Firefly3D Language Reference Manual

An Educational Programming Language for Creating 3D Graphics

Roy Aslan, Prerna Chikersal, Alex Shnayder -- {ra2752, pc2667, ajs2119}@columbia.edu

July 24, 2014

1. Introduction

This manual describes the Firefly3D (FF3D) language. Specifically, it explains the syntax and semantic conventions required to properly use the language. FF3D is a language that allows a program to construct 3D shapes and objects by drawing lines in three dimensional space. It is inspired by the Logo programming language (which is implemented two dimensions). The programmer can move the Firefly within space in straight lines, whereby each movement leaves a trail in a color that can be specified and altered by the programmer.

FF3D is an imperative programming language. The entry point for the program is the “main” code. The main code is the code enclosed within the curly braces, just after the “main” function declaration/definition. For example,

```
int alpha=2;
int newAlpha(int a){
        return a+10;
}
int main() {
        int beta = 30;
        int theta = newAlpha(beta);
        print(theta);
        return 0;
}
```

2. Lexical Conventions

There are five classes of tokens: identifiers, keywords, literals, operators, and separators. Blanks, horizontal and vertical tabs, newlines, and comments as described below (collectively, “white space”) are ignored except as they separate tokens. If the input stream has been separated into tokens up to a given character, the next token is the longest string of characters that could constitute a token.

2.1. Comments

FF3D supports block comments. Similar to C/C++ block comments, a comment block will being with /* and end with */. The compiler will ignore any comments embedded in the source code. Single line comments or nested comments are not supported.

2.2. Identifiers

An identifier is a sequence of characters that consists of alphabets, numbers and underscore. An identifier cannot begin with a number or underscore. It is used to label/reference a variable or a function.

2.3. Statement Separators

FF3D employs semicolons (;) to separate primary statements and return statements (see §8). Statements are the smallest unit of code to be executed and include function calls and variable assignments

2.4. White spaces

The FF3D compiler will require at least one white space character to distinguish between (and separate) tokens. It will ignore all white spaces such as spaces, tabs and newlines beyond a single space. Thus, it will treat any continuous stream of white space in the source code regardless of length as a separator between the preceding and trailing tokens.

2.5. Case sensitivity

FF3D is a case sensitive language. All tokens are case sensitive.

2.6. Keywords

FF3D supports the following keywords:

Keyword	Description
int	Integer data type designator
float	Floating point data type designator
bool	Boolean data type designator
string	String data type designator
vec3 <int/float>	Three dimensional vector data type designator
if, elseif, else	Specifies conditional expression. Else is compulsory
true/false	Possible values of a boolean variable
repeat	Designate a block of statements to be repeated a certain number of times
while	Designates a block of statements to be executed until an expression is qualified as being true
return	Denotes the expression whose value will be returned by the function

2.7. Operators

Operator	Description
+ - * /	Addition, Subtraction, Multiplication and Division arithmetic operators
== != > < >= <=	Equal to, Not equal to, Greater than, Less than, Greater than and equal to, Less than and equal to comparison operators

=	Assignment operator
&& ||	And and Or logical operators

2.8. Standard Library

FF3D has the following built-in standard library functions:

Function	Description
forward(float distance)	Advances FireFly by the distance in the direction of its current orientation
rotate(string direction, int degrees)	Rotates the firefly in either up, down, left or right directi by the number of specified degrees (relative to its curren orientation)
setPen(bool truthVal)	activates or deactivates the Firefly's pen
getPen()	returns true if the pen is active and false otherwise
setPenColor(vec3<int> color)	sets the Firefly's Pen color to a particular R, G and B value
getPenColor()	returns a vec3 containing the R, G and B values of the Firefly's pen color
getX()	Returns the X component of a vec3
getY()	Returns the Y component of a vec3
getZ()	Returns the Z component of a vec3
clearScr()	Clears the screen, without affecting the firefly's position
resetPos()	Resets the firefly's position to the initial default
print(int/float/vec3/string)	Prints any integer, float, vec3 or string value to the cons for debugging purposes
main()	Defines the entry point of the program that will be executed initially

2.9 Literals

FF3D supports the following literal types: integer-literal, float-literal, string-literal, bool-literal.

2.9.1 Integer

An integer is a sequence of one or more digits [0-9] with no other characters located anywhere in the sequence .

2.9.2 Float

A float is a sequence of digits with a single period located somewhere in the sequence (including the beginning or end).

2.9.3 String

A string is a sequence of characters surrounded by double-quotes. For certain special characters, the below escape sequences can be used:

Character	**Escape Sequence**
Newline	\n
Tab	\t
Backslash	\\
Double-quote	\”

2.9.4 Bool

A bool is either equal to true or false (both of which are keywords used to represent a boolean value).

3. Type Conversions

FF3D does not support type conversions. The programmer has to explicitly define the data type of each identifier.

4. Meaning of Identifiers

Identifiers can refer to two possible types of things: basic types or derived types.

4.1 Basic Types

There are four fundamental types:

int:	integer type
float:	floating number type
bool:	boolean type
string:	string of characters type

4.2 Derived Types

Besides the basic types, there are two derived types constructed from the fundamental types in the following ways:

vec3: a vector containing 3 elements of the same type (int or float);
function: returning objects of a given type.

5. Expressions and Operators

5.1 Precedence and Associativity

The order of precedence for evaluating expressions will match the ordering of the below sub-sections. For example, the first sub-section listed below (§5.1) refers to expressions with the lowest precedence. All operators within the same sub-section are left-associative.

5.2 Primary Expressions

Primary expressions are identifiers, literals, or expressions in parentheses.

primary-expression:
- *identifier*
- *literal*
- *vec3*
- *(expression)*

5.2.1 Identifier

An identifier is a name that refers to a variable or function. If the identifier refers to a variable (either a literal or vec3), and is used as an expression, then the return value will simply be the value of the variable that it refers to. An identifier that refers to a function cannot be used as an expression unless it is being used as part of a function application (see §5.3).

5.2.2 Literal

A literal evaluates to itself. See §2.9 for the available types of literals.

5.2.3 Vec3

A vec3 also evaluates to itself, and corresponds to a tuple of either three ints or three floats. For example, a vec3 may correspond to the tuple (3,5,2).

5.2.4 Parenthesized Expression

A parenthesized expression is simply an expression surrounded by parentheses. The return value is the evaluation of the enclosed expression.

5.3 Function Application Expression

A function is applied by specifying the identifier of the function, followed by a list of comma-separated arguments enclosed in parentheses. If the function definition does not specify any parameters, then the function call must still include an open and close parentheses with no values in between them. A function application expression evaluates to the return value of the function.

5.4 Arithmetic Expression

An arithmetic expression evaluates two expressions and performs an arithmetic operation on their values based on the given operator. The two expressions must return a value of the same type, which must be either a float type or an int type. The result of the arithmetic expression will have the same type as the provided expressions.

arithmetic-expression:
int-expression arithmetic-operator int-expression
float-expression arithmetic-operator float-expression

arithmetic-operator: one of
+ - * /

Note that arithmetic operators follow the standard arithmetic order of precedence (i.e. multiplication and division have higher precedence than addition and subtraction).

5.5 Logical Expression

A logical expression evaluates two expressions and performs a logical operation on their values based on the given operator. The two expressions must return a bool value. The result of the logical expression will be either true or false.

logical-expression:
bool-expression logical-operator bool-expression

logical-operator: one of
&& ||

5.6 Comparison Expression

A comparison expression evaluates two expressions and compares their values based on the given comparison operator. The expression will return either true or false.

comparison-expression:
expression comparison-operator expression

comparison-operator: one of
== != > < >= <=

6. Function Definition

Functions can be defined and called from anywhere in the program. Functions must be fully defined at the point of declaration. Arguments (or parameters) for functions can be specified using any of the supported data types. Functions must have a return value and must therefore have a designated data type. A compulsory “main” function must be defined and will be executed as the starting point of the program (from which other functions can be called). Functions cannot be declared/defined within other functions.

7. Variable Declaration & Initialisation

Variables can be declared with a designated data type and assigned a literal value. A variable must be declared before it is assigned a value. Declaring and assigning a variable in a single statement is not supported. For example, the following code is supported:

```
float x;
x = 1.0;
```

The following code is not supported:

```
float x = 1.0;
```

8. Statements

A FF3D program can contain the following types of statements:

8.1. Primary statement

A primary statement can be a variable declaration, a variable assignment/initialisation, or a function call, followed by a semicolon.

8.2. If Statement

If statements are can be of the following forms:

(i) A if-else statement

```
if (expression_to_evaluate){
        statement_to_execute
}
else{
        statement_to_execute
}
```

(ii) A if-elseif-else statement

```
if(expression_to_evaluate){
        statement_to_execute
}
elseif(expression_to_evaluate){
        statement_to_execute
}
else{
        statement_to_execute
}
```

NOTE:

- The “else” clause is compulsory.
- The curly brackets (“{“ and “}”) are mandatory to specify the scope of each clause.

8.3. Repeat Statement

The repeat statement causes a set of statements to execute a fixed number of times.

```
repeat N {
        statement_to_execute_1;
        statement_to_execute_2;
        …
}
```

NOTE:

- ‘N’ is the integer which denotes the number of iterations.
- This loop only terminates after the block of statements have been executed N times.

8.4. While Statement

While loops allow a set of statements to be executed any number of times. The while loops in FF3D are of the following form:

```
while(expression_to_evaluate){
        statement_to_execute_1;
        statement_to_execute_2;
        …
}
```

NOTE:

- The loop runs as long as “expression_to_evaluate” returns true.
- The “expression_to_evaluate” is evaluated at the beginning of every iteration.

8.5. Return Statement

The last statement inside a function definition, which causes the function to return a value. It is of the form:

```
return <type>;
```

9. Scope

The FF3D compiler will accept a single file as source code. All variable declarations outside of a function are globally accessible (for evaluation or assignment) from anywhere in the program. Variables declared within functions are (locally) accessible within the function only. Variables declared within a flow/conditional control block (e.g., while/repeat loop, if/elseif/else, etc.) are only accessible within that block.