# Dataflow Visual Programming Language Debugger Supported by Fisheye View

Yangyi Sui, Lili Pang, Jun Lin
*College of Instrument Science and Electronic Engineering, Jilin University, Changchun, China, 130061*
*suiyangyi@163.com*

Xiaotuo Zhang
*School of Communication Engineering, Jilin University, Changchun, China, 130022*

## Abstract

*Programs developed by Dataflow Visual Programming Languages (DFVPLs) often contain lots of visual information. If more useful information could be displayed in the limited range of screen, the efficiency of debugging would be improved. We took full advantage of the characteristic of fisheye view that can display both "local detail" and "global context" simultaneously, improved the previous implementation models and algorithms, and solved the problems of using fisheye view on DFVPL such as nest hierarchy, the difference in nodes' size is big and wires are complex. We combined the debugger with fisheye view and proved its practicability and feasibility by some contrastive experiments.*

**Keywords**: Dataflow visual programming language, debugger, fisheye view, information visualization, program visualization.

## 1. Introduction

Dataflow visual programming languages have many advantages such as simple structure, inherent parallel potential, easily understandable graphic representation [1]. Therefore, they have been widely used in virtual instrumentation, digital signal processing, parallel computing etc. In practical applications dozens or even hundreds of nodes often appear in the DFVPL programs. Thus it is difficult to display complete information in the limited visual range on computer screen. Particularly during the debugging process, the serious non-intuitive obstacle of human computer interaction makes programmers synthesize the relationship between nodes in programs and debugging information in their own minds, which greatly reduces the efficiency of debugging [2]. The common solutions of this problem are rolled view, enlarged view and the multi-view. But the effect is not satisfied.

To simulate the feature of fisheye, construct the fisheye lens and display the image which is observed through lens into fisheye view. A fisheye view might magnify nearby objects while shrink distant objects. It has the feature of simultaneous showing both "local detail" and "global context". Based on the research by Furnas and other people [3-5], many scholars have proposed a variety of ways to apply fisheye idea in wide fields [6-12] such as dataflow visual programming environment, the large graphic visualization, PDA, three-dimensional reconstruction, virtual touring, and image-based rendering and so on.

In this paper we first outline LabScene, the dataflow visual programming language, which is developed by us. Then we aim at the features of DFVPL and focus on the dataflow semantics, improve the implementation of models and algorithms and combine fisheye view with debugger. It makes both the information of current running node and the overall dataflow semantics display simultaneously in the limited screen range. Therefore, the efficiency of debugging is advanced.

## 2. LabScene -- One of DFVPLs

LabScene is an experimental virtual instrument oriented DFVPL [13] like LabVIEW [14-16]. It is abstracted the basic elements of a circuit and mapped a program that defined by DFVPL.

A program represented by DFVPL is a composition of nodes, wires and containers. It is represented by a 3-tuple:

$$Program = (Nodes, Wires, Containers)$$

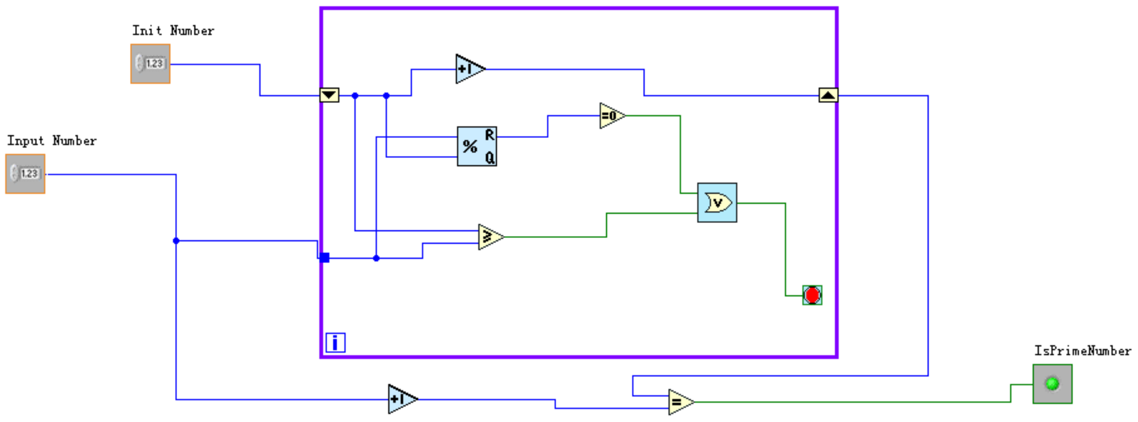

**Figure** 1. Instance of calculating prime number

*Nodes* are a set of *node*. *Node* is displayed by different icons. It contains a set of *input terminal* and a set of *output terminal*. *Wires* are a set of *Wire*. *Wire* is a set to represent data flow route. *Containers* are a set of *Container*. *Container* can contain *nodes* or *containers*. *Container* is called as *Parent* of *node* if it contains *node*. *Container* also can be considered a special node. The instance of calculating prime number is shown in Figure 1. The *While Container* is displayed by the rectangular frame and has ability of controlling cycle. *Nodes* are connected by *wires*. The node on the border of container called *border node*. If the outside nodes and the border nodes are connected, the container is connected with the outside nodes.

## 3. Implementation Model of Fisheye View

First we introduce the correlative terms. The normal view of a DFVPL program is shown in figure 1. The coordinates of the normal view is called *normal coordinates* and the coordinates of the fisheye view is called *fisheye coordinates*. Every node has different shape, size and some attachments like label. *Position* and *size* are the two properties of the node. The node concerned by users during debugging is called *focus*. *API* denotes the importance of every node; *VW* denotes the worth of visualization of every node and is used to distinguish the nodes' size. Nodes in the same layer have the same parent container.

### 3.1. Framework of Model

The point $P_{norm}$ under the *normal coordinates* transforming to the point $P_{feye}$ under the *fisheye coordinates* is achieved by the improved method of Sarkar [5], that is:

$$P_{feye} = \left\langle \begin{array}{c} T\left(G\left(\dfrac{D_{norm_x}}{D_{max_x}}\right)D_{max_x} + P_{foucs_x}\right), \\ T\left(G\left(\dfrac{D_{norm_y}}{D_{max_y}}\right)D_{max_y} + P_{foucs_y}\right) \end{array} \right\rangle \cdots (1)$$

and $G(x) = \dfrac{(d+1)x}{dx+1} \cdots\cdots\cdots\cdots\cdots (2)$

In the formula 2, $d$ is an adjustable constant. It is called distortion factor. In the formula 1, $D_{norm_x}$ is the horizontal distance between the point to be transformed and the *focus* in the same layer under the *normal coordinates*. $D_{max_x}$ is gotten according to the situation that which layer the transformed point in: if $P_{norm}$ is in the top layer, setting the largest rectangular area which could surround all nodes and wires as $A_{max}$, the $D_{max_x}$ is the horizontal distance between the *focus* and the $A_{max}$ boundary under the *normal coordinates*; if $P_{norm}$ is in a container, the $D_{\max_x}$ is the horizontal distance from the *focus* to the boundary of the container. $T$ is a transformation function which is responsible for transforming the calculated relative coordinates into the absolute coordinates to paint.

The node's worth of visualization in the same layer is designed as:

$$VW(n,f) = API(n) + \beta \cdots\cdots\cdots\cdots (3)$$

and $\beta$ is an adjustable constant to ensure $VW$ not to be 0. The node's size is designed as:

$$S_{feye}(n,f) = s(c \bullet VW(n,f))^{e} \cdots\cdots (4)$$

In formula 4, $c$、$e$、$s$ are all the adjustable constants, $s$ is called zoom factor. Notice that the nodes' size got from formula 4 couldn't guarantee there are no overlaps among the nodes. It is guaranteed by the algorithm behind.

## 3.2. Framework of Main Algorithms

The framework of main algorithms is shown in Figure 2, and we will mainly introduce the 5 steps in the middle.

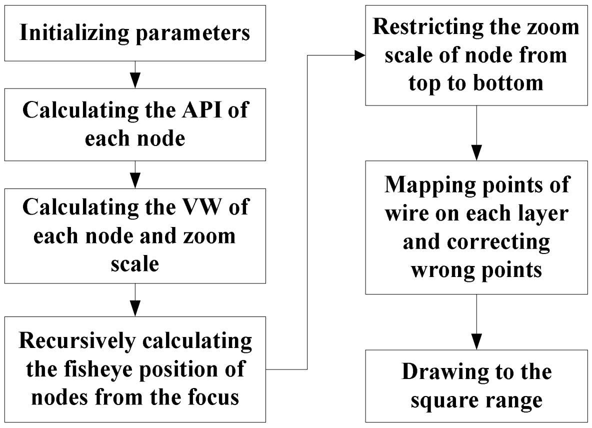
Initializing parameters
Calculating the API of each node
Calculating the VW of each node and zoom scale
Recursively calculating the fisheye position of nodes from the focus
Restricting the zoom scale of node from top to bottom
Mapping points of wire on each layer and correcting wrong points
Drawing to the square range

**Figure** 2. Framework of main algorithms

## 3.3. Implementation

**3.3.1. Calculating API.** *API* in ordinary fisheye view is a pre-set value which is assigned to each node at the beginning of algorithm. For outstanding the dataflow semantics, the improved method is dynamically calculated density of dataflow between the *focus* and the other nodes. The following algorithm is based on the truth that nodes with connection are closer than those without connection; nodes with direct connection are closer than those with indirect connection.

Algorithm of *API* is as follows:

(1) If the target node is the *focus*, *API* equals 1.

(2) Otherwise, traverse all the nodes connected with the input terminals of the node by starting from the input terminals of the *focus*. If the target node can not be found, traverse all the input terminals of the traversed nodes. If the target node is found, *API* equals $1/(n+1)$ and number $n$ is the traversed layers. If input terminals have no connected nodes, turn to step (3) to continue finding the target node.

(3) Do the same thing as step (2) along the output terminals of the *focus*. If the target node is found, *API* equals $1/(n+1)$ and number $n$ is the traversed layers. Otherwise, the two nodes have no dataflow relationship and *API* equals 0.

According to the algorithm, *API* is a real number between [0-1] and the reciprocal of traversed layers reflects the density relationship between the *focus* and the other nodes.

**3.3.2. Calculating Zoom Scale.** Nodes might be overlapped after zooming, which is not obvious when the original sizes of nodes are almost the same and the overlaps can be reduced by interactive parameters adjustment. But in DFVPL, containers which have control ability such as *while container*, contain plenty of nodes, so their sizes are much different from ordinary nodes. In order that guarantee containers do not overlap with the other nodes and the normal zoom scale of nested nodes, we calculate the zoom scale of every node first and then modified these overlapped nodes from the top layer to the bottom layer.

The key of calculating zoom scale is *VW*, because only *VW* of the same layer can be calculated with formula 3. So, algorithm is needed to reflect nested hierarchy, which is as follows:

(1) Start from the *focus*, seek up its immediate parent containers recursively along the nested hierarchy and get the collection called $List_p$.

(2) Search $List_p$, take the outmost parent container as the current calculating container and set it as *CurrentContainer*. Traverse $List_p$ from outside to inside and go to step (3).

(3) According to *API* of all nodes in the *CurrentContainer* and formula 3, calculate *VW* and modify it if those nodes only exist in containers. If the node is a child of one container of $List_p$, multiply *VW* of the upper layer by the got *VW*, gain the last *VW*. If the node is not a child of one container of $List_p$, take its parent *VW* as its own.

**3.3.3. Calculating Position.** In DFVPL, the nested hierarchy makes it special to express the node's position, which is defined by the relative position between the node and its parent container. If we still use the previous method to calculate the node's position in fisheye view, the nest hierarchy will be lost after it is transformed. Therefore, it is necessary to modify the function $G$ according to the *focus*. Details are as follows:

(1) Get the parameters needed in formula 1 according to the information of the layer where the focus lies in and figure out the position $P_p$ of the other nodes in the same layer. If the target node is a *container*, all nodes in it will inherit its position. Then set the relative position of the node in the container as $P_s$, the zoom scale of this container as $S_p$, so the relative position of the node in *fisheye coordinates* is $P_p+P_s \times S_p$. Finally, use the function $T$ to get the coordinates of the node for printing.

(2) Set the parent container of the *focus* as a new *focus*, if the parent container is not the top container, then go to step (1), otherwise stop.

**3.3.4. Limiting Zoom Scale.** The zoomed containers according to formula 4 often cover other nodes. Thus we propose an algorithm to limit zoom scale. This algorithm not only modifies the zoom scale of container but also affects the positions of nodes in it. Therefore it is necessary to adjust the parameters together. Details are as follows:

(1) Call the top container to be calculated as *CurrentContainer*.

(2) Traverse the nodes in *CurrentContainer*. If the zoom scale of *CurrentContainer* has been modified, adjust the positions of these nodes using the new zoom scale. If the traversal node is a container, figure out the minimum distance of coordinates between the right bottom corner of the container and all the other nodes in the same layer, marked $\triangle min_x$ and $\triangle min_y$. According to the minor and positive one of $\triangle min_x$ and $\triangle min_y$, and the original zoom scale of the container, get a new zoom scale of the container. When the *CurrentContainer*'s traversal is over, set a child container as the new *CurrentContainer*, repeat step (2) until the whole DFVPL's traversal is completed.

This algorithm is mainly used for adjusting containers in DFVPL. In fact, ordinary nodes also might overlap each other after it is zoomed. Since its influence is not obvious, it will be neglected in order to reduce calculation.

**3.3.5. Mapping Wires.** The algorithm of mapping wires is similar to calculate position. After it is zoomed, the points which are connected close to the terminals of the node would not be connected with the terminals correctly because the size of the node is not obtained from the right bottom corner of the node by using formula 1. So, we need to correct the position of wrong point, and keep the segments horizontal or vertical. Details are as follows:

(1) If the output terminal of the node is the first point of segment, *point*[0], figure out the rectangle area of the transformed node, according to the relative position of the output terminal with the node, get a new position $P_n$, assign $P_n$ to *point*[0].

(2) If the segment formed by *point*[0] and *point*[1] are horizontal before transforming, assign the ordinate of $P_n$ the same with the new ordinate of *point*[1]. Do the same thing along the point collection, till the segment formed by the two points is vertical. If *point*[0] and *point*[1] are vertical before transforming, do the similar modification, till the segment is horizontal.

By analogy, we can correct the position from the last point of wire (the input terminal of the node) as above.

## 4. Integrating Debugger with Fisheye View

### 4.1. Integral Structure

The debugger and fisheye view can be integrated in MVC pattern, as shown in Figure 3. Fisheye view belongs to view layer, debugger belongs to control layer and DFVPL programs belong to model layer. Taking nodes or containers as units, LabScene supports single-step debugging, animation debugging and breakpoint debugging. Debugger records the process of debugging as a series of events, which can be replayed after debugging. Interactive view can transmit the edit files to debugger in real time and is the chief debugging view. Fisheye view does well tracking the current debugging state. User can use both interactive view and fisheye view in the mean time to improve debugging efficiency.

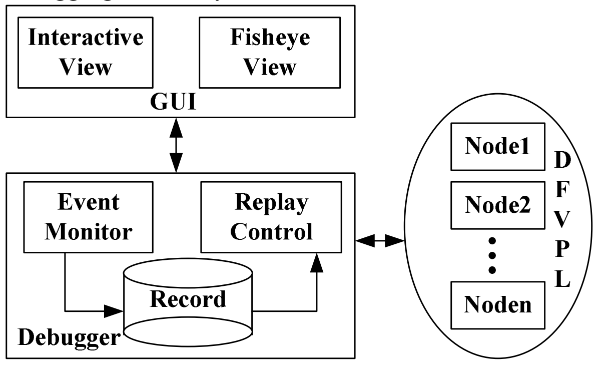

**Figure** 3. Structure of fisheye debugger

### 4.2. Practicality Discussion

LabScene starts up a thread in each debugging mode and draws rectangles four times during debugging in interactive view with a delay of 100ms each time, to get a zooming out animation effect. If the operation of fisheye view can be finished within less than 400ms, users will not feel the stagnancy that the fisheye view brings. For a desktop computer (Pentium 4, CPU 1.80GH, RAM 1G) with different amount of nodes, the time (ms) that the algorithm of fisheye view spent is shown in Table 1. It is found that the amount of nodes is linear with approximate time. When the amount is added to 352, consumption of time would be still less than 300ms. DFVPL is a structural design, which is a program can be sealed to use as a function node. Actually, in one program, it is hardly to find thousands of nodes but hundreds of sealed nodes to accomplish functions. So the debugger supported by fisheye view is practical.

The debugger has the characteristic of time delay, which is suitable for displaying the fisheye view, if show fisheye view according to mouse click, it will cause confusion and efficiency problem because of the sharp movement of the mouse. That is why LabScene does not integrate interactive editor with fisheye view.

**Table** 1. Runtime testing

| amount / test time | 7 | 55 | 114 | 223 | 352 |
|---|---|---|---|---|---|
| 1st | 15 | 62 | 109 | 203 | 282 |
| 2nd | 16 | 63 | 109 | 219 | 281 |
| 3rd | 15 | 78 | 109 | 217 | 297 |
| 4th | 15 | 62 | 94 | 202 | 285 |
| 5th | 16 | 62 | 93 | 210 | 266 |
| average | 15.4 | 65.4 | 102.8 | 210.2 | 282.2 |

## 5. Experiment

As the example of calculating prime number shown in Figure 1, when the debugging node is "+1", the effect is shown in Figure 4. The focus of the upper part is "+1" in the outer layer, and the focus of the lower part is "+1" in *While circle*. The constants are: $d$=2, $\beta$=0.2, $s$=2, $c$=1, $e$=1.

It is easy to observe the nodes that have dataflow relationship with "+1", reducing the attention of programmers to the nodes which are not close related to "+1". For example, in the upper part of Figure 4, While circle is close to "+1", but it is minified. It is significant that this kind of view can be shown clearly to programmers about dataflow semantics, concerned details and whole information. It is proved through the experiment that the model and the algorithms we proposed are feasible.

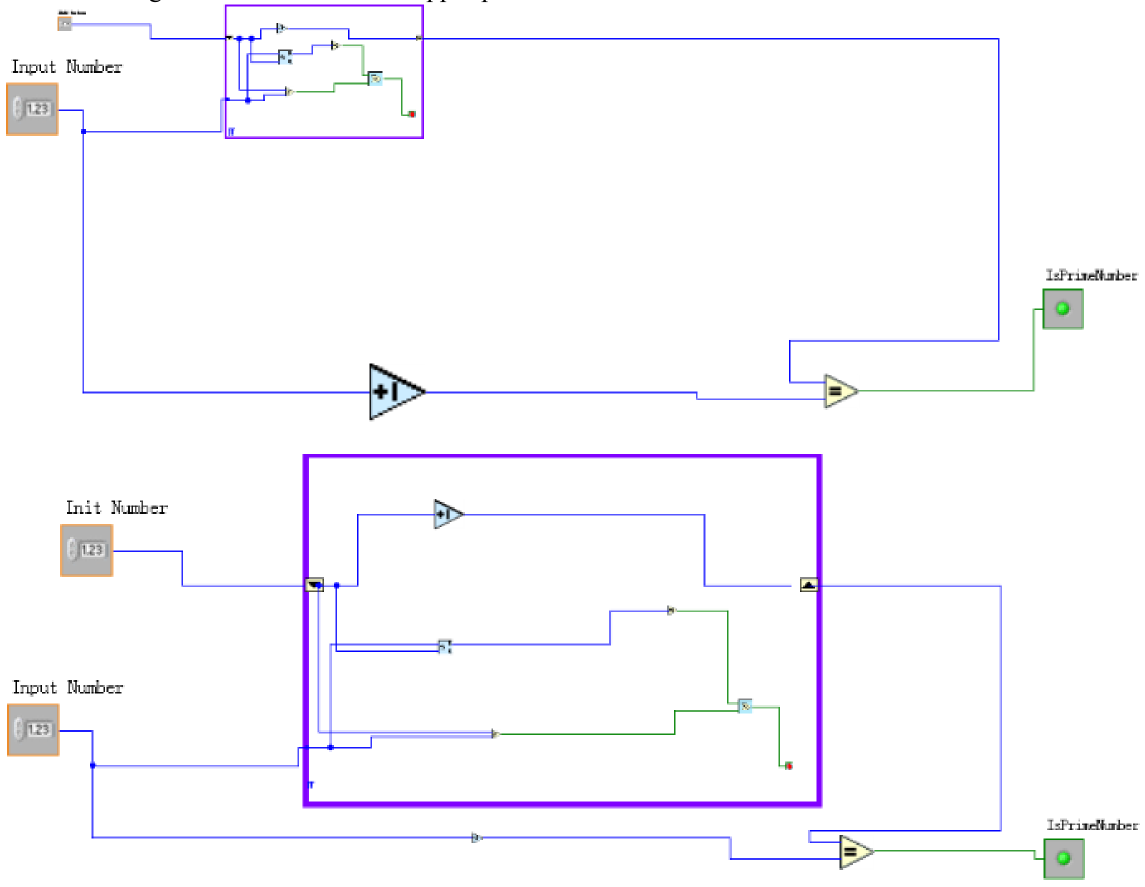

**Figure** 4. Calculating prime number in fisheye view

## 6. Related Work

The closest related work is the similar dataflow visual programming environment, KLIEG [2, 6-8]. The design of fisheye on it mainly show the effect of fisheye by reducing details of non-focus nodes and interfacing by dialog box. There are no corresponding solutions for the features of DFVPL such as nest hierarchy, the difference in nodes' size is big and wires are complex. However, if these characteristics are ignored, it will cause problems like losing nest hierarchy, covering nodes each other and losing positions of wires.

Some famous DFVPLs such as LabVIEW [14-16], Prograph [17] and VEE [18] support some tools to directly display the execution of DFVPL programs. But they have no fisheye view to help displaying visual information.

## 7. Conclusions and Future Work

This paper analyzes some problems when applying ordinary models and algorithms of fisheye view straightly in DFVPLs, such as loss of nested hierarchy, overlap of nodes and dislocation of wires etc. Aiming at these problems we proposed a new model and corresponding algorithms. It is proved that combining debugger with the improved fisheye view is practical and feasible through experiments.

The constants in fisheye view model can be used to adjust displaying effect. At present, we only choose the suitable constants through experiments. In order to improve the adaptive capacity of fisheye view, the following work is studying the relationship between layouts of DFVPLs and constants.

## Acknowledgments

This research was supported by the Programs of Science and Technology Development Plan of Jilin Province and National Science and Technique Supporting Plan Major Programs of China during the 11th Five-Year Plan Period. Xuansong Xie has contributed to earlier versions of LabScene.

## References

[1] WM Johnston, JRP Hanna, and RJ Millar, "Advances in Dataflow Programming Languages", *ACM Computing Surveys*, ACM Press, USA, 2004, vol. 36, no. 1, pp. 1-23.
[2] B Shizuki, E Shibayama, and M Toyoda, "Static visualization of dynamic data flow visual program execution", *Proceedings of the Sixth International Conference on Information Visualisation*, IEEE Press, London, 2002, pp. 713-718.
[3] GW Furnas, "Generalized fisheye views", *ACM SIGCHI Bulletin*, ACM Press, USA, 1986, vol. 17, no. 4, pp. 16-23.
[4] YK Leung and MD Apperley, "A Review and Taxonomy of Distortion-Oriented Presentation Techniques", *ACM Transactions on Computer-Human Interaction*, ACM Press, USA, 1994, vol. 1, no. 2, pp. 126-160.
[5] M Sarkar and MH Brown, "Graphical Fisheye Views", *Communications of the ACM*, Brown University, Providence, USA, 1994, vol. 3, no. 2, pp. 73-83.
[6] M Toyoda, B Shizuki, S Takahashi, S Matsuoka, and E Shibayama, "Supporting Design Patterns in a Visual Parallel Data-flow Programming Environment", *Proceedings of IEEE Symposium on Visual Languages*, IEEE Computer Society Press, Isle of Capri, Italy, 1997, pp. 76-83.
[7] M Toyoda and E Shibayama, "Hyper Mochi Sheet: a predictive focusing interface for navigating and editing nested networks through a multi-focus distortion-oriented view", *Conference on Human Factors in Computing Systems*, ACM Press, USA, 1999, pp. 504-511.
[8] B Shizuki, M Toyoda, E Shibayama, and S Takahashi, "Smart Browsing among Multiple Aspects of Data-Flow Visual Program Execution, Using Visual Patterns and Multi-Focus Fisheye Views", *Journal of Visual Languages and Computing*, Academic Press, UK, 2000, vol. 11, no. 5, pp. 529-548.
[9] ER Gansner, Y Koren, and SC North, "Topological Fisheye Views for Visualizing Large Graphs", *IEEE Transactions on Visualization and Computer Graphics*, IEEE Computer Society Press, United States, 2005, vol. 11, no. 4, pp. 457-468.
[10]T Büring, J Gerken, and H Reiterer, "User Interaction with Scatter plots on Small Screens - A Comparative Evaluation of Geometric-Semantic Zoom and Fisheye Distortion", *IEEE Transactions on Visualization and Computer Graphics*, IEEE Computer Society Press, United States, 2006, vol. 12, no. 5, pp. 829-836.
[11]C Zhang and JY Wang, "3D Model Reconstruction from Fish-Eye Images and Virtual Walk-through", *Journal of Computer-aided Design & Computer Graphics*, Institute of Computing Technology, China, 2004, vol. 16, no. 1, pp. 80-89.
[12]XH Ying and ZY Hu, "Fisheye Lense Distortion Correction Using Spherical Perspective Projection Constraint", *Chinese Journal of Computers*, Science Press, China, 2003, vol. 26, no. 12, pp. 1702-1708.
[13]XS Xie, YY Sui, and J Lin, "Hardware virtual model for G language", *Chinese Journal of Scientific Instrument*, Science Press, China, 2006, vol. 27, no. 9, pp. 1112-1115.
[14]GM Vose and G Williams, "LabVIEW: Laboratory Virtual Instrument Engineering Workbench", *BYTE*, McGraw-Hill Companies, USA, 1986, vol. 11, no. 9, pp. 84-92.
[15]R Jamal, "Graphical object-oriented programming with LabVIEW", *Nuclear Instruments and Methods in Physics Research Section A*, Elsevier, North-Holland, 1994, vol. 352, no. 1-2, pp. 438-441.
[16]R Jamal and L Wenzel, "The Applicability of the Visual Programming Language LabVIEW to Large Real-World Applications", *IEEE Proceedings of the 11th International IEEE Symposium on Visual Languages*, IEEE Computer Society Press, Darmstadt, Germany, 1995, pp. 99-106.
[17]PT Cox and T Smedley, "A Visual Language for the Design of Structured Graphical Objects", *IEEE Proceedings of 1996 IEEE Symposium on Visual Languages*, IEEE Computer Society Press, Boulder, USA, 1996, pp. 296-302.
[18]M Klinger, "Reusable test executive and test programs methodology and implementation comparison between HP VEE and LabView", *IEEE Systems readiness technology conference*, IEEE Computer Society Press, San Antonio, USA, 1999, pp. 305-312.