Simple Image Generation Language (SIGL)

Phong Pham Abelardo Gutierrez Alketa Aliaj

Programming Languages and Translators
Spring 2007

In this project, we propose developing a new language for 2D image generation, Simple Image Generation Language (SIGL). SIGL can be seen as a combination of two commonly used drawing languages: Virtual Reality Modeling Language (VRML), widely used for specifying 3D models, and Logo programming language. It deploys the familiar image specification methodology and syntax of VRML while providing more control and flexibility with conditional branching and loop similar to that of Logo. Our designing goal for SIGL is to provide users with a simple, natural and yet flexible way to draw 2D images.

1 Motivation

Why a drawing language? With the availability of a lot of drawing software available, drawing images are just the matter of simple mouse clicks. However, not all images can be easily drawn by hand. For example, it would be very tedious to draw a grid, or a set of many co-center circles. A drawing programming language would make repetitive drawing task easy. It also allows drawing of complicated mathematical curves.

Why another drawing language? While VRML is a widely used standard, it is not suitable for programmers. In most cases, VRML files are generated by drawing softwares. On the other hand, Logo is a powerful drawing language, but it uses a very different drawing methodology known as "turtle". Many people who are used to ordinary coordinate system might find Logo hard to use. In our language, we provide a drawing system using same OpenGL drawing methodology as VRML, empowered with additional flow control found in common programming languages. We also introduce a unique feature of point binding to allow natural reference to points during drawing.

Why a totally new language? Even though it is possible to build a similar library in general purpose languages such as Java/C/C++, the drawing syntax would be complicated by the parent language. The users also need to know the parent languages at the first place. In contrast, SIGL provides a natural and easy-to-understand syntax which a beginner can catch up in a short period of time.

2 Basic drawing operations

2.1 Primitives

SIGL provides all basic 2D primitives including points, lines, ellipses, and polygons. For convenience, SIGL will also provides additional primitives such as circles, rectangles, squares. Additional utility shapes can easily be added to the language or defined by users.

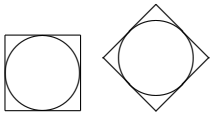
Similar to OpenGL, all primitives are supposed to be in object coordinate. These primitives can then be put into appropriate position in world coordinate using transformation operators.

2.2 Transformations

SIGL supports 3 basic transformations: scaling, translation, and 2D rotation.

2.3 Example use

```
polygon{                // we would like to draw a polygon
    {0,0},              // list of vertices of the polygon
    {0,1},              // here the vertices will form unit square
    {1,1},
    {1,0}
}
translate(0.5,0.5){     // draw a circle inside the square
    circle(0.5);
}
// we'll draw the same square here
// except that this square would be rotated and translated
translate(2,0){         // translate it in horizontal direction by 2 units
    rotate(45){         // rotate it counter-clockwise by 45 degree
        polygon{        // the same drawing instructions as above
            {0,0},
            {0,1},
            {1,1},
            {1,0}
        }
        translate(0.5,0.5){
            circle(0.5);
        }
    }
}
```

which produces 

3 Reusability

3.1 Point binding

In order to facilitate natural drawing, SIGL features a special binding for points to variables. Here is an example of drawing a unit square with 2 diagonals

```
var A, B, C, D;
polygon{                // we would like to draw a polygon
    A = {0,0},          // list of vertices of the unit square
    B = {0,1},
    C = {1,1},
    D = {1,0}
}
// draw the 2 diagonals
line(A,C);
line(B,D);
```

When a point is assigned to a variable, the variable is bound to that point and all its transformations at the time of assignment. The variable can later be used be used as a reference to point in other drawing commands. This allows more readable and natural drawing code. This is the unique feature offered by our language.

3.2 User-defined functions

A group of drawing commands can be defined as a function, for example, the example in Section 2.3 can be shorten into

```
function subpic() {
    polygon{            // we would like to draw a polygon
        {0,0},          // list of vertices of the polygon
        {0,1},          // here the vertices will form unit square
        {1,1},
        {1,0}
    }
    translate(0.5,0.5){ // draw a circle inside the square
        circle(0.5);
    }
}
// draw the custom shape
subpic();
// we'll draw the same square here
// except that this square would be rotated and translated
translate(2,0){         // translate it in horizontal direction by 2 units
    rotate(45){         // rotate it counter-clockwise by 45 degree
        subpic();       // draw the same shape again
    }
}
```

4 Other features

To allow maximal flexibility, SIGL also provides other features commonly used in other general purpose languages. However, users can still perfectly draw images without using any of these features.

- Dynamic type system.
- Possible types for variables are *int*, *double*, *boolean*, *point*, and *array*.
- All basic arithmetic operations (addition, subtraction, multiplication, integer division, floating point division) as well as other commonly used mathematic functions (log, min, max, exp, sin, cos, tan).
- Boolean logical operations (and, or, not, xor)
- Flow control structures *if*, *for*, and *while* will also be supported and have syntax similar to Java/C/C++.