Programming Paradigms and the Semantics of Geometric Symbols
(extended abstract)

W M Beynon[*], *S B Russ*[*], *Y P Yung*[*], *A J Cartwright*[†]

[*]Dept of Computer Science and [†]Dept of Engineering,
University of Warwick, Coventry CV4 7AL, UK

Introduction

The development of interactive computing has inspired interest in
- the use of geometric symbols rather than textual messages
- the presentation of active *environments* rather than passive *documents*.

Two fundamental technical problems respectively associated with these concerns are:
- developing computer representations that reflect the semantics of geometric data
- constructing comprehensible state-based models for environments.

Our main thesis is that there is a strong connection between these apparently separate problems. We shall argue that
- the semantics of a geometric object has to be understood with reference to its potential transformations,
- the effective representation of such transformations requires more powerful and expressive state-based computer models than have yet been developed.

Modelling the Semantics of Geometric Objects

An association between the semantics of geometric objects and the transformations to which they can be subjected was first proposed by Felix Klein in his "Erlanger Program" address (1872). The geometry of a figure is defined by those properties that are invariant under a specified class of transformations. In order to define the semantics of a geometric object, we associate with a specified object the class of permissible transformations that can be performed on it. The technical problem we have to address is the precise specification of this class of transformations. The validity of our specification will depend upon whether the possible transformations of symbols we describe conform to the possible appropriate state changes within the "real-world" application that the symbols purport to model.

As a familiar example, the icons of a desktop display typically represent accessible files. The nature of these files (e.g. whether they are executable or are data for a system application program) is indicated by the form of the icon. The deletion of a file is associated with removing an icon from the display. A distinguished subset of geometric transformations that can be performed on a displayed set of icons is in 1-1 correspondence with meaningful operations upon the associated physical files, such as the deletion or creation of a file or its conversion into a new format. The validity of the geometric interface depends upon the fact that this set of geometric transformations of the display corresponds precisely to the set of meaningful operations on physical files.

The transformations of geometric data that symbolically describe file management are very simple compared with those that might arise in applications where complex geometric models are used, but the same principles can be applied in general. By way of further illustration, consider Figure 1. Under one interpretation, the figure denotes a counter that is currently displaying "89". If we wish to model the behaviour of the counter faithfully, there will be two appropriate transformations, one associated with incrementing the counter to display "90", the other with resetting the display to "00". Such semantically significant transformations of a symbol will be called *interpretable*.

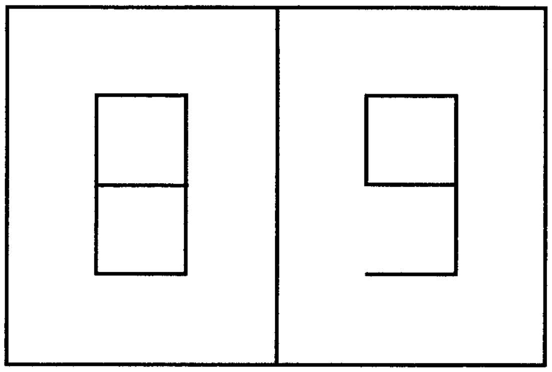

Figure 1

Under another interpretation Figure 1 is a plan for the furniture layout in a pair of adjacent rooms. The "8" symbol is a filing cabinet that is currently open, the "9" symbol the floor plan of a desk. The rooms have doors that are presently closed. In this case, the interpretable transformations of Figure 1 are of quite another kind, and correspond to actions an occupant might perform to change the state of a room, such as opening the door, moving a desk, or closing a filing cabinet.

Figure 1 has many other possible interpretations. The class of interpretable transformations is typically to be understood with reference to what agent is acting in the application and what privileges the agent possesses. Privileges express restrictions on the transformations that can be applied, for example ensuring that a desk does not intersect a wall. The consideration of agents enables us to express the fact that the same image can simultaneously have several alternative semantics. For example, reorganising the icons on a desktop display does not affect the status of the associated physical files, but is meaningful as a mode of presenting file information to the user.

The transformations of Figure 1 that correspond to updating a counter do not involve changing the presentation format for the digits. If on the other hand, the agent acting in the application is a graphical designer, it is necessary to consider transformations that modify the size and choice of representation for the digits "8" and "9", their relative position, or the colour of the background display. Similarly, when conceiving Figure 1 as an architectural plan, an architect might wish to modify the dimensions of the room, or relocate the door.

The above discussion motivates the development of a method of representing a geometric object so that we can formally describe:

- the set of interpretable transformations that can be applied to it
- the set of agents that can perform such transformations upon it
- the conditions that constrain the performance of these transformations.

Our progress towards this objective is based upon of a new style of programming to be briefly outlined and illustrated below.

A Programming Paradigm for Graphics

The drawing process is typically viewed as a means to an end: that of describing a particular static image (such as Figure 1) without regard for constructing a significant abstract representation. From our perspective, a geometric object cannot be adequately represented by a static image; to reflect the semantics of the object it is essential to describe its geometric form in conjunction with appropriate protocols for transformation. This reflects the distinction between the frozen figure that must be displayed in a document and the dynamic figure that can be modelled in an interactive environment.

Traditional programming paradigms for graphics have not been designed with the formal representation of transformations of geometric objects in mind. A procedural drawing package supplies a plethora of transformations that can be used to construct an image, but no framework within which the concept of "interpretable transformations of an image" can be expressed. Declarative methods e.g. those based upon the use of constraints give inadequate support to the concepts of state and transformation. A constraint is effectively an assertion about the form of a static object. Information about how an object is transformed when maintaining constraints is supplied only in an implicit manner: via a constraint-satisfaction system.

The fundamental technical problem to be addressed is the formal specification of the set of interpretable transformations of an image. In conventional state-based computer models, such as are provided by a procedural graphics package, it is possible to describe transformations of a geometric object in an informal way but not to distinguish between interpretable and uninterpretable transformations. For example, a basic drawing package will provide a sequence of updating operations to modify the counter in Figure 1 so that the display "89" is transformed into "90", but this might take the form of introducing and deleting line segments in such a way that the counter temporarily displayed "88","80" and "90" in sequence. What is required is a method of expressing the indivisible nature of the transition from the state of displaying "89" to that of displaying "90".

A full discussion of the programming principles we apply to the solution of this problem is beyond the scope of this abstract. Our approach is based upon the representation of state by means of a set of interrelated definitions of variables, where each definition either specifies the value of a variable explicitly or defines it as the value of a formula that references other variables (without cyclic definition). The redefinition of a single variable changes the values of all variables whose value is dependent upon it in a conceptually indivisible fashion. The principle is similar to that applied when updating the cells of a spreadsheet.

The nature of the formulae used in the definitions depends upon the application. It is determined by choosing an underlying algebra of data types and operators over which to evaluate expressions. Figure 2 specifies Figure 1 as a room layout using the definitive (definition-based) notation DoNaLD [3] in which the underlying algebra consists of points, lines and shapes comprising sets of points and lines. The basic transformations - such as that corresponding to opening the door - are described by the redefinition of a single variable. As an example, Figure 2 shows two files of DoNaLD definitions to describe the symbol "8" in Figure 1. In that on the right, the symbol "8" is transformed to "9" by incrementing the variable digit; on the left, the open filing cabinet represented by the symbol "8" is closed by setting the variable open to **false**. The protocols below prescribe the interpretable transformations.

In our research, similar principles have been applied to the description of a variety of geometric models; these include Cayley diagrams (finite automata introduced as graphical representations of groups) [2] and more complex geometrical objects appropriate for 3-dimensional modelling [5]. Their application is not confined to graphics, however: our methods have been applied to general screen layout and to concurrent systems modelling [6]. In each of these domains, the essential principle that makes the spreadsheet such a valuable computational tool can be observed: the state transformations that occur faithfully reflect perceived state changes in the application domain.

```
openshape cabinet
within cabinet {
    int      width, length
    point  NW, NE, SW, SE
    line    N, S, E, W

    N = [NW, NE]
    S = [SW, SE]
    E = [NE, SE]
    W = [NW, SW]

    width, length = 300, 300

    SW = {100, 200}
    SE = SW + {width, 0}
    NW = SW + {0, length}
    NE = NW + {width, 0}

    openshape drawer
    within drawer {
        boolean       open
        int           length
        line          N, S, E, W

        length = if open then ~/length
                            else 0
        open = true

        N = [~/NW + {0, length},
            ~/NE + {0, length}]
        S = [~/NW. ~/NE]
        W = [~/NW + {0, length}, ~/NW]
        E = [~/NE + {0, length} , ~/NE]
    }
}

protocol {
    open -> open = false
    ! open ∧ ! locked -> open = true
    locked -> locked = false
    ! open ∧ ! locked -> locked = true
}
```

```
openshape led
within led {
    int          digit
    point        p1, p2, p3, p4, p5, p6
    line         L1, L2, L3, L4, L5, L6, L7
    boolean   on1, on2, on3, on4, on5,
                 on6, on7

    digit = 8

    p1 = {100, 800}
    p2 = {100, 500}
    p3 = {100, 200}
    p4 = {400, 800}
    p5 = {400, 500}
    p6 = {400, 200}

    on1 = digit != 1 ∧ digit != 4
    on2 = digit != 0 ∧ digit != 1 ∧ digit != 7
    on3 = digit != 1 ∧ digit != 4 ∧ digit != 7
    on4 = (digit == 0 ∨ digit >= 4) ∧
             digit != 7
    on5 = digit == 0 ∨ digit == 2 ∨
             digit == 6 ∨ digit == 8
    on6 = digit != 5 ∧ digit != 6
    on7 = digit != 2

    l1 = if on1 then [p1, p4] else [p1, p1]
    l2 = if on2 then [p2, p5] else [p2, p2]
    l3 = if on3 then [p3, p6] else [p3, p3]
    l4 = if on4 then [p1, p2] else [p1, p1]
    l5 = if on5 then [p2, p3] else [p2, p2]
    l6 = if on6 then [p4, p5] else [p4, p4]
    l7 = if on7 then [p5, p6] else [p5, p5]
}

protocol {
    true -> digit = | digit | + 1
    true -> digit = 0
}
```

Figure 2

References

[1] W M Beynon, *Definitive notations for interaction*, Proc. hci'85, "People and Computers: Designing the Interface", ed Johnson and Cook, CUP 1985, 23-34
[2] W M Beynon, *Definitive principles for interactive graphics*, NATO ASI Series F:40, Springer-Verlag 1988, 1083-1097
[3] W M Beynon, Y W Yung, *Implementing a definitive notation for interactive graphics*, New Trends in Computer Graphics, Springer-Verlag 1988, 456-468
[4] W M Beynon, *Evaluating definitive principles for interactive graphics*, New Advances in Computer Graphics, Springer-Verlag 1989, 291-303
[5] W M Beynon, A J Cartwright, *A definitive programming approach to the implementation of CAD s/w*, Intell CAD Syst 2, Springer-Verlag 1989, 126-145
[6] W M Beynon, M T Norris, R A Orr, M D Slade, *Definitive Specification of Concurrent Systems*, Proc UKIT'90, Southampton, March 1990 (to appear)