End-User Programming in Three Dimensions

Natalia Kowalczyk
Technical University of Gdansk, Poland

The objective of our research is to identify requirements for, and to prototype, a three-dimensional visual programming language intended for the automation of repetitive tasks by end-users not trained as programmers. The language may also be used for solving domain-specific problems and creating prototypes. We analyse existing methods that simplify programming tasks and draw conclusions about their applicability for three-dimensional programming languages.

End-User Programming

The end user is a non-programmer, a person who uses a computer but without special training in programming. She may be a professional in other fields and thereby acquire the basic knowledge essential for programming, such as mathematics or logic, but the general trend in software development is to insulate the average computer user from the craft of traditional programming.

There are several situations, however, where the end-user may benefit from programming ability. The end-user may be, for example, an architect who knows the rules for the transformation of two-dimensional drawing into a three-dimensional model; an engineer or scientist who wishes to visualise a process; or a surgeon requiring support during operation planning; or a teacher skilled in sharing her knowledge; or an administrator who must produce a complex report. These end users often perform well-structured tasks that are suitable for automation. Furthermore, the user often has sufficient experience in the problem domain that she knows how the solution to a problem should be automated but does not know how to instruct the computer to accomplish it.

An experienced end user who can create a program that automates sequences of operations is able to pass on her knowledge of how to solve a task. End-users can also prepare a prototype to be developed further by a professional programmer.

Due to the growing popularity of computer usage and availability of non-specific software, it has been suggested that “end users must map their activities into the capabilities of the generic applications” (Cypher, 1994). The level of technology incorporated into application programs is now mature enough to let users not only customize programs to their needs but also to use parts of programs as building blocks to create custom solutions to their particular problems. Again the crucial issue is making available the right set of programming tools that are suitable for end users.

The problem examined in this paper is how to give the user the opportunity to solve her problems using available ready-to-use components without requiring her to gain extensive programming knowledge.

Three-Dimensional User Environment

Presently, the important change that is taking place in how we interact with computers is a transition from flat two-dimensional environments to spatial three-dimensional user environments. A change that is analogous to the movement
from textual to graphical environments that happened less than two decades ago. We believe that three-dimensional environments will become a basic means of human-computer interaction in the not too distant future, primarily because we live in a three-dimensional world and therefore, more naturally think and work in three dimensions. Direct manipulation of spatial objects is a natural human activity.

As hardware for image generation improves, we observe the incorporation of three-dimensional graphics in standard user interfaces. Along with the proliferation of hardware and software designed to imitate reality, human-computer interaction technology will eventually move toward three-dimensions in both input and output aspects.

In terms of software development, a spatial environment has an important advantage over a flat two-dimensional environment in that a spatial environment is able to present more information. This property is put to good use in the construction engineering, scientific and commercial visualizations.

Traditional computer programmers, skilled in the construction of textual commands, and may not necessarily benefit from such environments. They may in fact be hindered by the necessity to learn how to manipulate a new layer between themselves and the machine. It is instead the non-programmer (doctor, architect, teacher, etc.) who may gain most by being able to express her expertise through an intuitive three-dimensional computer environment which is much more like the world in which she operates.

We define a three-dimensional environment as a computer system that uses the illusion of three dimensions to present objects and to interact with the user. We do not specify the hardware or software needed to achieve such an environment, and indeed there are degrees of three-dimensional computing. A three-dimensional environment might consist of a typical graphics workstation capable of displaying three-dimensional graphics quickly and realistically enough to conjure up the appearance of a three-dimensional world on a world on a two-dimensional screen (Pang, & Wittenbrink, 1997). A more realistic implementation of a three-dimensional environment can be attained though an immersive virtual reality setup that combines the hardware and software necessary to create the illusion of reality for a user immersed in, and not simply looking into, an artificial three-dimensional environment (Cruz-Neira, Sandin, D. J., & DeFanti, T. A. 1993, Bolas, 1994). Finally, a three-dimensional computer environment can be constructed by augmenting the perception of reality with hardware and software that projects virtual artifacts into reality, so the user can both manipulate virtual objects in a real environment and manipulate real objects in a virtual environment (Feiner, MacIntyre, Haupt, & Solomon, 1993).

Three-dimensional user environments share one common trait, a more direct manipulation of the world. B. Shneiderman (1992) states the following principles of direct manipulation systems: “visibility of the objects and actions of interest; rapid, reversible, incremental actions; replacement of complex command-language syntax by direct manipulation of the objects of interest.” (p. 183). Programs operating in three-dimensional environments and tools facilitating end-user programming should be based on these principles.

Methods for Simplifying Programming Tasks

The end-user wishing to automate a task needs a method to instruct a computer. There are have been several attempts to simplify programming for the non-programmer. We consider existing techniques and point out their drawbacks for a three-dimensional environment.

A scripting language is a simple programming language used to drive a particular application such as a database or system shell. It maintains a similarity to the user’s natural language in syntax and vocabulary. It operates on compound objects instead of generic programming concepts like pointers or memory blocks. There are some widely accepted scripting languages such as Tcl/Tk (Welch, 1995), Perl (Wall, Christiansen, & Schwartz, 1996), JavaScript (Flannagan, 1997) and Visual Basic (Halvorsen, & Kinata, 1997).

When employing a scripting language, the user still has to be familiar with the general programming concepts of variables, loops and conditionals. The main disadvantage of scripting languages, however, is not the usage of traditional programming constructs but the requirement that such programs be represented textually which makes them best suited for textual, command driven user interfaces. That is why scripting languages are even less useful in three-dimensional environments than in two-dimensional ones.

Macro recorders are special programs that record the actions performed by a user on the computer and play them later at the user’s command. They are extremely easy to use, learn and support. A. Cypher (1993) discusses the capabilities and limitations of macro recorders. Cypher’s main point is that a macro recorder as a method of automating repetitive tasks is highly effective when tasks are identical and practically useless when tasks are merely similar. This is because the lack of parameters makes a recorded sequence of actions unaware of differences between tasks and the context in which it is replayed. This property applies to three-dimensional environments as well (Balaguer & Gobbetti, 1995). Macro recording is sometimes used at the beginning of program development which continues by other means such as a scripting language (Halvorson, & Kinata, 1997).

Programming by demonstration, also called programming by example, is a direct extension of macro recording. In macro recording, a sequence of operations is merely recorded for later playback. In programming by demonstration, the user shows the system an exemplary sequence of operations and the system generalizes the user’s actions by inferring her intent. Cypher (1994) states that “the motivation behind Programming by Demonstration is simple and compelling: if a user knows how to perform a task on the computer, that should be sufficient to create program to perform the task.” (p. 1).

The basic limitation in programming by demonstration is the need to formulate a set of examples that will allow the