PIXL

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

Description

Point processing is the enhancement or modification of an image through individual pixel manipulation. PIXL is a Java-like language that utilizes the pixel as a primitive type in order to more easily process images and apply various filters to image signals. Individual pixels can be added to one another, subtracted from one another, and negated, while pixel arrays can also be added, subtracted, and negated, as well as masked, intersected, blurred, sharpened, and antialiased, among other transformations.

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 custom transformations that are not defined in the language, PIXL includes special 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.

Data Types

Data Type	**Documentation**	**Declaration**
int	Integer value	int x = 10;
float	Floating point value	float x = 10.0;
File	A file	File f = “image.ppm”;
String	A string	String s = “hello”;
pixel	Length 4 tuple (r,g,b,a)	pixel x = (255,255,255,0.5);

boolean	True/False Value	boolean x = True;
array	List of data that is declared in a way similar to java; primarily for pixels	pixel[ ] x = new pixel[5];
matrix	2d array	pixel[ ][ ] x = new pixel[5][5];

Arithmetic and Logical Operators

+, -, =, *, /, %, ++, --, ==, !=, &&, ||, <, >, <=, >=, !
Comments begin with //
Multi-line comments /* … */
Semicolons (;) end a statement

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);

Keywords

Keywords	**Description**
if	if (cond) {}
else	else {}
else if	else if (cond) {}
for()	for(int i=0; i<4; i++){}
for(pixel x, int i,j : pmatrix1) {}	Enhanced for loop that makes traversing pixel arrays/matrices easy. The documentation on the left provides a way to loop through a pixel matrix left to right (when the end of a row is reached the loop moves one row down and repeats the process). The loop provides an i and j integer as indices.

Sample Code

```
// Sets an image to grayscale
File f1 = “image1.ppm”;

pixel[][] p1 = f1.load();

for (pixel x, int i,j : p1)
{
	int avg = (x.getRed() + x.getGreen() + x.getBlue())/3;
	p1[i][j] = (avg, avg, avg, x.getAlpha());
}

// Keeps only the common pixels of two images (AND) at the matrix level
File f1 = “image1.ppm”;
File f2 = “image2.ppm”;

pixel[][] p1 = f1.load();
pixel[][] p2 = f2.load();
pixel[][] out = p1 && p2;

// Keeps only the common pixels of two images (AND) at an individual pixel level
File f1 = “image1.ppm”;
File f2 = “image2.ppm”;

pixel[][] p1 = f1.load();
pixel[][] p2 = f2.load();
pixel[][] p3 = p2.zero();

for (pixel x,y, int i,j : p1,p2)
{
	p3[i][j] = x && y;
}

// Negates an image
File f1 = “image1.ppm”;

pixel[][] p1 = f1.load();
```

```
for (pixel x, int i,j : p1)
{
       p1[i][j] = (255-x.getRed(),255-x.getGreen(),255-x.getRed(), x.getAlpha());
}
```