RayForth: A Forth and Ray Caster Implemented in C++

Jeffrey Drake

25 Mar 2012

1 Introduction

RayForth started out as a simple attempt to make a ray tracer. In the process of building the ray tracer, a method of decoupling the definition of the scene from the compiled source, or more simply, a configuration file.

In evaluating the various means to parse a configuration file language, Boost Spirit was the top thing on the list, but it did not seem like there would be enough time to learn it well enough. The idea of implementing a Forth came with the thoughts of the utmost simplest way of representing data.

A Forth is a postfix oriented concatenative programming language where its simplest expression is a series of numbers progressively put on a stack and function calls to consume them and put a result back on the stack.

2 Building RayForth

RayForth uses cmake, boost, and libgd. In the default Debian GNU/Linux install with appropriate packages, everything will be found. It assumes libgd is located in /usr/lib.

The short version of building is as follows:

```
cd RayForth
mkdir build
cd build
CXX=g++-4.7 CC=gcc-4.7 cmake ..
make
./ray
```

As of this writing, gcc 4.7 is required for C++11 support.

3 RayForth Usage

RayForth supports input from a file or input from the console with a prompt. When executing ray with the help option, it will display the options as shown

in figure 1.

```
ray <options>
options:
  -h [ --help ]           display options
  -v [ --version ]        displays version
  -i [ --input-file ] arg input file
```

Figure 1: RayForth Options

To render the sample sphere shown in figure 4, type the text shown in figure 2 and when it is rendered, it will say 'ok' on the line after.

```
0.5 0.5 0.5 vec3 0.25 make-sphere 640 640 "image3.png" render-scene
```

Figure 2: Sphere Commands

Operationally, this will place a three element vector on the stack followed by a number, which are the parameters for the sphere. The sphere on the stack becomes followed by the width and height of the scene, and the output file.

4 RayForth Words

Figure 3 shows the words that are implemented in RayForth as of this writing. The stack specification should be read as what is on the stack before the call is on the left side of the – and the results are on the right. N refers to a number, $ is a string, A and B are any type, V3 is a three element vector, S is a sphere, and T is a triangle.

5 RayForth File Overview

RayForth is implemented through four main modules shown in figure 5. The main module sets up the options for the forth engine, the forth module interprets input, and the ray module implements all of the linear algebra and ray tracing required.

6 Main Module

The main module is purposed to explain to the user how to run RayForth and starts up the forth interpreter if there are no options or an input file specified. Boost::Program_Options is used to simplify the processing of input options (shown in figure 1, and Boost::FileSystem is used to verify a file exists if specified. Alternatives, are a report of version or requesting help.

Word	Stack	Description
+	(N N – N)	Adds two numbers on the stack
-	(N N – N)	Returns the difference of two numbers on the stack
*	(N N – N)	Multiplies two numbers on the stack
/	(N N – N)	Divides two numbers on the stack
%	(N N – N)	Finds the modulus of two numbers on the stack
.	(A –)	Prints the object on the stack
cr	(–)	Prints a new line
swap	(A B – B A)	Swaps the top two items on the stack
drop	(A –)	Drops the top item on the stack
vec3	(N N N – V3)	Creates a 3 element vector
make-sphere	(V3 N – S)	Creates a sphere with centre and radius
make-triangle	(V3 V3 V3 – T)	Creates a triangle
render-scene	(S/T N N $ –)	Renders a scene with an object, width and height of scene, and file output
bye	(–)	Quits (alternatively, EOF)
.s	(–)	Outputs what is on the stack without consuming it

Figure 3: Words available

Once the state is setup, Forth::executeForth is called with the input source, either cin or a file stream.

7 Math Module

Ray intersection is inherently a linear algebra problem. The Math module implements three element vectors exclusively, by using a std::array containing three elements of float. Float was used for expediency, and the use of double would cause double the memory to be used. An important generalization would have been to template the type away, but it was deemed unnecessary for the purposes of this project.

7.1 Notable Functionality

The function *feq* was written to support easy comparison of two floating point numbers, as it is possible for two floating point numbers to differ by a very little amount and it would escape equality tested with *operator ==*.

The functions *dot*, *operator -*, *operator +*, *operator **, and*operator /* were all implemented using the standard library algorithm *inner_product*, and this use of the standard library reduced the amount of code written and made it more generic at the same time.

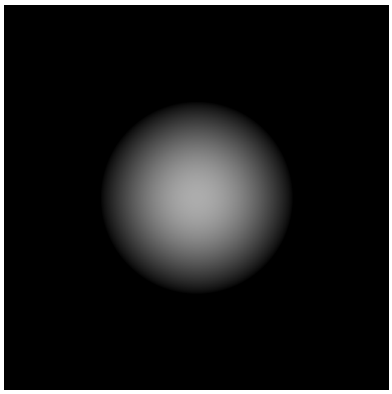


Figure 4: Sphere Rendering Output

Source File	Module	Description
main	main	Contains the initialization code to start the forth environment.
rayforth	forth	Processes input and delegates the stack.
forthparse	forth	Tokenizes input.
forth_basicwords	forth	Implements the basic words for forth.
forth_raywords	forth	Implements the words required to define the ray caster and render.
geometry	ray	Implements the intersection testing and underlying geometry required.
linear_algebra	math	Implements the required linear algebra functions.

Figure 5: RayForth Modules

The functions *cross*, *distance*, *unit*, and *norm* are all implemented using combinations of the previous functions. An additional ostream *operator «* was implemented to make it easy to output the vectors.

8 Ray Module

The Ray Module (implemented in Object namespace) implements everything necessary to define Spheres, Triangles, Rays, and Intersections. A tuple was chosen to represent all geometric data over a structure to reduce the amount of code dedicated to the types themselves.

This choice makes some polymorphism more difficult, but the boost library included the idea of the boost variant and this was used to merge a sphere and triangle into a general object.

A design decision was made to avoid exceptions in the intersection handling,

so taking a cue from Haskell, a MaybeIntersection was implemented with a pair involving a bool and intersection. This could be improved with a boost variant or better a boost optional.

The two objects implemented, Sphere and Triangle, each have a *test_intersection* function that tests its intersection with a Ray. Both return a MaybeIntersection. Polymorphism between the types has been implemented with the visitor pattern from boost variant in the functor *intersection*.

The function *intersection_colour* converts an intersection to a three element array containing the RGB values associated with an angle of intersection. If no intersection was present, then the colour is the background colour.

9 Forth Module

9.1 forth.hpp

All of the specific types required by the forth components are provided in this header file. The parsing token is always a bool, a double, std::string, or a Symbol struct. The stack types are implemented in the variant Generic, and include doubles, strings, three element vectors, spheres, and triangles.

Every time a new type is supported, the type must be operated on by the echo and echoTypes visiting functor. This is similar to a Haskell program implementing the Show typeclass on a data type.

9.2 forthparse

The problem of parsing the forth input was one of the most practical of problems that I learned. The solutions evaluated included *Boost::Spirit*, and regular expressions. Spirit was determined to be more complicated than time would allow, and regular expressions were proving to be hard to become unambiguous with the input. Regular expressions use a state machine, so I quickly learned more about state machines, and came up with a custom state machine that managed all the possible transitions. *Note: The state machine diagram is shown on the last page of this document because of its size.*

The states are represented by an enum struct, with a debugging list implemented with a map over that enum initialized by an initializer list. Some utility functions are declared and then used in the transfer table, also a map used with an initializer list. An example transfer function is shown in figure 6.

The actual parser functor operates from that table, checking for inconsistent state and special cases such as a Number, String, and Symbol. Drawbacks from the transfer function, is that it does not provide any means to perform a function, and that complicates the parser functor by requiring special handling.

9.3 forth_basicwords

All of the words implement a pattern of taking a Stack& and return a Status. The Stack is a vector of Generics (a variant), and thus able to cover all the

```
{ State::neutral,
{
{ isSign, State::sign },
{ isDigit, State::digit_sequence },
{ isDecimal, State::decimal_before_digits },
{ isSpace, State::neutral },
{ isQuote, State::string_start },
{ isNarrowPrintable, State::symbol_char }
} },
```

Figure 6: Transfer Function

possible types. Status is a tuple containing a bool and a string. The bool containing true, means everything is alright. If false, then the string contains details.

Each function tests its size(), and then the data types. An error indicator is returned, and the stack backtrace is printed and the stack is restored to the state before the line was processed.

9.4 forth_raywords

These functions implement *vec3*, *makeSphere*, *makeTriangle*, and *renderScene*. The *renderScene* function is the function that drives the actual ray casting. It is worth discussing what generally occurs for a ray cast operation.

A ray is created for every pixel from 0 to 1 subdivded by width and height. A vector of MaybeIntersection now contains the intersection status. The function *Object::intersection_colour* transforms it into a vector appropriate for holding RGB image data. At this time, libgd is used to create the image and save. Unfortunately, a side effect of using libgd in this way, is that I must use fopen to open files instead of a proper ofstream.

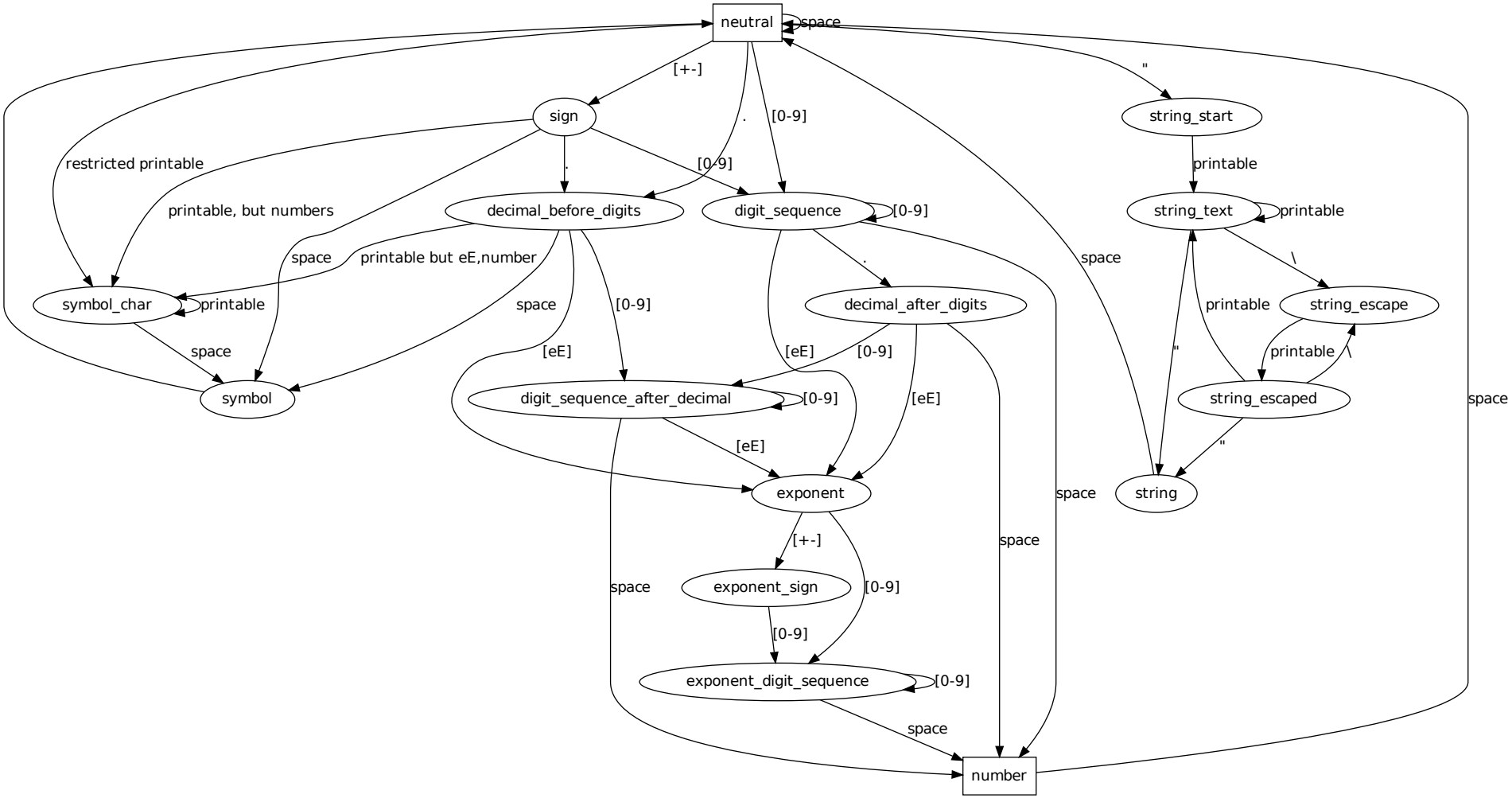
9.5 rayforth

Rayforth brings all of the other modules together to create the forth engine. All of the words defined in the previous two modules, are stored in an overall word map. A functor *_process* provides an easy way to deal with each token (number, string, symbol) and does the right thing when it is applied with the visitor pattern. This functor pushes strings and numbers, and calls symbols. When a symbol is not found or a symbol call fails, an appropriate error and stack traceback is given.

10 Future Directions

I see two main directions that I can take this project. It has really gone from a Ray Tracer to a Forth interpreter. I would like to see more of the ray tracing code be able to be done directly in the interpreter instead of the forth just being an interface to the C ray tracer.

Further investigation of state machines and parsers would be advisable, and eventually adding support for lambdas and variables would complete the language to make it general purpose.



RayForth FSM