Curve

Kun An, John Chan, Dave Mauskop, Wisdom Omuya, Zitong Wang
{ka2401, jcc2220, dbm2130, awo2108, zw2193}@columbia.edu

Columbia University
COMS W4115: Programming Languages and Translators

September 26, 2012

Motivation

Curve is a vector graphics manipulation language specifically targeted for graphics processing. It is a text-based graphics language that directly creates scalable vector graphics (SVG) to describe and manipulate images.

Although there exists graphical utilities that edit SVG files, it is often difficult to find ones that include batch job functions that allow it to process many images at a time. Furthermore, this process can be tedious and counterintuitive. Curve will allow designers more flexibility in manipulating vector graphics while leveraging powerful programming constructs, available whilst dynamically rendering images.

Key Features

The goal of the Curve syntax is to make conceptually simple graphics and manipulations using simple, yet powerful, language constructs. The only basic type in Curve is – yes, you guessed it – Curve. This type, while simple, is very expressive. e.g. a group of curves may be used to represent a layer, a group of layers used to form an image. In Curve, we will allow some operators to be overloaded to provide a shorter notation for common functions. Rotation, scaling, and layering are some of the effects that can be executed using these operators.

Example

Curve leverages a very powerful basic type that allows the user to create more complex and interesting components using even more expressive operators.

```
// "$" means create a curve.
// The type of the variable is not declared explicitly, as it can be inferred.

// Create a point.
```

```
myPoint = $(0, 0);

// Create a line segment.
myLine = $(0, 0)(1, 1);

// Connect two line segments.
myConnectedLines = myLine1 + myLine2;

// So a line segment can also be created by writing
myLine = myPoint1 + myPoint2;

// Create a Zigzag. A zigzag can be thought of as a special kind of
// curve. Typical curves make smooth turns, whereas zigzag has sharp turns.
myZigzag = $(0, 0)(1, 1)(2, 0)(3, 1);

// Create a triange.
myTriangle = $(0, 0)(1, 1)(1, 0)(0, 0);

// Move myTriangle to a new location.
// >>(10, 0), for example, means that move myTriangle
// upward by 5 units, and move it to the right by 10 units.
myTriangle>>$(10, 5);

// Equivalently, we have
myTriangle<<$(-10, -5);

// A layer is a group of curves.
myLayer = [myCurve1, myCurve2, myCurve3];
```

Built-in Types and Operators

Types

We strove for a minimal set of built-in types. Graphics languages often have built-in types such as rectangles, polygons, circles, ellipses, lines, curved lines, and points. We started out with this list, and realized that Curve, as we have defined it, is a generalization of all these types. We can even replace Strings with curves! Our hope is that by reducing the number of built-in types we will simplify the job of writing a compiler for our language. We hope to create an extensive standard library that offsets the loss in convenience that comes with fewer built-in classes.

`Integer, Double:`

We will have integers and doubles, which will be defined in the conventional way. For these types, the operators we will support are: multiplication, addition, subtraction, division, conversion to a string, greater than, less than, equals, greater than or equal to, and less than or equal to.

`Curve`

A curve is made up of pieces. Each piece is defined by two points which are its endpoints and some number of points which are the intermediate control points of the Bezier curve which connects the endpoints. When constructing a curve, we assume that pieces are listed in sequence. That is, the second endpoint of the i^{th} piece is the first endpoint of the $(i+1)^{th}$ piece. The curve may either be connected (the second endpoint of the last piece is the first endpoint of the first piece) or not. The built in operators on curves are: translation, scaling, rotation, and coloring with an object of type Color. (Note that reflection about a line can be accomplished with rotations and translations.)

`Color`

Minimally, we will support RGB colors that can be used to change the color of a curve. If time allows, we hope to support more complex coloring schemes, such as gradients.

Keywords

`if, elif, el, for, while, break, continue, switch`

Reserved

`>>, >>, $, +, -, /, ., *, (, ), ;, ], [, ,,`