Compatability and Interaction Style In Computer Graphics

George W. Fitzmaurice and Bill Buxton
Alias|Wavefront Inc.

Introduction

Recent trends in human computer interaction have focused on representations based on physical reality [4, 5, 6, 8]. The idea is to provide richer, more intuitive handles for control and manipulation compared to traditional graphical user interfaces (GUIs) using a mouse. This trend underscores the need to examine the concept of manipulation and to further understand what we *want* to manipulate versus what we *can* easily manipulate. Implicit in this is the notion that the bias of the UI is often incompatible with user needs.

The main goal of UI design is to reduce complexity while augmenting the ability of users to get their work done. A fundamental belief underlying our research is that complexity lies not only in what is purchased from the software and hardware manufacturers, but also in what the user creates with it. It is not just a question of making buttons and menus easier to learn and more efficient to use. It is also a question of "Given that I've created this surface in this way, how can it now be modified to achieve my current design objective?" (The observation is that how the user created the surface in the first place will affect the answer to the question.) Our thesis is that appropriate design of the system can minimize both kinds of complexity: that inherent in accessing the functionality provided by the vendor, *and* that created by the user. The literature focuses on the former. In what follows, we investigate some of the issues in achieving the latter. In so doing, we structure our discussion around questions of *compatibility*.

Three Perspectives on Compatibility

When we consider manipulation in the context of 2D and 3D graphics applications, we often consider properties of the input device such as degrees of freedom, and how well the device is capable of moving points or shapes in this space. That is, how well can the user move and adjust interactive widgets as well as directly transform geometry (e.g., adjust points, curves and surfaces). What is often not considered, however, is the ease of manipulating the underlying structure of the graphics (e.g., the deep structure of the geometry, as represented by the scene graph, etc.), even when these are data that were directly or indirectly created by that same user.

Many users, especially artists, do not understand the deep structure, or how it is represented — in fact, many prefer not to. However providing access and acquiring such understanding is often necessary for users to achieve their goals. Thus, we must find ways of exposing the deep structure to the user in ways that are compatible, intuitive and efficient, and which enable the user to work at this level when appropriate.

To explore this issue, we examine three styles of interaction in terms of their ability to support manipulation of both the deep and surface levels of the graphics. We do so by considering the issue of *compatibility* between input and output devices in addition to the ability of the user to manipulate internal representations ("deep structure") as well as external representations ("surface structure") of the application data. This is somewhat akin to the model-view-control organizing structure of SmallTalk [7]. Let us look at each one of these compatibilities.

- *Input with output devices*: Does the input device match the capabilities of the output display?
- *User interface with the ability to manipulate internal representation (access to "deep structure"):* Can the user interface effectively manipulate the internal representation of the application data stream?
- *User interface with the ability to manipulate external representation (access to "surface structure"):* Can the user interface effectively manipulate the artifacts generated by the internal representations?

For example, let us consider a form-based UI on a database of records that use an alphanumeric keypad input device. From the first perspective, this has good compatibility since the primary type of data being input and displayed is alpha-numeric. The deep structure might be considered the layout of the forms, and what attributes of the underlying database are exposed and in what relationship. Since layout, at least, is a spatial thing, the keyboard would likely have low compatibility, compared to a mouse, in terms of interacting at this level. Finally, in interacting with the fields exposed, the surface structure, there is medium to good compatibility. In entering the alphanumeric data the compatibility is high. However, tabbing from field to field using the keyboard may often be less compatible than doing so by selection with a mouse.

In this paper we explore the progression of manipulation as it relates to input devices, deep structure and surface structure. We do this by giving two historical examples and then discuss a new trend which we call "interactive assemblages." We will frame our discussion within the context of sophisticated graphics applications such as computer aided design, modeling, compositing and animation.

Example 1: APL and Teletype

I/O Compatibility — High
UI to Manipulate Deep Structure — High
UI to Manipulate Surface Structure — Low

To begin, consider the use of the programming language APL in computer graphics in the 1960s. The language was cryptic and terse, but the matrix handling was wonderful. "The power of APL comes from its direct manipulation of n-dimensional arrays of data. The APL primitives express broad ideas of data manipulation. These rich and powerful primitives can be strung together to perform in one line what would require pages in other programming languages."[1] Interaction was via an IBM Selectric typewriter-like terminal, and compared to the card-punch readers typical of the era, it was very interactive.

I/O Compatibility — High
The input devices (characters on the keyboard) and the output (printed characters) had a strong compatibility. IBM Selectric typewriters were very popular and familiar to users. Moreover, the symbolic nature of the language lent itself to typing.

UI to Manipulate Deep Structure — High
There was a high compatibility in the manipulation of the deep structure of the graphics. This was by virtue of the language's facility in manipulating n-dimensional arrays, performing matrix multiplication operations and programming new functionality.

UI to Manipulate Surface Structure — Low
However, there was a strong incompatibility in the manipulation of the surface structure of the graphics. APL processed numeric arrays of data, not graphics images per se. While the numeric arrays may represent a graphical image, such as a set of curves, a user could not *directly* adjust the contour of a curve within the graphical domain.

Consequently, the user was forced to function exclusively at the abstraction level of the graphics, manipulating the internal representations, or deep structure. The user interface was designed and optimized to facilitate the manipulation of the graphics as represented by the APL notation, not by way of the pictures themselves. Nevertheless, APL was used in the 1970s to produce a number of innovative animations such as by Judson Rosebush's company, Digital Effects Inc.

Example 2: GUI Direct Manipulation

I/O Compatibility — High
UI to Manipulate Deep Structure — Low
UI to Manipulate Surface Structure — High

In this example we leap ahead approximately 20 years and consider the shift towards *direct manipulation* through the development of the graphical user interface (GUI). The GUI significantly improved our ability to manipulate the surface structure which was so lacking in the previous example.

I/O Compatibility — High
The high compatibility of the input and output devices with GUIs is based on the use of graphical metaphors. Tools and other entities are represented graphically, often as icons, and one interacts with them using generic actions such as pointing and dragging. Interaction is mediated through the use of a graphical input device such as a mouse, tablet, trackball or touch screen, typically in conjunction with a pixel addressable graphics display. While graphical representations and metaphors are used to regulate input and output, there is still a level of indirection employed which serves as an abstraction above that of physical reality (for example, with a GUI one doesn't grab a document to move it up or down. One does this indirectly, using a widget such as a scroll bar or scroll arrow, which is itself a metaphoric icon).

Moreover, while graphically the GUI's output is explicitly representational, the input is limited and generic. For example, while I may be presented with a graphical ruler, the typically one-handed, single device input mechanisms available only permit me to perform a small subset of the overall bimanual actions or gestures that I might employ with a real ruler. Just compare how you perform actions like bending, flipping, squeezing, shaking, tilting or moving objects in the physical world with how you do so using so-called "direct manipulation" with a GUI.

In summary, while input and output are compatible in manipulating the surface structure, we need to keep in mind that this holds true only within the range of actions available. Between how one interacts with the graphical objects in the GUI and their counterparts in the physical world, there is limited compatibility at best, and incompatibility at worst.

UI to Manipulate Deep Structure — Low
The underlying structure of complex graphical scenes and objects does not generally lend itself to effective graphical representation, that is, in terms of actions that you want to perform on it. While I may "know how" to manipulate things in the micro sense (point and click) I still may have problems achieving my goals in a macro sense. Consider Alias|Wavefront's *Maya* animation package [9], as an example. In this application, 3D geometry is available to the user in a number of representations. One is a 3D perspective view. Another is as a 2D dependency graph which reflects the deep structure of the graphics. Users are able to manipulate the graphics within this view. However, while this exposes the internal representation to the user in a form that can be manipulated, actually doing so can still be quite cumbersome.

This is sometimes due to a notational issue: the graphical representation or interaction techniques available may not be appropriate to the task at hand. Other times, however, it is due to the data itself being structured in a way that is incompatible with the manipulation that the user wants to perform (even though it was likely that same user who structured it that way.)

The former notational issue can often be addressed by offering another form of representation and manipulation. This is partially why packages such as Side Effects *Houdini* and *Maya* still support procedural scripting in addition to the GUI: this enables the user to manipulate the geometry and perform global, large scale manipulations at a variety of granularities. This scripting ability is more akin to the previous APL example than direct manipulation.

The latter point, the compatibility of the underlying structure itself to the types of manipulations that the user wants to perform is addressed by, and motivates, Example 3, below.

UI to Manipulate Surface Structure — High
There is a high compatibility between the GUI and the ability to manipulate the surface structure of the graphics. This is because it enables the artist to work directly on an image or 3D model. Note that this is not always done. Rather, almost more often than not, intermediate dialog boxes are used to manipulate parameters resulting in what might more properly be described as *indirect manipulation*.

Some work has been done to address this issue, such as building 3D manipulators and attaching them to 3D geometry [3]. Such manipulators provide efficient handles which afford control over key parameters of the underlying structure. They bring such control closer to the geometry itself than is the case with dialogue boxes, making control more direct.

In summary, GUI-based systems generally do a fairly good job of supporting the capability to work directly on the graphics. However, they also have affordances that bias and deter users from understanding the deep structure. Thus, users often have trouble manipulating the graphics at the appropriate level of detail or abstraction for the task at hand. The bias is towards the surface structure. Consequently, as the complexity of the (user created) graphical scene, object or animation increases, the effectiveness of the UI breaks down. Without a grasp of the deep structure and an effective handle on it, users are mired in too much detail, detail which becomes overwhelming regardless of how simple any single step of that detail is.

Resolving the Dilemma

Given these two examples, we are presented with a dilemma. How can we design interactive systems which maintain the balance of manipulability and compatibility over deep and surface structure? Already we have alluded to a hybrid approach (*Houdini* and *Maya*) which offers both scripting and direct manipulation solutions. But from the perspective of artists, direct manipulation is generally preferred since it is closer to how they work in the physical world. Consequently, intense scripting is typically delegated to a programmer (technical director) who is fluent in the scripting language.

What we would like to do in the remainder of this paper is discuss an alternative approach to finding a balance between control over the deep and surface structure of the graphics. This approach has more to do with how the user creates the graphics in the first place, rather than the mechanics of the UI per se. Here, one assembles the graphics using higher order primitives. The assumption is that much of the complexity in the deep structure in conventional systems stems from the primitives used being at too low a level. If the granularity of how one thinks about a model, for example, matches that of the primitives used to construct or modify it, then the assumption is that the underlying complexity for the user will be significantly reduced. Such primitives may be procedural or declarative. The key point is that they be compatible with how the user thinks, and that they afford control at the appropriate level of detail or granularity appropriate for achieving the user's goals. We will refer to this approach as *interactive assemblages*.

Example 3: Interactive Assemblages

I/O Compatibility — High
UI to Manipulate Deep Structure — Moderately High
UI to Manipulate Surface Structure — High

By interactive assemblages we mean the construction of graphical scenes, models and animations out of higher level components than is traditionally the case. Such components may be declarative (such as a canonical form or a generic skeleton with all of the IK handles and dynamics built in), procedural (such as a

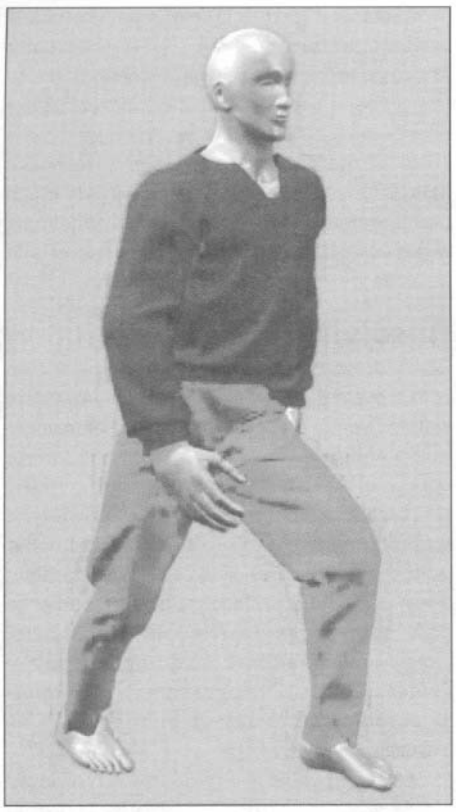

Figure 1: Example of cloth simulation on a computer character [2]. ***See page 103 for color image.***

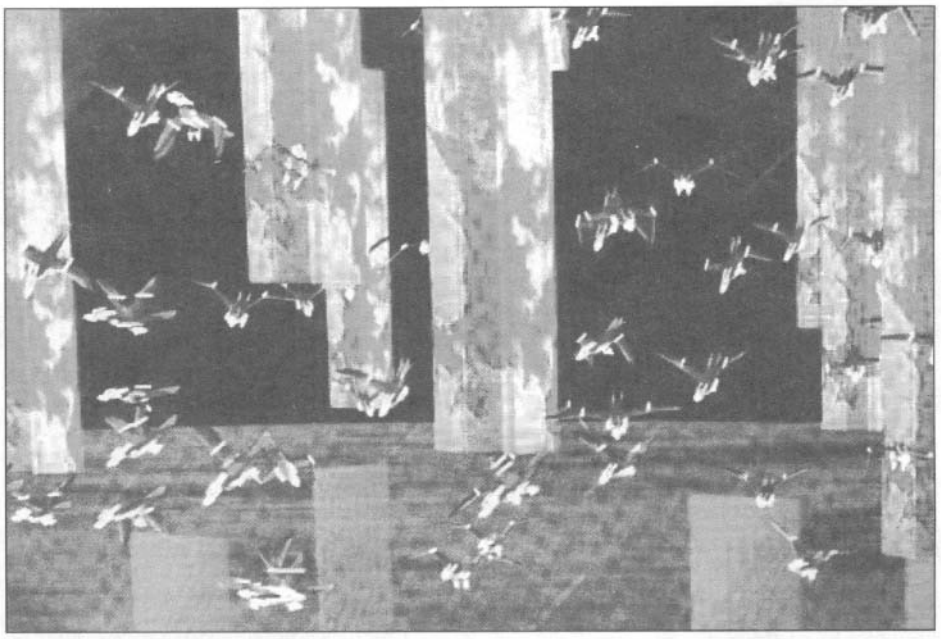

Figure 2: Example of a flock of birds [10]. ***See page 103 for color image.***

flocking module) or some hybrid of the two. Each integrates sophisticated behaviours or data into a single component having a manageable but rich set of appropriate operating parameters. These operating parameters, or handles, may be exposed to the user via virtual means (such as dialog boxes or 3D manipulators) or through physical means (such as specialized input devices like physical sliders, dials or customized graspable objects).

The deep structure now reveals the relationship among these components, rather than the low-level primitives traditionally used. Thus, there is an increased likelihood that any representation of the deep structure will be both more comprehensible to users, and afford manipulation at a level appropriate for the task at hand.

Let us describe three example modules to illustrate this idea: modeling cloth, specifying flocking behaviour and car design.

Modeling Cloth

Imagine animating a super hero who is wearing a cape. Specifying the behaviour of the cape through a sequence using keyframe techniques is as tedious as it is time consuming, so much so that the ability to experiment with different variations is extremely limited. On the other hand, if one has a generic parametrized cloth module (see Figure 1), the generation of such variations becomes relatively easy. First, one "dresses" the character. The artist does so at a high level, specifying parameters such as size, seams, fabric properties, texture and how tightly the material should fit or follow the character. The cloth module simulator then can calculate the behaviour of the cloth as the character moves — factoring in the physical dynamics of the cloth material and real-world properties such as gravity and object collisions. With this approach, notice that the specification of the character's movement is independent of the specification of the behaviour of the cape. Certainly the former affects the behaviour of the other. But unlike traditional animation, if I change the character's movement, I don't have to reanimate the cape, and if I change the material of the cape, I don't have to reanimate the whole thing. The animator addresses things at the appropriate level with the ensuing freedom to explore a far broader range of possibilities with relatively little increase in cost. In terms of the UI, one can imagine defining virtual or physical sliders and controllers to adjust the parameters of the cloth. Perhaps a set of instrumented pieces of fabric may facilitate specification of cloth behaviours. The idea is that the deep structure of the cloth module is abstracted and users only need to set a few high level parameters to get sophisticated behaviours.

Specifying Flocking Behaviour

A second example is the animation of the behaviour of a crowd of people, a school of fish or a flock of birds. Traditionally, one animates each member of the group individually. But if it is the character of the group as a whole which is of concern, one can offer this level of control (see Figure 2). First, the artist specifies a few representative characters, then specifies an area where the crowd should occupy. The flock module then populates the area with a random placement of instances of the representative characters. High-level parameters can then be defined which move the crowd in a particular direction, make them more or less active, and control the percentage of the crowd looking in a certain direction, etc. Again, it would be very tedious for the artist to have to manipulate each individual character within the crowd consisting of hundreds of characters.

Car Design

The previous two examples were procedural. Our third example involves a module that consists of declarative data. It is a car module in which the designer is presented with a canonical model of a car. This is in contrast with current practice where the designer starts with a blank sheet and works from there. The model contains and encapsulates the necessary engineering specifications and manufacturing constraints, as well as the essence of the "style" of that particular product line (see Figure 3). The notion is that by modifying the base model, rather than building each concept from scratch, one can work faster, stay within the style, conform to the engineering criteria and end up with a much better initial geometry.

Given these three examples, let us now consider the manipulation and compatibility issues of interactive assemblages.

I/O Compatibility — High

This approach builds upon the GUI and direct manipulation paradigm. However, since we are exposing more handles of the deep structure to modify, assemble and control the surface structure, we have a stronger opportunity to

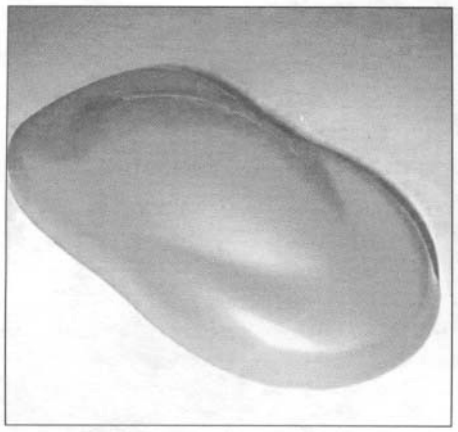

Figure 3: Example of a pre-fabricated base model of car. Model components compliments of General Motors. ***See page 103 for color image.***

manipulate these parameters using dedicated input devices. Thus, there are two main differences compared to the traditional GUI approach. First, the quantity of input is increased when we move towards offering multiple, dedicated physical input transducers for adjusting high level controls. This is in contrast to conventional GUIs having one graphical input device, such as the mouse, which serves as a "time-multiplexed" physical handle being repeatedly attached and unattached to a variety of logical functions of the GUI. Second, we shift to using more specialized form factors and capabilities for our input devices. These dedicated physical handles can serve as iconic input devices (physical icons) as well as graphical output.

UI to Manipulate Deep Structure — Moderately High

There is a moderately high compatibility with manipulating the deep structures. We are able to place direct manipulation UI on relevant parameters of the deep structures. These parameters are passed to procedural scripts or simulations which perform the real work. However, we are constrained by using only the parameters that are exposed to the user and by the components which are available for assembling. Nevertheless, we are now dealing with higher level components with meaningful handles.

UI to Manipulate Surface Structure — High

As with the GUI and direct manipulation example above, there is a high compatibility with manipulating the surface structure. Moreover, the user is augmented by the extensions mentioned above with the handles on higher level components. This gives users capabilities they would never consider doing by hand (e.g., adjusting each bird in a flock of 200 birds).

The results of these interactive assemblage components are a consequence of complex relationships among a number of variable parameters. For example, the cloth module may have 20 parameters to adjust to generate an enormous variety of cloth behaviour. Therefore it is not enough to understand the effect of changing one parameter but instead one needs to know the relationships among a set of parameters. The fundamental point is that having simultaneous manipulation through dedicated handles allows us to more effectively and rapidly explore these relationships (i.e., the parameter space).

There are at least three characteristics of the interaction that result from the interactive assemblages approach:

- Active exploration through the provision of rich control, including input handles (physical and conceptual) with good compatibility.
- A move from a "specification" to "exploration" style of interaction. The goal is to create a fertile ground for exploration but also to provide rich points of departure rather than starting from nothing.
- The exposure of portions of the deep structure at a high-level of abstraction. This allows users to leverage off of the deep structure using traditional GUI elements without as much of a need for an expert understanding of the micro level of the internal structure.

The net result is to place a manageable and compatible handle both physical and cognitive on the task of creation in this space.

Conclusions

In UI design we want to manage complexity through the use of structure. Today our elementary building blocks are at a very fine granularity and the onus is on the user (artist) to deal with the high overhead of creating, manipulating and managing structure from these atomic elements.

Our belief is that a significant part of the complexity of current systems lies within the structure of the data created by the user rather than in questions like "what does this button do?" The problem is, when building from scratch, one often doesn't know what structural features are needed until long after decisions have been made which make changing that structure, or working within it, overly difficult, expensive or even impossible. We see the same thing in software engineering where it often takes us multiple iterations to get our data structures correct. In this paper we have argued that we can address these problems for our users in much the same way we have dealt with them for ourselves: increase the granularity of the building blocks and shift the balance from a "creation" approach to one of "clone, modify and assemble."

There is a potential for strong tension between manipulating the surface and deep structure when there is a UI or representational discontinuity between the two structures. One step towards alleviating this is to display both structures and show how the deep structure changes as one manipulates the surface structure. Thus, users can build an understanding (perhaps limited) of the deep structure. Alternatively, we want to strategically migrate deep structure up to the surface. We suggest defining interactive assemblages as a way of bridging the gap between these two structures. The assemblages encapsulate the deep structure and offer high level control and handles within the same UI representation of the surface structure.

The key is to focus on manipulation and transformation rather than creation from scratch. Further, we want to manipulate relationships, both temporal and spatial, rather than individual parameters. Graspable systems should expose the deep and surface structure through the use of cognitive and physical handles of control.

Acknowledgments

This research was undertaken under the auspices of the User Interface Research Group at Alias|Wavefront. Special thanks to Gordon Kurtenbach for his discussions and for reviewing early drafts.

References

1. ACM SIGAPL, **http://www.acm.org/sigapl**.
2. Baraff, D. and A. Witkin. "Large Steps in Cloth Simulation," *SIGGRAPH 98 Proceedings*, 1998, pp. 43-54.
3. Conner, D. B., S. S. Snibbe, K. P. Herndon, D. C. Robbins, R. C. Zeleznick, and A. van Dam. "Three-dimensional widgets," *Proceedings of the ACM Symposium on Interactive 3D Graphics*, 1992, pp. 183-188.
4. Fishkin, K.P., T. P. Moran and B. L. Harrison. "Embodied User Interfaces: Towards Invisible User Interfaces," To appear in *Proceedings of EHCI'98* (Heraklion, Greece), 1998.
5. Fitzmaurice, G.W. Graspable User Interfaces, Ph.D. dissertation, University of Toronto, **http://www.dgp.toronto.edu/people/GeorgeFitzmaurice/home.html**, 1996.
6. Fitzmaurice, G.W., H. Ishii and W. Buxton. "Bricks: Laying the Foundation for Graspable User Interfaces," *ACM Proceedings of CHI'95*, 1995, pp. 442-449.
7. Goldberg, A. and D. Robson. *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Reading, MA, 1983.
8. Ishii, H. and B. Ullmer. "Tangible Bits: Towards Seamless Interfaces between People, Bits, and Atoms," *ACM Proceedings of CHI'97*, 1997, pp. 234-241.

9. *Maya* 1.0. 1998, Alias|Wavefront, **http://www.aw.sgi.com/**.
10. Reynolds, C.W. "Flocks, herds and schools: A distributed behavioural model," *SIGGRAPH 87 Proceedings*, 1987, pp. 25-34.

George Fitzmaurice is a Research Scientist at Alias|Wavefront, Inc. His research interests include computer augmented reality, physical-virtual interfaces, interactive 3D graphics and haptic input devices. He holds a Ph.D. in computer science from the University of Toronto, an M.Sc. in computer science from Brown University and a B.Sc. in mathematics with computer science from the Massachusetts Institute of Technology.

George W. Fitzmaurice & Bill Buxton
Alias|Wavefront Inc.
Toronto, Ontario
Email: **{gf, buxton}@aw.sgi.com**
Web: **http://www.dgp.toronto.edu/people/GeorgeFitzmaurice/.**

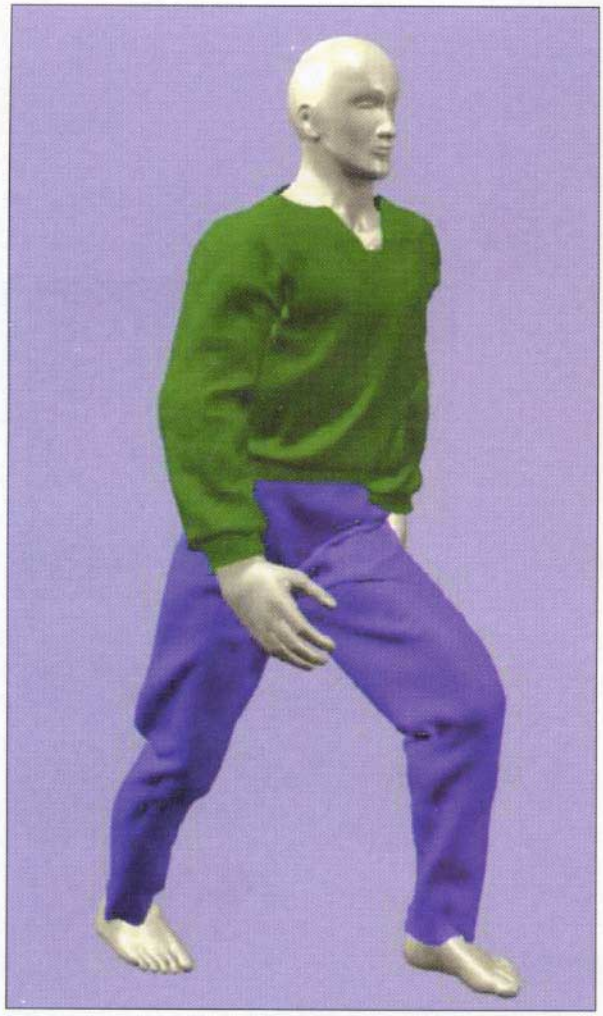

Figure 1: "Compatability and Interaction Style in Computer Graphics" by George W. Fitzmaurice and Bill Buxton. See pages 64-68.

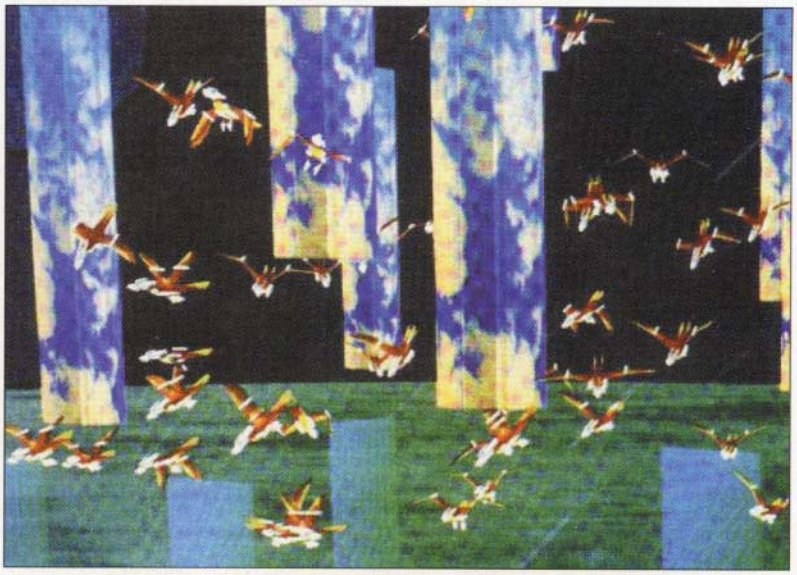

Figure 2: "Compatability and Interaction Style in Computer Graphics" by George W. Fitzmaurice and Bill Buxton. See pages 64-68.

Figure 3: "Compatability and Interaction Style in Computer Graphics" by George W. Fitzmaurice and Bill Buxton. See pages 64-68.