



HAL
open science

A multi-layer graphic model for building interactive graphical applications

Jean-Daniel Fekete

► **To cite this version:**

Jean-Daniel Fekete. A multi-layer graphic model for building interactive graphical applications. Proceedings of Graphics Interface '92, May 1992, Vancouver, Canada. pp.294-300, 10.20380/GI1992.34 . hal-02357999

HAL Id: hal-02357999

<https://inria.hal.science/hal-02357999>

Submitted on 11 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Multi-Layer Graphic Model for Building Interactive Graphical Applications

Jean-Daniel Fekete

LRI – CNRS URA410
Bat 490
Université de Paris-Sud
F-91405 ORSAY Cedex
+33 (1) 69 41 65 91

2001 S.A.
45, rue Camille Desmoulins
F-92230 CACHAN
+33 (1) 45 46 10 00
jdf@lri.lri.fr

Abstract

In this article, we present a model based on multi-layer graphics for building interactive graphical applications. With this model, a variety of components of an interactive graphical editor (transient objects like the selection rectangle, selected objects, grids, cursors), as well as the handling of input events, acquire a clearer status. We show how an interactive graphical application can use layered graphics in a simple and effective way to describe direct manipulation, some graphic constraints and events handling using various input devices. By giving a clear status to abstract elements of an interactive application, the model helps to describe a significant part of the dynamic aspect of the interaction.

Résumé

Dans cet article, nous présentons un modèle basé sur le graphique multi-couches pour le développement d'applications graphiques interactives. En utilisant ce modèle, il est possible d'attribuer un statut clair à de nombreux éléments composant un éditeur graphique interactif (les objets «fugaces» comme le rectangle utilisé pour sélectionner, les objets sélectionnés, les grilles d'alignement ainsi que les curseurs).

Nous montrons comment une application graphique interactive peut tirer profit du modèle graphique multi-couches pour décrire simplement la manipulation directe, certaines contraintes graphiques ainsi que la gestion des événements produits par divers périphériques d'entrée. En attribuant un statut clair à ces composantes abstraites des applications graphiques interactives, le modèle aide à la construction d'une partie importante de la description du comportement dynamique de l'interaction.

KEYWORDS: User Interface Design, Interface Metaphors, Input Devices.

Introduction

Building interactive graphical programs is still a difficult task. Some toolkits [13, 10] propose a framework for building task specific editors, but the kind of editors they can produce are stereotyped in several ways:

- they only consider keyboard and a mouse input,
- their graphic model is close to the PostScript model,
- their interaction is based on stereotyped graphic objects like button boxes and menus,
- they offer some stereotyped mouse and keyboard based manipulations.

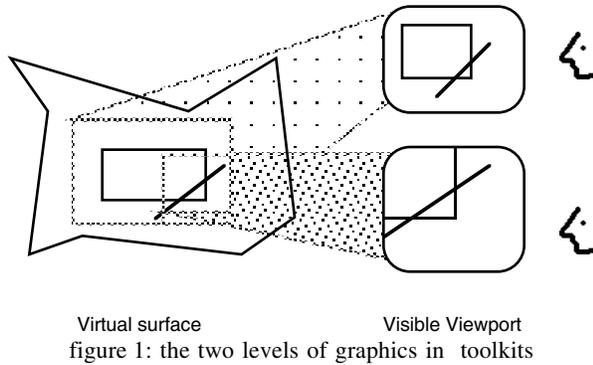
We believe that these limitations have become too strong, because of easily available new input devices, both 2D (Wacom pressure sensitive stylus, touch screens with multi touch sensitivity, eye trackers) and 3D (data gloves, flying mice, 3D stylus). Since the list of devices recognized by current toolkits is “hardwired” into the toolkit, almost no support is provided for application programmers to handle these new kinds of devices for input and to support graphic feedback of their actions.

We propose here a model based on multi-layered graphics and multi-layered input handling.

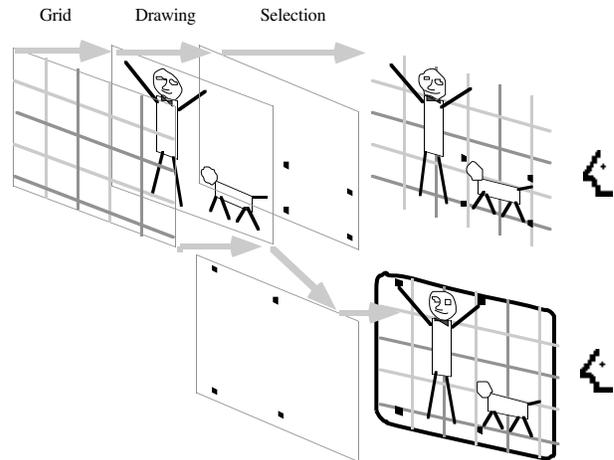
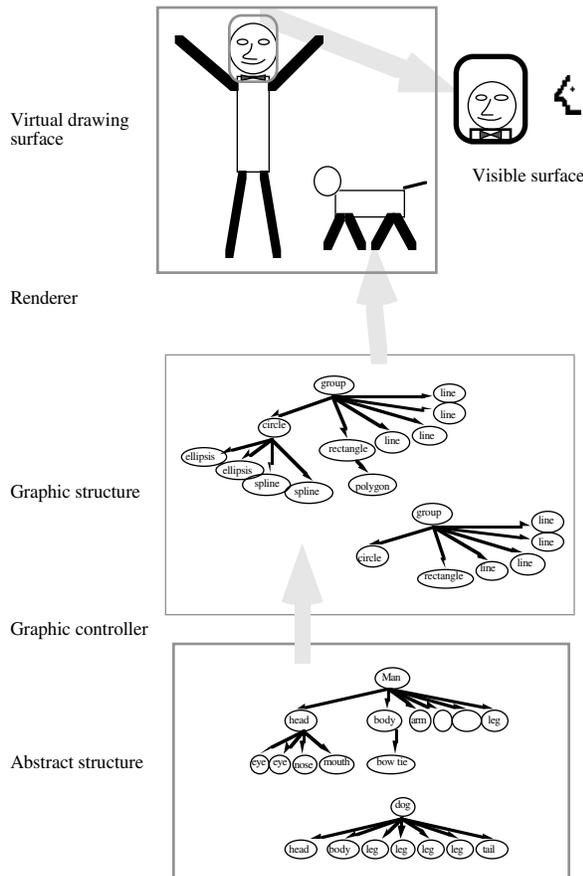
This model is also suitable for describing traditional mouse and keyboard based interactive programs, giving a clearer description of the handling of interaction, the handling of transient shapes, of selected graphic objects, of representations of constraints (*e.g.* a grid) and of device feedbacks.

Most graphical toolkits distinguish two levels for displaying graphics: a virtual surface where the graphics are drawn and one or several viewports which show some part of the virtual surface (see fig. 1).

In this model, the graphics on the virtual surface are produced by a **graphic controller**, a module which transforms an abstract data structure into a graphic data structure which can be displayed. Passing from the graphic data structure to the virtual surface is



done by a **renderer** module (see fig. 2). Projecting the virtual surface on the visible viewport is done through a transformation and a clipping zone. This model is used by most graphical systems and the mechanisms to implement it can be found in windowing systems like X [11], the Macintosh Toolbox [2], by graphical toolkits like InterViews [8], ET++ [14] or Garnet [10]. This model is well adapted to the display of graphics but is of little help for the interaction. For instance,



transient objects, like handles or sliding shapes, have no clear graphic status. They do not belong to the graphic data structure so their display and management is done by an internal part of the toolkit. This internal part is hard to adapt to new interaction and appearance styles.

The Multi-Layered Graphic Model

We propose a model, similar to the Xtv model, where the visible surface can display several virtual surfaces, like stacked transparent celluloids (see figure 3). A virtual surface still presents a graphical structure but a visible surface can display several stacked virtual surfaces. The same virtual surface can be shown on several different visible surfaces, at different levels. This is an extension to the multi-view concept.

This model is useful to specify both the display of graphics and the handling of events. Layers are drawn from the bottom (background) to the top (foreground) whereas events are handled from the topmost layer to the bottommost. In the following sections, we discuss the use of layers in typical applications, the handling of graphics, the event handling, the status of various components in our model and finally an example of its use in a real program.

The Layers

While using the Xtv toolkit developed at LRI [3], we have experimented with several interactive graphical layouts which led us to distinguish between the following layers:

- *Background/model layer*: this layer displays a background image. Conceptually, areas where no other layer has drawn contain the background image. Usually, window systems handle this layer in a special way. X has, for each window, a background image, usually a gray texture. In

programs like Illustrator™, this layer can contain a drawing which is then used as a template for the main drawing, like with tracing paper. In our model, this layer shows “default” graphics and handles events ignored by the other layers.

- *Graphical constraints visualization layer*: this layer usually displays a grid or another graphical formalism for representing geometrical constraints.
- *Application data layer*: this layer, as in figure 1, displays the graphical objects representing the application’s internal objects. Event handling in this layer is described later.
- *Selected objects layer*: this layer displays the selection, using shapes (e.g. handles) expressing the kinds of manipulations available. Event handling in this layer supports the direct manipulation.
- *Lexical operations representation layer*: this layer displays shapes expressing the status of input devices, like the cursors, as well as transient shapes (e.g. “zoom animation”).

Obviously, this list is not exhaustive, other layers can be found for specific applications. It does however apply to many applications (MacDraw, Illustrator, Idraw, etc.)

The model can manage many visible surfaces (multi-views) with some original properties: each virtual surface can be seen by any number of visible surfaces and each visible surface can display a virtual surface at any level.

Hence, the selection in a given view can differ from the selection in another view. The constraint grid can appear in one view and not in another, and be different in a third. A view can use a visible surface as a model whereas another uses it as the application data layer.

The Transformation Model

As explained in the previous section, the display of a virtual surface on a visible surface is done with a transformation and through a clipping region (usually the boundings of a window containing the visible surface). The graphic structure is displayed on the visible surface in a way similar to the way structured graphics packages (GKS [6], PHIGS [7]) build complex objects: by composition of simple objects geometrically transformed (see figure 4).

In the mono-surface model, the visible surface is responsible for scrolling and zooming in 2D, or changing the view point in 3D (these operations are sometimes called *non-semantic* manipulations.) The transformation is therefore associated to the visible surface.

Our model retains similar properties but requires that some objects are aware of the transformation applied on them and can manage it explicitly when being

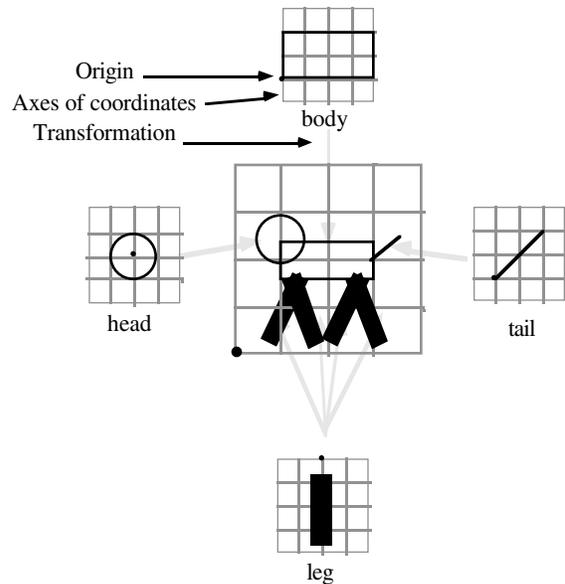


Figure 4: composition of structured graphic objects.

drawn on the visible surface. Specifically, objects can behave in three ways to redraw themselves with a specific transformation:

- use the transformation for each component,
- use the transformation for positioning but not for dimensions (lines width, handles width, ...),
- don’t use the transformation at all.

By construction, the application data layer follows the transformation for each component, since the transformation model has been chosen to represent its contents. The objects of the selection and the grid use the second behavior, they usually draw lines using a constant width of one pixel. The third is sometimes used to present graduated rules or display informations like the coordinates of the locator.

Implementation Issues

With the mono-layered model, most toolkits use a “lazy” redisplay mechanism: when a part of the visible surface should be updated, a function is called for the visible surface with the region to repaint as an argument. The toolkit is then responsible for redrawing the region. This mechanism is used either because that part has been made visible or because the graphics under it have changed (i.e. an object is created, modified or removed.) In this case, the data structure is updated and, for each visible surface where the virtual surface appears, the region to update is cleared and the redisplay mechanism is triggered.

In order to keep the simplicity of this mechanism with the multi-layer model, graphic objects should be able to actively control their drawing process to bypass the standard visualizing transformation. As explained in the previous paragraph, the redisplay

algorithm calls a drawing function with a region to draw as an argument. In our model, the visualization transformation should also be passed as a parameter to the drawing function of each layer.

It should be noted that when a graphic structure is modified and the redisplay algorithm is notified that an area should be updated, the area cannot be computed just once for the abstract surface and propagated to each visible surface where it appears, because, here again, the treatment of the transformation has to be taken into account for each visible surface in order to compute the exact area.

The lazy redisplay mechanism used by the MacIntosh toolbox as well as InterViews, ET++ and Xtv can easily be adapted to our model.

Note that no assertion is made about the graphic model used to draw in the application data layer. The multi-layered model can be used with bitmap graphics with an alpha value, enhanced bitmap graphics (supporting zoom), bitmap graphics with some structured primitives (QuickDraw [2], X), device-independant 2 1/2D painting (PostScript [1]) or 3D (PHIGS).

Handling of 3D

Note also that the model is, to some extent, suited to interactive manipulation of 3D graphics. Most 3D editors use a “wire frame” representation of 3D structures. The vertices are displayed using a transformation and manipulated using some 2D or 3D locator. The virtual surface contains 3D objects (virtual volume would be a better name) and the visible surfaces present a projection — through a visualizing transformation — of the virtual volume. The list of layers given for a conventional 2D graphical application still applies for a 3D editor:

- *the background* usually displays a solid color (either black or grey),
- *the graphical constraints visualization layer* can display a 3D grid or a trajectory,
- *the application data layer* displays the 3D objects,
- *the selected objects layer* displays handles of selected vertices or objects,
- *the lexical operations representation layer* displays the representation of the input device (the projection of a 3D mouse for instance).

The model can not be used when objects are edited with their hidden surfaces removed because the cursor and selection should be consistent with the 3D structure. This case is very unusual.

Handling of Events

A major benefit of multi-layer graphics is the description of interaction. Each layer handles events for which it has expressed interests. In this way, the control of the interaction is distributed. When it receives an event, a layer can either:

- ignore it and pass it to layers below,
- handle it,
- handle it first and let it pass to layers below after,
- handle it first, transform it and pass it to layers below.

For example, the figure 5 describes how the events are handled by each layer if the **selection** mode in a program works in the following way:

- When the mouse is clicked outside any object, the selection is cleared.
- If the mouse is then dragged, a selection rectangle is drawn and follows the mouse.
- If the mouse button is released, objects inside the rectangle are selected and the rectangle is deleted.
- When the mouse is clicked on a graphic object, it becomes selected.

Layer	Event	Behavior
Lexical:	PointerMoveDown	If a selection rectangle exists on the lexical layer, set its moving corner to the mouse position. Else, pass the event.
	PointerUp	If a rectangle exists on the lexical layer, select the objects of the main data layer inside the rectangle, then, delete the rectangle. Else, pass the event.
Selection:	PointerMoveDown	If the selection was being moved, move it with the pointer. If it was not moving and a ghost is under the pointer, move it with the pointer. Else, pass the event.
Data:	PointerDown	If an object is under the pointer, select it (create a ghost for it). Else, pass the event.
Background:	PointerDown	Clear the selection, then, create a rectangle on the lexical layer at the mouse position.

Figure 5: description of the **selection**

- If the mouse is clicked over a selected object and then dragged, a ghost¹ of the selection follows the pointer.
- When the mouse buttons are released, the ghosts are deleted and the objects they represented are moved to the position of their ghost.

This table also shows the declarative aspect of event handling with our model.

For exotic devices, the lexical layer can also represent a more sophisticated feedback than the traditional cursor. In our animation program, we use a pressure sensitive wireless Wacom digitizer where the hot spot is represented as a cross and the pressure as a circle, with a radius proportional to the pressure. The width of the cross is the maximal amplitude of the pressure, which depends on the selected pen thickness. A sketching program using the digitizer to draw in a similar way as a pencil consists simply of adding some actions to the layers above (see figure 6).

Layer	Event	Behavior
Lexical:	DigitMoveUp	Draw a cross (in Xor) centered at the hot spot and pass the event.
	DigitDown	Draw the cross and put the position and the pressure in a new list.
	DigitMoveDown	Erase the cross, draw a line from the previous point to the current point using the current pressure to compute its width, draw the cross at the current point and add the point and pressure to the list. Pass the event.
	DigitUp	Erase the cross, mark the area where the lines were traced as an area to redisplay, then, pass a new event called DigitTrace containing the list.
Data:	DigitTrace	Create an object with the list of points and select it.

Figure 6: adding actions for a pressure sensitive digitizer.

Logical Status of Layers

Most toolkits use a model inspired from the Small-talk Model-View-Controller (MVC) model: ET++ uses MVC, InterViews calls it Subject-View, Garnet uses a one-way constraint system. The idea of these models is to distinguish between the states of an object and its representations and to provide a mechanism to keep them coherent. The PAC

¹ We call **ghost** a transient graphic object which represents the selection of a graphic object within the main data layer.

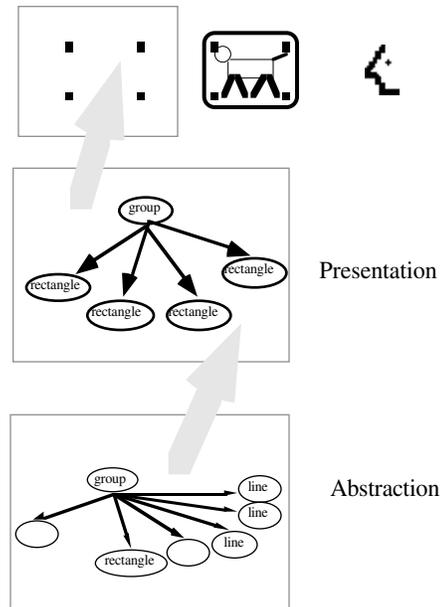


Figure 7: relationship between a graphic and its selection.

(Presentation, Abstraction, Controller) model of [4] is a generalisation of this concept of separation. By using it, we can assign a clear status to layers in our model.

In graphical applications, some abstract structure is to be displayed (see fig. 2). The graphic structure is considered as the presentation of the abstract structure. The graphic controller is responsible for keeping the graphical structure coherent with the abstract structure. It is also responsible for receiving input events and handling them according to the graphic semantics of the application. In most interactive graphic applications, the controller of this layer performs only "hit detection", that is, receive input events and interpret them as creating or deleting ghosts of objects designated by the events (e.g. mouse clicks). The manipulation – either direct or through menus – is usually performed on selected objects.

With our model, we can describe the relationship between the graphic structure and the representation of its selection (its ghost) by a similar process (see fig. 7). The handles representing the selection is a presentation of the state "being selected" of the graphic structure.

As shown in the previous section, the ghost appears in the selected objects layer. In our model, the ghost is considered as a view of the graphical data structure representing the object in the main data layer. Most of the time, only geometric information is required to draw a ghost. The graphical aspect of the ghost can therefore be defined independently of the application. Of course, in an object oriented

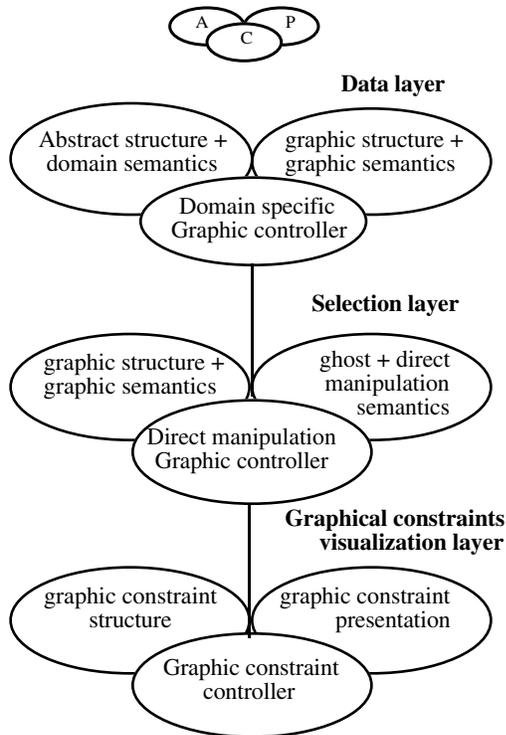


Figure 8: PAC relations between layers.

implementation, the ghost may be specialized to display some application specific information. The controller is responsible for maintaining the graphical coherence between the graphic structure of the main data layer and its ghost. It also interpret input events and handle the direct manipulation. When a manipulation is started, the controller maintains a graphic echo of the manipulation going on and can try to support it by applying constraints to each atomic manipulation. The constraints here can be either lexical, syntactic or semantic, depending mainly on performance issues. Lexical constraints (like grid alignment constraints) are independant of the graphic data structures and of the semantics of the abstract data structure beeing manipulated. Syntactic constraints (like non overlapping constraints) depend on the graphical structure. Semantic constraints (like hilighting only valid targets when interactively connecting two components of a graphic structure) depend on the graphic structure and the semantic structure. Once the manipulation is terminated, the controller can modify the abstract data structure or ignore the effects if the manipulation is not considered valid. This above structuration shows a recursive PAC organization (see figure 8). The PAC model is also applicable to the graphical constraints visualization layer, which displays a presentation of some constraints. For example, if we

use a simple grid alignment as constraint, the abstraction is composed of four values: the grid spacing and the offset from the origin. From this abstraction, the grid can be displayed. The abstraction is then used during the direct manipulation phase to align the input device events to grid values (see figure 8).

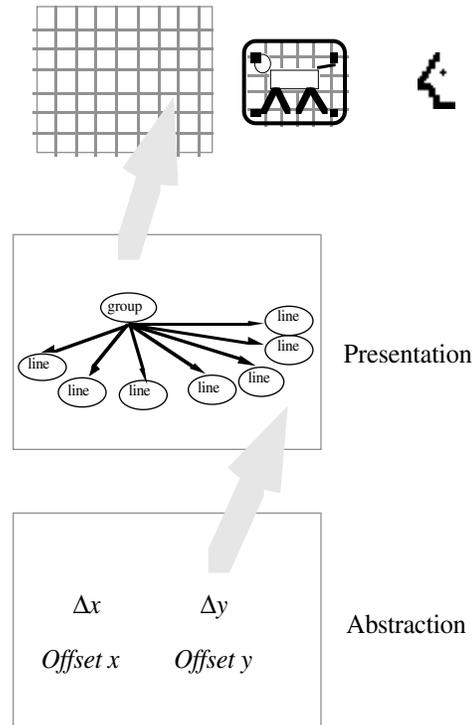


Figure 8: PAC model for a grid alignment constraint.

Finally, with the PAC model, the lexical operations representation layer displays some presentations of devices states (the mouse cursor for instance which presents the coordinates of the mouse device). Each layer contains the presentations and handle the interaction of a specific category of abstractions.

Example of Use in an Application

With the multi-layer model, we have built several graphical editors, handling both a mouse and a pressure sensitive Wacom stylus with multiple levels of pressure. We have used the layers of figure 10.

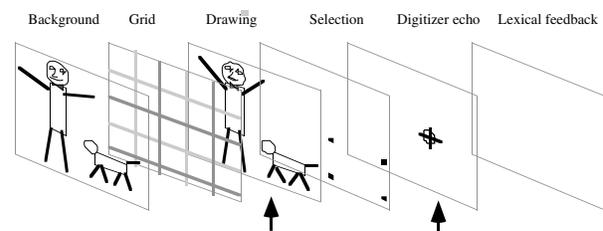


Figure 10: layers of a 2D animation graphic editor.

Compared with a conventional 2D graphic editor, we have added the *Digitizer echo layer* which displays a cross centered at the hot spot of the stylus, with a width equal to the selected drawing line width. When the stylus touches the digitizer, a circle appears with a radius proportional to the pressure applied.

The graphic editor can show multiple view of the same drawing (in the main data layer). Each view also contains the Digitizer echo layer.

[Dire que le fond contient des autres celluloses]

This has been implemented on a computer aided animation program where “in-betweeners” animators used two key drawings as background in order to sketch an intermediate drawing. Finally, some echo of direct manipulation can be visible in each visible surface, providing better feedback. We have experimented with this by managing a soft cursor which appears on all the visible surfaces where a virtual surface is displayed. Users usually use two visible surfaces, one showing the whole drawing and the other showing the part being currently edited. The presence of the soft cursor appearing on both surfaces has been found very useful.

Related Work

The multi-layer model is not completely new; previous systems have used multiple layers for graphic output or event handling. However, no system generalizes this notion as our model does.

Some window systems offer support for stacked windows, either for graphic output or for event handling. NeWS [12] has a special type of window called “overlay canvas” which is used to display transient data. Overlay canvases optimize the redisplay by relaxing the drawing model of PostScript. It usually uses either an overlay plane of the screen on the root window when the hardware offers such a device, or draws all the shapes using Xor raster operations. Overlay canvases offer just one level of layering, which is a strong limitation.

The X window system offers transparent input-only windows for event handling. As their name implies, no graphic output can be done on such windows.

The HyperCard [5] system has a two layer model for handling graphics and events, which is very close to our model. However, HyperCard is not extensible and can not be considered as a complete toolkit.

It is also interesting to notice that some real devices do provide multi-layered graphics, like heads-up displays on military planes.

Future work

We are working on a graphic editor to design interactive graphical applications, using this model to express the event handling graphically.

Conclusion

The multi-layered multi-view model simplifies both the management of graphical output and the description of event handling for interactive graphics applications. It gives a clear status to objects appearing in a graphical application, like the selection, the cursor, the grid or the selection rectangle. It also permits a clear description of event handling for a variety of input devices. We believe it will be useful to unify the handling of a variety of problems that are currently solved with ad hoc approaches.

Acknowledgments

Thanks to Michel Beaudouin-Lafon, Stéphane Chatty and Chris Weikart for their comments and suggestions.

Bibliography

- [1] Adobe, PostScript Language Reference Manual, Addison Wesley, Reading Mass., 1985.
- [2] Apple Computer, Inside Macintosh, Volume I, Addison Wesley, Reading Mass., 1986.
- [3] M. Beaudouin-Lafon, Y. Berteaud, S. Chatty, Créer des applications à manipulation directe avec Xtv, Actes IHM'90.
- [4] J. Coutaz, Interface Homme-Ordinateur : Conception et Réalisation, Dunod, 1990.
- [5] G. Harvey, Understanding HyperCard for Version 1.1, Sybex Books Publishers, 1988.
- [6] International Organization for Standardization, Information processing systems – Computer Graphics – Graphical Kernel System (GKS) functional description, ISO IS 7942, July, 1985.
- [7] International Organization for Standardization, Information processing systems – Computer Graphics – Programmer's Hierarchical Interface to Graphics (PHIGS) functional description, ISO DP 9592, October 1986.
- [8] M. A. Linton, J. M. Vlissides and P. R. Calder, Composing User Interfaces with InterViews. IEEE Computer, February 1989, pp. 8–22.
- [9] X Toolkit Library – C Language Interface, X protocol Version 11, MIT, 1987.
- [10] B. A. Myers *et al*, Garnet : Comprehensive Support for Graphical, Highly Interactive User Interface, IEEE Computer, November 1990, pp. 71–85.
- [11] R. W. Scheifler, J. Gettys, The X Window System, ACM Transactions on Graphics 5(2), April 1986, pp. 79–109.
- [12] SUN Microsystems Inc. : NeWS Manual ; SUN Microsystems Inc., 2250 Garcia Avenue, Mountain View, CA 94043.
- [13] J. M. Vlissides, M. A. Linton, Unidraw: A Framework for Building Domain-Specific Graphical Editors, ACM-Transactions on Information Systems, 8, 3, July 1990, pp. 237–269.
- [14] A. Weinand, E. Gamma, R. Marty, ET++ — An Object-Oriented Application Framework in C++, in ACM-OOPSLA'88 proceedings, San-Diego, SIGPLAN Notices, 23, 11, November, 1988, pp. 46–57.