The Art of Computer Graphics Programming: Translating Pioneer Programs

Viviane Alencar
State University of Campinas, Brazil
vivisalencar@gmail.com

Gabriela Celani
State University of Campinas, Brazil
celani@fec.unicamp.br

Abstract

Considering the importance of the use of programming languages for teaching computational design to architects, this paper proposes the translation of computer programs from a pioneer work in this field into a more contemporary programming language. The book *The Art of Computer Graphics Programming: A Structured Introduction for Architects and Designers* was published in 1987 by William J. Mitchell, Robin Ligget and Thomas Kvan, and remains an important reference for architects. The original Pascal codes in the book were translated into Processing, and made available through an Internet website, along with images and comments, in order to give late Prof. Mitchell's work the consideration it deserves.

Keywords: Processing; Pascal; Computer graphics**.**

Introduction

In the 80's, at the MIT Media Lab, Harel and Papert (1991) proposed the use of computers as a learning tool, in order to stimulate creative processes that should be involved in learning. In the same line of thinking, architecture educators have also proposed the use of programming for teaching design, such as Schmitt (1988) and Coates (1995) (Celani, 2008).

One of the very first books that was published with this purpose was *The Art of Computer Graphics Programming: A Structured Introduction for Architects and Designers* (1987), authored by William J. Mitchell, one of the most influential authors in the field of Computational Design theory, along with Robin Ligget and Thomas Kvan. The book was one of the first to establish a relationship between architecture and computer science by means of computer programming code that implemented computational design ideas. However, this book has never been republished and it is almost forgotten since the codes in it were written in Pascal, an obsolete language.

The Art of Computer Graphics Programming is much more than a Pascal programming manual. In the Preface, the authors deliberately affirm that they "are at least as concerned here with issues of design theory and visual aesthetics as we are with computer technology" (p.viii). The book introduces programming techniques along with computational design concepts, through a number of shape generation concepts that can be implemented in Pascal (as in any other computer language): variables that can take different values, thus defining parametric shapes; symmetry and repetition, which create different composition from the same vocabulary; conditionals, lead to environment-dependent solutions; encapsulation of shapes, which creates hierarchic structures; and finally, transformation operations, which carry objects from one state to another. The code examples and exercises always establish a relationship to existing architectural examples, validating the concepts presented as actual architectural design generation strategies. Along with The Logic of Architecture (Mitchell, 1990) and The Poetics of Gardens (Moore, Mitchell and Turnbull, 1988), the book can be seen as part of a trilogy (Celani, Duarte, Vaz, 2011).

This educational matter is one of the principal aspects of this work, and also of this book. Not only have the authors aimed to teach programming skills, but also they intended to explain the inner works of the computers, printers and plotters available at the time, in 1987. It is easily notable how much the technological equipment has changed since that, which made a substantial part of the book obsolete. Nevertheless, it is important to observe that even with this specific point being unnecessary to the current readers, *The Art of Computer Graphics Programming* is filled with interesting comments about the relationship between technology and architecture, especially considering the period.

The project herein described aimed at rescuing *The Art of Computer Graphics Programming* from the past, through the translation of its codes into a more contemporary programming language, thus allowing the younger generation to get in touch with an early work of one of the pioneers of CAAD. The main intention of this work was to preserve the originality of the

content of this book as one of the first to focus on the importance of programming knowledge for the modern architect.

The renewed codes were made available through an Internet website, which also includes information about the new programming language used. All the book's codes were translated into this language, and each of them is followed by comments, explanations and an illustration, allowing anyone to easily use them. By translating and making these codes available we expect to contribute with the education of a new generation of architects who understand that computers can be much more empowering when one knows how to program them.

Structure of the Book

The first part of *The Art of Computer Graphics Programming,* "Introduction to the Medium", presents information about hardware and software. The final part of the book, again very technical, discusses graphic packages for 2D and 3D-modeling, and for rendering still images and animations. In Part 2, "Elementary Graphics Program", after a brief introduction to Pascal syntax basics, the book introduces computational design concepts along with programming techniques, all of which are illustrated with sample codes. The codes on Chapters 5 through 15 were translated in the present research. The main computational concepts presented in the book are described below.

Chapter 8, "Graphic Vocabularies", introduces the "distinction between the essential and accidental properties of an object" (p. 167), i.e., between the general description of an object (its type) and its actual instantiation. The same chapter describes the "Parameterization of Graphic Elements" (p. 166), including a discussion about ranges of parameters, and the degrees of freedom of a parameterized element. The chapter ends with a section called "Defining vocabularies" of graphic elements".

Chapter 9, "Repetition", introduces "Principles of regular composition" (p. 201), and proceeds with the "Use of Control Structures to Express Compositional Rules" (p. 202), which means looping through code with statements such as For/Next, While/Do and Repeat/Until. This chapter ends with a discussion about generate-and-test-procedures.

Chapter 11, "Conditionals", presents structures that allows "to vary conditionally, according to context" (p. 273): If/Then/Else and Boolean variables. This concept is exemplified by many design situations, such as choosing among many design alternatives (state-action diagrams), exterior and interior conditions, conditional insertion of architectural elements to generate rhythms, and generate-and-test-procedures.

Chapter 12, "Hierarchical structures", shows how to create subsystems and spatial relations by specifying "the relation between its constituent vocabulary elements" (p. 324). It also introduces the concepts of recursion and recursive subdivision. Finally, Chapter 14, "Transformations", introduce Euclidean operations and show how to combine them with hierarchical structures, parameters and loops to generate symmetries.

Choosing a new Programming Language

In a previous work, we have proposed the translation of the codes from The Art of Computer Graphics Programming into a CAD scripting language, VBA for AutoCAD (Celani and Lazzarini, 2007). However, we realized that that language would soon become obsolete, because AutoDesk was shifting towards Dot Net. Besides, associating the scripts with a specific CAD software was very limiting. For that reason, we decided to look for a more general, free-standing programming language, and after considering different alternatives, we decided to use Processing.

Since its creation, in the early 2000's, Processing was intended to guide architects, designers and anyone without previous skills through learning a simple, but powerful, programming language. It is described in its website as "an open source programming language and environment for people who want to create images, animations, and interactions (…) developed to teach fundamentals of computer programming within a visual context". Therefore, it is a perfect match with the ideas presented in the book.

Processing was developed by two students from MIT Media Lab to teach computer programming concepts for architects and artists. However, due to its full potential, it is now widely used for professional language in architecture, arts and computing itself, in areas such as embedded systems and artificial intelligence, as can be seen in the tutorials and examples found on the internet.

At this moment, there are different versions available for Processing. The latest version, 2.0, was released in June 2013. For this project, version 1.5.1 was chosen, since it was the most stable version when the translation process started. In theory, the codes developed in this version can run in future ones, since only the most basic aspects of the language are used. More information about the language and its development environment can be found and downloaded for free from Processing website.

Methodology

The development methodology adopted for the project involved the following stages:

Preliminary study

A preliminary research was conducted on programming languages in general and Processing specifically. As it is possible to see by this article's references, many books on Processing have recently been published. However, they look more like manuals than text books. The most effective way to get objective information proved to be on the Processing forum. In this forum one can ask for help and suggestions, ask questions to other more experienced programmers and discuss technical aspects of the language. Many applets, applications and development environments for

Processing were found on the Internet, which confirmed that this language is very popular.

Analytical phase

The second phase consisted of creating an online support website for Processing users. The site was created with an open source online Content Management System, (Concrete 5), which allows using add-ons for inserting content, such as text, images and even computer codes. The website was divided in two parts: Languages and Projects. On Languages, a page describing Processing presents a brief description of the language, with links to online resources that had been previously identified, such as websites, e-books, videos, and apps for tablets.

The Projects page contains a sub-page that introduces the book The Art of Computer Graphics Programming and its authors. In this page we included links for each chapter of the book, as well as to a page explaining how the information is organized and how the website should be used.

The idea is to add other computer languages and translation projects to the websites in the future.

Implementation phase

In this step, the following procedures were performed for each code in the book:

a. Translating the code from Pascal to Processing, using Processing's Integrated Development Environment - IDE;

b. Generating an output image;

c. Writing comments on the implementation of the code;

d. Pasting the translated code and corresponding image in the corresponding chapter's page, and adding comments about the translation process. In some cases, a flow chart was also created to help explaining the code structure.

During the translation process, many issues arose. They were solved by consulting e-books, by posting questions on the online forum, or even by using the tablet application. A total of 199 small programs were translated into Processing (Table 1).

Evaluation phase

When all the pages are finished, the website will be evaluated by potential users in terms of usability and contribution to the teaching of computational design concepts.

Four examples of translated programs and their resulting graphics are presented in Figure 1.

Table 1:. Programs translated from each book Chapter

Chapter	Programs translated
5	9
6	5
7	10
8	24
9	31
10	17
11	29
12	8
13	20
14	36
15	10
Total number of programs	**199**

Discussion

During the completion of the translation some conceptual differences between the Pascal and Processing programming languages were found. Some of them did not cause relevant differences in the codes, such as the orientation of the coordinate axes being opposed. However, we realized that some exercises in the book should be moved to later chapters because, although they involve very basic concepts in Pascal, they require a bit more knowledge in Processing. This can be exemplified by the function that takes input values: in Pascal it requires just a simple command line asking for the input value, but in Processing it requires a function. A document explaining these differences will be included in the website. Despite these observations, the translation of Pascal codes into Processing was considered to be very feasible, allowing the examples and exercises in the book to maintain their original pedagogical aims.

The choice for the Processing language over other languages surveyed (such as C, Java , Python , Design By Numbers, POV Ray and Cinder) proved consistent as it is a programming language that has focused on teaching since it was conceived. The pedagogical aspect was one of the main issues considered of the present work, as well as in the book. The authors not only sought to teach programming techniques, but also intended to propose a new, computational way of understanding architecture. Thus, one of the challenges of this work was to preserve the originality of the contents of the book, focusing on the fact that it was one of the first to emphasize the importance of programming knowledge for the contemporary architect.

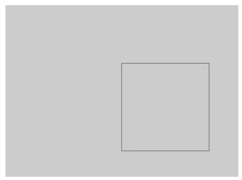
Conclusion

Despite the use of an obsolete language, *The art of computer graphics programming* remains a useful guide to learning programming for anyone, not just architects. Its pedagogical way of teaching each concept based on the previous is unique and could inspire other computer science educators. Moreover, the relationships established between computational concepts and architectural design in the book remain an important and actual issue, so we expect that the translated version of the codes will make it possible to give this book again the credits and the consideration it deserves.

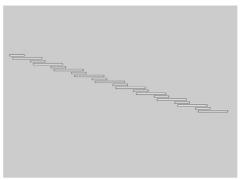
```
PROGRAM SQUARE;
      USES GRAPHICS;
      VAR X1,X2,Y1,Y2,SIDE :
INTEGER;

      BEGIN
 { Independent variables }
      X1 := 400;
      Y1 := 200;
      SIDE := 300;

 { Dependent variables }
      X2 := X1 + SIDE;
      Y2 := Y1 + SIDE;

 { Draw the square }
      START_DRAWING;

          MOVE (X1,Y1);
          DRAW (X2,Y1);
          DRAW (X2,Y2);
          DRAW (X1,Y2);
          DRAW (X1,Y1);

      FINISH_DRAWING;
 END;
```

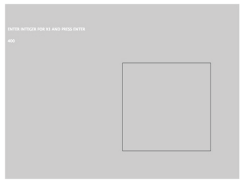
```
void setup(){
  size(800,600);
}

void draw(){

  //independent variables
  int x1 = 400;
  int y1 = 200;
  int side = 300;

  //dependent variables
  int x2 = x1 + side;
  int y2 = y1 + side;

  line(x1,y1,x2,y1);
  line(x2,y1,x2,y2);
  line(x2,y2,x1,y2);
  line(x1,y2,x1,y1);
}
```

PROGRAM SQUARE (1)

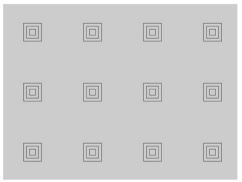


PROCEDURE STAIRS





PROGRAM SQUARE (2)



PROGRAM GRID

```
PROGRAM SQUARE;
USES GRAPHICS;
  { Declare the coordinate
variables }
VAR X1,X2,Y1,Y2 : INTEGER;

BEGIN
  { Prompt for and read in
coordinates }
WRITELN ('Enter integer for X1');
READLN (X1);
WRITELN ('Enter integer for X2');
READLN (X2);
WRITELN ('Enter integer for Y1');
READLN (Y1);
WRITELN ('Enter integer for Y2');
READLN (Y2);

  { Draw the square }
START_DRAWING;

    MOVE (X1,Y1);
    DRAW (X2,Y1);
    DRAW (X2,Y2);
    DRAW (X1,Y2);
    DRAW (X1,Y1);

FINISH_DRAWING;

END;
```

```
void setup(){
  size(800,600);
}

void draw(){
text("ENTER INTEGER FOR X1
AND PRESS ENTER",10,90);
   text(saved,10,130);

int x1 = int(saved);
   int y1 = 200;
   int x2 = 700;
   int y2 = 500;

if(x1 != 0){
   line(x1,y1,x2,y1);
   line(x2,y1,x2,y2);
   line(x2,y2,x1,y2);
   line(x1,y2,x1,y1);
  }
}

String typing = "";
String saved = "";

void keyPressed(){
if (key == '\n' ) {
   saved = typing;
   typing = "";
 } else {
   typing = typing + key;
  }
}
```

```
PROCEDURE STAIRS
(X_INITIAL,Y_INITAL,DEPTH,WIDTH,
X_INCREMENT,Y_INCREMENT,
NUM_OF_STAIRS,N : INTEGER);
  VAR COUNT,X,Y,
  LANDING_DEPTH : INTEGER;

  BEGIN
     LANDING_DEPTH := 2 * DEPTH;
     X := X_INITIAL;
     Y := Y_INITIAL;

     FOR COUNT := 1 TO
NUM_OF_STAIRS DO
     BEGIN
 { Determine when to insert landing }
       IF COUNT MOD N = 0 THEN
          BEGIN
             RECTANGLE
(X,Y,LANDING_DEPTH,WIDTH)
             X := X + DEPTH;
          END
       ELSE
          RECTANGLE
(X,Y,DEPTH,WIDTH);

     X := X + X_INCREMENT;
     Y := Y + Y_INCREMENT;
  END;
END;
```

```
void setup(){
  size(800,600);
}

void draw(){

  int x_initial = 25;
  int y_initial = 180;
  int depth = 50;
  int width1 = 5;

  int x_increment = 10;
  int y_increment = 10;
  int num_of_stairs = 20;
  int n = 5;

  int landing_depth = 2 * depth;
  int x = x_initial;
  int y = y_initial;

  for(int count = 1; count <=
num_of_stairs; count++){
     if(count % 2 == 0){

rect(x,y,landing_depth,width1);
        x = x + depth;
     }else{
       rect(x,y,depth,width1);
     }

     x = x + x_increment;
     y = y + y_increment;
  }
}
```

```
PROGRAM GRID;
USES GRAPHICS;

PROCEDURE SQUARE (X,Y,SIDE :
INTEGER);
    VAR X1,Y1,X2,Y2 : INTEGER;
    BEGIN
 { Calculate values for X1,Y1,X2,Y2 }
       X1 := X - (SIDE DIV 2);
       Y1 := Y - (SIDE DIV 2);
       X2 := X1 + SIDE;
       Y2 := Y1 + SIDE;
 { Move to bottom left corner }
       MOVE (X1, Y1);

 { Draw the four sides of the square }
       DRAW (X2, Y1);
       DRAW (X2, Y2);
       DRAW (X1, Y2);
       DRAW (X1, Y1);
  END;

PROCEDURE NESTED_SQUARES
(X,Y,DIAMETER, INCREMENT, NEST :
INTEGER);
    VAR COUNT : INTEGER;
    BEGIN
   { Loop to nest squares centered around X,Y
}
       FOR COUNT := 1 TO NEST DO
           BEGIN
              SQUARE (X,Y,DIAMETER);
              DIAMETER := DIAMETER -
INCREMENT;
           END;
    END;

PROCEDURE ROW
(INITIAL_X,Y,DIAMETER,INCREMENT,NEST,
SPACING,NUM_COLUMNS : INTEGER);
    VAR X, COUNT_COLUMNS : INTEGER;
    BEGIN
       X := INITIAL_X;
    {Loop to place row of nested squares }
       FOR COUNT_COLUMNS := 1 TO
NUM_COLUMNS DO
           BEGIN
              NESTED_SQUARES
(X,Y,DIAMETER,INCREMENT,NEST);
              X := X + SPACING;
           END;
    END;

 { Main program }
VAR Y,COUNT_ROWS : INTEGER;
    BEGIN
       START_DRAWING;
          Y := 500;
      { Loop to draw grid of nested squares }
           FOR COUNT_ROWS := 1 TO 3 DO
            BEGIN
                ROW (100,Y,60,20,3,200,4);
                Y := Y - 200;
            END;
        FINISH_DRAWING;
    END.
```

```
void setup(){
  size(800,600);
}

void draw(){
  int y = 500;
  for(int count_rows = 1;
       count_rows <= 3;
       count_rows++){
    row(100,y,60,20,3,200,4);
    y = y - 200;
  }
}
void row(int initial_x, int y,
int diameter1, int increment, int
nest,
int spacing, int num_columns){
  int x = initial_x;
  for(int count_columns = 1;
count_columns <=
num_columns;
count_columns++){

nested_squares(x,y,diameter1,
increment,nest);
    x = x + spacing;
  }
}

void square(int x, int y, int
side){
  int x1 = x - side/2;
  int y1 = y - side/2;
  int x2 = x1 + side;
  int y2 = y1 + side;

  line(x1,y1,x2,y1);
  line(x2,y1,x2,y2);
  line(x2,y2,x1,y2);
  line(x1,y2,x1,y1);
}

void nested_squares(int x, int
y, int diameter1, int increment,
int nest){
  int count;

  for(count = 1; count <= nest;
count++){
    square(x,y,diameter1);
    diameter1 = diameter1 -
increment;
  }
}
```

Figure 1: Four examples of translated programs and the resulting graphics.

Acknowledgments

The authors would like to acknowledge CNPq for supporting this research, by providing a one-year scholarship to Viviane Alencar.

References

Celani, M. G., & Lazarini, K. (2007). Using CAD for generating architectural form: Reviewing and translating pioneer programs. Proceedings of the V Mathematics & Design International Conference.

Celani, M. G. (2008). Teaching CAD programming to architecture students. Gestão & Tecnologia de Projetos, São Carlos. Retrieved from http://www.iau.usp.br/posgrad/gestaodeprojetos/index.php/gestaodeprojetos/article/view/73

Celani, M. G., Duarte, J. P., & Vaz, C. V. (2011). The gardens revisited: The link between technology, meaning and logic?. Proceedings of the 16th International Conference on Computer Aided Architectural Design Research in Asia / The University of Newcastle, Australia, 643-652. Retrieved from http://cumincad.scix.net/cgi-bin/works/Show?caadria2011_061

Coates, P., & Thum, R. (1995). Generative modelling – student workbook. London: University of East London.

Concrete5 (n.d) Retrieved from http://www.concrete5.org

Greenberg, I. (2007). Processing: creative coding and computational art. Friendsof.

Harel, I.; Papert, S. (1991) Constructionism. Norwood: Ablex.

Mitchell, W. J., Ligget, R. S., & Kvan, T. (1987). The art of computer graphics programming, a structured introduction for architects and designers. New York, NY: Van Nostrand Reinhold Company.

Mitchell, W. J., Moore, C. W., & Turnbull, W. (1988). The poetics of gardens. Cambridge, MA: The MIT Press.

Mitchell, W. J. (1990). The logic of architecture: design, computation, and cognition. Cambridge, MA: The MIT Press.

Processing.org (n.d) Retrieved from http://www.processing.org

Project Website (n.d) Retrieved from http://www.revistaparc.fec.unicamp.br/lapac/index.php/re/programming

Reas, C., & Fry. (2007). Processing: a programming handbook for visual designers and artists. Cambridge, MA: The MIT Press.

Schimitt, G. (1988). Microcomputer Aided Design for Architects and Designers. New York; John Wiley & Sons.

Shiffman, D. (2008). Learning Processing: a beginner's guide to programming images, animation, and interaction. Morgan Kaufmann.