# Graphics Programming with "Archetypes"
# - A Preliminary Study -

Fernando Mario Martins
Jose Nuno Oliveira

C.C.E.S.
Universidade do Minho
Rua D.Pedro V,88
4719 Braga Codex
Portugal

JOSÉ N. OLIVEIRA
16SET85

This paper is a brief report on the initial phase of the formal development of a graphics programming system. At this stage of the specification, the system architecture is just outlined and attention is focussed on the conceptual level. The abstract notion of a graphic 'archetype' is introduced and proposed as a basis for the style of graphics programming to be implemented. The formal description of this meta-concept of the system is sketched.

## 1.- Introduction

This paper is a brief report on the preliminary phase of the design of a graphics programming system which we intend to use as a test-bed for specification/'inferential programming' techniques [1,2,3].

As is stressed by Partsch [4], the proper upgrading of formal methods for software development depends on "feedback" coming from experience in applying the existing techniques to as diversified branches of software design as possible. We found a stimulus for applying formal methods to computer graphics in the local industrial environment (viz. Northern Portugal textile and metallurgical industry). We are aware that automatic production and/or computer-aided pattern design is either unsatisfactory or not yet implemented.

We suspect, however, that a reason for dissatisfaction with the use of existing graphics design tools is lack of formal analysis of the graphic objects which this kind of software deals with. Concepts such as 'picture', 'form', 'image', etc. are usually introduced in an intuitive way and later manipulated too informally. Experience already consolidated in more ordinary environments (eg. numerical data-types) has shown how important it is for software development to know the mathematical essence of the objects we manipulate and the properties which interrelate them [5]. Lack of this kind of concern in the graphics environment has led to cumbersome and complex tools. Not only is this software difficult to understand and maintain but it is also time/space consuming.

The group we belong to has been addressing the problem of reducing the complexity of computer graphics software. On the one hand, we believe that the desired level of conciseness and efficiency may be achieved by working "top down" from a formal, rigorous basis (eg. starting from an algebraic or abstract-model characterization of the graphic environment). On the other hand, the potential users of our software are likely to require small, inexpensive systems (eg. microcomputers).

We found this preliminary phase more difficult, and more interesting, than we would have thought. This text is a brief description of our goals, of the ideas which have been around and of the difficulties encountered during this phase. A survey on graphics formal certification and verification literature is presented next.

## 2.- Background

As happens in other branches of software engineering, the use of formal methods in computer graphics programming is a relatively recent endeavour. A sample of the available literature is surveyed below. We show how it mirrors the major competing trends in current software technology, for example wrt. the use of 'declarative' or 'imperative' languages.

In the former class of languages above, 'functional programming' is advocated as being the easiest vehicle for picture description. In this kind of approach (pioneered by Henderson [6,7]) pictures are described by sets of functional equations. The basic idea is that these equations form a purely functional program which may run if encoded in some available functional language, eg. LISP. Running such programs should have the overall effect of plotting the corresponding pictures. These are regarded as elements of an elaborated data-type which comprises operations for picture manipulation (eg. building pictures from other pictures).

This approach is followed by Arya [8] who exploits advanced features of the functional programming language HOPE [9] (such as polymorphism, abstract-data constructors, overloading, higher-order functions, etc.) in creating and manipulating pictures. These are regarded as hierarchical entities built by so-called 'primitive' constructors, 'convenience' constructors (for concise picture description) and 'intelligent' constructors (cf. Henderson's operators [7]).

In the purely functional approach a program is regarded as a "runnable specification" and this completes the design process (if additional optimization is required one proceeds via program-transformation [10]). Minkowitz [11] combines these ideas with constructive software development from formal specifications. The design of a graphics editor is described which develops Henderson's [7] 'functional geometry'. Mathematics is used in specifications in the same way as eg. in the Vienna Development Method (VDM) [12] but operations on data-types are defined explicitly, which means that they are directly runnable as functional programs. (Cf. the Stirling school of 'rapid prototyping' [13] advocated as a means for shortening software development cycles).



From the algebraic point of view, the above corresponds to "running the definitional axioms" of abstract-type-operators. This is a principle beneath so-called "runnable (algebraic) specification languages" such as ORDINARY [14] or OBJ [2]. In the former work a completely self-contained specification is given of geometrical constructions such as distances, lines and circles. The algebraic approach to computer graphics formal description is also put forward by Mallgren [15,16], Gnatz [17,18] and Carson [19,20].

Mallgren [15,16] describes a method for algebraic specification of interactive graphics programming languages. Special graphic data types are specified and embedded in the programming language PASCAL. User interaction is specified via 'event algebras' (an extension of the standard algebraic technique) which is suitable for describing resource sharing.

Carson [19] suggests that computer graphics systems be specified via the techniques employed in the specification of concurrent systems. Existing programming-language specification techniques (some of them listed in an ANSI document [21]) are found to be unsuitable for the specification of the semantics of computer graphics systems 'with no formally defined syntax'. The desirable properties of a formal specification are also pointed out and references are made to alternative computational models which fit into the graphics environment, namely the distributed process-message based one (which is mathematically treatable via Milne and Milner's 'flow algebras' [22] and via the syntactic theory of message-passing developed by Ward and Halstead [23]). A proposal for computer graphics formal specification is also presented which supports the idea that the fundamental objects of a specification must be interfaces (between graphics and high level languages, and from graphics to graphic devices), the structure of the graphics system and the functions it performs. A practical example is presented which refers to interesting excerpts of the PMIG specification [24] and illustrates the suitability of algebraic-based techniques for nontrivial graphics software specification problems.

Both Gnatz [17,18] and Carson [20] are particularly concerned with formal specification of graphics standards (namely GKS). Gnatz [17,18] uses the framework for algebraic specification developed by the Munich School [5,3]. Carson [20] adopts a similar style of writing data type specifications. The need for such formal standards has been felt in the approach to graphics 'certification by verification', as opposed to the less reliable 'certification by testing' [25]. Gnatz [17,18] sketches a hierarchical, abstract-type approach to computer graphics. An algebra of 'images' is presented first as a semigroup and later enriched into a more elaborate structure. Classes of models of these types are studied and it is suggested that this study may form the basis for a taxonomy of graphics hardware. The CIP algebraic-framework [3] is also shown to be adequate for defining the semantics of GKS-like metafiles.

Duce et al [26], Marshall [27,28], Minkowitz [29] and Duce and Fielding [30] have also been working on formalizing parts of the GKS standard. However, these works have preferred the model-based strategy to algebraic ('property oriented') specification. In [26,30] VDM is used to describe 'implicit regeneration' and 'attribute handling' in GKS. Minkowitz [29] applies the so-called 'me too' methodology to the former problem. The 'me too' model (based on the VDM model) leads to a prototype straightforwardly

derived from the specification. This makes it possible to observe directly the behaviour of the system by exercising the prototype (ie. the specification is "animated").

Marshall [27] addresses the problem of specifying graphics devices and, in particular, the problem of output approximation to an ideal picture. Encouraging results are obtained on proving that some existing algorithms produce the right kind of output. Marshall [28] studies the specification of 'workstations', a fundamental concept in GKS. It is shown that formal specifications for a simplified abstract GKS output workstation (as well as for two specific output devices) are straightforwardly achieved. However, Lynn Marshall admits that a significant amount of work is still needed to produce a "polished specification" of the complete GKS standard.

Finally, the use of logic programming in computer graphics has been recently put forward by Pereira [31], who suggests ways to do graphics in PROLOG.

## 3.- Main Goals

As it has been pointed out above, the main goals of the project presented here are the application of formal specification and development methods (eg. the algebraic [2,3] and the constructive or abstract-model based methods [1]) to the design of a graphics software environment. From the experience in exploring such techniques we expect to draw useful conclusions about their strength and applicability to this kind of software systems design.

Following experience already gained in this area [30], such techniques are the tools we believe we need to guide us in the rigorous development of our graphics software system. This has been characterized by a set of behavioural, architectural and implementational properties and capabilities (which are our initial informal design specifications) that must be observed in the overall project. For example, the final product should be able to run on some locally widespread microcomputer systems, namely MS-DOS, CP/M or UNIX-like systems.

## 4. - The Conceptual Level

### 4.1 - Why Archetypes?

The physical world which surrounds us is made of imperfect, rough things whose geometrical appearance is far from being regular. And yet, one is able to find some abstract geometrical structure in most of them. The authors found in Plato's "reminiscence theory" (whereby such abstract entities are explained to be the "archetypes of things", ie. just the reminiscence of a sublime world populated with all-perfect things which we remember because we all once lived there) an interesting (also "philosophical") departure point for the formalization of graphics objects.

In this way, 'graphic archetypes' are regarded as the result of abstracting from graphic objects their roughness and imperfections. Because computers deal with formal models (abstractions) of the physical world, we found this concept likely to be useful in formal



computer graphics.

Moreover, many centuries of research in geometry have taught us that such "pure forms" are characterized by useful algebraic properties. Therefore, the "archetype view" of computer graphics seems to be very much in the spirit of modern software disciplining tools such as data-type abstraction. For example, one may try to "encapsulate" Plato's world of "graphic reminiscences" in a library (cluster or module) recording all facts about archetypes and then reliably build graphic objects on top of such a formal knowledge. One achieves thus correctness, modularity, structured development and (hopefully) trims down the complexity of computer graphics as a whole. The main difficulty resides in formally describing such a concept of graphic-archetype.

### 4.2 - From Archetypes to Images

We often refer (intuitively) to geometrical objects by merely identifying them, and no ambiguity arises from such loose references. This is so because in certain domains of discourse we are not interested in the details of these entities and simple, general properties are sufficient for characterizing them and distinguishing them from others. For example, we frequently refer to square, triangular, rectangular, linear-shaped (concrete) objects. Such geometric entities are regarded as object-attributes and are intrinsically abstract. However, for some kind of visualization of them to be possible, additional (pictorial) information is needed, which is not inherent in the geometrical entities themselves, but serves as a vehicle to a particular representation of them.

We must distinguish the properties of geometric abstractions ("archetypes") from the properties of some particular instantiation of them (not necessarily for visual purposes). At the most abstract level we must be concerned only with the properties that will make possible a classification and grouping of them in well defined classes. Each class will provide means for deciding whether a given object belongs to it (thus retrieving its properties). For example, discussing equality between two squares at such a level is nonsensical, but it is possible to establish a rigorous distinction between square and triangle-shaped objects (just because they are instantiations of different archetypes (1)).

Conceptually, the archetype level must capture the syntactic and semantic properties of the most useful (and intuitively recognized) graphic objects, eg. points, lines, line-segments, circles, rectangles, triangles, etc. So, our archetypes embody all usual 'primitive graphic data types', to be specified either via imperative constructs or by an applicative, "higher order" approach

--------------

(1) By analogy with a common situation in strongly-typed imperative languages, if eg. we declare two objects of type Integer, we are (at that level) distinguishing them from all other implemented data types. Even if throughout the code we never check whether the two objects represent equal integer quantities, any attempt to assign to one of them any non-integer quantity will result in an error. However the user may not be bothered about the integer data-type semantics.

as described in section 5.

Archetypes form the most abstract level of 'picture realization', which is the one responsible for object/operation overall consistency. However, from "simple" primitive archetypes we must be able to construct more complex ones. Some constructors will thus be provided for this purpose. For instance, there must be a way of defining the archetype which abstractly models the class of all square-circle composite-objects. All these archetype-level constructors, which differ from instantiation-functions transforming 'archetypes' into 'forms' by means of archetype instantiation-attributes (see Figure 1), are closed over the archetype algebra.

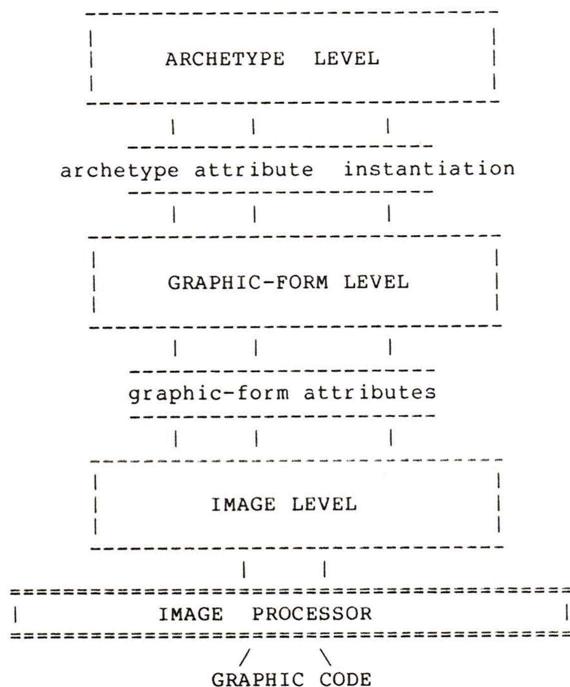


Figure 1 - Graphic-object hierarchy

Archetype instantiation leads to a lower level which we term the 'graphic-form' level (Figure 1). This level is made of less abstract objects characterized by configurational and positional attributes. The underlying model at this level may be the Cartesian plane, where forms may be represented by sets of points (Cartesian coordinates or any equivalent). Congruence relations such as form-equality make sense at this level, and other relations (eg. symmetry) may be defined on top of equality.

Graphic-form constructors are available for defining arbitrarily complex graphic-forms. The use of a positional reference model



enables the usual operations such as rotation, translation, juxtaposition, etc. This is again a closed space under these operations, which are different from others that will map graphic-forms on 'images' via graphic-form-attribute instantiation. Images are the only objects which can be directly displayed.

Figure 1 above sketches a possible scheme for these conceptual levels. A more detailed specification will be given in [32]. New ideas (or changes to the existing ones) are likely to arise during formal development.

## 5.- Topics for Discussion

The (intuitive) notion of a 'graphic archetype' sketched above raises several issues for discussion. Firstly, we need to formally define this concept. Secondly, we need to experiment with graphics specification and development using candidate algebras for archetypes. Practical insight thus gained will provide useful "feedback" which we hope will improve the concept and its formal description. Finally, on exit from such a "cycle of improvements" we intend to implement these ideas into our practical graphics environment (see the appendix for a brief description of this environment).

We are currently working at the first and second levels above. The first task appears to be somewhat dependent on which final graphics programming style will be adopted by the system. If a procedural style is chosen, we are led to a "static" view of archetypes (considered just as another kind of object within the system, and stored somewhere in a graphics-knowledge database). By contrast, if the functional approach (in the style of Henderson [7]) is to be retained, we may use a "dynamic", higher-order approach to archetypes. For example, consider the following excerpt of an algebraic specification of circles:

```
. . .
mkPoint: Real X Real -> Point
mkCircle: Point X Real -> Circle
. . .
```

whereby eg. the term 'mkCircle(mkPoint(0,0),10)' specifies the circle centred upon the origin of the Cartesian plane whose radius is 10. What does it mean to write the expression

'mkCircle(mkPoint(0,0))'?

We may regard it as a "higher-order" term assuming that "currying" of arguments is an admissible meta-operation of our algebraic notation. Such an expression no longer specifies a circle, but rather the 'class' of all circles centred upon the origin of the Cartesian plane. Defining

```
mkCircles:  Real -> Circle
mkCircles  = λr.mkCircle(mkPoint(0,0),r)
```

mkCircles(10) will specify the same (concrete) circle as above.

Now, what does it mean to simply write

'mkCircle' (ie. 'λcr.mkCircle(c,r)')?

Since both radius and centre are "left unspecified" we are actually referring to the broadest class of circles (namely the class of all circles) - ie. the archetype 'circle'. Since higher-order functions are "first-class citizens" in functional programming, we may rely on the implementability of archetypes viewed in this way.

However, we wish a more flexible functional-processor with respect to "currying". Such a framework should take isomorphism laws such as

$$C \uparrow (A \times B) \cong C \uparrow (B \times A)$$

ie. A -> (B -> C) ≅ B -> (A -> C), into account. This will make higher-order terms such as 'mkCircle(10)' (cf. "all circles whose radii are 10") also valid in an immediate, elegant way. In summary, our algebraic framework must encompass the notions of "incomplete (higher-order ) term" and argument-ordering-independent "currying".

The design of a graphics editor which supports archetypes (as described above) will be discussed in [33]. This editor is parameterized wrt. the algebra chosen for specifying the graphical objects to be edited (this will make it easier for us to experiment with alternative algebras for graphics). This editor is also syntax-oriented, ie. it is governed by the signature of its parameter in a way such that only valid terms of the word-algebra can be built.

We are currently studying the formal implications of the ideas sketched above. We do not want to commit ourselves to a particular programming style before we dwell a bit longer on these problems. The formal components of our project are still in their infancy and further research is necessary in order to "polish" the ideas sketched in this preliminary report.

## Appendix - Outline of the AGSYS Architecture

The AGSYS graphics system under design will provide the user with "intelligent", interactive graphic pattern design tools. Its main components are the NUCLEUS, the GRAPHICS EDITOR, the DATA BASE, the IMAGE PROCESSOR and the DISPLAY SURFACE (Figure 2), cf. [32] for a more elaborated description.

The core of AGSYS is the NUCLEUS which represents the user's actual working context during any stage of a picture design. The NUCLEUS is initially empty. This means that, at the starting stage of a pattern design process, no graphical objects have yet been defined. However (optionally) it may be "bootstrapped" with some "initial knowledge" resulting from previous sessions.

The NUCLEUS may contain three distinct classes of graphical entities, corresponding to different stages of a picture generation process. These entities are archetypes (the "geometric knowlegde" of the system), abstract forms (the representation of the geometrical features of each graphical object) and abstract images (graphically well defined objects which represent pictures to be visualized).



The EDITOR provides the user with high-level conversational commands for creating, retrieving or composing such entities. In fact, the system's overall "intelligence" is partly provided by this editor [33].

At a lower level, the NUCLEUS must be supported by (and interact with) a graphic-knowledge DATA BASE. The interaction is realized by means of predefined commands. For example, "bootstrap" commands may be invoked to initialize a new nucleus with a useful set of predefined graphics-manipulating functions, models, forms, images, etc.

The user works over the NUCLEUS and the DATA BASE in an interactive way via the EDITOR. Alternatively, high-level languages (HLL) may serve as an interface. Whilst editing graphic objects the user may visualize them on a display device. This helps in controlling the evolution of the objects that will constitute the pattern.

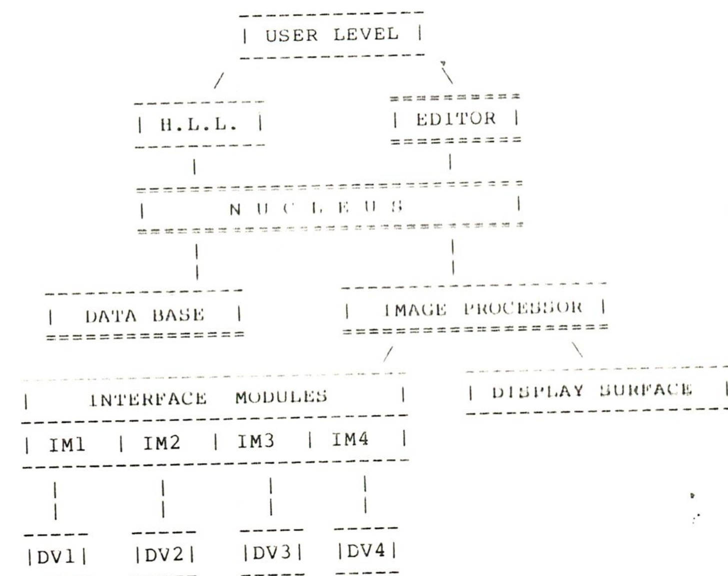


Figure 2 - AGSYS Architecture

We intend to minimize the device-dependence of the generated code by means of the definition of a single virtual (abstract) device. The IMAGE PROCESSOR generates graphic-code for this abstract device. This code will produce pictures on different graphic devices provided that adequate device interface-modules have been implemented. Additional device interface-modules may be plugged in the system, adding power without sacrificing the system's portability. We are also concerned with the source-code portability and so the final system will be written in a widespread language (we believe C to be a sensible choice).

Acknowledgements

We are grateful to our colleagues at CCES who have contributed to the ideas sketched in this paper. In particular, the concept of "archetype" was first suggested by Prof. J. Valenca. On the pragmatic side, Leonor Barroca and Jose Joao provided much stimulus by implementing a 'Functional Geometry'-based package suitable for the experimental part of the project.

Finally, we wish to thank the anonymous referees for many comments and suggestions which improved the initial draft of this paper.

References

[1] Jones C. B., "Software Development - A Rigorous Approach", Prentice-Hall International, Inc., Series in Comp. Sc. (Hoare C. A. R. -Ed. ), Englewood Cliffs, New Jersey, 1980.

[2] Goguen J. A., "Parameterized Programming", Proc. of Workshop on 'Reusability in Programming', (Perlis A. -Ed. ), 1983.

[3] CIP Language Group, "The Munich Project CIP", Vol 1 - 'The Wide Spectrum Language 84', T. Univ. Munchen, December 1983.

[4] Partsch H., "Structuring Transformational Developments: A Case Study Based on Earley's Recognizer", Science of Computer Programming 4, 17-44,1984.

[5] Bauer F. L. and Woessner H., "Algorithmic Language and Program Development", (Gries D. -Ed. ), Springer-Verlag, 1982.

[6] Henderson P., "Functional Programming: Application and Implementation", Prentice-Hall International, Inc., Series in Comp. Sc., (Hoare C. A. R. -Ed. ), Englewood Cliffs, New Jersey, 1980.

[7] Henderson P., "Functional Geometry", Proc. of the 1982 ACM Symp. on LISP and Functional Programming, 179-187, 1982.

[8] Arya K., "A Functional Approach to Picture Manipulation", Comp. Graphics Forum 3, 35-46, 1984.

[9] Burstall R. M., MacQueen D. B. and Sannella D. T., "HOPE: An Experimental Applicative Language", Int. Report CSR-62-80, Edinburgh Univ., May 1980 (updated February 1981).

[10] Darlington J., "Program Transformation", in 'Functional Programming and Its Applications: An Advanced Course', Newcastle Univ., 20-31 July 1981 (Darlington J., Henderson P. and Turner D. A. -Eds. ), Cambridge Univ. Press (England), 1982.

[11] Minkowitz C., "A Methodology for the Prototyping of Software Design", Tech. Report TR. 14, Dept. of Comp. Sc., Stirling Univ., September 1984.

[12] Bjorner D. and Jones C. B., "Formal Specification and Software Development", Prentice-Hall International, Inc., Series in Comp. Sc., (Hoare C. A. R. -Ed. ), Englewood Cliffs, New Jersey, 1982.



[13] Henderson P., "Specifications and Programs", in 'Software Requirements, Specification and Testing' (Anderson T. -Ed. ), Blackwell Scientific Publications, 1984.

[14] Goguen J. A., "Geometrical Constructions", LNCS 134 (Proc. of the Workshop on Program Specification, Aarhus, Denmark, August 1981), 31-46, Springer-Verlag, 1982.

[15] Mallgren W. R., "Formal Specification of Interactive Graphics Programming Languages", Doctoral Dissertation, Washington Univ., 1982 (publ. by ACM-MIT Press Distinguished Dissertation Series, June 1983).

[16] Mallgren W. R., "Formal Specification of Graphic Data Types", ACM ToPLaS 4(4), 687-710, October 1982.

[17] Gnatz R., "An Algebraic Approach to the Standardization and the Certification of Graphics Software", Comp. Graphics Forum 2(2/3), 153-166, August 1983.

[18] Gnatz R., "Approaching a Formal Framework for Graphics Software Standards", Comp. and Graphics 8(1), 39-50, 1984.

[19] Carson G. S., "The Specification of Computer Graphics Systems", IEEE Comp. Graphics & Applications, September 1983, 27-41.

[20] Carson G. S., "An Approach to the Formal Specification of Computer Graphics Systems", Comp. & Graphics 8(1), 51-58, 1984.

[21] ANSI X3H3 Formal Specifications Subcommitee, "The Formal Specification Aspects of the Standard Graphics System", ANSI doc. X3H3/79-35, 1979.

[22] Milne G. and Milner R., "Concurrent Processes and Their Syntax", JACM 26(2), 302-321, April 1979.

[23] Ward S. A. and Halstead R. H., "A Syntactic Theory of Message Passing", JACM 27(2), 365-383, April 1980.

[24] Carson G. S. and Prost E., "A Formal Specification of the Programmer's Minimal Interface for Graphics", ANSI doc. X3H3/82-156,November 1982.

[25] EEC, "Formal Specification of Graphics Software Standards", Report on the EEC Workshop on Graphics Certification, Steensel (the Netherlands), 8-9 June 1982.

[26] Duce D. A., Fielding E. V. C. and Marshall L. S., "Formal Specification and Graphics Software", RAL-84-068, Rutherford Appleton Laboratoy, August 1984.

[27] Marshall L. S., "A Formal Specification of Line Representations on Graphics Devices", Int. Report, Dept. of Comp. Sc., Manchester Univ., September 1984.

[28] Marshall L. S., "GKS Output Workstations: Formal Specification and Proofs of Correctness for Specific Devices", Int. Report, Dept. of Comp. Sc., Manchester Univ., September 1984.

[29] Minkowitz C., "Specification to Prototypes — A Comparison of Two Formal Methods of Software Design", Tech. Report TR.18, Dep. of Comp. Sc., Stirling Univ., December 1984.

[30] Duce D. A. and Fielding E. V. C., "Better Understanding Through Formal Specification", RAL-84-128, Rutherford Appleton Laboratory, December 1984.

[31] Pereira F., "Can ~~Graphics~~ [Drawing] Be Liberated From The Von Neumann Style? ", Comp. Sc. Laboratory, SRI International, Menlo Park, California, [June] 1983. [(Technical Note 282)]

[32] Martins F. M., Oliveira J. N., "Rigorous Specification of a Graphics Environment", Int. Report, CCES, Minho Univ., ~~March 1985~~ (in preparation).

[33] Oliveira J. N., Martins F. M., "Algebraic Generation of Syntax-[d]Directed Editors - a Graphics Application", Int. Report, CCES, Minho Univ., ~~March 1985~~ (in preparation).