The Design and Implementation of 2D Vector Graphics Interactive Tools Based on "Smart Handle"

Li Xiang[a], Wang Shuang[b], Huayixin[a]
[a] Institute of Surveying and Mapping, Information Engineering University, Zhengzhou , China ,
[b] LRES, Institute of Geographic Science and Natural Resources, CAS, Beijing, China

ABSTRACT

This article summarizes several kinds of interactive tasks and tools in 2D vector graphic system, analyses an ordinary design idea for interactive tool. We find that it is difficult to find a point of balance between user experience and code's maintainability and extensibility. So we present a design idea for smart handle-based interactive tool. Handles are usually expressed as rectangles or circles when graphic-cells are selected. Smart handle is a concept proposed in this paper. Compared with the handle, smart handle knows how to operate its own graphic-cell. So a complicated interactive task is assigned to every smart handle. It proves that this is a better solution to solve a contradiction between user experience and code's maintainability and extensibility.

Keywords: Handle，Interactive tasks，Interactive tools，2D vector graphics system

1. INTERACTIVE TASKS AND TOOLS IN 2D VECTOR GRAPHICS SYSTEM

In general 2D vector graphics system, drawing graphics is mainly through interactive way. In computer graphics, there are many researches of interactive technology. There are some basic interactive tasks and composite interactive tasks[1], but it is mainly focused on the human-machine interface. For a 2D vector graphics system, we believe that there are four general interactive tasks:

a). Graphic-cell construction task, it means that we construct the basic graphic-cell in the view of 2D vector graphics system. The basic graphic-cell includes line, rectangle and so on. Take a line construction for an example. The common way is that we press the left mouse button to designate the initial point of the line, drag the mouse, and up the left mouse button to designate the end point of line. So a construction process is finished.

b). Graphic-cell selection task, it means that choose one or more graphic-cells from the graphic-cell set. There are two kinds of selection ways, one is point selection and the other is rectangle selection. Point selection means that selected by mouse clicking. If the mouse clicks position in the graphic-cell or graphic-cell within the envelope, graphic-cell in the selected state, the contrary is not selected in the state and vice versa. Rectangle selection means that selected by the rectangle which is drawn by a series of mouse actions. Then we determine whether the drawn rectangle and the graphic-cell are intersected or not. If they are intersected, the graphic-cell is in the selected state, the contrary is not selected in the state and vice versa. There are round selection, polygon selection, etc besides point selection and rectangle selection.

c). Graphic-cell dynamic operation task: it contains that scaling, rotating, moving, anchor points moving and control points moving tasks. As shown below:

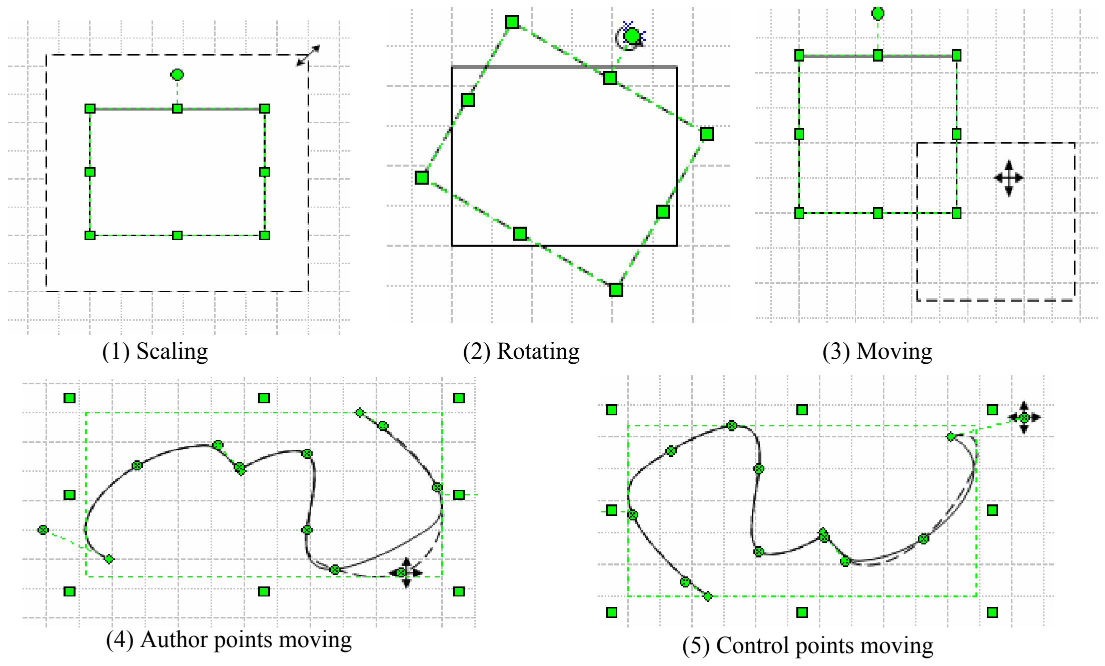

Fig 1 Graphic-cell dynamic operation tasks category

d). Graphic-cell property setting task: it means that setting graphic-cell attributes such as line size, line style and angle, primarily through dialog box.

The basic tasks in the 2D vector graphic system are grouped into the following table:

Table 1 interactive tasks in 2D vector graphics system

Interactive tasks	Interactive sub-tasks	Interactive methods
Graphic-cell construction	Line construction	Mouse and control key
	Rectangle construction	
	Circle construction	
	Ellipse construction	
	....	
Graphic-cell selection	Point selection	Mouse and control key
	Rectangle selection	
	Circle selection	
	......	
Graphic-cell dynamic operation	Scaling	Mouse and control key
	Rotating	
	Moving	
	Author points moving	
	Cntrol points moving	
	......	
Graphic-cell property setting	Setting line style	Dialog box or tool control bar
	Setting line width	
	......	

Ordinary 2D vector graphics system designers will design many interactive tools corresponding to the different interaction tasks or sub-tasks. Interactive tools are offered to users in a form of the menu or tool control bar. As shown below:

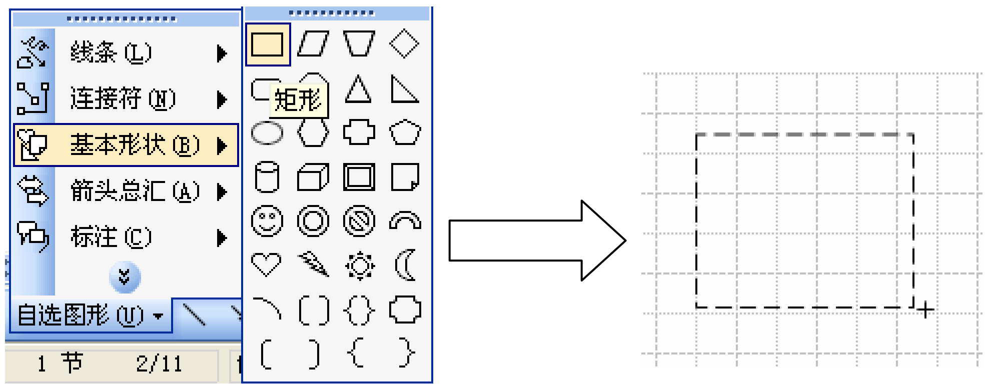

Fig2 2D vector graphics system interactive tools

When users select menu item [basic shape / - a rectangle], it means that the rectangular construction tool is selected, when the mouse moves to the view through the following mouse actions: press the left mouse button – drag mouse - raise your left mouse button. you can construct a rectangular graphic-cell.

2. AN ORDINARY DESIGN IDEA ABOUT 2D VECTOR GRAPHICS INTERACTIVE TOOLS

In object-oriented programming methods, designers will design many interactive tools corresponding to the graphic-cell construction task, graphic-cell selection task and graphic-cell dynamic operation task. All the tools will inherit an abstract tool class. It defines a number of mouse and keyboard response methods. When users select a tool, the system framework will send mouse and keyboard messages to the active tool. So the tool completes the appropriate task. (Because graphic-cell attribute task often uses the dialog to provide user to set attribute, does not relate to the mouse and keyboard messages. So this paper doesn't discuss how to design and realize graphic-cell attribute tool.)

The relationship between tools and tasks are not one-to-one, the designers will generally consider two points to design interactive tools:

a). Good user experience;

b). Code's maintainability and scalability.

When sub tools and tasks are one-to-one, code's maintainability and scalability is very high. Because every tool class only responds to one sub task in mouse and keyboard events. The amount of tool class code is very small. So code's maintainability is very high. If user adds a new tool, user only needs a tool inherited an abstract tool class. So code scalability is very high. But user experience is not very good. For example, if user completes the following task: constructing rectangle, rotating 45 degrees and moving rightward 20 pixels, user needs to select 4 tools: rectangle construction tool-point selection tool (rectangle selection tool)-rotating tool-moving tool.

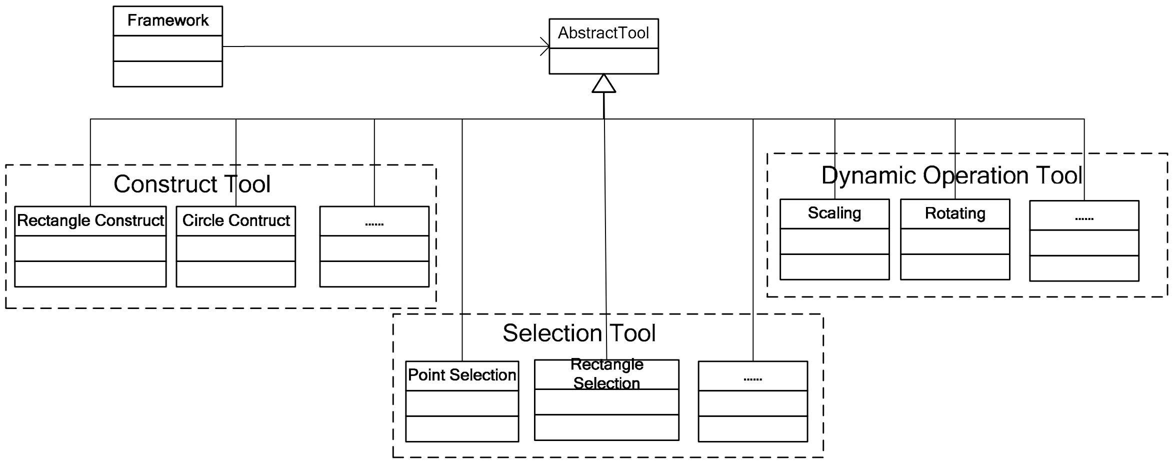

Fig 3 The class diagram for general 2D vector graphic interactive tools

Good user experience is that user can complete all the tasks only by selecting one tool. The interactive tools can be divided into two kinds:

a). The construction tool: It completes all the graphic-cells construction task via setting some parameters;

b). The selection and dynamic operation tool: It completes two sub tasks in selection task as point selection and rectangle selection; and five sub tasks in dynamic operation task as scaling, rotating, moving, moving anchor point and moving control point.

The designers can solve automatic conversion of the construction tool to the selection and dynamic operation tool after construction by the locking concept. When the construction tool is locked, user constructs graphic-cells until the construction tool is unselected. When the construction tool is unlocked, it switches the active tool to the selection and dynamic operation tool after constructing the graphic-cell. From the maintainability and scalability of code, the complexity of the structure tool has not changed significantly, and its maintainability and scalability are still good. But the amount of the selection and dynamic tool code has increased. So it resulted in the increase of complexity, maintainability and scalability will inevitably decline.

Both design methods are very difficult to find a point of balance between the user experience and code's maintainability and extensibility. This paper, based on JhotDraw open source projects and other vector graphics system, presents a method which to solve the contradiction between the code's maintainability and extensibility and good user experience.

3. THE DESIGN AND IMPLEMENTATION OF AN INTERACTIVE TOOL BASED ON "SMART HANDLE"

The concept of handle helps to provide the graphic-cell scaling, rotating, moving, author point moving and control point moving. When graphic-cell in the selected state, a small rectangle or a solid circle is used to represent a handle. Generally, the distributive positions of the handle have three situations:

a). The corner and the center border of graphic-cell envelop;

b). The vertex of line or area graphic-cell;

c). The control point of curve.

User picks up the corresponding handle through the mouse to implement scaling, rotating, moving, author point moving and control point moving.

Smart handle is a concept proposed in this paper. Actually, handle is abstracted into a class of object; each handle is linked to a graphic-cell. A graphic cell has one or more handles. Compared with the handle, smart handle knows how to operate its own graphic-cell.

3.1 The design thinking

This is a class diagram of the interactive tools. As shown below:

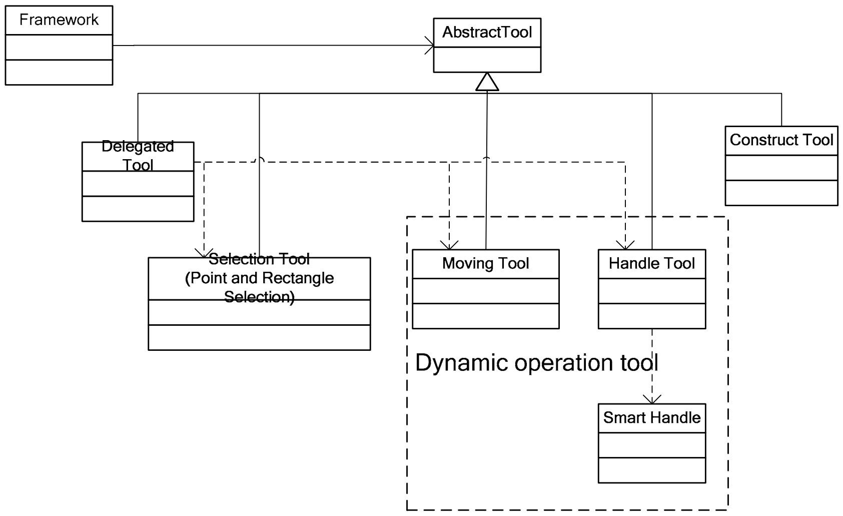

Fig4 A class diagram of interactive tools based on smart handle

Table 2 Interactive tools and Interactive tasks

Interactive tools		Interactive tasks
The delegated tool		This tool delegates mouse and keyboard events to a specific tool such as the selection tool, moving tool and based on handle tool.
The selection tool		This tool includes point selection tool and rectangle selection tool.
The construction tool		This tool can construct line, rectangle and circle graphic-cell by passing different parameters.
The dynamic operation tool	The moving tool	This tool takes charge of moving graphic-cell.
	The handle tool	This tool takes charge of scaling, rotating, author point moving and control point moving.

The design has two different points:

a). Dynamic operation tool is divided into two tools, the moving tool and the handle tool. Because moving a graphic-cell does not need picking up a handle, we design the moving tool. Because we scale, rotate a graphic-cell and move anchor points and control points by picking up handles, we design the handle tool. The handle tool is implemented by different kinds of smart handles.

b). We design the delegated tool above the selection tool, moving tool and handle tool. This tool delegates mouse and keyboard events to a specific tool. So user selects one tool to complete all the tasks. This tool doesn't implement any edit operation. So the amount of the delegated tool code is small. The code's maintainability and scalability are very high.

3.2 The implement of the delegated tool

According to general user's habits, the delegated tool judges the type of task in two mouse events, the left mouse button press event and the mouse moved event, and delegates tasks to different tools.

a). The left mouse button press event: the delegated tool judges whether it captures a handle according to the position which the left mouse button pressed. If the mouse captures a handle, the delegated tool will delegate a task to the handle tool. if not, the delegated tool judge whether the mouse captures a graphic-cell. If it captures a graphic-cell, the delegated tool will delegate task to the moving tool. If not, the delegate tool will delegate a task to the selection tool;

b). The mouse moved event: the rule in this event is the same as in the left mouse button press event;

c). The left mouse button raised event: the delegated tool determines whether the task has been assigned to other interactive tools. If the task has been assigned to other interactive tools, the delegate tool delegates mouse event to a specific tool. If not, it doesn't make any operation.

d). Other mouse and keyboard events: the delegated tool delegates mouse and keyboard events to a specific tool.
The flow diagram of the delegated tool:

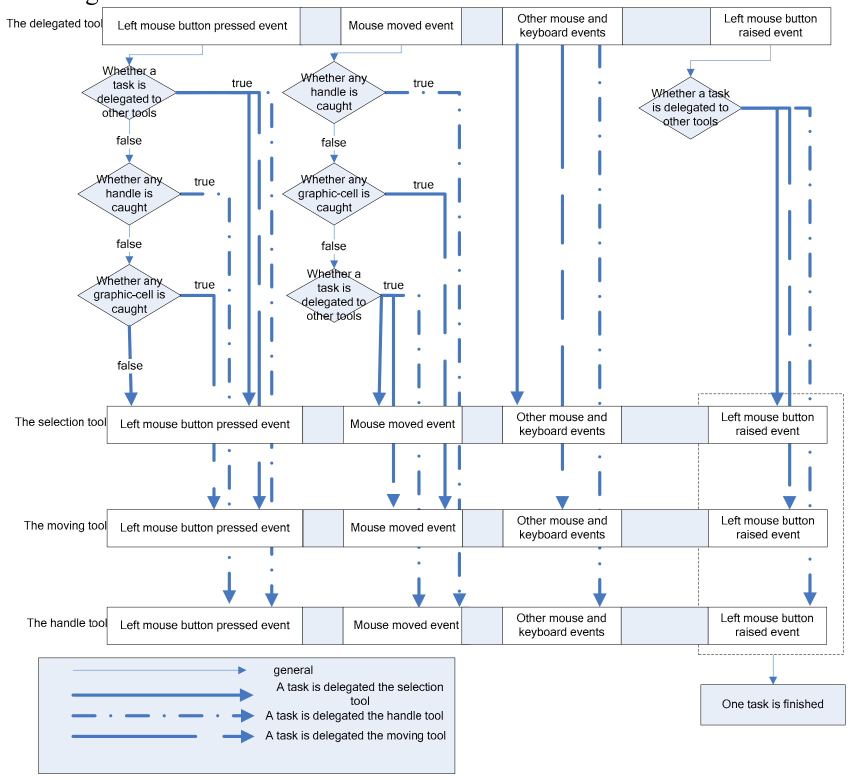
The delegated tool
Left mouse button pressed event
Mouse moved event
Other mouse and keyboard events
Left mouse button raised event
Whether a task is delegated to other tools
true
false
Whether any handle is caught
Whether any graphic-cell is caught
The selection tool
The moving tool
The handle tool
One task is finished
general
A task is delegated the selection tool
A task is delegated the handle tool
A task is delegated the moving tool

Fig 5 The flow diagram of the delegated tool

It is worth to notice a point in the diagram. This is a judgment which determines whether the current task is delegated to other tools in the left mouse button down event and mouse moved event. But there is a little difference between the left mouse button down event and mouse moved event. In the left mouse button, this judgment is the first. But it is the last in the mouse moved event. The difference is determined by the character of the mouse moved event. For example, assuming that the current tool is the moving tool, if a handle is caught in the mouse moved event, the current tool should be changed to the handle tool. If the judgment is the first which determines whether the current task is delegated to other tools, the current tool is always the moving tool.

3.3 The implement of the handle tool based on smart handle

A smart handle class encapsulates graphics operations (rotating, scaling, etc) and the cursor resources. Every smart handle is linked to a graphic-cell. A graphic-cell has one or more smart handles. All the handles must implement I Handle interface. It has three important methods, InvokeStart， InvokeEnd and InvokeStep (Table 3).When a handle is captured, the delegated tool delegates followed mouse and keyboard events to the handle tool. The handle tool delegates tool to the captured smart handle. The InvokeStart is invoked in the left mouse button pressed event.InvokeStep is invoked in the mouse dragged event.InvokeEnd is invoked in the left mouse button raised event. The cursor is set in the mouse move event.

Table 3 methods in smart handle class

Declaration	Description
void InvokeStart(int x, int y)	It is invoked in the left mouse button pressed event
void InvokeStep (int x, int y, int anchorX, int anchorY)	It is invoked in the left mouse button pressed event
void InvokeEnd(int x, int y, int anchorX, int anchorY)	It is invoked in the left mouse button pressed event
IFigure* Owner() const	The graphic-cell linked to smart handle
void SetCursor()	Set the handle's cursor to the current cursor

According to the functions, it can be divided into several types

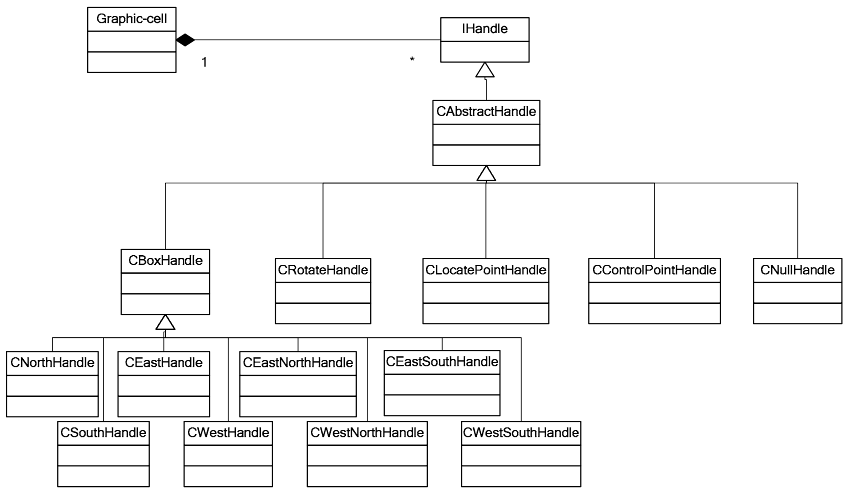

Fig 6 handle class diagram

Table 4 handle classes

Name	Description
IHandle	Smart handle class's interface. It determines several methods which are must be implemented
CAbstractHandle	It implements part of methods such as function called Owner.
CBoxHandle	It is the base class of handles which resize their own graphic-cell.
CNorthHandle， CSouthHandle, CNorthWestHandle, etc	See also the figure 7
CRotateHandle	It rotates its graphic-cell.
CLocatePointHandle	It moves its graphic-cell's anchor points.
CControlPointHandle	It controls its grapic-cell's control points.
CNullHandle	It has no operation except that it represents its graphic-cell is selected.

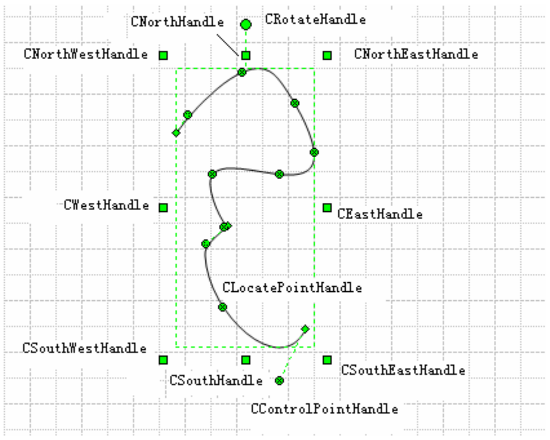

Fig 7 A different kind of smart handles

You can create other handles to implement other advanced edit functions. But all the handles must implement IHandle interface.

4. The potential problem and solving methods

4.1 The potential problem

The amount of every tool's code is relatively small because of the division of the dynamic operation tool and introduction of smart handles. At the same time, user only needs to select one tool, the delegated tool; to implement all tasks. It is easy to use. But this design also brings a potential problem, a large amount of memory spending. Because one graphic-cell needs four smart handles at least. It means when you create a graphic-cell, you must create four handles in memory. Although user does not feel slowly when he edits a graphic-cell, but based on two reasons, we must attach importance to this issue.

a). The actual situation is that only a graphic-cell is in the selected state, its handles are displayed. And at most time we do not need handles;

b). User does not feel slowly when he edits a graphic-cell in a PC machine. But in the network environment, the bandwidth is limited; we must take this issue seriously.

4.2 The solution of solving problem

The thinking is that when a handle is needed, it is created, when it is not needed, it is destroyed. The problem is when a handle is live and when a handle is dead.

There are two methods to define when a handle is live and dead. To live or die, this is really a problem.

a). When a graphic-cell is created, associated handles are to live. When a graphic-cell is destroyed, associated handles are dead.

b). When a graphic-cell is selected, associated handles are live. When a graphic-cell is unselected, associated handles are dead.

The latter method can greatly reduce memory spending. Take example for drawing 1000 graphic-cells in a graphic system.

According to the first definition, 4000 handles are resident in memory at least. But according to the second definition, the amount of handles lies on the current number of the selected graphic-cells. In fact we have little time to select all the graphic-cells when graphic system is in edit state.

The follow figure represents the process of life and death.

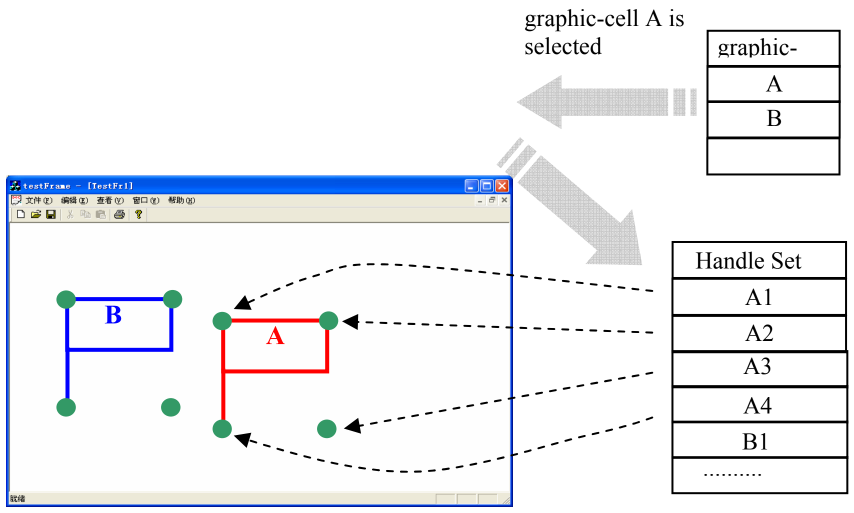

Fig 8 The handle's life

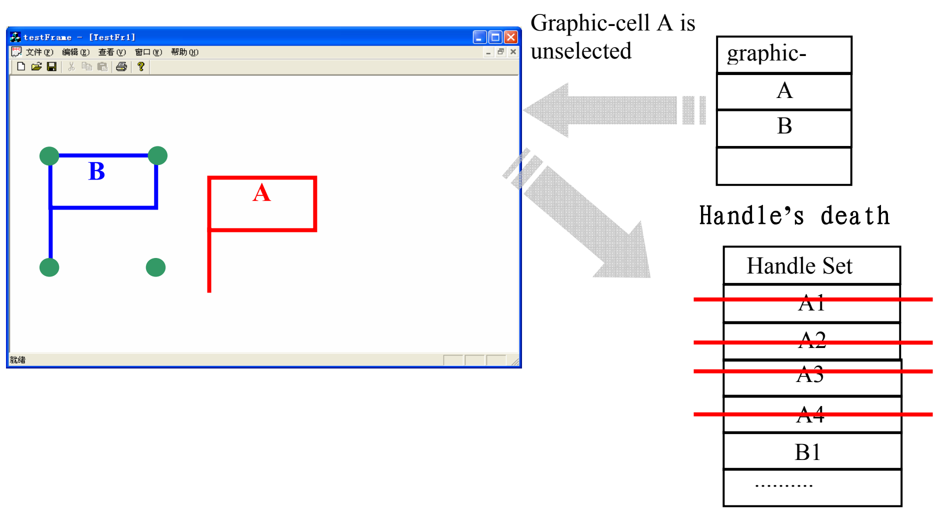

Fig 9 The handle's death

5. CONCLUSION

This design idea has the following main advantages:

The amount of every tool's code is relatively small because of the division of the dynamic operation tool and introduction of smart handles. At the same time, user only need to select one tool, the delegated tool, to implement all tasks. It is easy to use. This is a better solution to solve the confliction between code's maintainability and scalability and easy-to-use.

We find a potential problem, a large amount of memory spending, because of the introduction of the smart handle. Through analyzing the process which is from the handle's life and death, we solve this problem.
However, the design still needs to be improved: As graphic-cell and its handles are not a whole, when we draw, add and delete a graphic-cell, we must notify the framework to add or delete handles. This realization is not so good.

REFERENCES

1. James D. Foley, etc "Computer Graphics Principles and Practice (Second Edition in C)," China machine press. 254-273 (2004).
2. Erich Gamma, etc "Design Patterns Elements of Reusable Object-Oriented Software," China machine press. (2000)
3. http://www.jhotdraw.org