Bull. Nov. Comp. Center, Comp. Science, 14 (2001), 59–64
© 2001 NCC Publisher

The construction of graphic interfaces*

M.B. Ostapkevich, D. Shashkov

The Object Visualization Engine library developed by the authors is described in the paper and its architecture is substantiated. This library provides an easy way for a user to implement the modules for custom visualization of cellular objects. These modules are called Object Visualization Drivers (OVD). The algorithm of the OVD construction is given. Sample drivers on the rectangular and hexagonal grids by means of colors, directions, and numeric values are presented.

1. Introduction

Fine grain computation models have versatile applications from the design of massively parallel computing devices to research in physics, chemistry, and biology. This simulation process for such type of algorithms typically has a huge number of simultaneously performed data transformations. Thus it is not an easy task for a researcher to comprehend the phenomena that take place in the simulating model. This understanding must be facilitated by the custom visualization modes which are most appropriate for a particular application. Different fine-grain applications typically do not have the same form of intermediate data and results [1].

The goal of the work is to provide a WinALT [2] user with a means for creation and insertion of custom cellular arrays visualization modes. This would allow to enrich the universality of the system for new potential applications. Such a means was constructed in a form of a library called Object Visualization Engine (OVE), which is a part of the WinALT graphic interface subsystem. The OVE includes the set of external libraries, each of which implements one mode of visualization. These libraries are called Object Visualization Drivers (OVD). A user may include his own drivers in this set.

The design of the system is based on the analysis of existing approaches to the construction of visualization tools such as graphic editors. The approaches to the creation of systems that have open architectures based upon publicly available interfaces of functions [4] were also taken into consideration.

*Supported by the Russian Foundation for Basic Research under Grant 99-07-90422.

2. Substantiation of the OVE architecture

2.1. The requirements to the OVE. Some features of data representation which are general for the whole class of fine-grain algorithms can be obtained from the investigation of Parallel Substitution Algorithm [3] that is the theoretical basement for this class of algorithms:

1) all the data objects (further, cellular arrays) are represented by sets of data elements (further, cells) of the same type;
2) cells are indexed by one, two or three dimensions depending on the dimension of a cellular array;
3) the cell values are altered in the discrete time (typically a part of them at a single step).

Thus, a cellular array has to be painted as a rectangle. The drawing of any two cells in one array has to be done in the similar way. As the size of an array may be huge for the real world tasks, a capability to visualize its part must be implemented. The visible rectangle ought to slide easily.

The obligatory operations for the object management are: creation, deletion, and modification of attributes (name, scale, and visualization modes). The set of such operations was formed in graphic editors. Some of them were implemented in the WinALT graphic subsystem, thus not all of them has to be in the OVE. At the same time, the interoperability between the WinALT graphic subsystem and the OVE should be established.

The cell values are not interpreted in the same way in different applications. The set of these applications is not predefined. Thus, a fixed set of the visualization modes may not reside in the OVE itself. Instead, the implementations of visualization modes ought to be in a form of external modules which form a flexible set. A user may add new modules to this set and exclude the existing ones.

2.2. The OVE architecture overview. The requirements formed in the passage above lead to the following architecture. It must be open and ought to be based on the same principles as that of the whole WinALT [5]. The subsystem is divided into fixed and flexible parts. The fixed part contains the operations, which are the same for all the visualization modes. And besides, this part manages the external libraries (or Object Visualization Drivers, the OVD), each of which implements a certain visualization mode. The set of the OVDs forms the flexible part of the subsystem.

3. The OVE architecture

The fixed part is represented by the following modules: the OVD manager (OVDMgr), visual object manager (VisObjMgr), standard interface element

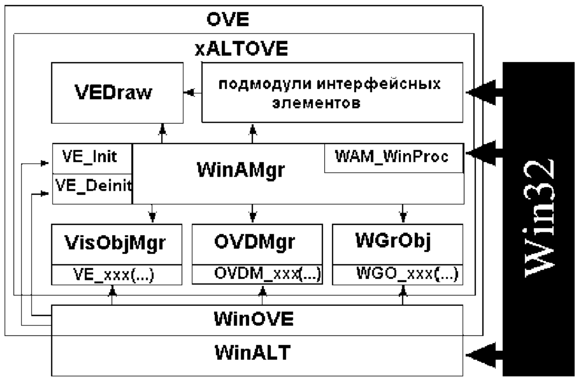
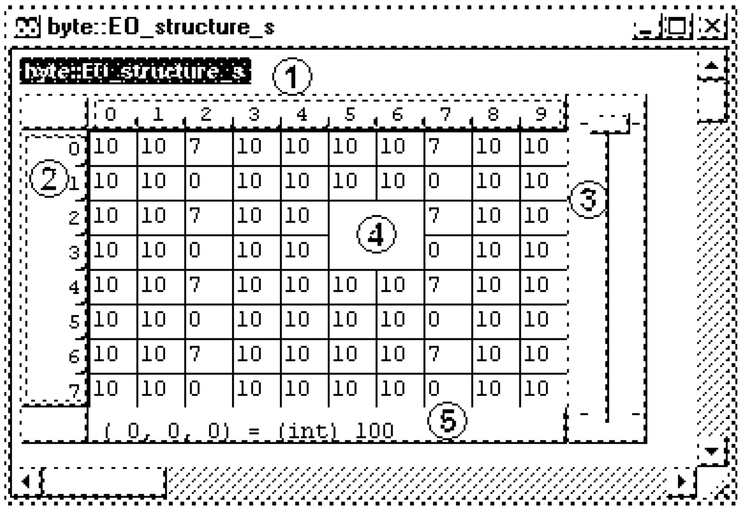


Figure 1. OVE architecture



	0	1	2	3	4	5	6	7	8	9
0	10	10	7	10	10	10	10	7	10	10
1	10	10	0	10	10	10	10	0	10	10
2	10	10	7	10	10			7	10	10
3	10	10	0	10	10			0	10	10
4	10	10	7	10	10	10	10	7	10	10
5	10	10	0	10	10	10	10	0	10	10
6	10	10	7	10	10	10	10	7	10	10
7	10	10	0	10	10	10	10	0	10	10

```
( 0, 0, 0) = (int) 100
```

Figure 2. OVE standard interface elements

painter (VEDraw) and the interface modules between WinALT and OVE (WinOVE), Win32 and OVE (WinAMgr) (Figure 1).

The OVDMgr implements the OVD loading. It calculates the number of objects being visualized under a certain mode. When a driver is no longer in use, the manager unloads it. The module includes as well the operations for inclusion and deletion of a visual mode and obtaining the list of the registered visual modes. It is the OVDMgr who brings extensibility and scalability to the OVE.

Some elements of the visual interface are common for all the visual modes. Namely, these are the rulers, edit, and status lines (Figure 2). Their drawing is implemented in the VEDraw module.

The main data object, that is processed by the OVE, is visual object which includes a cellular object and the parameters of its visualization, such as cell scale, the visibility of rulers and edit line, current mode of visualization and so on. The operations on the visual objects, which are common for all the visual modes, are implemented in the VisObjMgr module. And the rest of them are located in the OVDs.

The integration of the OVE, that is based upon Win32 SDK, into WinALT, which uses the MFC class library, is performed by the WINOVE module.

4. Object visualization drivers and their interface

As it was mentioned above, the painting is not done by the OVE itself. It is up to a driver to draw. A driver is a dynamically linked library. The OVE calls it via its interface that is the same for all OVDs and hides the differences between the particular drivers.

The interface consists of two functions: `OVD_Paint` and `OVD_Request`. `OVD_Paint` is the main OVD function, as it performs the drawing of an object. `OVD_Request` contains the code for all the other OVD operations. It takes an integer value that denotes subfunction code and two values. Their meaning depends on a particular subfunction. Not all the subfunctions are obligatory for the implementation of a commonplace driver. The OVE can call, for example, a subfunction in the OVD to determine if a driver requires the visibility or rulers or edit line. Or the OVE may demand the correction of cell sizes.

An OVD may not paint directly in a window. It draws in a memory region instead. The OVE caches the created images so as to speed up the drawing or scrolling and improve the overall performance.

The set of the Win32 API functions used in a driver is quite small, thus the code is quite portable because these functions can be easily reimplemented. In Linux WINE package for the Win32 emulation can be used.

5. Steps of the OVD creation

The OVE distribution contains the source of the simplest driver. A user may commence the code from the scratch or basing upon this skeleton, which has to be modified in the following locations.

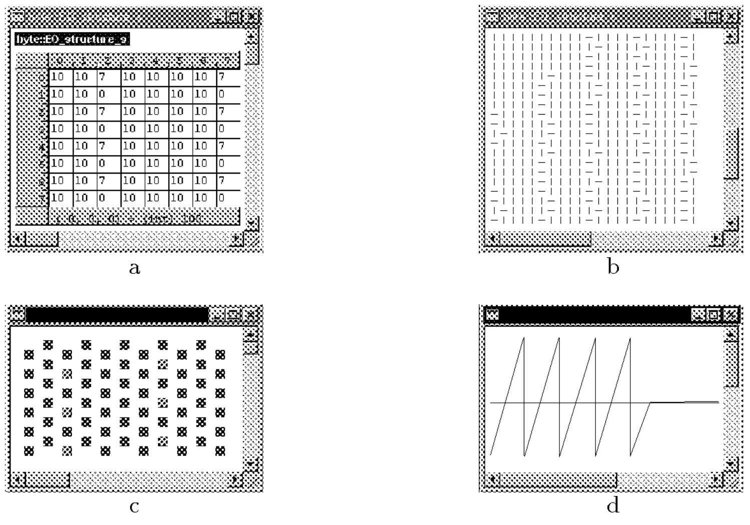
The loop body in `OVD_Paint` must contain the code for cell drawing for one or several visualization submodes. Only the main submode is obligatory.

The OVE passes the submode number in `SETSUBMODE` subfunction. The number is kept in nSubMode variable in the skeleton. The main submode redraw is requested by default. If a user have press Ctrl and/or Shift, other modes are activated. After the key was released the main mode is restored in one second.

All the other code modifications are done in the `OVD_Request` function. First, the initialization code has to be inserted into the `INITIALIZE` subfunction. It is activated after the OVD has been loaded. Then, the code for cell size correction is implemented if required. in `CORRECT_SIZES` subfunction. If a driver has no fixed, maximal, or minimal cell sizes, this step can be omitted. The last place to modify is `DEMAND_PROPERTIES` subfunction. The OVE calls it to determine the capabilities of an OVD (e.g., whether it is capable to edit) and what the OVD demands from the OVE (e.g., if the rulers have to be visible).

6. The OVD samples overview

A number of samples were designed for the most widely used visual modes. The spreadsheet mode outputs the values as numbers in rectangular cells (Figure 3a). The stream mode interpreters the cell values as directions and paints them with arrows (Figure 3b). Many models in physics, such as diffusion, use the hexagon grid mode (Figure 3c). The visual mode for the 1D objects that contains signal samples (e.g., sound) is depicted in Figure 3d.



	0	1	2	3	4	5	6	7
0	10	10	7	10	10	10	10	7
1	10	10	0	10	10	10	10	0
2	10	10	7	10	10	10	10	7
3	10	10	0	10	10	10	10	0
4	10	10	7	10	10	10	10	7
5	10	10	0	10	10	10	10	0
6	10	10	7	10	10	10	10	7
7	10	10	0	10	10	10	10	0

(0, 0, 0) = (int) 100

Figure 3. Visual modes

7. Conclusion

The further development of the OVE includes first of all the construction of new OVDs. The standard WinALT visual modes in the graphic subsystem have to be converted into the OVD form. The unification of intermodular interaction mechanisms will be done by converting the OVE interfaces into event-driven style with the help of DCMS [6].

References

[1] Toffoli T., Margolus N. Cellular Automata Machines. – Moscow: Mir, 1991.

[2] Beletkov D., Ostapkevich M., Piskunov S., Zhileev I. WinALT, a software tool for fine-grain algorithms and structures synthesis and simulation // LNCS. – Springer, 1999. – № 1662. – P. 491–496.

[3] Achasova S.M., Bandman O.L., Markova V.P., Piskunov S.V. Parallel Substitution Algorithm. Theory and Application. – Singapore: World Scientific, 1994.

[4] Wirth N. A plea for lean software // IEEE Comp. – 1995. – Vol. 28, № 2. – P. 64–68.

[5] Ostapkevich M. The open architecture of WinALT // NCC Bulletin, Series Comp. Science. – Novosibirsk: NCC Publisher, 1998. – Issue 9. – P. 93–106.

[6] Ostapkevich M. Event-driven tools for open system design // NCC Bulletin, Special Series. – Novosibirsk: NCC Publisher, 1999. – Issue 1. – P. 15–22.