ISSN 1512-1232

A Domain-Specific Language for Line Drawings

Aashi Jain, Archita Goyal and Pinaki Chakraborty*
Division of Computer Engineering, Netaji Subhas Institute of Technology, New Delhi 110078, India
*E-mail: pinaki_chakraborty_163@yahoo.com

Abstract:
A domain-specific language to draw line drawings is introduced in this paper. The compiler converts the specification of a line drawing written in this language into an image file. The language is suitable for drawing line drawings used in scientific literature.

Keywords: *Line drawing, domain-specific language, graphic language, compiler.*

1. Introduction

Scientists and engineers are regularly required to draw line drawings. Line drawings appear in books, articles, dissertations and various other documents. Line drawings can be drawn using a wide range of software tools colloquially called graphics editors. Most of the available graphics editors use a canvas-based approach in which a diagram is drawn by dropping and dragging various shapes. This approach is easy to use and requires no training. However, this approach suffers from some drawbacks. It is difficult to draw large or intricate diagrams using the canvas-based approach. Making modifications, even minor ones, is also difficult if the canvas-based approach is used. These problems can be avoided using the alternate domain-specific language based approach. This approach was first used for typesetting technical documents about fifty years ago. In this paper we introduce a domain-specific language, named Line Drawing Description Language (LDDL), in which the specification of a line drawing can be written. We have implemented a tool that converts such a specification into an image file in the JPEG format. Although non-programmers will require some training to use LDDL, users with some experience in programming can start drawing line drawings using LDDL readily.

2. Related Work

In the last fifty years several line drawing languages have been designed. Unfortunately, many of them were described only in in-house technical reports which are now difficult to locate. However, there are also a few well-known line drawing languages which require a brief review. In an early study, Frank [1] developed a line drawing language named B-LINE. B-LINE provided a small set of basic graphics statements. Programs were written using these and conventional Fortran statements. A macro-processor converted such a program into a conventional Fortran program. B-LINE was supposed to be used for producing illustrations in the Bell System Technical Journal. Around the same time, Kulsrud [2] developed an elaborate compiler-compiler based framework to design line drawing languages. The use of this framework was demonstrated by designing a language whose syntaxes looked like those of an assembly language. The language supported various advanced features like those to analyze and manipulate the diagrams. More than a decade later, van Wyk [3] and Kernighan [4] designed two line drawing languages named IDEAL and PIC, respectively. IDEAL was a well-developed procedural language with support for several advanced features like multi-layered pictures. Alternatively, PIC had features of procedural languages and languages processed by preprocessors. Both IDEAL and PIC were processed using preprocessors. Several versions and re-implementations of these languages were also developed in the later years. Additionally, some languages have been also designed for particular subtypes of line drawings like flowcharts, graphs and chemical structures.

ISSN 1512-1232

3. The Language

LDDL is a high-level language. To be more specific, it is a procedural language having syntaxes and semantics similar to, but not exactly same as, those in the C and C++ programming languages. LDDL has been intentionally kept small and simple. A minimal set of features essential for describing a line drawing is supported. The only data type supported in LDDL is the 16-bit `int`. LDDL supports `if-else` and `while` statements. LDDL also supports functions. Arguments may be passed to a function, only pass-by-value is supported however. A function may or may not return a value. The arguments and return value, if any, must be of the type `int`. Recursion is allowed. A function must be defined before the main subroutine. Common arithmetic and logic operators are also supported.

LDDL provides eighteen types of specialized graphic statements. Twelve of them are used to draw various shapes (Table 1) and the rest are used to specify properties of the shapes (Table 2).

Table 1. Statements to draw various shapes

Statement	Description
`pixel (int x, int y);`	Draws a pixel at (x,y).
`line (int x1, int y1, int x2, int y2);`	Draws a line between (x1,y1) and (x2,y2).
`rectangle (int left, int top, int right, int bottom);`	Draws a rectangle with (left, top) as the upper left corner and (right, bottom) as the lower right corner.
`roundedrectangle (int left, int top, int right, int bottom);`	Draws a rectangle with (left, top) as the upper left corner and (right, bottom) as the lower right corner, and rounds the corners.
`polygon (int n, int left, int top, int right, int bottom);`	Draws a regular polygon with n sides that can fit inside the rectangle with (left, top) as the upper left corner and (right, bottom) as the lower right corner.
`circle (int x, int y, int radius);`	Draws a circle with center at (x,y) and a specified radius.
`circlefit (int left, int top, int right, int bottom);`	Draws the largest circle that can fit inside the rectangle with (left, top) as the upper left corner and (right, bottom) as the lower right corner.
`ellipse (int x, int y, int xradius, int yradius);`	Draws an ellipse with center at (x,y) and specified radii along the x- and y-axes.
`arc (int x, int y, int startangle, int endangle, int radius);`	Draws a circular arc with center at (x,y) from startangle to endangle and a specified radius.
`beziercurve (int x1, int y1, int x2, int y2, int u1, int v1, int u2, int v2);`	Draws a cubic Bezier curve between (x1,y1) and (x2,y2) and influenced by the points (u1,v1) and (u2,v2).
`arrowhead (int x, int y);`	Draws an arrowhead pointed at (x,y) facing right.
`text (int x, int y, char string[]);`	Displays a string starting at (x,y).

Table 2. Statements to specify various properties of shapes

ISSN 1512-1232

Statement	Description
`setdimensions (int maxx, int maxy);`	Sets the dimensions of the canvas with (maxx,maxy) as the lower right corner.
`setcolor (int r, int g, int b);`	Sets the color for all lines and texts.
`setfillcolor (int r, int g, int b);`	Sets the fill color for all shapes.
`setrotation (int angle);`	Rotates shapes by a specified angle.
`setlineweight (int width);`	Sets a specified width, in pixels, for all lines.
`setlinestyle (int style);`	Sets the line style, 1 for solid line, 2 for dashed line and 3 for dotted line.

Statements are available to draw pixels, lines, rectangles, rectangles with rounded corners, circles, ellipses, arcs, Bezier curves and arrowheads. In each of these statements, coordinates and other necessary parameters have to be provided. The `text` statement is available to position a text starting at a specified coordinate. The other group of statements is used to specify important properties of the shapes like the color of lines and texts, fill color, angle of rotation, and width and style of lines. A property specified by such a statement is applicable to all shapes drawn subsequently till the same statement is used again to modify that property. In the `setcolor` and `setfillcolor` statements, the values of the red, green and blue components of the color may have their values from 0 to 255. The `setdimensions` statement is used to specify the size of the drawing canvas. The canvas is white and of 100 pixel x 100 pixel in size by default. By default, the color for drawing lines and writing texts is black, there is no rotation, line weight is of one pixel, and lines are solid in style. Additionally, LDDL supports the #include preprocessor directive to include header files containing function definitions. LDDL supports both single- and multiple-line comments.

4. The Compiler

We have implemented a compiler for LDDL. The LDDL compiler uses many of the principles and techniques used in typical compilers but also has a few peculiar features. The compiler has five phases which have been arranged in two passes (Figure 1). The syntax analyzer used in the LDDL compiler is a backtracking-based recursive-descent parser. The intermediate code generator produces a low-level equivalent of the source program for a hypothetical stack machine. The synthesizer *executes* the low-level program produced by the intermediate code generator. It reads an instruction from the program and performs the operations denoted by it. This may include drawing a shape on the canvas, specifying properties of shapes to be drawn next, performing arithmetic and logic operations, and changing the flow of control of the program. The synthesizer uses a stack to store local variables, evaluate expressions and implement function calls. On successful compilation, the LDDL compiler saves the canvas as an image file in the JPEG format. The LDDL compiler has been implemented in C++.

5. Sample Programs

Figure 1 has been drawn using LDDL. Figures 2-5 present some more drawings drawn using LDDL. The source code of Figure 3 is given below to show how a typical LDDL program looks like.

```
main
{setdimensions(400,400);
```

ISSN 1512-1232

```
 int i;
 i=0;
 while(i<8)
     {if(i%2)
          setfillcolor(255,0,0);
      else
          setfillcolor(0,0,255);
      circle(i*50+5,200,i*5+5);
      i++;
     }
}
```

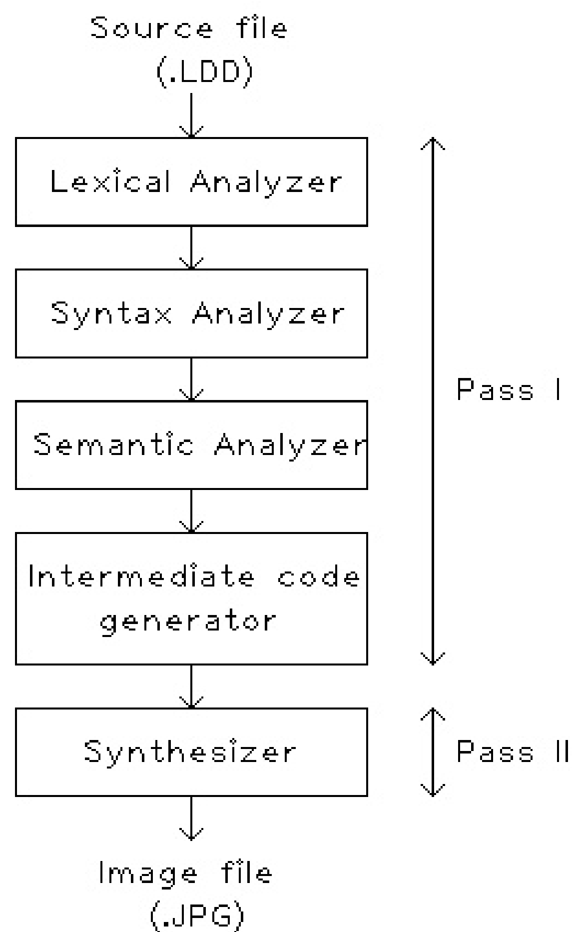

Figure 1. Block diagram of the LDDL compiler.

Pixel

Line

Line Color

Rectangle

Fill Color

Rounded Rectangle

Rotation

Polygon

Line Style

Ellipse

Line Weight

Circle

Arc

Bezier Curve

Arrow Head

Figure 2. Different features supported by LDDL.

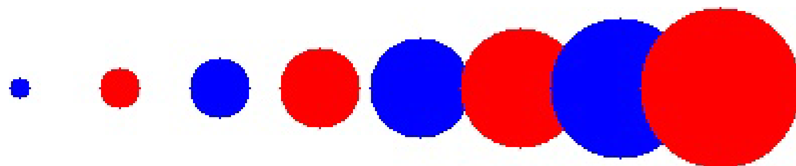

Figure 3. A sample drawing using iterative statement.

ISSN 1512-1232

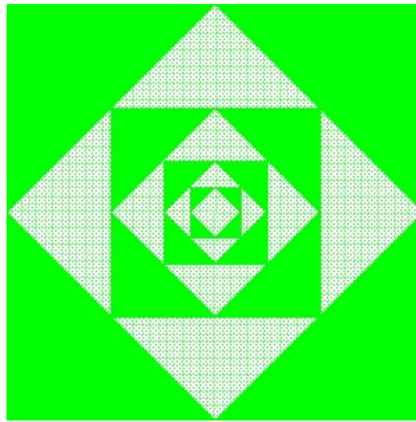

Figure 4. A sample drawing using recursive function call.

Figure 5. A sample entity-relationship diagram drawn using LDDL.

6. Conclusions

ISSN 1512-1232

We have defined a line drawing language whose syntaxes and semantics are similar to those of typical high-level languages. Implementing the tool to process it as a compiler made LDDL more powerful than its predecessors. The compiler produces ready-to-publish image files. The compiler, in both source and executable forms, and some sample programs are available at https://drive.google.com/folderview?id=0B6CbAq8pewJOSENrdWxUUmkwR3c&usp=sharing. A grammar for LDDL has been also made available.

In our opinion, a graphics editor should support both canvas- and language-based approaches to draw diagrams. A user should be free to use either of these approaches to draw a diagram. Such a graphics editor will be suitable for drawing technical as well as non-technical diagrams, and will be equally popular among scientists, engineers and other users.

References

[1] Frank, A. J. B-LINE, Bell line drawing language. *Proceedings of the AFIPS Fall Joint Computer Conference*, 1968, 179-191.
[2] Kulsrud, H. E. A general purpose graphic language. *Communications of the ACM*, 1968, **11**(4), 247-254.
[3] van Wyk, C. J. A high-level language for specifying pictures. *ACM Transactions on Graphics*, 1982, **1**(2), 163-182.
[4] Kernighan, B. W. PIC — A language for typesetting graphics. *Software: Practice and Experience*, 1982, **12**(1), 1-21.

Article received: 2016-08-22