PIXL

Language Reference Manual

Maxwell Hu (mh3289)
Justin Borczuk (jnb2135)
Marco Starger (mes2312)
Shiv Sakhuja (ss4757)
Jacob Gold (jeg2213)

6. Statements

    6.1. Block
    6.2. Conditional Statement
    6.3. For Loops
    6.4. Enhanced for loop
    6.5. While Loops
    6.6. Return Statement

1 Introduction

PIXL is a 2D matrix manipulation language that uses a pixel as a primitive type in order to more easily process images and apply filters using image signals. Pixel operations include adding, subtracting and negating individual pixels; pixel arrays are able to use the same operators as pixels, where the operation will be applied to each corresponding pixel pair in the matrices. PIXL also contains standard library functions that support transformations such as masking, intersecting, blurring, sharpening, etc. PIXL makes it easier to apply algorithms, transformations, and combinations to images.

In order to easily work with image files, PIXL has file I/O capabilities. More importantly, an image file can be easily loaded into a pixel matrix in order to process the image signal. A pixel matrix can also easily be written into a file for export.

For easy manipulation of pixel matrices, PIXL uses enhanced loops that can easily loop through all the pixels of an image in order to apply a common function to each pixel, and can do this for multiple images as well for functions that manipulate images based on another image's pixel values.

2 Lexical Conventions

2.1 Comments

//	Single Line Comment
/* … */	Multi Line Comment

2.2 Identifiers

Identifiers must have an upper or lowercase letter as a first character, which can be followed by any assortment of uppercase or lowercase letters, underscores, and numbers.

2.3 Keywords

Keywords	Description
if	Enters statement if condition is met
else if	Paired with if; evaluated only when if/else if statements above in the same block haven't been satisfied
else	Default if no other conditions are satisfied in the block
for	Repeats until a condition is satisfied
while	Repeats until a condition is satisfied
return	Returns value from function
break	Exits loop containing statement and continues at the first statement outside the loop
main	The main function is the starting point of a program
true	Boolean literal for 1
false	Boolean literal for 0
void	Keyword used for functions

2.4 Type Specifiers

boolean	Boolean value
char	ASCII character
int	Integer
float	Float
pixel	One pixel's color and transparency information

String	Encapsulated character array representation
File	Encapsulated file pointer

2.5 Punctuators

A punctuator is a symbol that does not specify a specific operation to be performed but rather has syntactic value to the compiler in order to format code

Symbol	**Definition**
;	Statement terminator
{ }	Block of statements

2.6 Operators

2.6.1 Arithmetic/Logical Operators

Arithmetic/Logical Operator	Description
=	Assignment Operator
+	Additive Operator
-	Subtraction Operator
*	Multiplication Operator
/	Division Operator
^	Exponentiation Operator
==	Returns 1 if values are equal, 0 otherwise
+=	Adds value on the left to the value on the right and stores in the left variable

!=	Returns 1 if values are not equal, 0 otherwise
>	Greater than operator
<	Less than operator
>=	Greater than or equal to operator
<=	Less than or equal to operator
&&	Logical AND operator
||	Logical OR operator
!	Logical NOT operator

2.6.2 Pixel Operators

Operator	**Description**	**Example**
+	+ works the same way as Java. However, when adding two pixels together you add the corresponding r,g,b values in each pixel together to create a new tuple. If the sum of two corresponding values exceeds 255, then 255 is used as the sum value. + can also be used as an operand between two matrices of the same dimensions: the + operator is applied to each corresponding pixel pair and adds them using the pixel + operator.	pixel x1 = (100,100,200,0.5) pixel x2 = (50,50,100,0.5) pixel x3 = x1 + x2 // x3: (150,150,255,0.5)

-	Works the same way as addition, except you subtract the two tuples. Absolute value is used to avoid negative integers. - can also be used as an operation on matrices. Like addition, each corresponding pixel pair is subtracted.	pixel x1 = (100,100,200,0.5);
		pixel x2 = (50,50,100,0.5);
		
		pixel x3 = x1 - x2;
		
		// x3: (50,50,100,0.5)

=	The equals assignment operator sets	pixel x = (100,50,100,0.5);

	the value of the left variable equal to the value of the right side.	pixel y = x;
		
		// y: (100,50,100,0.5)

==	The equality check for pixels returns True if the pixels have the same rgba values, and False if any of the rgba values differ.	pixel x = (100,100,100,0.5);
		pixel y = (100,100,100,0.4);
		boolean b = x == y;
		
		// b: False

&&	Logical AND is applied to two pixels in the following way:
	
	1)If the pixels are the same, return the pixel.
	
	2) If the pixels are different, return (0,0,0,0).
	
	The Logical AND operator can also be applied to two matrices. In this case, it takes corresponding pixel pairs in two matrices of the same size and applies the two rules above to output a third matrix.
		pixel x = (100,100,100,0.5);
		pixel y = (100,100,100,0.5);
		pixel z = x && y;
		
		// z: (100,100,100,0.5);

2.6.3 Operator Precedence

The following table displays operator precedence from highest to lowest.

Operator Symbol	Description
!	Logical NOT operator
* /	Multiply, divide
+ -	Add, subtract
> < >= <=	Greater than, less than, greater than or equal to, less than or equal to

== !=	Equality, inequality
&& ||	Logical AND, logical OR
=	Assignment operator

3. Syntax Notation

3.1 - Function Declarations

Function headers are made in the same way as java. The return type is mentioned first, or void is used if there is no return type. Then, the function name is written, followed by 0 or more parameters.

type function_name(arguments){...}

3.2 - Function Calls

A function can be called by including another file that contains the function, and then calling the function by using the following syntax:

functionName(parameter1, …)

3.3 - Variable Declarations

type variable = new type();

3.4 - Postfix/Prefix Expressions

i++; // increments i by 1, returns i
++i; // returns i, increments i by 1
i+=3; // adds 3 to i, returns i
i--; // decrements i by 1, returns i
--i; //returns i, decrements i by 1
i-=3; // subtracts 3 from i, returns i

3.5 - Matrix Declaration

A matrix is declared by using double brackets, and the size of the matrix must be declared. The first integer is the number of rows, and the second integer is the number of columns.
pixel[ ][ ] pMatrix = new pixel[5][6]; // This creates a matrix of 5 rows and 6 columns

3.6 - Matrix Access

A specific value in a matrix can be accessed by integers i and j, where i indicates the $i+1^{th}$ row and j indicates the $j+1^{th}$ column (index starts at 0).
pMatrix[i][j] // Accesses the value at row i+1 and column j+1

int[ ][ ] iMatrix = new int[3][4];

// fill iMatrix

[1] [2] [3] [4]
[5] [6] [7] [8]
[9] [10] [11] [12]

iMatrix[2][1] // gets the value in the 3rd row and 2nd column, which is 10

4. Standard Library Functions

4.1 - Pixel Manipulations

Built-in library functions will allow user to apply several different filters to pixel matrices. These filters include:

-Gray Scale
-Tint
-Black and White
-Sharpen
-Invert

4.2 - File I/O

Standard Library Functions will be available for File I/O to create pixel matrices from ppm files for manipulation. The syntax for File I/O is as follows:

File f1 = new File("image1.ppm"); //when declared will set instance variables for
//dimensions of the image

pixel [][] p1 = f1.load() //Because f1 has dimensions as instance variables, they will be
//returned and set as dimenstions for p1.

5. Semantics

5.1 Scope

Scope rules are similar to Java.

Variables declared within a block – for example, in a loop, or in a function, are *local* to the code block.

Example:

```
if (0 < 1) {
    int a=1;
}
print(a);

>> ERROR
```

For example, the program above will throw an error because the variable *a* is only available within the code block.

Variables declared outside every block are *global*, i.e – accessible throughout the program.

Example:

```
int a = 0;
if (0 < 1) {
    a=1;
}
print(a);
```

```
>> 1
```

5.2 Recursion

Recursion occurs when a function is called within the function itself.

6. Statements

6.1 Blocks

Code blocks are 1 or more lines of code which are surrounded by curly braces. They are most commonly used in conditionals, loops and methods.

Note: Code blocks have local scope, so variables declared within a code block are only available within that code block.

6.2 Conditional Statement

Conditional statements include **if**, **else if** and **else** statements, which work as they do in Java. Conditional statements must also contain a code block using braces to be executed if the condition is met.

```
if (condition) { // code to execute }
```

Example of conditional statement:

```
if (foo == bar) {
      print(“bar”);
}
else if (foo == 23) {
      print(“yay!”);
}
else {
      print(“oops”);
}
```

6.3 For Loops

For loops can consist of two types.

Standard for-loop: This is a standard for loop. It requires an initialization statement, a conditional statement and a code block to execute.

Example:

```
for (int i=0; i<5; i++) {
      // code to execute
}
```

6.4 Enhanced for loop

This is a special kind of for loop designed to iterate through rows of pixels, specifically for the purpose of image manipulation.

```
for (pixel p : matrixName) {
      // code to execute
}
```

6.5 While Loops

This kind of loop evaluates a condition, and if it is true, executes the code block following the condition. At the end of the code block, it returns to the line with the condition, and repeats the process until the condition is no longer true.

Example:

```
while (foo < bar) {
      // code to execute
}
```

6.6 Return Statement

Return statements are used in methods to return a value. A return statement ends the method. The lines following a return statement are not executed.

All methods with a return type other than void must have a return statement. The return type must match the method signature.

Example:

```
int getRedValue() {
      // some code to execute
```

```
        return red;
    }
```