graphics introduces multiple pixel matrixes with associated index tables, RasterOp operations on them and new, raster oriented, viewing transformations called views which can be combined to form a final image on the viewsurface.

During the second half of the grant period we have concentrated on developing key concepts needed in a graphics programming language. The overall structure consists of geometric modeling, viewing, interaction management, and display management subsystems. Within each area, we have been developing descriptions of the semantic capabilities required. Many of these semantics are included in our 1984 report.

Several important themes run through these capabilities. The themes are important because they will form the basis for constructs in the programming language. One of the themes is that of dynamic links. Links maintain dependencies between data variables (which might include graphic/geometric objects). Thus when variable A changes, so too do the other variables which have functional dependencies on A.

Another theme is that of coordinate system. It seems natural that a graphics language deal with coordinate systems, but none have. We will have named coordinate systems, and declare an object's coordinate system as one of its attributes. We then use another theme, that of a transformer, to relate one coordinate system to another. Transformers are useful in input (to relate the coordinate system of a mouse or tablet to the world coordinate system), in modeling (to relate objects to subobjects), and in viewing (to relate world coordinates to the screen).

Another concept used is that of additional dimensions as a means of specifying attributes. The intensity of a point, for example, is just another (non-geometrical) coordinate of the point. By treating attributes in this way, the attribute values can be operated on by transformers: a separate language construct is not needed.

Separately, we have started implementing the look-up table compiler for composing separate images into a single view. The concepts of composition, composition frames and image priority are all working.

BIBLIOGRAPHY OF PAPERS & REPORTS

Acquah, J.B., J.D. Foley, J.L. Sibert, and P. Wenner, "A Conceptual Model of Raster Graphics Systems," Computer Graphics 16, 3, July 1982, (Proceedings of 1982 SIGGRAPH Conference) pp. 321-328.

Abstract

In this paper we present a conceptual model of raster graphics systems which integrates, at a suitable level of abstraction, the major features found in both contemporary and anticipated graphics systems. These features are the refresh buffer; the image creation (scan-conversion) system; the single address-space architecture which integrates the address space of the refresh buffer with those of the image creation system and the associated general-purpose computer; the Raster-Op or BitBlt instructions found in some single address-space architectures; the video look-up table, and refresh buffer to screen transformations. Also included are the major components from the conceptual model of vector graphics systems which are embodied in the ACM/SIGGRAPH/GSPC Core System. Using the conceptual model as a base we proceed to sketch out the capabilities we have defined in a substantial addition to the core system. The capabilities are currently being implemented as part of the George Washington University Core System.

Acquah, J.B., J.D. Foley, and P. Wenner, Reference Manual for Advanced Raster Graphics Extensions to ACM/SIGGRAPH Core System. IIST Report 82-21, Department of Electrical Engineering and Computer Science, The George Washington University, Washington, D.C., March 1982.

Abstract

This reference manual contains detailed functional descriptions of advanced raster graphics extensions developed for the ACM/SIGGRAPH Core Graphics System. It should be read in conjunction with the paper describing the philosophy behind these extensions: "A Conceptual Model of Raster Graphics Systems" by James Acquah, James Foley, John Sibert and Patricia Wenner. (GWU/EE/CS/IIST Report 81-29, November 1981). The original specification for the Core Graphics System is found in "Report of the Graphics Standards Planning Committee", published by Association for Computing Machinery, special Interest Group in

Graphics. (Computer Graphics, Vol 13, No 4, August 1979). The section headings and numbering are a continuation of that document and are therefore not necessarily sequential.

Acquah, J.B., J.D. Foley, and C.F. McMath, Graphics Programming Language Research. IIST Report 84-49, Department of Electrical Engineering and Computer Science, The George Washington University, Washington, D.C., July 1984.

Abstract

This report describes the state of our graphics programming language research. The work covers two areas: 'look-up table' compilers, and general graphics programming languages. We have refined our earlier look-up table language, and are now preparing to implement it. We have developed the basic concepts needed in a graphics programming language. Both efforts are described herein.

Acquah, J.B., The IMAGYS Graphics Programming Language: Description and Analysis. IIST Report 84-50, Department of Electrical Engineering and Computer Science, The George Washington University, Washington, D.C., November 1984.

Abstract

Imagys (Interactive Modelling And Graphics System) is a high-level modelling and graphics programming language being developed at GWU. The language is designed to considerably reduce the difficulty of writing highly interactive engineering-oriented graphics programs, and is based on the paradigms of object-oriented systems and functional, or value-oriented systems. Although these are generally considered alternative approaches to designing languages, they are combined in Imagys to produce a language which we believe combines the best features of both and has few of the drawbacks of either.

Data abstraction in general, and object-oriented systems in particular, are naturally suited to any system that creates a model of real-world entities and processes. The principle of hiding irrelevant details and encapsulating system-dependent features is an extremely powerful tool. Of even greater importance in a language for graphics is that such languages are extensible, adapting to meet new requirements and demands.

Value-oriented computation is adopted in Imagys for two reasons. Firstly, many graphics programs involve considerable amounts of numerical computation, especially for geometric transformations. Concurrency is the key to high-speed computation. With any given

hardware technology, massive increases in computing speed can only be achieved by operating multiple processors in parallel. This is greatly facilitated by a language that permits the greatest degree of concurrency to be extracted from a program. such languages must be free from side effects and must have no sequencing requirements not derived from data dependencies in the program. Functional, or applicative languages which perform all numerical computation by t'e application of functions to values have such properties.

But in an interactive graphics language there is an additional reason for developing a language which imposes no superfluous sequenci g constraints on an algorithm. Human factors considerations require that a system be as flexible as possible in order to accomodate different user preferences and styles. A non-procedural or declarative style of programming is the one best-suited to these requirements.

This paper presents an introduction to the Imagys base language. A companion report (Standard Graphics and Modelling Classes of the Imagys Language [IIST 84-51]) describes in detail how the language is extended for high-level graphics.