“3D-PP”: Three-Dimensional Visual Programming System

Takashi Oshiba and Jiro Tanaka
University of Tsukuba
1-1-1 Tennodai Tsukuba Ibaraki 305-8573 Japan
e-mail: {ohshiba, jiro}@softlab.is.tsukuba.ac.jp

1 Introduction

Our research group [1] is doing research in the field of visual programming. We are interested in three-dimensional visual programming environment in particular. The conventional visual programming systems mainly focused on two-dimensional pictorial programming. A big problem is that two-dimensional visual programming systems fail to manage a lot of pictorial programming elements. This paper proposes a new pragmatic three-dimensional visual programming system “3D-PP” to solve the problem.

2 Programming Paradigm of “3D-PP”

“3D-PP” is based on the concurrent logic programming language *GHC* [3] which is one of the high level declarative programming languages. A declarative programming language is suitable to be visualized by a visual programming system because visual programming is also declarative. A logic programming language is also suitable to be visualized because a logic programming language requires comparatively fewer number of programming elements than a procedural language.

The clause of *GHC* is composed as follows:

```
predicates(arguments, ... ) :- guard | body.
```

Both the `guard` and `body` can contain more than two goals. The principal program elements of *GHC* are atom, list, input-output data, goal and built-in goal.

The programmers of “3D-PP” can describe visual programs as follows:

2.1 Three-Dimensional Icon

The visual program of “3D-PP” is built by combining pictorial programming elements. We prepare three-dimensional miniature objects as the principal programming elements mentioned above (Figure 1). The three-dimensional miniature objects are placed as three-dimensional icons. When programmers click one of the three-dimensional icons, the pictorial programming element appears in the center of the window. The shape of the three-dimensional icon corresponds to its original pictorial programming element.

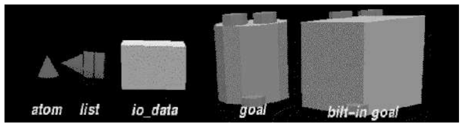

Figure 1. Three-dimensional icons.

2.2 Direct Manipulation

The left side of `:-` in the textual *GHC* code is visualized as a parent pictorial programming element in “3D-PP” (parent goal). The right side of `:-` in the textual *GHC* code is visualized as a child pictorial programming element (child goal). The visual program of “3D-PP” is composed of hierarchical nesting boxes of pictorial programming elements.

When the programmers manipulate a three-dimensional program element by using a mouse, the program element should be moved in accordance with the mouse movement. However, they have the difficulty to specify the position of the program element because the mouse is two-dimensional. To solve this problem, we apply direct manipulation technique [2] to operate a three-dimensional program element.

2.3 Extended Drag-and-Drop Technique

When a programmer describes a visual program, the programmer often move a child goal into its parent goal by using drag-and-drop of mouse. However, the drag-and-drop technique in three-dimensional space is difficult for programmers.

To solve the problem, we propose extended three-dimensional drag-and-drop technique to suit a three-dimensional space. We prepare a plane which restricts the movement of a program element (Figure 2). The restrictive plane is always parallel to the display of a computer. The normal of the restrictive plane is (d_x, d_y, d_z). When programmers move the program element while holding down the left button of the mouse, the program element is moved along the restrictive plane. The programmers can move the program elements in the three-dimensional space as if the program elements are placed in a two-dimensional space. The programmers do not need to mind whether the goal which is being dragged is collided with the other goal to be dropped or not. The program element is represented semi-transparently while it is being dragged. Other program elements are not hidden by this semi-transparent representation.

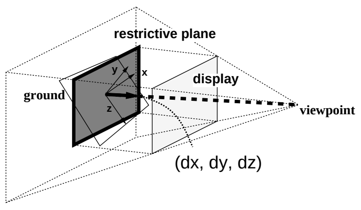

Figure 2. The restrictive plane.

3 Sample Program

GHC sample code: begin

```
:-module main.
main:-primes(1000,LP),io:outstream([print(LP),nl]).
primes(Nth,LP):-gen_primes(AB,Ps),
        pkup(Ps,Nth,AB,LP).
gen_primes(AB,Ps):-gen(AB,2,Ns)@lower_priority,
        sift(Ns,Ps).
gen(abort,_,Ns):-Ns=[].
alternatively.
gen(AB,N,Ns):-Ns=[N|Ns2],N2:=N+1,gen(AB,N2,Ns2).
sift([],Zs):-Zs=[].
sift([P|Xs],Zs):-Zs=[P|Zs2],filter(P,Xs,Ys),
        sift(Ys,Zs2).
filter(_,[],Ys):-Ys=[].
filter(P,[X|Xs],Ys):-X mod P=\=0 |
        Ys=[X|Ys2],filter(P,Xs,Ys2).
filter(P,[X|Xs],Ys):-X mod P=:=0 |
        filter(P,Xs,Ys).
pkup([P|_], Nth,AB,LP):-Nth=:=1 |
        LP=P, AB=abort.
pkup([_|Ps],Nth,AB,LP):-Nth=\=1 |
        Nth2:=Nth-1,pkup(Ps,Nth2,AB,LP).
```

GHC sample code: end

The sample code of *GHC* shown above is the program which can calculate and output the value of the 1000th prime number. `primes` is the goal which calculates and outputs the value of the 1000th prime number. `gen_primes` generates a sequence of prime numbers. `pkup` picks up the value of the 1000th prime number and transfers to `primes`. `io:outstream` registers the output data from `primes`. Figure 3 is the visual program which corresponds to the sample code of *GHC*.

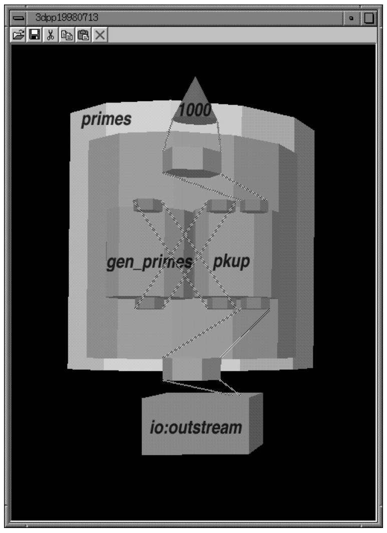

Figure 3. Snapshot of "3D-PP."

4 Related work

PrologSpace[4] adopt three-dimensional program representation based on Prolog. PrologSpace is composed of at least four panes. In this paper, our visual programming system "3D-PP" is composed of a single window.

References

[1] Takashi Oshiba and Jiro Tanaka: Three-Dimensional Modeling Environment "Claymore" Based on Augmented Direct Manipulation Technique, In *Proceedings of The 8th International Conference on Human-Computer Interaction (HCI International '99)*, Munich, Germany, August 22th to 27th, 1999 *(to appear)*.

[2] Ben Shneiderman: Direct Manipulation: A Step Beyond Programming Languages, *IEEE Computer*, Vol.16, No.8, pp.57-69, 1983.

[3] Kazunori Ueda: Guarded Horn Clauses, *ICOT Technical Report*, TR-103, 1985.

[4] Masoud Yazdani and Lindsey Ford: Reducing the Cognitive Requirements of Visual Programming, In *Proceeding of 1996 IEEE Symposium on Visual Languages (VL'96)*, pp.225-262, 1996.