SOS

Tojo Abella, Sitong Feng, G Pershing, Sheron Wang

Introduction

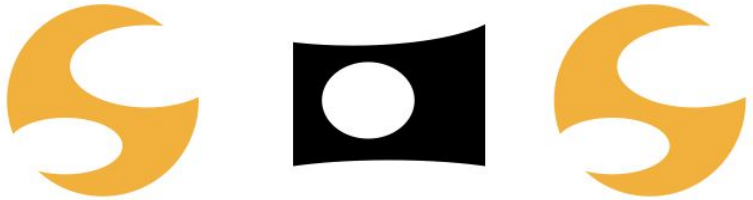
of

SOS

(Shape Open System)

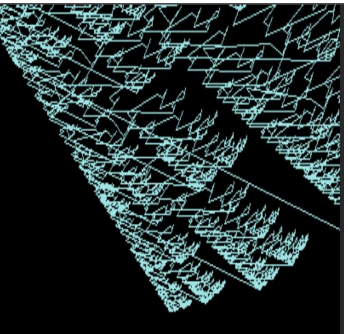
- Introduction
- Motivation

Introduction

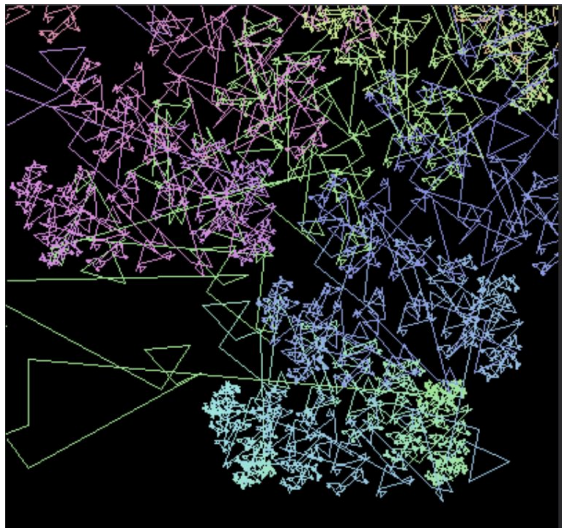


SOS (Shape Open System) Language

- Imperative
- Focus on 2D image support, especially complex shapes
- Emerges from C and OCaml like syntax
- Employs OpenGL for graphics utils
- Provide math operation based on C



< - artistic image rendered instantly - > leaf.ppm



Motivation

- Existing Languages/API
  - takes long time to learn
  - takes too much code to do complicated operations on shape
  - no great support on shapes alone

- SOS
  - simple and quick to master
  - create mathematically complex images elegantly

Features

of

SOS

(Stylistically Observed Structures)

- Basic Features
- Unique Features
- Import
- Math Library
- Graphics Library

Basic Features

- Types
  - `bool, int, float`
  - `array, struct, func`
- Expressions
  - Most statements have a value
  - No statement end syntax—expression ends are unambiguous
  - Arithmetic, if/else, sequencing
- Recursion
  - We have it!

```
n : int = 1
f : float = 6.2832
l : array int = [1, 2, 3]
struct point = {x: float, y: float}
p : point = {-1.3, 4.6}

fac : (n: int) -> int =
    if n==0 then 1
    else n * fac(n-1)
fac_ref : func int->int = fac

print(fac(l[2]))
```

Unique Features

- Struct arithmetic
  - Addition, scaling, dot product
  - Matrix multiplication
- Implicit Array iteration
  - For operators
  - For functions

```
struct point = {x: float, y: float}
struct mat2 = {a11: float, a21:
float, a12: float, a22: float}

p   : point = {1.0, 1.3}
q   : point = 3*p + {0.0, 2.0}
dot: float = p * r
ccw: mat2   = {0.0, 1.0, -1.0, 0.0}
q = ccw ** q

double : (n: int) -> int = 2*n
double([0,1,2]) // = [0, 2, 4]
a: array array int = [0,1]+[2,3]
  // = [[2, 3], [3, 4]]
```

Import and standard libraries written in SOS

- Naive import
  - Works like `#include` in C
  - Replaces the line to codes in another file
  - Increases extendibility of our language
  - Duplicate files detection
- Standard Libraries written in SOS
  - Makes OpenGL calls easier to use
  - Define functions and structs using SOS
  - List of libraries:
    - `renderer.sos`
    - `point.sos`
    - `shape.sos`
    - `color.sos`
    - ... and more in future!

color.sos

```
struct color = {r: float, g: float,
b: float, a: float}

alias colors = array color
```

helloworld.sos

```
import color.sos

c1 : color = {255.0, 0.0, 0.0, 0.8}

c2 : color = {0.0, 255.0, 0.0, 0.8}

color_arr : colors = [c1, c2]
```

Math Library

- Could link to C math library **easily** in LLVM
- Use by `import math.sos`
- Functions that we support as a graphic language:
  - `float sqrt(float x)`
  - `float sin(float x)`
  - `float cos(float x)`
  - `float tan(float x)`
  - `float asin(float x)`
  - `float acos(float x)`
  - `float atan(float x)`
- Yet another function that we implemented with C math library utilization:
  - `float toradians(float x)`
- Several math functions implemented with pure SOS:
  - `float floor(float x)`
  - `float ceil(float x)`
  - `float frac(float x)`
  - `float max(float x)`
  - `float min(float x)`
  - `float abs(float x)`
  - and more!

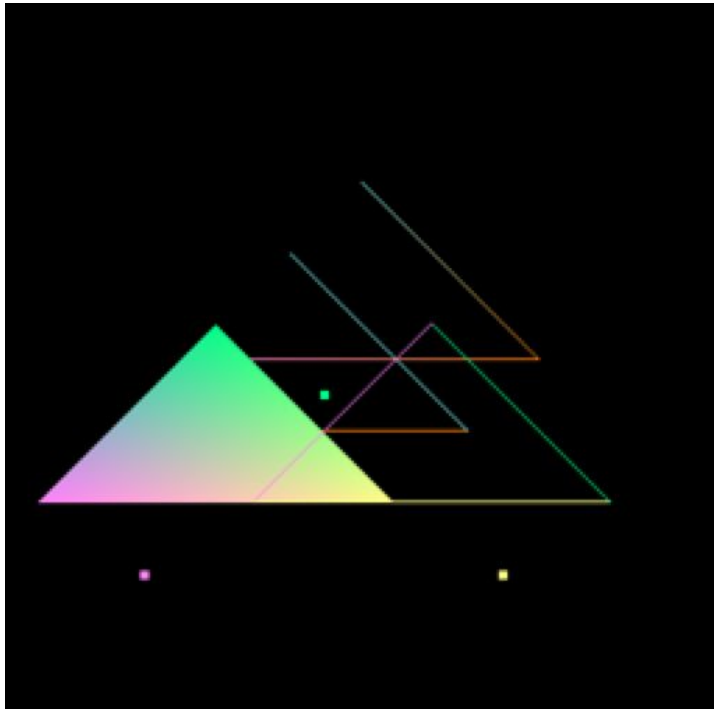
Graphics Library - Renderer.sos

Canvas functions: given an SOS canvas, start or end OpenGL context

- `startCanvas(...)` starts MESA context with appropriate window size
- `endCanvas(...)` ends MESA context, saves window to .ppm file

Drawing functions: given a point array and color array, draw objects

- `drawPoints()` draws points on current canvas
- `drawPath(...)` draws path on current canvas
- `drawShape(...)` draws shape on current canvas



Graphic Library - OpenGL

CODE TO CREATE AN OPENGL CONTEXT: VERY COMPLICATED!

```
void gl_startRendering(int width, int height){
    ctx = OSMesaCreateContextExt(OSMESA_RGBA, 16, 0, 0, NULL);
    if (!ctx){
        printf("OSMesaCreateContext failed!\n");
    }

    buffer = malloc( width * height * 4 * sizeof(GLubyte) );
    if (!buffer) {
        printf("Alloc image buffer failed!\n");
    }

    // Bind the buffer to the context and make it current
    if (!OSMesaMakeCurrent(ctx, buffer, GL_UNSIGNED_BYTE, width, height)) {
        printf("OSMesaMakeCurrent failed!\n");
    }
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glMatrixMode(GL_MODELVIEW);
   glClear(GL_COLOR_BUFFER_BIT);
   glPushMatrix();
   glEnableClientState(GL_VERTEX_ARRAY);
   glEnableClientState(GL_COLOR_ARRAY);
   glColor4f(1.0, 1.0, 1.0, 1.0);
```

Compiler

of

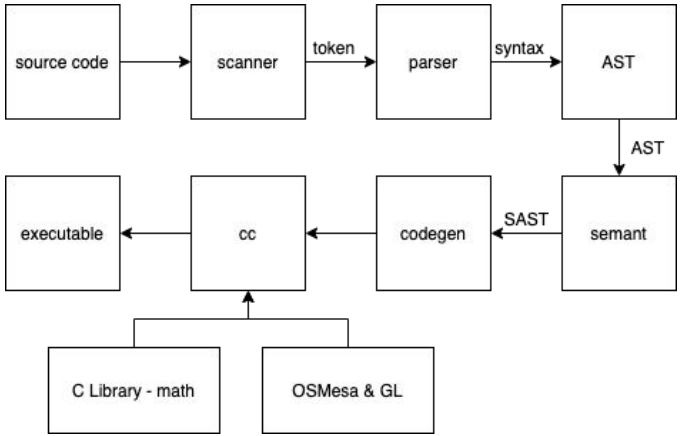
SOS

(Sad Oblique Shapes)

- Architecture
- Docker Image

Compiler Architecture

- Math bindings supported by LLVM
- Utilize **The Mesa 3D Graphics Library**, one open source software implementation of OpenGL
- Could run inside of the docker image provided by us, thanks to **Off-Screen Rendering Mesa**
    - render into main memory without any window system or operating system dependencies
    - save graphics to *.ppm files when rendering ends



Reference: https://docs.mesa3d.org/osmesa.html

```
docker pull sheronw1174/sos-env
```

If you are seeking a Docker Image with OpenGL & LLVM etc.

Wrap up

of

SOS

(SOS Object System)

- Challenges
- Future Work

Challenges

Running OpenGL is hard, and running OpenGL **inside of Docker** is especially hard.

— Sheron the one who says she's going to help but she's not

Codegen is easy as long as you don't value your sanity.

— G the God

Using OpenGL is hard. Thank G, I mean thank God I have SOS!!

— Tojo the peasant

SOS has saved us from complicating our brain.

— Sitong the editor

Future Work

Basics

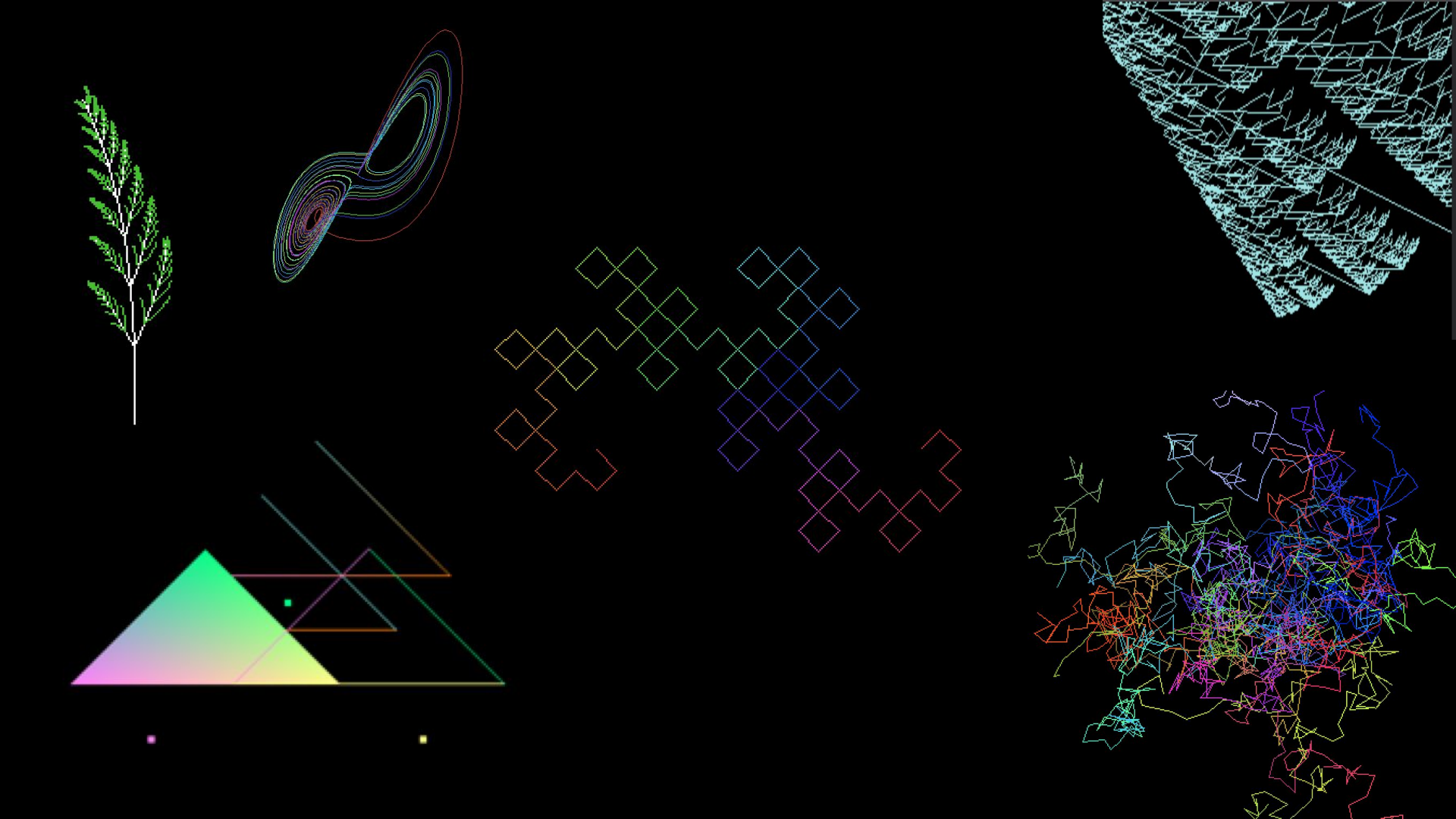
- Incorporate more OpenGL utilities such as line type - glLineWidth
- Add function scope
- Add basic built-in shapes
- Memory Management

Advanced

- Add 3D Shape Support/Plot Support
- Add more third-party API support
- Allow real-time interactivity

DEMO for SOS

a.k.a. Silly Odd Shapes



Thank you!