SOS Language Project Proposal

Terric Abella, Sitong Feng, G Pershing, Sheron Wang
tta2117, sf3049, gap2148, xw2765

February 03, 2021

1. Description/Overview

SOS is a functional-like language with C inspired syntax designed to create mathematically interesting images, especially self-similar fractals (realistically only in 2D). With recursive functions that can be passed and manipulated as in OCaml, the users can elegantly build their desired objects. The language will also utilize the OpenGL library for its graphics features. Our goal is to present a language that provides convenience for creating very complicated (and nice-looking) images with as few lines of code as possible, and render them efficiently.

2. Syntax/Language Details

2.1 Data Types

2.1.1 Primitive Types

- `int`
- `float`
- `boolean`

SOS will support standard arithmetic (`+, -, *, /`) and comparison (`>, <, >=, <=, ==, !=`) operators for ints and floats.

2.1.2 Structured Types

- `list`
  - `[a,...,n]` where items a - n are items in a list
  - Elements must have the same type
  - A linked list like implementation similar to Ocaml
  - Concatenation
  - Slicing
- `struct`
  - Access with `.property` like in C

2.1.3 Premade Types

- `point = struct (float x, float y)`
  - Add/subtract (gives a `vector`)
  - Affine application (`*`)
- `vector = struct (float x, float y)`
  - Add/subtract
  - Add/subtract to a `point` (gives another `point`)
  - Scalar multiplication (`*`)
  - Affine application (`*`, ignores translations)
  - Dot product (•), cross product (`*`)
- `curve = point list`
  - Points to be rendered in a continuous path
  - Slice `[:]`
  - Concatenation
  - Select
- `affine`
  - A 3D matrix representing an affine transformation.
  - Matrix multiplication (`*`)
- `shape = point list`
  - Similar to curve, but the shape is interpreted as a loop that is filled in
  - Same operations as curve (maybe different concatenation)
- `color = struct (float r, float g, float b, float a)`
  - RGBA color, can also construct as HSV
  - Some color operations, (like mix and multiply)

2.2 Variable Declaration

Variables can be declared globally or locally. A global variable is declared with an assignment operator, =. A local variable is declared with a `let` … `in` … expression.

2.3 Functions

- A function declaration must specify the function name and parameters. If the function returns a value, the function declaration must also specify the return type.
- Functions are first-order
- Pattern matching, guards, if-then-else

For example:

```
// the function sum takes two ints and returns an int
let int sum (int x,int y) = x + y
```

2.4 Control Flow

Conditionals:

- If then else

2.5 Library Functions

SOS will support an extensive standard library of useful math and graphics functions. Here are some examples:

2.5.1 `sqrt : int -> int, float -> float`
Given an int/float m, return an int/float that takes the square root of m

2.5.2 `scale : float -> affine`
Given a float m, returns an affine representing a uniform scale by a factor of m.

2.5.3 `rotate : float -> affine`
Given a float m, returns an affine representing a counterclockwise rotation by m.

2.5.4 `trans : float * float -> affine, vector -> affine`
Given two floats or a vector, returns an affine representing a translation by the given vector.

2.5.5 `reverse : list -> list`
Reverses a given list.

2.5.6 `render`
Given a curve/shape, and some parameters (curve color, background color) renders the image using OpenGL.

2.5.7 `mousepos : () -> vector`
Returns current mouse position, which can be used as a parameter for `render` function to indicate where the image will be rendered

2.5.8 `animate : shape -> func -> float -> shape`

Takes a shape, a function that represents the transforming action (e.g. scaling, rotating, translation and etc.), and a float that represents the time duration of the action, the resulting shape will be animated accordingly (change from the original shape to the transformed shape, then alternates back and forth)

2.6 Keywords

The following words are reserved keywords in SOS:

let	Specify variables and functions
in	Allow local variable binding in expression
if, then, else	Specify conditional expression
true, false	Specify boolean logic
int, float, struct, boolean	Specify types of variables

2.7 Comments

The comments are C like but can be nested. Single-line comments use //, while multi-line comments use /* */.
For example:

```
/*  /* nested */ */
```

3. Example Programs

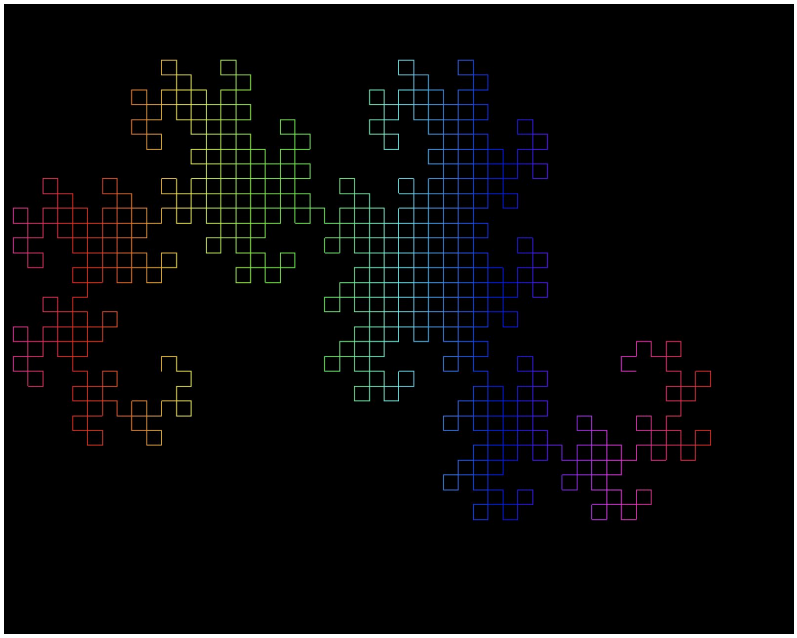
```
// set up affine transformations using primitives
// e.g. scale, rotate, translate, shear
affine A = scale(sqrt(2.0)) * rotate(45)
affine B = trans(0.5, 0.5)*scale(sqrt(2.0))*rotate(135)*trans(-1, 0)

// recursively defines the dragon curve
curve dragon (int n) =
     if n == 0
     then [(0, 0), (1, 0)]
     else let d = dragon (n-1) in
```

```
      /* affine application pointwise to a curve, followed by curve
      appending (+) */
      A * d + B * reverse(d)

// interpolates a color along the line a-b
color rainbow (point a, point b, vector position) =
     // hsv color picker, • for vector dot product
      hsv (position • (b-a) / (b-a) • (b-a), 1, 1)

/* does the work to render an image
 * color is passed as a vector->color function, which automatically
assigns different colors to different segments of the curve
 * background is passed as a color, could also be passed as a
vector->color function for a gradient effect */
render (translate (60, 200) * scale(360) * dragon (10),
color=rainbow((0, 240), (480, 240)), background=rgb(0, 0, 0))
```



Approximate Render

```
// (Approximately) Equivalent code in C#/Unity, for comparison

Vector3[] DragonCurve (int n) {
     const float sqrt2on2 = 0.7071067812f;

     if (n == 0) {
          Vector3[] ret = new Vector3[2];
          ret[0] = Vector3.zero;
          ret[1] = new Vector3(1, 0, 0);
          return ret;
     }

     Vector3[] d = DragonCurve(n - 1);
     Vector3[] combined = new Vector3[d.Length * 2 - 1];
     for (int i = 0; i < d.Length; i++) {
          // Transform by A, B (this is SO HARD TO DEBUG)
          Vector3 Ad = sqrt2on2 * (Quaternion.Euler(0, 0, 45) *
d[i]);
          Vector3 Bd = sqrt2on2 * (Quaternion.Euler(0, 0, 135) *
(d[d.Length - 1 - i] - new Vector3(1, 0, 0))) + new Vector3(0.5f,
0.5f, 0);
          combined[i] = Ad;
          combined[i + d.Length - 1] = Bd;
     }

     return combined;
}

Color RainbowColor (Vector3 a, Vector3 b, Vector3 pos) {
     Vector3 dir = b - a;
     float h = Vector3.Dot(dir, pos - a) / dir.sqrMagnitude;
     return Color.HSVToRGB(h, 1, 1);
}

Vector3[] d = DragonCurve(10);
for (int i = 0; i + 1 < d.Length; i++) {
     Vector3 a = d[i] * 360 + new Vector3(60, 200);
     Vector3 b = d[i + 1] * 360 + new Vector3(60, 200);
     DrawLine(a, b, RainbowColor(new Vector3(0, 240), new
Vector3(480, 240), 0.5f * (a + b)), 60f);
}
```