Simplex Language Guide

Version 3.0
October 28, 2021

1. Introduction

What is Simplex?

Simplex is the term used for Fracton's compiler that uses regular double precision variables. The simplex language takes a lot more lines to do the same function as the triplex compiler. The simplex language offers more control over every aspect of making a model. The math for simplex is complete and is very similar to the familiar C language.

What would an equation in Simplex look like?

Writing an equation for a 3D model in simplex would probably not look familiar to someone comfortable with 2D fractals. For example a 3D Mandelbulb looks like:

```
// Mandelbulb
fx = 0;
fy = 0;
fz = 0;
power = 8;
@beginloop;
r = sqrt(fx * fx + fy * fy + fz * fz);

// atan2(fy,fx)
if (fx > 0) { phi = atan(fy / fx); }
elseif (fx < 0 && fy >= 0) { phi = atan(fy / fx) + pi; }
elseif (fx < 0 && fy < 0) { phi = atan(fy / fx) - pi; }
elseif (fx == 0 && fy >= 0) { phi = pi / 2; }
elseif (fx == 0 && fy < 0) { phi = -pi / 2; }

// arccos(z/r)
if (r > 0) { theta = acos(fz / r); }
else { theta = 0; }

rpower = r^power;
phipower = phi * power;
thetapower = theta * power;

fx = rpower * sin(thetapower) * cos(phipower) + tpx;
fy = rpower * sin(thetapower) * sin(phipower) + tpy;
fz = rpower * cos(thetapower) + tpz;

test = (fx*fx + fy*fy + fz*fz) < 4;
```

The built in variables (tpx,tpy,tpx) near the end are the point in 3D where the fractal equation is calculated.

2. Predefined Variables

Some variables are pre-defined and are used as an interface between the language and the application. The following variables are pre-defined: fx, fy, fz, pi, tpx, tpy, tpz, p1x, p1y, p1z, through p6x, p6y, p6z, maxit, n, test.

fx, fy, fz:

fx, fy, fz are the main triplex variables and are usually iterated. For a Julia type expression, fx, fy, fz are initialized to the value tpx, tpy, tpz respectively. For a Mandelbrot type expression, fx, fy, fz are initialized to zero. The magnitude of fx, fy, fz is tested to see if it is larger than bailout. If so, the iteration is stopped and f is declared to have escaped. The number of iterations required before fx, fy, fz escapes is used to determine the outer surface of the 3D fractal. The value of fx, fy, fz after the last iteration is returned to Fracton but is not used at this time.

pi:

pi is the constant (3.14…,0,0).

e:

e is the constant (2.718…,0,0).

tpx, tpy, tpz:

tpx, tpy, tpz is the point in 3D fractal space where the formula will be evaluated. The formula should make no assumption about the limits of tpx, tpy, tpz. The value may be in any direction and may not be visible on any view.

p1x, p1y, p1z, through p6x, p6y, p6z:

p1x, p1y, p1z, through p6x, p6y, p6z are values supplied to the formula from the settings view. The values are related to tpx, tpy, tpz and can be used to offset, rotate, and scale tpx, tpy, tpz without any other arithmetic. p1x, p1y, p1z, through p6x, p6y, p6z are also used for animation.

p1x, p1y, p1z, through p6x, p6y, p6z = scale(rotate(tpx, tpy, tpz - position))

maxit:

maxit is the maximum number of iterations from the settings view.

n:

n is the iteration number of the loop. n is zero based so the first time through the loop n = 0.

test:

test is the escape test result for the fractal. If test is false, the fractal is declared escaped and the loop stops. If test is true the loop continues as long as the maximum number of iterations has not been exceeded. Calculate the magnitude squared of fx, fy, fz to determine the value of test. The value of test is returned to Fracton after each iteration.

Example: test = (fx*fx + fy*fy + fz*fz) < 4;

3. User Defined Variables

To add a user defined variable, simply use it in an equation. You should set its value before you read its value.

4. Initialization of Variables

Statements before the @beginloop; compiler directive are only executed on the first iteration of the loop.

Example:
```
fx =0; // Initializes fx to 0
@beginloop; // Start of the loop
```

5. Constants

Constants are double precision. Exponents are IEEE compliant.

Example:

```
a = -3.14e-10;
b = 4E10;
c = 3;
```

6. Command Summary

Arithmetic: + - * / ^ %

Assignment: =

Comment: // /* */

Compare: < > <= >= == != && ||

Conditional Expression: ?:

Non-Trigonometric Functions: abs ceil floor mag magsqr pow sqrt

Trigonometric Functions: acos acosh asin asinh atan atan2 atanh cos cosh exp log log10 sin sinh tan tanh

Removed: flip imag real conj cosxx

7. Arithmetic

In all of the descriptions below a and b are double precision numbers.

+

Adds two triplex numbers a and b and returns a triplex number.
a + b

Example: c = a + b;

-

Subtracts two numbers a and b and returns a number.

a - b

Example: c = a - b;

Unary minus negates a variable or parenthesized expression.

Example: b = -a; b = -(a * 2);

*

Multiplies two numbers a and b and returns a number.

a * b

Example: c = a * b;

/

Divides two numbers a and b and returns a number.

a / b

Example: c = a / b;

^

Raises the number a to the power of b and returns a number. The operator ^ is right associative so a ^ b ^ c = a ^ (b ^ c). The c language uses pow(a , b) instead of the ^ operator.

a ^ b

Example: c = a ^ b;

%

Performs the scalar modulus operation on a number. Returns 0 for b <= 0.

a % b
if (b > 0) c = a % b
else c = 0

Example c = a % b;

8. Assignment

=

The equal copies the value of an expression to a variable. Every expression must start with a variable followed by = .

a = b

Example: a = b;

9. Comment

//

Any time // is encountered the rest of the line is ignored by the compiler.

Examples:

```
// This line is a comment
a = b + c; // Everything to the right of // is a comment
```

/*

The start of a block comment.

*/

The end of a block comment.

Example:

```
/* All of these three lines
a = b * c;
are comments */
```

10. Compare

<

a < b returns 1 if a is less than b. Otherwise, it returns a 0.

Example c = a < b;

>

a > b returns 1 if a is greater than than b. Otherwise, it returns a 0.

Example c = a > b;

<=

a <= b returns 1 if a is less than or equal to b. Otherwise, it returns a 0.

Example c = a <= b;

>=

a >= b returns 1 if a is greater than or equal to b. Otherwise, it returns a 0.

Example c = a >= b;

==

a == b returns 1 if a is equal to b. Otherwise, it returns a 0.

Example c = a == b;

!=

a != b returns 1 if a is not equal to b. Otherwise, it returns a 0.

Example c = a != b;

&&

a && b returns 1 if both a and b are greater than 0. Otherwise, it returns a 0.

Example c = a > 1 && a < 0;

||

a || b returns 1 if either a or b are greater than 0. Otherwise, it returns a 0.

Example c = a > 1 || a < 0;

11. Conditional Expression:

?:

The conditional expression a ? b : c returns b if a > 0 otherwise it returns c. This differs from the C language in that all of the expressions are evaluated whether they are used or not.

Example: f = f >= 0 ? (f - 1) : (f + 1);

12. Non-Trigonometric Functions

abs

Returns the absolute value of a number. For the modulus of the x component only see % in the arithmetic section.

abs(a)

Example: b = abs(a);

ceil

Returns the ceiling of a number.

ceil(a)

Example: b = ceil(a);

floor

Returns the floor of a number.

floor(a)

Example: b = floor(a);

mag

Returns the argument unchanged. This function is included for completeness.

mag(a)

Example: b = mag(a);

magsqr

Returns the square of a number. This function is included for completeness.

magsqr(a)
b = a * a

Example: b = magsqr(a);

pow

Raises the number a to the power of b and returns a number. This is the same as the ^ operation above.

pow(a , b)
// See ^

Example: c = pow(a , b);

round

Returns the round of the individual components of a triplex number.

round(a)
b.x = round(a.x)
b.y = round(a.y)
b.z = round(a.z)

Example: b = round(a);

trunc

Returns the trunc of a number.

trunc(a)

Example: b = trunc(a);

sqrt

Returns the square root of a number.

sqrt(a)

Example b = sqrt(a);

13. Trigonometric Functions

Trigonometric functions work with double precision numbers. The trigonometric functions are: acos acosh asin asinh atan atan2 atanh cos cosh exp log log10 sin sinh tan tanh.

14. Precedence:

( )

Changes the precedence of an expresssion.

Example: a = 2 * (1 + 3); // a is 8

15. Control Flow

if, if else, and else are used to allow logical control of program execution.

if

The if operator must be followed by an expression and one or more statements enclosed by brackets. If the expression is not equal to zero the expression is true and the statements are executed. Unlike C, the brackets are required around statements.

```
if(expression) {one or more statements}
```

else if

The else if operator is executed only if the previous if or else if operators were false. The else if operator must be followed by an expression and one or more statements enclosed by brackets. If the expression is not equal to zero the expression is true and the statements are executed.

```
else if(expression) {one or more statements}
```

else

The else operator is executed only if the previous if or else if operators were false. The else operator must be followed by one or more statements enclosed by brackets.

```
else {one or more statements}
```

Example

```
if( a < 5) {
a = 0;
b = b + 1;
}
else if( a > 10){
a = 0;
}
else {
a = 10;
b = 0;
}
```