IPL
Image Processing Language

Youngjin Yoon Wookyun Kho Jianning Yue
{yy2223, wk2153, jy2246}@columbia.edu

December 19, 2006

Contents

1 Introduction **1**
- 1.1 Background and Goal . . . 1
  - 1.1.1 Ease of use . . . 1
  - 1.1.2 No experience needed . . . 1
  - 1.1.3 Productivity . . . 2
- 1.2 Features of IPL . . . 2

2 Tutorial **3**
- 2.1 Variable Defination and Array Manipulation . . . 3
- 2.2 Image Manipulation . . . 4
  - 2.2.1 Scaling . . . 4
  - 2.2.2 Rotating . . . 4
  - 2.2.3 Fading . . . 5
  - 2.2.4 Positioning . . . 5
  - 2.2.5 Grouping . . . 6
- 2.3 Show Picture . . . 8
- 2.4 Animate It! . . . 8
- 2.5 Using functions . . . 8
- 2.6 Practical Example . . . 9

3 Language Reference Manual **11**
- 3.1 Lexical Conventions . . . 11
  - 3.1.1 Comments . . . 11
  - 3.1.2 Identifiers(Names) . . . 11
  - 3.1.3 Keywords . . . 11
  - 3.1.4 Constants . . . 12
- 3.2 What's in a name . . . 12
- 3.3 Objects and lvalues . . . 12
- 3.4 Conversions . . . 12
- 3.5 Expressions . . . 12
  - 3.5.1 Expressions . . . 12
- 3.6 Boolean Expression . . . 14
  - 3.6.1 Unary Operator . . . 14
  - 3.6.2 Image Operators . . . 14
  - 3.6.3 Arithmetic Operators . . . 15
  - 3.6.4 Assignment Operators . . . 16
  - 3.6.5 Relational and Equality operators . . . 16
  - 3.6.6 Logical Operators . . . 16
- 3.7 Declarations . . . 17

3.7.1 Type Specifiers 17
3.7.2 Declarators 17
3.8 Statements 17
3.8.1 Normal Statements 18
3.8.2 Defunc statement 19
3.8.3 Miscellaneous 19
3.9 Scope rules 19

4 Project Plan **21**
4.1 Team Responsibilities 21
4.2 Project Timeline 21
4.3 Software Project Environment 21
4.3.1 ANTLR 21
4.3.2 Operating Systems 22
4.3.3 Java 1.4.2 22
4.3.4 CVS 22
4.4 Project Log 22

5 Architecture Design **23**

6 Testing Plan **25**
6.1 Overview 25
6.2 Phase I 25
6.3 Phase II 25
6.4 Final testing Programs and Integration 25

7 Lessons Learned **28**

A Language Syntax **29**
A.1 Lexical rules 29
A.2 Syntactic rules 29

B Code Listing **32**
B.1 Parser 32
B.1.1 src/grammar.g 32
B.1.2 src/walker.g 36
B.2 JAVA 2D Animation Module 46
B.2.1 src/Affines.java 46
B.2.2 src/ImageINFO.java 49
B.2.3 src/IPLImageDisplay.java 51
B.3 IPLDataType and Main Module 64
B.3.1 src/IPL.java 64
B.3.2 src/IPLArray.java 64
B.3.3 src/IPLBool.java 68
B.3.4 src/IPLBoolArray.java 69
B.3.5 src/IPLCoord.java 69
B.3.6 src/IPLCoordArray.java 70
B.3.7 src/IPLDataType.java 71
B.3.8 src/IPLFunc.java 72
B.3.9 src/IPLImage.java 73
B.3.10 src/IPLImageArray.java 75
B.3.11 src/IPLImageHistory.java 77

B.3.12 src/IPLNumber.java 79
B.3.13 src/IPLNumberArray.java 81
B.3.14 src/IPLRange.java 81
B.3.15 src/IPLSymbolTable.java 82
B.4 JAVA 2D Applet Module 87
B.4.1 src/IPLApplet.java 87
B.5 Testing cases and samples 93
B.5.1 testingcase/animate.ipl 93
B.5.2 testingcase/burger.ipl 93
B.5.3 testingcase/coordinate.ipl 93
B.5.4 testingcase/defination1.ipl 93
B.5.5 testingcase/defination2.ipl 93
B.5.6 testingcase/display.ipl 94
B.5.7 testingcase/fading.ipl 94
B.5.8 testingcase/function.ipl 94
B.5.9 testingcase/functiondefinition.ipl 94
B.5.10 testingcase/imageassignment.ipl 94
B.5.11 testingcase/imageinserting.ipl 95
B.5.12 testingcase/imagelistadding2.ipl 95
B.5.13 testingcase/imagelistadding.ipl 95
B.5.14 testingcase/joining2.ipl 95
B.5.15 testingcase/joining.ipl 95
B.5.16 testingcase/positioning.ipl 95
B.5.17 testingcase/rotating.ipl 96
B.5.18 testingcase/scaling.ipl 96
B.5.19 testingcase/strawberry.ipl 96
B.5.20 testingcase/animate2.ipl 96
B.5.21 testingcase/while.ipl 97
B.5.22 testingcase/scope1.ipl 97
B.5.23 testingcase/scope2.ipl 97
B.5.24 testingcase/demo1.ipl 97
B.5.25 testingcase/demo2.ipl 98
B.5.26 testingcase/demo3.ipl 98
B.5.27 testingcase/demo4.ipl 99

Chapter 1

Introduction

1.1 Background and Goal

The main goal of our programming language is to implement and achieve a simple and ease-of-use programming language for animation production. We want our language to enable users to easily produce 2D animations and simple image concatenation. By simply using the functions and operator provided by IPL such as image displaying, rotation, moving, and modifying transparency, programmer can make useful animations easily.

Existing language is not an easy way for rookies to implement an animation because programmers need to know every details of information for provided library. Moreover, sometimes programmer should know how to apply the transformation using arrays, which is related with linear algebra.

With this background, we want to design a language that does not needs this preparation phase for programmer. By using simple operator and function, the programmer can handle fundamental operation and animation for image, such as rotation, scaling and transformation and frame animation.

Thus the goal for our language are quite explicit:

1.1.1 Ease of use

With animate() and basic operator for image type, the programmer easily implement what she want to do. For example, without any further thinking, one can create a scenario that an apple moves from location A to location B, while doing some size-scaling and rotating during the movement with just one single line of program by using animate(). For a more complicated animation, the user simply needs to describe the behavior of several such ones then IPL will combine them altogether. Moreover, the programmer can easily define a function or use some control statements such as while and if because we applied our function definition and control flow similar to C.

1.1.2 No experience needed

Somebody may have known that with Java 2D API programmers can make animations and this API provides very specific handling mechanism. However, using the API requires a lot of programming experiances and background knowledge such as linear algebra. Therefore, it is not such an easy way to use.

1.1.3 Productivity

By using IPL, the programmer easily generate the Java Applet using export(). Because those exported image file is very tiny compared with Animated GIF file, we can easily upload our animated picure to the Internet by using our provided applet code.

1.2 Features of IPL

IPL is designed to facilitate the animation production for programmers who have little professional knowledge in these fields. It is a straightforward, interesting, strong, robust and productive language.

Straightforward

IPL is a programming language that builds animations in a straightforward way. We build our language to achieve the concept that "what you want to express is actually what you want". For example, "Move the apple from the left-down corner in 4 seconds while rotating 180 degrees and enlarge by 1.2 times" could be expressed in a sentence, which is as same sequence as natural langue, in IPL. Even if you do not have any experiences about language, you can easily figure out by tracing what you want.

Interesting

We try to eliminate the tedious elements of common programming language and complex syntax constraints for a high flexibility. Besides making the language easy to use, we try to make IPL always arouse programmers interests in animation producing. The production is almost nothing about how the language works but the programmer′s idea of the animation he want to show. With this feature, user no longer needs to care about the language itself but to focus on what kind of animation he wants to design.

Strong

IPL includes the basic image operator for images. By using scale, rotate, set, transparency, and concat operator, the programmer can easily build up their target image. Moreover, IPL provides strong animate() to animate the image object. By using those expression with animate(), the programmer can build any animation what he want. Also, IPL provides function definition. By using defunc keyword, the programmer define their own function to handle the image properly.

Robust

IPL provides good error checking for each operand types for each operator. By strongly prohibiting type casting, it also blocks the possibility of error.Moreover, the programmer can fix their problem by provided column and line printed in error message.

Productive

IPL is a productive language. Programmers can make animations with few prepared materials, And those animated image can be exported as a JAVA applet. If he want to show an apple falling down from a tree on the web, he only needs pictures of a tree and a apple. What needs to be done is to write two separate lines to describe this with export().

Chapter 2

Tutorial

2.1 Variable Defination and Array Manipulation

IPL has a very flexible array manipulation style which dedicates to facilitate image processing. This part is somewhat verbose but really helpful, especially when you want to manipulate a data structure. Image type variable and array store not only static pictures, but also a moving images.

We can define a variable like below:

```
image imgA, imgB;

imgA = "sshield.jpg";
```

The program defines two image-typed variables named by imgA and imgB, and then make imgA store a static picture in the path ./sshield.jpg. (We will see what these composing pictures are manipulated.)

We can also define an array by doing like below:

```
image[] imgListA;

imgListA = {"pear_red.jpg","apple.jpg","orange.jpg"};
```

This example defines an array named imgListA and then makes it store three different pictures. The array is not specified with size (how many elements it will contain), and this is one of the key feature of IPL that makes users easily manipulate dynamic array structure. We can also replace the first element with:

```
imgListA[0] = "strawberry.jpg";
```

IPL provides truly flexible array manipulation functionality with only few operators. See the following example:

```
imgListA[1~3] = { "apple.jpg","pear_red.jpg","orange.jpg" };

imgListA[2+] ={"strawberry","apple.jpg"};
```

The first statement with s̃ymbol, which means ”from to”, replaces the second, third and the fourth elements of imgD with three new pictures on the right side. By writing down the code like the one in the second statement which has a + symbol (this means ”insert after), you can insert two pictures right after the third element, and you can also shift the following elements to the rightside.

The following codes will pop off the last picture and produces the same result as above.

```
imgListA[1~3]={"apple.jpg","pearred.jpg","orange.jpg","apple.jpg" };
```

Note that if you use + and ˜ together, you can get the different functionality:

```
imgListA[2~4+] = {"pear_red.jpg","apple2.gif","orange.jpg"};
```

This code will shift the elements which are in between the second and the fourth in imgListA, and insert following four pictures to the space.

2.2 Image Manipulation

2.2.1 Scaling

You can easily change the size of a picture by using ôperator. Here is an example:
Example 2.2.1: Scaling

```
image imgA = "sshield.jpg"‘(400,300);

display(imgA^2);

display(imgA);
```

The result is shown in Figure 2.1.

2.2.2 Rotating

Also, operator can make you rotate an image. Simple example is below:
Example 2.2.2: Rotating

```
image imgA = "sshield.jpg"‘(400,300);

display(imgA);

display(imgA@90);
```

The result is shown in Figure 2.2.

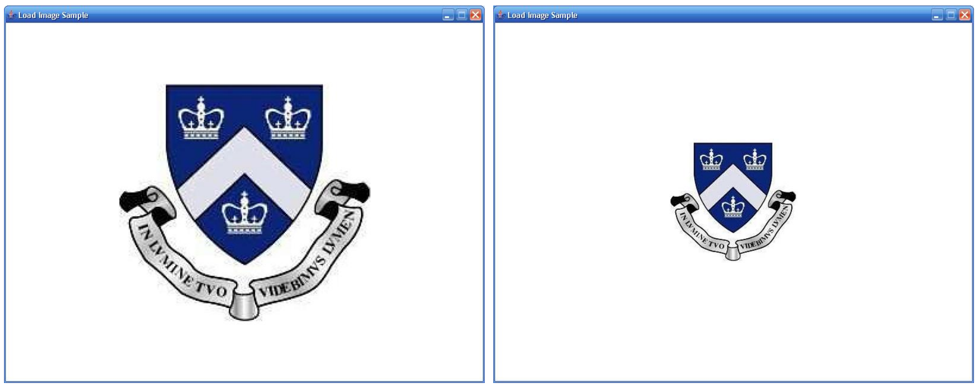

Figure 2.1: Scaling

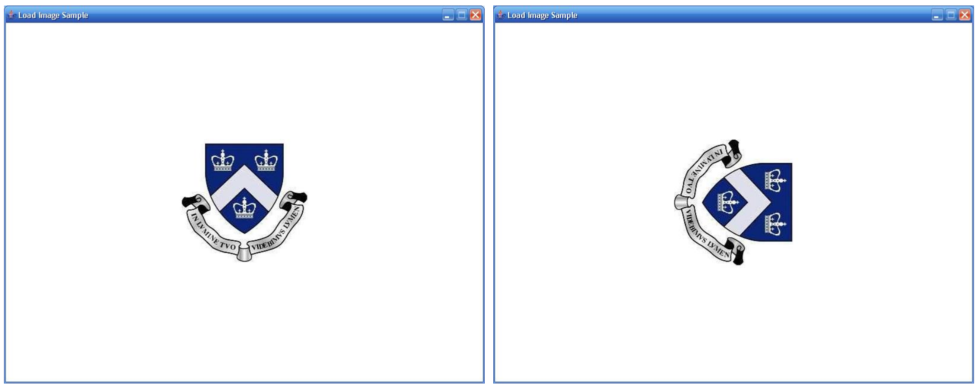

Figure 2.2: Rotating

2.2.3 Fading

Sometimes, we might need this kind of functionality to manipulate a picture that is disappeared gradually or showed up slowly. In IPL, you can easily achieve this functionality by using : operator. In fact, this operator can control transparency.

Example 2.2.3: Fading

```
image imgA = "sshield.jpg"‘(400,300);

imgA = animate(imgA:-300, 5);

display(imgA:-40);
```

The result is shown in Figure 2.3.

2.2.4 Positioning

Positioning is an important function that is to specify where a picture needs to be showed up or where an object will start to move and stop. You can do this by using ‘ operator. Note that the (0,0) is the left-up corner of the screen.

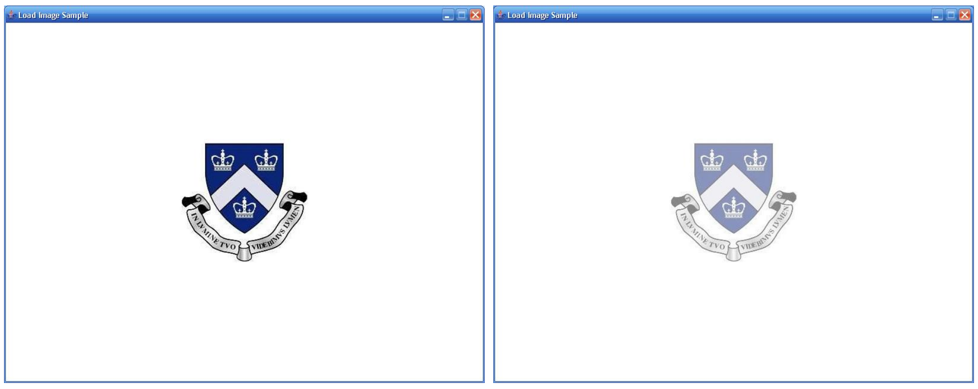

Figure 2.3: Fading

Example 2.2.4: Positioning

```
image imgA = "sshield.jpg"‘(300,200);

display(imgA);

display(imgA‘(700,400));
```

The result is shown in Figure 2.4.

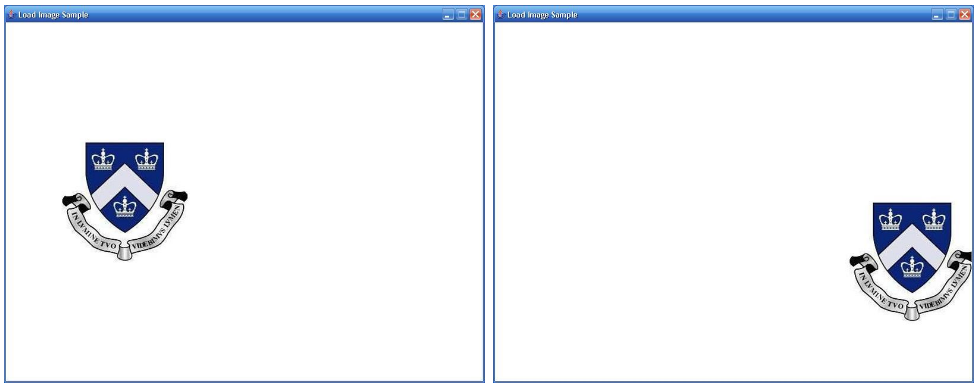

Figure 2.4: Positioning

2.2.5 Grouping

Let’s look back at the example we started at the beginning:

Example 2.2.1:

```
image imgA = "sshield.jpg",imgB,imgC,imgD;
```

```
coord coA,coB,coC,coD;
coA = (105,105);
coB = (-105,105);
coC = (-105,-105);
coD =(105,-105);
imgA = imgA‘(500+xof(coC),300+yof(coC));
imgB = imgA‘(500+xof(coD),300+yof(coD))@90;
imgC = imgA‘(500+xof(coA),300+yof(coA))@180;
imgD = imgA‘(500+xof(coB),300+yof(coB))@270;

display(imgA $ imgB $ imgC $ imgD);
```

The @ operator followed by a number returns a picture named imgA that has been rotated clockwise. $ operator will place the left operand right over the right operand horizontally. Finally the right part of the assign will return a hybrid of transformed pictures and assign it to imgA. The picture is shown in Figure 2.1.

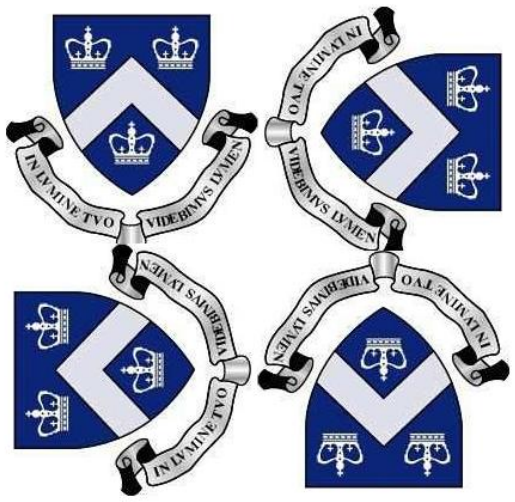

Figure 2.5: Image Rotating and Combining

See another example:

```
imgA = (imgA$imgA@90 )$(imgA@270$imgA@180);

imgB[3] = imgA‘(0,1):30 @ 120 ;
```

What does imgB[3] will store after this? . followed by a coordinate sets the imgA to the position (0,1), then set the picture transparency to 30 percent (we can see a half-transparent picture after this), and finally rotate it by 120 degree clockwise.

2.3 Show Picture

We have declared some pictures in the code, imgA and imgB. To show these pictures, the only thing that you have to do is making only 4 short lines of code.

```
image imgA, imgB;

imgA = "sshield.jpg"‘(400,200);

imgB=(imgA$imgA@90)$(imgA@270$imgA@180);

display(imgB);
```

We have seen the usage of ”display” in some examples. ”display” is a built-in function in IPL that will display image or animation.

2.4 Animate It!

We are now anxious about how to move images and make animations. Here is the simple animation making code:

```
image[] imgListA , imgListB;

image imgA;

imgListA[0+] = "strawberry.jpg"^2;

imgListA[1+] = "apple.jpg";

imgListA[2~5+]
     ={"pear_red.jpg","tomato.jpg","melon.jpg","orange.jpg"};

imgA = animate(imgListA‘(600,400),50); display(imgA);
```

2.5 Using functions

In order to get a higher flexibility, programmer can define functions. Syntax for function definition can be found in language reference manual section. Here is an example program that defines a new function, ”showall”.

```
image imgA = "pear_red.jpg"; image imgB = "apple.jpg";

defunc showall ( image imgS, image imgQ ) image imgR {
    bool b = true;
    number p=100;
    number i=0;
    image[] imgC;
    while (i<5)
```

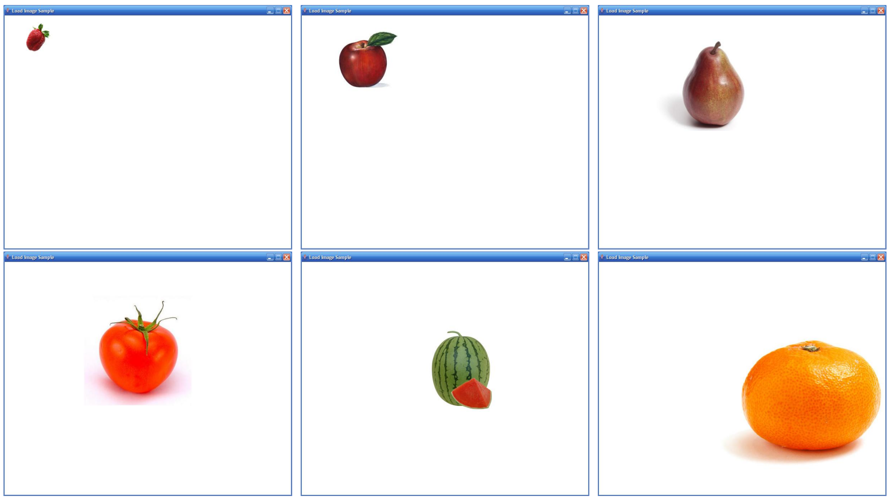

Figure 2.6: Animating

```
        {
            if(i%2 ==1) {imgC[i+] = imgS‘(p,p);}
            else {imgC[i+] = imgQ‘(p,p);}
            p=p+100;
            i = i+1;
        }
        image imgD = animate(imgC‘(600,400):-100,3);
        display(imgD);
    }

    showall( imgA, imgB );
```

The function reads two operands and assigns them to a picture array inside of while loop. Here is the output of this function.

2.6 Practical Example

Now we have almost enough conceptions about how IPL works. Let’s see a more practical example. Suppose that you want to make an advertisement for Computer Science Department.

```
image a = "sshield.jpg", b = "cucs.jpg",c = "columbia1.jpg", d = "columbia2.jpg";
image a1 = a‘(512,300):-100;
image b1 = b‘(600,300);
image c1 = c‘(100,600);
image d1 = d‘(800,600);
a1 = animate(a1^5@20:+100,10);
b1 = animate(b1^3:-300,10);
c1 = animate(c1‘(600,100):-300,10);
d1 = animate(d1‘(100,100):100,10);
display(a1 $ b1 $ c1 $ d1);
```

Here is the running output of this function (By capturing snapshot at some time).

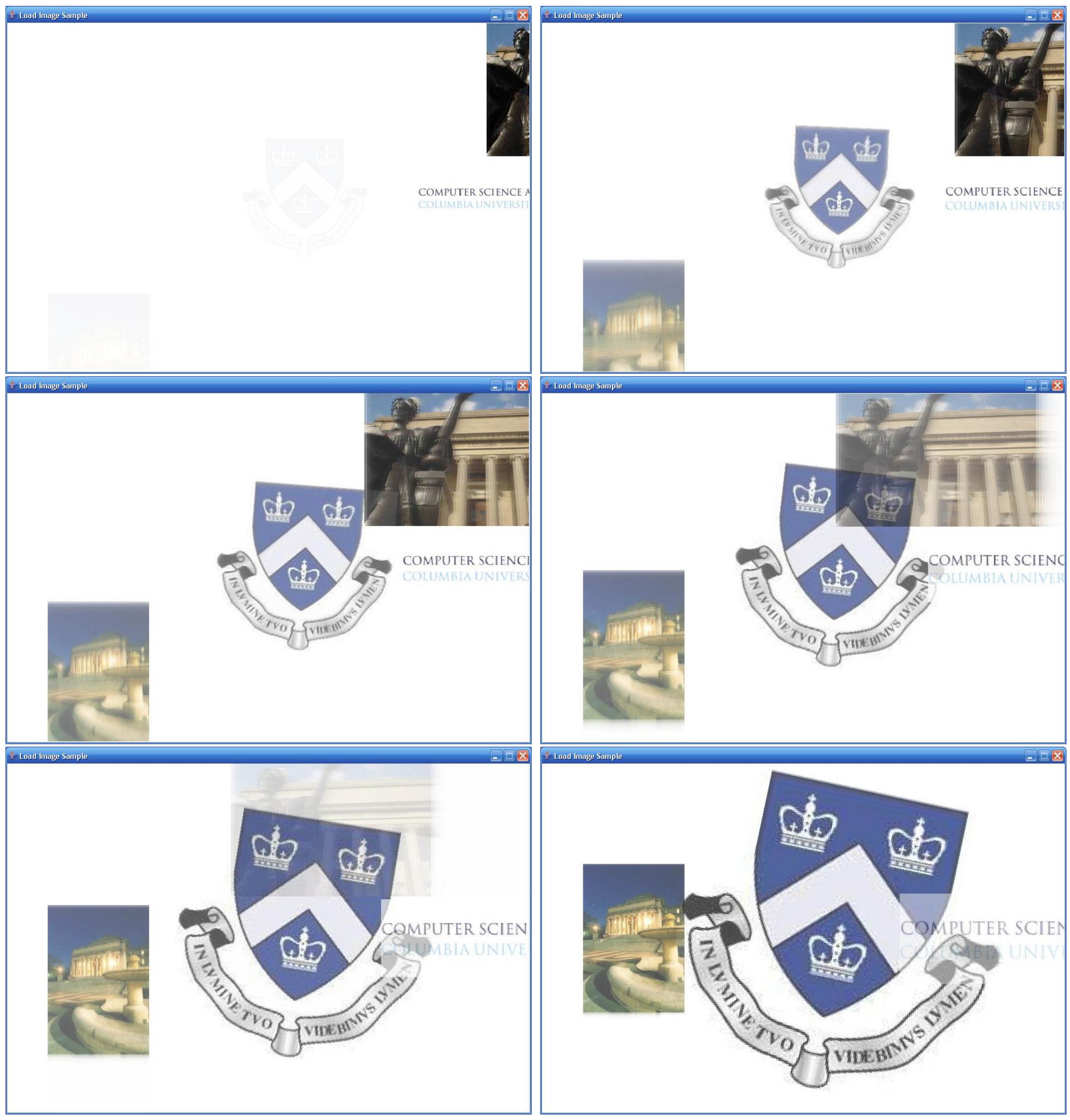

Figure 2.7: Practical example: an advertisement

Chapter 3

Language Reference Manual

3.1 Lexical Conventions

There are six kinds of tokens: identifiers, keywords, constants, strings, expression operators, and other separators. In general blanks, tabs, newlines, and comments as described below are ignored except as they serve to separate tokens. At least one of these characters is required to separate otherwise adjacent identifiers, constants, and certain operator-pairs. If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters which could possibly constitute a token.

3.1.1 Comments

The characters /* introduces a comment, which terminates with the characters */. // introduces a comment of a single line.

3.1.2 Identifiers(Names)

An identifier is a sequence of letters and digits; the first character must be alphabetic. The underscore _ counts as alphabetic. Upper and lower case letters are considered different.

3.1.3 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

if	while
number	image
coord	bool
break	continue
defunc	else
export	true
false	null
xof	yof
diplay	animate

3.1.4 Constants

There four kinds of constants in IPL: number constant, image constant, coordination constant, and bool constant. A number constant is a sequence of digits optionally followed by dot with sequence of digits. It represents both floating point number and integer number in C. An image constant is surrounded by quotes which denotes filename of the image. A coordination constant is surrounded by parenthesis and includes two number constants separated by comma. And a bool constant is “true” or “false”, which represents yes or true and no or false respectively.

3.2 What’s in a name

IPL bases the interpretation of an identifier upon contents of the identifier: image, number, bool and coordinate. If variables are invoked inside of a function, variables are local to each invocation of a function, and are discarded on return. Independently of invocations of the function; external variables are independent of any function. IPL supports four fundamental types of objects: numbers, images, bools and coordinates. Besides the four fundamental types there is a conceptually infinite class of derived types constructed from the fundamental types in the following way:

lists of objects of most types;
functions which return objects of a given type;

In general these methods of constructing objects can be applied recursively.

3.3 Objects and lvalues

An object is a manipulatable region of storage; an lvalue is an expression referring to an object. An obvious example of an lvalue expression is an identifier. The name value comes from the assignment expression ”E1 = E2” in which the left operand E1 must be an lvalue expression. The discussion of each operator below indicates whether it expects lvalue operands and whether it yields an lvalue.

3.4 Conversions

In IPL, conversion (type casting) is not available.

3.5 Expressions

The precedence of expression operators is the same as the order of the major subsections of this section (highest precedence first). Within each subsection, the operators have the same precedence. Left- or right- associativity is specified in each subsection for the operators discussed therein. The precedence and associativity of all the expression operators is summarized in an appendix.

Otherwise the order of evaluation of expressions is undefined. In particular the compiler considers itself free to compute subexpressions in the order it believes most efficient, even if the subexpressions involve side effects.

3.5.1 Expressions

We can manipulate image with their special operator. For static images, you can rotate, scale and set the position or transparency using variant expressions. IPL also supports logical and arithmetic expressions for number and bool type similar to C. Expression can be divided into three part based on their operands.

identifier

An identifier is an expression. Its type is specified when it is declared. it cannot be used when it is not declared properly.

constant

A decimal, or floating constant used in C is an expression whose type is a number in IPL. general string type in C++, surrounded by double quotes, is an expression whose type is image. “true” and “false” are expressions whose type is bool. two number constants separated by comma and surrounded by parenthesis represents a coordination, which is an expression whose type is coord. It denotes the coordination of the screen.

identifier (expr-list)

If you have identifier followed by parenthesis and containing a possibly empty, comma- separated list of expression which constitute the actual argument to the function.

identifier [range-expr]

A identifier followed by a range expression within square brackets is an expression. It denotes a list of numbers, images or coordinates. Range expression(range-expr) is a special expression to denote the range of the function. it can be just a number, numbers with operator which means the range of array, and number with operator and + or - operator which denote add element and delete elements respectively.

To specify, If there is +, it means that extend the total size of array so we can contain the rvalue. If there is -, it mean that returns the array except denoted range. because of this characteristics of + and -, you can only use + for lvalue and - for rvalue. It is explained by Figure below:

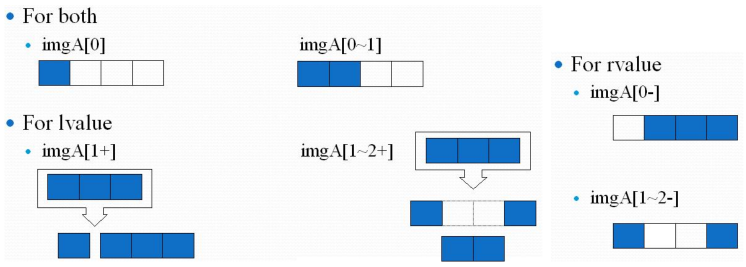

Figure 3.1: Array manipulation

Image Expression

Image expression is an expression for image manipulation. The expression returns image constant, which is one of the data type in IPL. The operator for image expression are:

```
^    @    ‘    :

$
```

There are precedence for these operators. The operators where the first line shows has a higher precedence.

Logical Expression

Logical expression is a expression for logical calculation. The expression returns type boolean constant, which is one of the data type in IPL, denoted as bool. The operator for logical expression are:

$| \quad \& \quad == \quad != \quad < \quad > \quad <= \quad >=$

Or by the unary operators:

!

Arithmetic Expression

Arithmetic expression is an expression for arithmetic calculation. The expression returns number constant, which is one of the data type in IPL, denoted as number. The operator for arithmetic expression are:

```
+    -    *    /    %
```

3.6 Boolean Expression

Boolean expression is an expression for boolean calculation. The expression returns boolean constant, which is one of the data type in IPL, denoted as bool. The operator for boolean expression are:

$| \quad \& \quad == \quad != \quad < \quad > \quad <= \quad >=$

Or by the unary operators:

!

there is boolean type in IPL, the result of those operation is both true or false.

3.6.1 Unary Operator

Expressions with unary operators group right-to-left.

!expression

The result of the logical operator ! is true if the value of the expression is false, false if the value of the expression is true. The type of the result is boolean. This operator is applicable only to boolean.

-expression

The result of the arithmetic operator - yields the negation of the expression. This operator is applicable only to number.

3.6.2 Image Operators

Image operators are used for transforming the show of pictures. It provides the user with easy and direct ways to modify the picture appearance with high flexibility. The operators will return the type image after being transformed according the operands. The precedence of $ is lower than the precedence of @ $\wedge$. and :. In each of their precedence the operators group left-to-right.

expression $ expression

$ operator indicates the concatenation of the two operands. Both operands must be of image type. It does not means that the image will be displayed in sequence. It should be displayed on their position. This operator returns an combined image, which can be contained in image type.

expression ∧ expression

The ∧ operator indicates scaling. The first operand must be image type which is the picture needed scaling, and second operand must be number type denotes how many times the picture will be scaled. Whether enlarging or shrinking the image is determined by the value of the second operand. The result is the image type object that has been scaled.

expression @ expression

The @ operator indicates rotating. The first operand must be image type which is the picture needed to rotate, and second operand must be number type denotes how degrees the picture will be rotated. The result is the image type object that has been scaled.

expression : expression

The : operator indicates alpha value setting, which is to set the transparency of a picture. The left operand must be image type which is the picture, and second operand should be number type denotes how transparent the picture will be shown. The result is the image type object that has been set.

expression ‘ expression

The ”‘’ operator indicates position setting. The first operand must be image type which is the picture needed to be positioned, and second operand must be coordinate type denotes where the picture should be placed. The result is the image type object that has been set.

3.6.3 Arithmetic Operators

Arithmetic Operator is used for arithmetic calculation of numbers.The binary operators *, /, %, +, and -, group left-to-right.

expression * expression

The * operator indicates arithmetic multiplication. The operands should be both number type, and the result type is a number.

expression / expression

The / operator indicates arithmetic division. The same type considerations as for multiplication apply.

expression % expression

The binary % operator yields the reminder from the division of the first expression by the second. The same type considerations as for multiplication apply.

expression + expression

The + operator indicates arithmetic addition. The result is the sum of the expressions. The same type considerations as for multiplication apply.

expression - expression

The result is the difference of the expressions. The same type considerations as for addition apply.

3.6.4 Assignment Operators

Assignment operator groups right-to-left. It requires a lvalue as its left operand, and the type of an assignment expression is that of its right operand. The value is stored in the left operand after the assignment has taken place.

lvalue = expression

The assignment operator requires a lvalue as its left operand, and the type of an assignment expression is that of its right operand. The value is the value stored in the left operand after the assignment has taken place. lvalue is an expression referring to an object which is a manipulatable region of storage. The value of the expression replaces that of the object referred to by the lvalue. An obvious example of an lvalue expression is an identifier.

The = operator supports the automatic array type transformation, which means that the lvalue type will automatically change to the type of the right operand if the two are different types depend on what kind of range-expressions the lvalue have.

3.6.5 Relational and Equality operators

The relational operators group left-to-right, the following are the relational operators:

expression > expression (larger)

expression >= expression (larger than equal)

expression < expression (smaller)

expression <= expression (smaller than equal)

The relational operators all yield false if the specified relation is “false” and “true” if it is true.

expression == expression

expression != expression

The == (equal to) and the != (not equal to) operators are exactly analogous to the another logical operators explained before except their lower precedence. (Thus “a<b == c<d” is true whenever a<b and c<d have the same truth value).

3.6.6 Logical Operators

expression | expression

The | operator groups left-to-right. The operands must be number type. The result is a number which is the bit-wise inclusive or of its operands.

expression & expression

The & operator groups left-to-right. Both operands must be number type. The result is a number which is the bit-wise logical and of the operands.

3.7 Declarations

Declarations are used anytime to specify the interpretation which IPL gives to each identifier; they do not necessarily reserve storage associated with the identifier. Declarations have the form

```
declaration:
       decl-specifiers declarator-listopt;
```

The declarators in the declarator-list contain the identifiers being declared. The decl-specifiers consist of at most one type-specifier.

3.7.1 Type Specifiers

The type-specifiers are type-specifier:

```
image
number
bool
coord
image[]
number[]
bool[]
coord[]
```

3.7.2 Declarators

The decl-stmt is appeared in a declaration is a comma-separated sequence of declarators which is defined by asgn-list. you can also define the initial values in declaration statement.

```
asgn-list:
        ID
        assignment
        ID, asgn-list
        assignment, asgn-list
```

Meaning of Declarations

If the declaration has the form image[] i; The declaration-specifier makes the identifier have the type "array". The bracket of the type-specifier can not be filled with any constant, identifier or any other thing. The declaration calls for a list named i, and so is the case with bool[], number[] and coord[].

3.8 Statements

Except as indicated, statements are executed in sequence. Statements can be categorized as defunc statements and others, named as normal statements. Normal statements tells what should be executed in IPL, while defunc statements describe what the specific function does in IPL. Defunc statements starts with defunc keyword, while normal statements does not. It is described as follows:

3.8.1 Normal Statements

Normal statements can be divided into assignment statements, function call, conditional statements, loop statements, break statements, return statements and continue statements. Because each statement will be executed by sequence in IPL, it is important to control the various data using conditional statements and loop statements.

Assignment Statement

In IPL language, Most statements are assignment statements. it is not allowed to use expression itself as a statement. expression should be assigned or used in while or if statement.

```
lvalue = expr ;
```

Function-Call Statement

Function call can be used as a statement. because the non-return type function can be defined, it can be used both rvalue for assignment and statement.

Conditional Statement

Conditional statement can be described as follows:

```
if ( expression ) { statement-list }
if ( expression ) { statement-list } else { statement-list }
```

It executes one or more statements if the return value of primary in parenthesis is true. statement after close parenthesis describes what to do if the primary is true. The statement-list can be expressed as both single statement or, multiple statements surrounded by brace such as:

```
statement-list :
      statement
      statement statement-list
```

While statement

Syntactically, there is only one loop statement in IPL. It is while loop expressed as:

```
while ( expression )  {statement-list}
```

It executes statements in braces iteratively until the return of expression in parenthesis becomes false. Statement after close parenthesis describes what to do while the expression is true.

Break statement

The statement

```
break;
```

causes termination of the smallest enclosing while statement; control passes to the statement follow the terminated statement.

Continue statement

The statement

```
continue;
```

causes control to pass to the loop-continuation portion of the smallest enclosing while statement; the is to the end of the loop.

Return Statement

The statement

```
return;
```

causes termination of the function invoke; control passes to the statement follow the terminated function calls.

3.8.2 Defunc statement

defunc statement starts with defunc reserved words which represents the definition of the function. It is free where you define function using defunc in IPL. However, the function should be defined before it is used. The syntax of defunc can be described as:

```
 defunc func\_identifier ( param-listopt ) return-identifier
{statement-list}
```

It defines the func_identifier as a name of function which has param-list as an optional parameter with parenthesis and return-identifier as a return value if it have the return-identifier. Moreover, statement after return-identifier represents what to do in the function.

3.8.3 Miscellaneous

param-list represents the parameter list for the function. It can be represented only one or more variables as follows:

```
param-list:
      identifier
      identifier, param-list
```

return_value represents the return value of the function. It can be represented only one variable or empty if not needed defined as:

```
return-identifier:
      identifier
```

3.9 Scope rules

Because IPL allows a block-structure surrounded by braces in case of defunc, if and while statements, it is important to define the scope of the variables. Variables in braces is restricted as local variable in brace only. For example, following causes an error because imgB defined as a local scope:

```
if (numA > 3)
{
      image imgB;
      imgB = "./foo.jpg";
}
display(imgB);
```

In case of function, it uses a static scope scheme.

Chapter 4

Project Plan

4.1 Team Responsibilities

Every team member is responsible for some part of this project. For the past three months, our team has held regular meetings on every weekend to discuss about our project and cowork with each other. We have set responsibilities for each team member to complete this project. Here are how we distributed our tasks to each member:

Wookyun Kho	Back-end (Animation Engine, Applet Code)
Youngjin Yoon	Front-end (Parser, Lexer, Tree Walker)
Jianning Yue	Testing and documentation

4.2 Project Timeline

Here is our project timeline:

Sep 26	White paper submitd
Oct 15	Grammar Specification
Oct 19	The language reference manual
Nov 15	Lexer and Parser clearance
Nov 25	Tree walker
Nov 26	The back-end Design
Dec 7	The back-end Implementation
Dec 10	Module Test
Dec 13	Integration Test
Dec 19	Final Demonstration and Documentation

4.3 Software Project Environment

This project is developed on Windows environment and we used Java SDK 1.4.2. The lexer and parser are manipulated by ANTLR to java code, and most of the other programs are written in Java. The back-end development and testing environment is Eclipse V.3.2.0.

4.3.1 ANTLR

The language parser is written in ANTLR, a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing C++ or Java actions.

4.3.2 Operating Systems

This project is developed in pure Java and can be used on any kind of machine that a Java Virtual Machine is running on. Our team used Windows to develop our compiler.

4.3.3 Java 1.4.2

Java 2D class has provided pretty strong implementation methods for creating animations. IPL translates its own language to a JAVA code to display animations on JFrame.

4.3.4 CVS

CVS is a version control system for developers. We kept our CVS repository on a Linux machine, and use SSH to check in/out our programs with secured network connections.

4.4 Project Log

Here are the dates of significant project milestones. Most of them came from the CVS logs.

Sep 13	The first version of white paper
Sep 26	White paper submitted
Sep 30	The first version of syntax
Oct 7	Modifications on syntax
Oct 13	The first version of grammar
Oct 19	The language reference manual
Oct 20	Modifications on syntax
Nov 15	Lexer and Parser first version
Nov 16	Grammar testing. Lexer and parser settled
Nov 18	Tree walker started
Dec 5	The back-end started
Dec 7	The back-end first version finished
Dec 11	Started final report
Dec 13	Tree walker and back-end tuning
Dec 16	Most tests are done
Dec 17	The first version of final report

Chapter 5

Architecture Design

The IPL compiler consists of five components: a lexer which reads the user input as a stream and translate it into some tokens, a parser that analyzes syntactic structure of the program and converts it into an abstract syntax tree, a tree walker that travel the abstract syntax tree and create and invoke the method, a symbol table which contains variable and functions with scope information, and image display engine which actually display our image. In case of lexer, parser and tree walker, we used antlr language to generate java code. It can be shown as figure below.

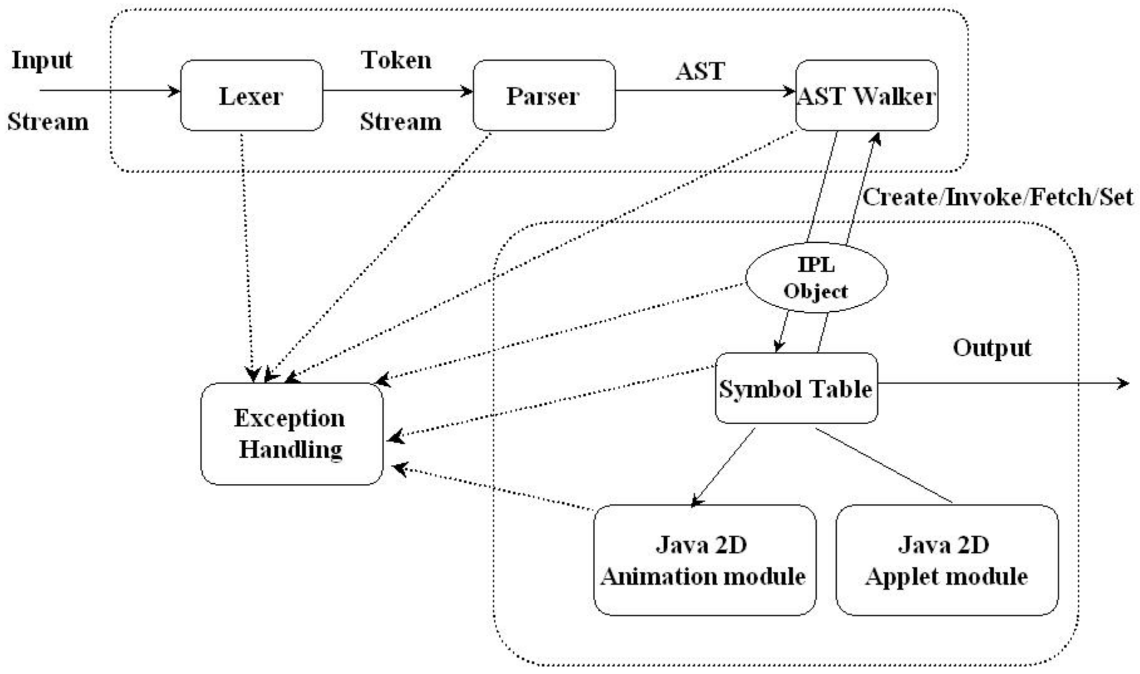

Figure 5.1: Compiler Architecture Design

The main method is in class IPL. It simply opens IPL language file and pass the stream into lexer. Then lexer make the token and try to pass it into parser. As a result, parser generates abstract syntax tree as a form of class AST. By using this AST, a Treewalker traverses AST and create object which is proper to our image type, invoke method in the object, manage the control statement such as if, else, while, continue, break, return, get the object from symbol table, and set the object into the symbol table. Every data-type object is derived from IPLDataType. figure below illustrates data types in IPL as a focus of implementation.

Because we have much flexible array range manipulation, we have IPLRange as abstract data type (which means not shown as a type in our language) in our implementation. Also, to save our function, we declare IPLFunc object.

Chapter 6

Testing Plan

6.1 Overview

IPL is to conduct animation production, and testing is to verify whether the compiler can handle every possible input correctly. In designing the testing plan, we are aiming to make sure every phase is the valid and effective basis of the next and the demonstration of each module will enhance the durability of the whole system will make it less error-prone. While the testing plan tries to cover as many as possible cases, it still can not necessarily cover all the situations the compiler may come across. Testing is undertaken throughout the compiler development process and our testing is to test each piece of our compiler and integrate them together to conduct the final test.

6.2 Phase I

This testing phase involved initial phases of development based on the incremental implementation of functionality of each individual module. Also, specifically, the initial testing phase includes the testing of lexer/parser to make sure our syntax is well defined. As specific methods or functionality were implemented small tests would be conducted in order to assure proper behavior of a specific feature or unit. The initial phase does not involve a systemic testing before they work as parts of a more integrated functional module. The verification of these units is the basis for our further testing phase.

6.3 Phase II

The next phase of testing involved integration of the module and to test whether they works fine with one another. The parser should pipe a valid syntax tree to the tree walker which is based on the validation of the parser. The tuning between the tree walker and back-end ensures will validate the functionality of our final testing. At this point, all the major classes have been written and pieced together in their final arrangement. We want to ensure that all the blocks of code are correctly interacting.

6.4 Final testing Programs and Integration

The final test aims to demonstrate the correctness of the whole system that user will see and use. We enter this phase after the development is finished. Before this, the white box tests and regression tests have been passed and the code is ready for final testing. Generally these phase include the frond-end functionality of the language. The testing cases are included in the appendix, here is an list of them:

1 `testingcase/animate.ipl`

2 `testingcase/burger.ipl`

3 `testingcase/coordinate.ipl`

4 `testingcase/defination1.ipl`

5 `testingcase/defination2.ipl`

6 `testingcase/display.ipl`

7 `testingcase/fading.ipl`

8 `testingcase/function.ipl`

9 `testingcase/functiondefinition.ipl`

10 `testingcase/imageassignment.ipl`

11 `testingcase/imageinserting.ipl`

12 `testingcase/imagelistadding2.ipl`

13 `testingcase/imagelistadding.ipl`

14 `testingcase/joining2.ipl`

15 `testingcase/joining.ipl`

16 `testingcase/positioning.ipl`

17 `testingcase/rotating.ipl`

18 `testingcase/scaling.ipl`

19 `testingcase/strawberry.ipl`

20 `testingcase/animate2.ipl`

21 `testingcase/while.ipl`

22 `testingcase/scope1.ipl`

23 `testingcase/scope2.ipl`

24 `testingcase/demo1.ipl`

```
25 testingcase/demo2.ipl

26 testingcase/demo3.ipl

27 testingcase/demo4.ipl
```

Chapter 7

Lessons Learned

Our group benefits a lot from the development the IPL compiler. We have been getting to know importance of collaborating and working as a team to accomplish our tasks. We regularly held meetings for discussion no matter how much other work our teammates needed to handle. The discussion sessions are often effective and productive that we exchanged our ideas of how our animation production language will be like and what it will do, the result of this is that IPL was improving from time to time and approaching what we have imaged what we have expected.

We progressed our project by setting new goal of each phase and divide the work during every discussion. During this we were making sure every team member is in the scope of the project in that we know if there exists somewhere that could not fully grasped by every member it may probably become a bottle neck for the efficiency of our work and even cause trouble to the progress of the project. In this regard, we found that the management of our team is crucial to the product of project even if we are just a 3-people group. Here the time management is important that the time line seem not very tight at the beginning before we noticed that we should speed up our pace, on the other hand, the fully research of the basis of language is determining the implementation phase of the project, so we can say our management is successful but still has something that could be improved.

As mentioned before, the distribution of our work is crucial that in order to learn something, everybody should do every procedure that we need to do, meanwhile we may not guarantee that every team member has gained the knowledge and skills that every part needs. Though it is hard, we should hold the attitude that to learn something needs real practice.

We were having a lot of fun in developing such an interesting language which arises our further interests in designing and implementation of programming language. This course has endowed us with not only the knowledge of languages and compilers but also the opportunity to exchange our ideas. We will also keep on improving IPL and make it approach perfection. We are loving IPL.

Appendix A

Language Syntax

A.1 Lexical rules

alpha → ’a’..’z’ | ’A’..’Z’ | ’_’
digit → ’0’..’9’
id → **alpha** (**alpha**|**digit**)*
number → ((digit)+ (’.’ (digit)+)?
whitespace →

```
' ' | '\t' | '\r'|'\r\n'|'\n'
```

comment →

```
( '/*' (~'*/')* '*/' )|'//'(~('\n' | '\r'))*
```

A.2 Syntactic rules

```
program →
                 statement-list
statement-list →
                 statement
                 statement statement-list
statement →
                 if-stmt
                 while-stmt
                 decl-stmt
                 break-stmt
                 continue-stmt
                 display-stmt
                 animate-stmt
                 func-call-stmt
                 assignment
                 defunc-stmt
if-stmt →
                 if ( expr )  statement  else-stmt
else-stmt →
                 else  statement
                 ϵ
while-stmt →
                 while ( expr )  statement
decl-stmt →
                 number asgn-list SEMI
                 image asgn-list SEMI
                 coord asgn-list SEMI
                 bool asgn-list SEMI
asgn-list →
                 ID
                 assignment
                 ID, asgn-list
                 assignment, asgn-list
break-stmt →
                 break ;
continue-stmt →
                 continue ;
return-stmt →
                 return ;
display-stmt →
                 display ( img-expr ) ;
animate-stmt →
                 animate ( img-expr, arith-expr) ;
func-call-stmt →
                 func-call ;
export-stmt →
                 export ( img-expr ) ;
func-call →
                 ID ( arg-list ) ;
                 ID () ;
arg-list →
                 expr
                 expr , arg-list
assignment →
                 lvar = expr ;
lvar →
                 ID
                 ID [ lrange-expr ]
lrange-expr →
                 NUM
                 NUM ~NUM
                 NUM ~NUM +
expr →
                 img-expr
img-expr →
                 img-term $ img-expr
                 img-term
img-term →
                 logical-expr :  img-term
                 logical-expr @ img-term
                 logical-expr ^img-term
```

```
                  logical-expr ‘ img-term
                  logical-expr
logical-expr →
                  logical-not & logical-expr
                  logical-not | logical-expr
logical-not →
                  !  logical-term
                  logical-term
logical-term →
                  arith-expr >= arith-expr
                  arith-expr <= arith-expr
                  arith-expr > arith-expr
                  arith-expr < arith-expr
                  arith-expr == arith-expr
                  arith-expr != arith-expr
                  arith-expr >= arith-expr
                  arith-expr
arith-expr →
                  arith-term + arith-expr
                  arith-term - arith-expr
                  arith-term
arith-term →
                  arith-unary * arith-term
                  arith-unary / arith-term
                  arith-unary % arith-term
                  arith-unary
arith-unary →
                  - r-value
                  r-value
r-value →
                  func-call
                  animate-stmt
                  NUM
                  coord
                  'true'
                  'false'
                  xof ( ID )
                  yof ( ID )
                  rvar
                  ( expr )
rvar →
                  ID [rrange-expr]
                  ID
rrange-expr →
                  NUM
                  NUM ~NUM
                  NUM ~-
coord →
                  ( arith-expr, arith-expr )
```

Appendix B

Code Listing

B.1 Parser

B.1.1 src/grammar.g

```
//////////////////// LEXER ///////////////////////////////////
class IPLLexer extends Lexer;
options {
k = 2;                // Need to decide when strings literals end
charVocabulary = '\3'..'\377'; // Accept all eight-bit ASCII characters
testLiterals = false;
     exportVocab = IPLAntlr;
defaultErrorHandler = false;
}
MOD    : '%' ; TIMES  : '*' ; DIV    : '/' ; //arithmatic operators
PLUS   : '+' ; MINUS  : '-' ;

SCALE  : '^' ; ROTATE : '@' ; //image operators
CONCAT   : '$' ;
SET  : '`' ; ALPHA  : ':';

NOT : '!' ; AND : '&' ; OR  : '|' ; // logical operators
LE  : "<="; GE  : ">="; LT  : '<' ; GT  : '>' ; EQ  : "=="; NE  : "!=";

FROMTO : '~' ; SEMI   : ';' ; LBRK   : '[' ; RBRK   : ']' ; COMMA  : ',' ;

ASSIGN : '=' ;
LBRACE : '{' ; RBRACE : '}' ;
LPAR   : '(' ; RPAR   : ')' ;
NUM :  DIGITS ('.' DIGITS )? ;


STRING : '"'! (options {greedy=false;}:.)* '"'!;

protected
DIGITS : ('0'..'9')+ ;
```

```
WHITESPACE : (
' '
| '\t'
| ("\r\n" | '\r' | '\n') { newline(); }) { $setType(Token.SKIP); };

COMMENT :
("/*"  (
options {greedy=false;}:
       ('\r' '\n') => '\r' '\n' { newline(); }
| '\r' { newline(); }
| '\n' { newline(); }
| ~('\n' | '\r')
)* "*/"
| "//" (~('\n' | '\r'))*
) { $setType(Token.SKIP); } ;


//identifier
ID options { testLiterals = true; }
: ('_' | 'a'..'z' | 'A'..'Z') ('_' | 'a'..'z' | 'A'..'Z' | '0'..'9')* ;


/////////////////// PARSER ////////////////////////////////
class IPLParser extends Parser;
options {
    buildAST = true; // Enable AST building
    k = 2;           // Need to distinguish between variable and function call
    exportVocab = IPLAntlr;
    defaultErrorHandler = false;
}
tokens {
 STATEMENT;
 STMT;
 FUNC_CALL;
 BLOCK_BODY;
 ARG_LIST;
 EXPR_LIST;
 ASGN_LIST;
 ASGN_BLOCK;
 NEGATION;
 COORDINATION;
 NUMBER_ARRAY;
 COORD_ARRAY;
 IMAGE_ARRAY;
 BOOL_ARRAY;
}
statement: (stmt)* EOF!
 {#statement = #([STATEMENT, "STATEMENT"], statement); };

stmt
: defunc_stmt
```

```
| exec_stmt //{#stmt = #([STMT, "STMT"], stmt); }
;

exec_stmt
: decl_stmt
| if_stmt
| while_stmt
| break_stmt
| continue_stmt
| display_stmt
| export_stmt
| func_call_stmt
| (assignment) => assignment
| return_stmt;
if_stmt : "if"^ LPAR! expr RPAR! block_body
    (options {greedy = true;}: "else"! block_body )?;

while_stmt : "while"^ LPAR! expr RPAR! block_body;

break_stmt : "break"^ SEMI!;

continue_stmt : "continue"^ SEMI!;

return_stmt : "return"^ SEMI!;

assignment : l_value ASSIGN^ (asgn_block | expr) SEMI!;
display_stmt : "display"^ LPAR! expr RPAR! SEMI!;

export_stmt : "export"^ LPAR! expr RPAR! SEMI!;
animate_stmt: "animate"^ LPAR! img_term COMMA! arith_expr RPAR! ;

func_call_stmt : func_call SEMI! ;

defunc_stmt : "defunc"^ ID LPAR! arg_list RPAR! (d1:decl i1:ID!{#d1 = #(d1, #i1);})?
           block_body;

arg_list: d1:decl i1:ID!
{ #d1 = #(d1, #i1); }
(COMMA! d2:decl i2:ID!
{ #d2 = #(#d2, #i2); } )*
{#arg_list = #([ARG_LIST, "ARG_LIST"], arg_list); }
| /* nothing */;

block_body : LBRACE! (exec_stmt)* RBRACE!
 {#block_body = #([BLOCK_BODY, "BLOCK_BODY"], block_body); };

decl_stmt: d:decl a:asgn_list! SEMI!
{ #decl_stmt = #(#d, #a); };

decl: ("number"^ (LBRK! RBRK!
{#decl = #[NUMBER_ARRAY, "NUMBER_ARRAY"]; })?
| "image"^ (LBRK! RBRK!
{#decl = #[IMAGE_ARRAY, "IMAGE_ARRAY"]; })?
| "coord"^ (LBRK! RBRK!
{#decl = #[COORD_ARRAY, "COORD_ARRAY"]; })?
| "bool"^ (LBRK! RBRK!
{#decl = #[BOOL_ARRAY, "BOOL_ARRAY"]; })?
      );

asgn_list: asgn_val (COMMA! asgn_val)*;

asgn_val: ID (ASSIGN^ (asgn_block | expr))?;

asgn_block : LBRACE! expr (COMMA! expr)* RBRACE!
{#asgn_block = #([ASGN_BLOCK, "ASGN_BLOCK"], asgn_block); };




expr: img_term ((CONCAT^) img_term)*;
img_term: logical_expr (((SCALE^ | ROTATE^ | ALPHA^) logical_expr | SET^ logical_expr))* ;
logical_expr: logical_not ((AND^ | OR^) logical_not)*;
logical_not: (NOT)? logical_term;
logical_term: arith_expr ((LE^ | GE^ | LT^ | GT^ | EQ^ | NE^) arith_expr)?;
arith_expr: arith_term (options {greedy = true;}:(PLUS^ | MINUS^) arith_term)*;
arith_term: arith_unary ((TIMES^ | DIV^ | MOD^) arith_unary)*;

arith_unary: PLUS! r_value
| MINUS! r_value
{ #arith_unary = #([NEGATION, "NEGATION"], arith_unary);}
|r_value ;

r_value: (func_call) => func_call | STRING | NUM | (coord) => coord |
"true" | "false"| animate_stmt |
"xof"^ LPAR! expr RPAR! | "yof"^ LPAR! expr RPAR! |
r_variable | LPAR! expr RPAR!;
r_variable: ID^ (LBRK! index (MINUS)? RBRK!)*;
l_value:  ID^ (LBRK! index (PLUS)? RBRK!)*;
index: arith_expr (FROMTO^ arith_expr)?;
//lindex: arith_expr (FROMTO^ arith_expr)? ( PLUS )?;
//rindex: arith_expr (FROMTO^ arith_expr)? ( MINUS )?;

func_call: ID LPAR! expr_list RPAR!
{ #func_call = #([FUNC_CALL, "FUNC_CALL"], func_call);};

expr_list: expr (COMMA! expr)*
{ #expr_list = #([EXPR_LIST, "EXPR_LIST"], expr_list);}
| /* nothing */;


coord :  LPAR! arith_expr COMMA! arith_expr RPAR!
{ #coord = #([COORDINATION,"COORDINATION"], coord); };
```

B.1.2 src/walker.g

```
{
 import java.io.*;
 import java.util.*;
}

class IPLTreeWalker extends TreeParser;
options
{
importVocab = IPLAntlr;
defaultErrorHandler = false;
}

{
static final int PROCEED = 0;
static final int ISBREAK = 1;
static final int ISCONTINUE = 2;
static final int ISRETURN = 3;

IPLSymbolTable ist = new IPLSymbolTable();
int control = PROCEED;
}

stmts
{
}
:#(STATEMENT (allPart:.{stmt(#allPart);})*);


stmt returns[IPLDataType r]
{
IPLDataType a,b,c;
Vector argp = null;
Vector retVal = null;
IPLFunc f;
r = null;
}

:#("number"  (numId:ID {
r = new IPLNumber(numId.getText());
ist.asgnVar(r);}
|(#(ASSIGN numId2:ID arith_asgn:. {
r = new IPLNumber(numId2.getText());
ist.asgnVar(r);
a = arith_expr(#arith_asgn);
if(a instanceof IPLNumber) ist.setVar(numId2.getText(),a);
else a.error("Cannot assign "+a.type+"into number "+numId2.getText());})))+)
|#("image" (imgId:ID {
```

```
r = new IPLImage(imgId.getText(),"<NULL>");
ist.asgnVar(r); }
|(#(ASSIGN imgId2:ID img_asgn:. {
r = new IPLImage(imgId2.getText(),"<NULL>");
ist.asgnVar(r);
a = img_expr(#img_asgn);
if(a instanceof IPLImage) ist.setVar(imgId2.getText(),a);
else a.error("Cannot assign "+a.type+"into image "+imgId2.getText());})))+)
|#("coord"  (cooId:ID {
r = new IPLCoord(cooId.getText());
ist.asgnVar(r); }
|(#(ASSIGN cooId2:ID a=coord_expr {
r = new IPLCoord(cooId2.getText());
ist.asgnVar(r);
if(a instanceof IPLCoord) ist.setVar(cooId2.getText(),a);
else a.error("Cannot assign "+a.type+"into coord "+cooId2.getText());})))+)
|#("bool"  (booId:ID {
r = new IPLBool(booId.getText());
ist.asgnVar(r); }
|(#(ASSIGN booId2:ID a=logical_expr {
r = new IPLBool(booId2.getText());
ist.asgnVar(r);
if(a instanceof IPLBool) ist.setVar(booId2.getText(),a);
else a.error("Cannot assign "+a.type+"into bool "+booId2.getText());})))+)
|#(NUMBER_ARRAY (numAId:ID {
r = new IPLNumberArray(numAId.getText());
ist.asgnVar(r);}
|(#(ASSIGN numAId2:ID a=expr {
r = new IPLNumberArray(numAId2.getText());
ist.asgnVar(r);
if(a instanceof IPLNumberArray) ist.setVar(numAId2.getText(),a);
else a.error("Cannot assign "+a.type+"into number[] "+numAId2.getText());})))+)
|#(IMAGE_ARRAY (imgAId:ID {
r = new IPLImageArray(imgAId.getText());
ist.asgnVar(r);}
|(#(ASSIGN imgAId2:ID a=expr {
r = new IPLImageArray(imgAId2.getText());
ist.asgnVar(r);
if(a instanceof IPLImageArray) ist.setVar(imgAId2.getText(),a);
else a.error("Cannot assign "+a.type+"into image[] "+imgAId2.getText());})))+)
|#(COORD_ARRAY (cooAId:ID {
r = new IPLCoordArray(cooAId.getText());
ist.asgnVar(r);}
|(#(ASSIGN cooAId2:ID a=expr {
r = new IPLCoordArray(cooAId2.getText());
ist.asgnVar(r);
if(a instanceof IPLCoordArray) ist.setVar(cooAId2.getText(),a);
else a.error("Cannot assign "+a.type+"into coord[] "+cooAId2.getText());})))+)
|#(BOOL_ARRAY ((ID)=>booAId:ID {
r = new IPLBoolArray(booAId.getText());
ist.asgnVar(r);}
```

```
|(#(ASSIGN booAId2:ID a=expr {
r = new IPLBoolArray(booAId2.getText());
ist.asgnVar(r);
if(a instanceof IPLBoolArray) ist.setVar(booAId2.getText(),a);
else a.error("Cannot assign "+a.type+"into bool[] "+booAId2.getText());})))+)

|#("while" expp:. bBody:BLOCK_BODY){
if(control == PROCEED) {
while(control == PROCEED) {
if(!logical_expr(#expp).getValue()) break;
ist.newScope(false);
ist.setWhileLoop(true);
invoke(#bBody);
ist.setWhileLoop(false);
ist.exitScope();
if(control == ISBREAK) { // if break
break;
}
else if(control == ISCONTINUE) // if continue
control = PROCEED;
}
control = PROCEED;
}
}
|#("if" a=logical_expr{ ist.newScope(false);}
#(BLOCK_BODY (thenp:.{if(control == PROCEED && (((IPLBool)a).getValue()))
stmt(#thenp);})*)
(#(BLOCK_BODY (elsep:.{if(control == PROCEED && (!((IPLBool)a).getValue()))
stmt(#elsep);})*))?)
{ist.exitScope();}
|"break"
{
if(!ist.isWhileLoop()) IPLDataType.error("Cannot use break out of while loop");
else control = ISBREAK;
}
|"continue"
{
if(!ist.isWhileLoop()) IPLDataType.error("Cannot use continue out of while loop");
else control = ISCONTINUE;
}
|"return"
{
if(ist.isGlobalScope()) IPLDataType.error("Cannot use return in global scope");
else control = ISRETURN;
}
|#("export" a=img_expr)
{
if(!(a instanceof IPLImage))
a.error("Cannot export"+a.type+"to files.");
else ist.export(a);
}
```

```
|#("defunc" defId:ID ((ARG_LIST)=>argp=arglist)? (("number" | "image" | "coord" |
"bool"|NUMBER_ARRAY|IMAGE_ARRAY|COORD_ARRAY
|BOOL_ARRAY)=>retVal=retlist)? blockp:.)
{
a = new IPLFunc(defId.getText(),argp,retVal,#blockp);
ist.asgnFunc(defId.getText(), (IPLFunc)a);
}
|#("display" a=expr)
{
ist.display(a);
}
|#(ASSIGN a=l_value b=expr)
{
c = ist.getVar(a.getId());
if(c instanceof IPLArray && ((IPLRange)a).getFrom() != -1)
((IPLArray)c).listAdd((IPLRange)a,b);
// if array with ~ and +, add it into the list
else c = b; // else just replace it!
ist.setVar(a.getId(),c);
}
|#(FUNC_CALL funcId:ID argp=exprlist)
{
if(control == PROCEED) {
f = ist.getFunc(funcId.getText());
f.invoke(this, argp);
control = PROCEED;
}
}
;

arglist returns [Vector r]
{
Vector v = new Vector(5);
r = null;
}
:#(ARG_LIST (#(type:. id:ID {
String[] s = new String[2];
s[0] = new String(id.getText());
s[1] = new String(type.getText());
v.add(s);
}))*){r = v;}
;

retlist returns [Vector r]
{
Vector v = new Vector(5);
r = null;
}
:#(type:. id:ID {
String[] s = new String[2];
s[0] = new String(id.getText());
s[1] = new String(type.getText());
v.add(s);
}){r = v;}
;

expr returns [IPLDataType r]
{
IPLDataType a,b;
IPLRange range;
IPLFunc f;
r = null;
Vector argp = new Vector(5);
}
:(ID)=>r = r_value
|(FUNC_CALL)=>#(FUNC_CALL funcId:ID argp=exprlist)
{
if(control == PROCEED) {
f = ist.getFunc(funcId.getText());
r = f.invoke(this, argp);
control = PROCEED;
}
}
|(PLUS|MINUS|TIMES|DIV|MOD|NEGATION|"setx"|"sety"|NUM)=>r = arith_expr
|(CONCAT|SCALE|ROTATE|SET|ALPHA|STRING|"null"|"animate") =>r = img_expr
|(LE|GE|LT|GT|EQ|NE|NOT|AND|OR|"true"|"false") => r = logical_expr
|coo:COORDINATION { r = coord_expr(#coo); }
|#(ASGN_BLOCK {r = new IPLArray(null); }
(a=expr {((IPLArray)r).listAdd(a);})*)
{r = ((IPLArray)r).convert();}
;

exprlist returns [Vector r]
{
IPLDataType a;
Vector v = new Vector(5);
r = null;
}
:#(EXPR_LIST (a=expr{v.add(a);})*) {r = v;}
;


img_expr returns [IPLImage r]
{
IPLImage a, b;
IPLCoord c;
IPLNumber d;
IPLRange range;
IPLDataType u;
IPLFunc f;
String s;
r = null;
```

```
Vector argp = new Vector(5);
}
:#(CONCAT a=img_expr b=img_expr)  {r = a.concat(b);}
|#(SCALE a=img_expr d=arith_expr)  {r = a.scale(d);}
|#(ROTATE a=img_expr d=arith_expr)  {r = a.rotate(d);}
|#(SET a=img_expr c=coord_expr)  {r= a.set(c);}
|#(ALPHA a=img_expr d=arith_expr)  {r= a.alpha(d);}
|str:STRING { r = new IPLImage(null,str.getText()); }
|"null" { r = new IPLImage(null,"<NULL>"); }
|id:ID { r = (IPLImage) r_value(#id);
if(!(r instanceof IPLImage))
r.error("Cannot use "+r.type+"in image operation");}
|#(FUNC_CALL funcId:ID argp=exprlist)
{
if(control == PROCEED) {
f = ist.getFunc(funcId.getText());
r = (IPLImage) f.invoke(this, argp);
if(!(r instanceof IPLImage))
 r.error("Cannot calculate with the return value of"+funcId.getText());
control = PROCEED;
}
}
|#("animate" aniExpr:. d=arith_expr)
{
IPLDataType.setAnimate(true);
u = ani_expr(#aniExpr);
if(u instanceof IPLImage) r = ((IPLImage)u).animate((IPLNumber)d);
else if(u instanceof IPLImageArray) r = ((IPLImageArray)u).animate((IPLNumber)d);
else u.error("Cannot use animate with "+u.type+" type.");

IPLDataType.setAnimate(false);
};

ani_expr returns [IPLDataType r]
{
boolean animated;
IPLDataType a;
IPLCoord c;
IPLNumber d;
IPLFunc f;
r = null;
Vector argp = new Vector(5);
}
:#(SCALE a=ani_expr d=arith_expr) {
if(a instanceof IPLImage) r = ((IPLImage)a).scale(d);
else if(a instanceof IPLImageArray) r = ((IPLImageArray)a).scale(d);
else a.error("Cannot use "+a.type+" in animation()");
}
|#(ROTATE a=ani_expr d=arith_expr) {
if(a instanceof IPLImage) r = ((IPLImage)a).rotate(d);
else if(a instanceof IPLImageArray) r = ((IPLImageArray)a).rotate(d);
else a.error("Cannot use "+a.type+" in animation()");
}
|#(SET a=ani_expr c=coord_expr)  {
if(a instanceof IPLImage) r= ((IPLImage)a).set(c);
else if(a instanceof IPLImageArray) r = ((IPLImageArray)a).set(c);
else a.error("Cannot use "+a.type+" in animation()");
}
|#(ALPHA a=ani_expr d=arith_expr) {
if(a instanceof IPLImage) r= ((IPLImage)a).alpha(d);
else if(a instanceof IPLImageArray) r = ((IPLImageArray)a).alpha(d);
else a.error("Cannot use"+a.type+"in animation()");
}
|id:ID { r = r_value(#id);
if(!(r instanceof IPLImage || r instanceof IPLImageArray) )
r.error("Cannot use "+r.type+"in animation");}
|#(FUNC_CALL funcId:ID argp=exprlist)
{
if(control == PROCEED) {
f = ist.getFunc(funcId.getText());
r = (IPLImage) f.invoke(this, argp);
if(!(r instanceof IPLImage || r instanceof IPLImageArray))
 r.error("Cannot calculate with the return value of"+funcId.getText());
control = PROCEED;
}
}
|#("animate" aniExpr:. d=arith_expr)
{
animated = IPLDataType.isAnimate();
IPLDataType.setAnimate(true);
a = ani_expr(#aniExpr);
if(a instanceof IPLImage) r = ((IPLImage)a).animate((IPLNumber)d);
else if(a instanceof IPLImageArray) r = ((IPLImageArray)a).animate((IPLNumber)d);
else a.error("Cannot use animate with "+a.type+" type.");
if(!animated) IPLDataType.setAnimate(false);
};

logical_expr returns [IPLBool r]
{
IPLNumber a,b;
IPLBool c,d;
IPLFunc f;
IPLDataType u,v;
IPLRange range;
r = null;
Vector argp = new Vector(5);
}
:#(AND c=logical_expr d=logical_expr) {r= c.and(d);}
|#(OR c=logical_expr d=logical_expr) {r= c.or(d);}
|#(NOT c=logical_expr) {r= c.not();}
|#(LE a=arith_expr b=arith_expr) {r = a.le(b);}
|#(GE a=arith_expr b=arith_expr) {r = a.ge(b);}
```

```
|#(LT a=arith_expr b=arith_expr) {r = a.lt(b);}
|#(GT a=arith_expr b=arith_expr) {r = a.gt(b);}
|#(EQ u=expr v=expr {
if(u instanceof IPLBool && v instanceof IPLBool)
r = ((IPLBool)u).eq((IPLBool)v);
else if(u instanceof IPLNumber && v instanceof IPLNumber)
r = ((IPLNumber)u).eq((IPLNumber)v);
else u.error("cannot compare between "+u.type+" and "+v.type);})
|#(NE u=expr v=expr {
if(u instanceof IPLBool && v instanceof IPLBool)
r = ((IPLBool)u).neq((IPLBool)v);
else if(u instanceof IPLNumber && v instanceof IPLNumber)
 r = ((IPLNumber)u).neq((IPLNumber)v);
else u.error("cannot compare between "+u.type+" and "+v.type);})
|"true" { r = new IPLBool(true);}
|"false" { r = new IPLBool(false);}
|id:ID { r = (IPLBool) r_value(#id);
if(!(r instanceof IPLBool))
r.error("Cannot use "+r.type+"in logical operation");}
|#(FUNC_CALL funcId:ID argp=exprlist)
{
if(control == PROCEED) {
f = ist.getFunc(funcId.getText());
r = (IPLBool) f.invoke(this, argp);
if(!(r instanceof IPLBool))
 r.error("Cannot calculate with the return value of"+funcId.getText());
control = PROCEED;
}
}
;



arith_expr returns [IPLNumber r]
{
IPLNumber a,b;
IPLCoord c;
IPLRange range;
IPLFunc f;
IPLDataType u;
r = null;
Vector argp = new Vector(5);
}
:#(PLUS a=arith_expr b=arith_expr)  {r = a.add(b);}
|#(MINUS a=arith_expr b=arith_expr)  {r = a.subtract(b);}
|#(TIMES a=arith_expr b=arith_expr)  {r = a.multiply(b);}
|#(DIV a=arith_expr b=arith_expr)  {r = a.divide(b);}
|#(MOD a=arith_expr b=arith_expr)  {r = a.modulo(b);}
|#(NEGATION a=arith_expr)  {r= a.negative();}
|num:NUM { r = new IPLNumber(Double.parseDouble(num.getText())); }
```

```
|id:ID { r = (IPLNumber) r_value(#id);
if(!(r instanceof IPLNumber))
r.error("Cannot use "+r.type+"in arithmetic operation");}
|#(FUNC_CALL funcId:ID argp=exprlist)
{
if(control == PROCEED) {
f = ist.getFunc(funcId.getText());
r = (IPLNumber) f.invoke(this, argp);
if(!(r instanceof IPLNumber))
 r.error("Cannot calculate with the return value of"+funcId.getText());
control = PROCEED;
}
}
|#("xof" c=coord_expr) { r=c.xof(); }
|#("yof" c=coord_expr) { r=c.yof(); }
;

coord_expr returns [IPLCoord r]
{
IPLNumber a,b;
IPLFunc f;
IPLDataType u;
IPLRange range;
Vector argp;
r = null;
}
:#(COORDINATION a=arith_expr b=arith_expr) { r = new IPLCoord(a,b); }
|id:ID { r = (IPLCoord) r_value(#id);
if(!(r instanceof IPLCoord))
r.error("Cannot use "+r.type+"in arithmetic operation");}
|#(FUNC_CALL funcId:ID argp=exprlist)
{
if(control == PROCEED) {
f = ist.getFunc(funcId.getText());
r = (IPLCoord) f.invoke(this, argp);
if(!(r instanceof IPLCoord))
 r.error("Cannot calculate with the return value of"+funcId.getText());
control = PROCEED;
}
}
;

r_value returns [IPLDataType r]
{
r = null;
IPLRange target = new IPLRange("<NULL>");
IPLNumber from,to;
}
:#(asgnId:ID  { r = new IPLRange(asgnId.getText()); }
((from=arith_expr{target = new IPLRange(from);}
|(#(FROMTO from=arith_expr to=arith_expr
{target = new IPLRange(from,to);})))
(MINUS {target.setOpt(IPLRange.MINUS);})?)?
{r=ist.getVar(asgnId.getText(),target);})
;
l_value returns [IPLRange r]
{
IPLNumber from, to;
r = null;
}
:#(asgnId:ID  { r = new IPLRange(asgnId.getText()); }
((from=arith_expr{
r = new IPLRange(from);
r.setId(asgnId.getText());}
|(#(FROMTO from=arith_expr to=arith_expr
{ r = new IPLRange(from,to);
  r.setId(asgnId.getText());})))
(PLUS { r.setOpt(IPLRange.PLUS);})?)?)
;

invoke
:#(BLOCK_BODY (st:.{if(control == PROCEED) stmt(#st);})*) ;
```

B.2 JAVA 2D Animation Module

B.2.1 src/Affines.java

```
import java.awt.geom.*;

public class Affines implements Cloneable{
AffineTransform translate;
AffineTransform scale;
AffineTransform rotate;
float alpha;

//long duration;
double duration;
Affines next;

public Affines()
{
translate = new AffineTransform();
scale = new AffineTransform();
rotate = new AffineTransform();

translate.setToIdentity();
scale.setToIdentity();
rotate.setToIdentity();
alpha = 0.0f;

duration = 0;
next = null;
}

public void init()
{
translate.setToIdentity();
scale.setToIdentity();
rotate.setToIdentity();
alpha = 0.0f;

duration = 0;
next = null;
}

public Affines(Affines affine)
{
translate = new AffineTransform();
scale = new AffineTransform();
rotate = new AffineTransform();

translate.setToIdentity();
scale.setToIdentity();
rotate.setToIdentity();
```

```
translate.concatenate(affine.getTranslate());
scale.concatenate(affine.getScale());
rotate.concatenate(affine.getRotate());
alpha = 0.0f;

duration = 0;
next = null;
}

public Affines(Affines affine, int flag)
{
translate = new AffineTransform();
scale = new AffineTransform();
rotate = new AffineTransform();

translate.setToIdentity();
scale.setToIdentity();
rotate.setToIdentity();

translate.concatenate(affine.getTranslate());
scale.concatenate(affine.getScale());
rotate.concatenate(affine.getRotate());
alpha = affine.alpha;

duration = affine.duration;
next = null;
}

public Affines clone()
{
return new Affines(this, 1);
}

public void concatAll(Affines af)
{
translate.concatenate(af.translate);
scale.concatenate(af.scale);
rotate.concatenate(af.rotate);
}

public void concatAll(IPLImageHistory hs)
{
translate.translate(hs.x, hs.y);
scale.scale(hs.scale, hs.scale);
rotate.rotate(hs.rotate);
}

public void preconcatAll(IPLImageHistory hs)
{
AffineTransform tmp = new AffineTransform();
```

```
tmp.setToIdentity();
tmp.translate(hs.x, hs.y);
translate.preConcatenate(tmp);
tmp.setToIdentity();
tmp.scale(hs.scale, hs.scale);
scale.preConcatenate(tmp);
tmp.setToIdentity();
tmp.rotate(hs.rotate);
rotate.preConcatenate(tmp);
}

public void addTranslate(double x, double y)
{
translate.translate(x, y);
}

public void addTranslate(AffineTransform at)
{
translate.concatenate(at);
}

public void addScale(double ratio)
{
scale.scale(ratio, ratio);
}

public void addScale(AffineTransform at)
{
scale.concatenate(at);
}

public void addRotate(double degree)
//public void addRotate(int degree)
{
//rotate.rotate(Math.toRadians(degree));
rotate.rotate(degree);
}

public void addRotate(AffineTransform at)
{
rotate.concatenate(at);
}

public AffineTransform getTranslate()
{
return translate;
}

public AffineTransform getScale()
{
return scale;
```

```
}

public AffineTransform getRotate()
{
return rotate;
}

public AffineTransform getAffine()
{
AffineTransform ret = new AffineTransform();
ret.setToIdentity();
ret.concatenate(translate);
ret.concatenate(scale);
ret.concatenate(rotate);

return ret;
}

public void updateAlpha(float al)
{
alpha += al;
}
}
```

B.2.2 src/ImageINFO.java

```
import java.awt.geom.*;
import java.awt.image.*;

public class ImageINFO {
String filename;
BufferedImage img;
Affines affine;
float alpha;

//public IPLImageHistory animation;

//long duration;
double duration;

//public int flag; // next 0, below 1;

ImageINFO next;
ImageINFO prev;
ImageINFO list;

Affines anim;

public ImageINFO(BufferedImage img)
{
this.img = img;
```

```
affine = new Affines();

alpha = 1.0f;
//flag = -1;

next = prev = list = null;
}

public void updateAlpha(float al)
{
alpha += al;
if(alpha < 0.0f) alpha = 0.0f;
else if(alpha > 1.0f) alpha = 1.0f;
}

public void addTranslate(double x, double y)
{
affine.addTranslate(x, y);
}

public void addTranslate(AffineTransform at)
{
affine.addTranslate(at);
}

public void addScale(double ratio)
{
affine.addScale(ratio);
}

public void addScale(AffineTransform at)
{
affine.addScale(at);
}

public void addRotate(double degree)
{
affine.addRotate(degree);
}

public void addRotate(AffineTransform at)
{
affine.addRotate(at);
}

public AffineTransform getTransform()
{
AffineTransform ret = new AffineTransform();
ret.setToIdentity();
ret.concatenate(affine.getAffine());
ret.translate(-(img.getWidth()/2), -(img.getHeight()/2));
```

```
return ret;
}
}
```

B.2.3 src/IPLImageDisplay.java

```
import javax.swing.filechooser.FileFilter;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
//import java.io.FileFilter;
import javax.imageio.*;
import javax.swing.*;
import java.awt.geom.*;
import java.util.*;
import java.lang.*;

/**
 * This class demonstrates how to load an Image from an external file
 */
public class IPLImageDisplay extends Component implements Runnable{
//LinkedList imgs;
IPLImageHistory history = null;
ImageINFO imgHead, imgTail;
//Alpha alphaHead, alphaTail;

    int frame;
    int delay = 100;
    Thread animator;

    Dimension offDimension;
    Image offImage;
    Graphics offGraphics;

    public IPLImageDisplay(IPLImageHistory history, int flag)
    {
     if (flag == 1)
     { // display image
     setBackground(Color.white);
         this.history = history;

         init();

         animator = new Thread(this);
         animator.start();
     }
     else
     { // save Image
     this.history = history;
     init();
     JFileChooser chooser = new JFileChooser();
     String filename;
         //chooser.setDialogType(JFileChooser.SAVE_DIALOG);
         int returnVal = chooser.showSaveDialog(this);
         if(returnVal == JFileChooser.APPROVE_OPTION)
         {
          filename = chooser.getName(chooser.getSelectedFile());
          save(filename);
         }
         else
         {
          System.out.println("You didn't specify filename.");
         }

     }
    }

    private void init()
    {
     imgHead = imgTail = null;
    }

    public Dimension getPreferredSize()
    {
     return new Dimension(1024,600);
    }

    private void addImage(String filename)
    {
     BufferedImage img = null;
     //Alpha alpha = null;
     try {
         img = ImageIO.read(new File(filename));
         //alpha = new Alpha();
     } catch (IOException e) {
     System.err.println("<Error>Recognition: File Not Found ("+filename+")");
     System.exit(0);
     }

     ImageINFO tmp = new ImageINFO(img);
     tmp.filename = filename;
     if(imgHead == null)
     {
     imgHead = imgTail = tmp;
     //alphaHead = alphaTail = alpha;
     }
     else
     {
     imgTail.next = tmp;
     tmp.prev = imgTail;
```

```
     imgTail = tmp;
     }
    }

    private void addTransform(ImageINFO img, Affines animation)
    {
     img.addTranslate(animation.getTranslate());
     img.addScale(animation.getScale());
     img.addRotate(animation.getRotate());
     img.updateAlpha(animation.alpha);
    }

    private void addTransform(ImageINFO img, IPLImageHistory hs)
    {
     img.addTranslate(hs.x, hs.y);
img.addRotate(hs.rotate);
img.addScale(hs.scale);
img.updateAlpha(hs.alpha);
    }

    //public static void main(String[] args) {
    public static void displayImage(IPLImageHistory hs)
    {
        JFrame f = new JFrame("Load Image Sample");
        f.setBackground(Color.white);

        f.addWindowListener(new WindowAdapter()
        {
         public void windowClosing(WindowEvent e)
         {
         System.exit(0);
         }
        });

        f.add(new IPLImageDisplay(hs, 1));
        f.pack();
        f.setResizable(false);
        f.setVisible(true);
    }

    public static void saveImage(IPLImageHistory hs)
    {
     new IPLImageDisplay(hs, 2);
    }

    private void parseHistoryTree()
    {
if(history == null) return;

// calculate all animation stuffs
switch (history.nodeID)
```

```
{
case IPLImageHistory.DEFAULT:
init();
addImage(history.filename);
history.rotate = Math.toRadians(history.rotate);
addTransform(imgTail, history);

if(history.animation != null)
{
handleDEF(history);
}
break;
case IPLImageHistory.CHILDREN:
init();
handleCHILDREN(history.children);

break;
case IPLImageHistory.IMGLIST:
init();
handleIMGLIST(history);
     double i;
     Affines af = new Affines();
     ImageINFO iptr = imgTail;
     Affines aptr;
     while(iptr != null)
     {
     addTransform(iptr, af);
     aptr = iptr.anim;
     while(aptr != null)
     {
     for(i = 0; i < aptr.duration; i += delay)
     {
     //af.concatAll(aptr);
     af.addTranslate(aptr.translate);
     af.addRotate(aptr.rotate);
     af.addScale(aptr.scale);
     af.updateAlpha(aptr.alpha);
     }
     aptr = aptr.next;
     }
     iptr = iptr.list;
     }

//int i;
     for(i = 0; i <= history.imgNum; i++)
     history = history.next;
break;
default:
// ERROR
}
    }
```

```
private void save(String filename)
{
 ImageINFO iptr;
 Affines aptr;
 int i;
 String content = "", tmp1, tmp2;
 int totalImgNum = 0, iNum = 0, aNum = 0;
 double[] rMatrix = new double[6];

 while (history != null)
 {
 parseHistoryTree();
 iptr = imgHead;
 iNum = aNum = 0;
 tmp1 = "";

 while(iptr != null)
 {
 iNum++;

 tmp1 += iptr.filename
 + " " + iptr.alpha
 + " " + iptr.duration
 + " " + iptr.affine.translate.getTranslateX()
 + " " + iptr.affine.translate.getTranslateY()
 + " " + iptr.affine.scale.getScaleX();

 iptr.affine.rotate.getMatrix(rMatrix);
 for(i = 0; i < 6; i++)
 tmp1 += " " + rMatrix[i];

 tmp2 = "";
 aptr = iptr.anim;
 aNum = 0;
 while(aptr != null)
 {
 aNum++;
 tmp2 += aptr.alpha
 + " " + aptr.duration
 + " " + aptr.translate.getTranslateX()
 + " " + aptr.translate.getTranslateY()
 + " " + aptr.scale.getScaleX();

 aptr.rotate.getMatrix(rMatrix);
 for(i = 0; i < 6; i++)
     tmp2 += " " + rMatrix[i];

 tmp2 += " ";
 aptr = aptr.next;
 }
 tmp1 += " " + aNum + " " + tmp2;
 iptr = iptr.next;
 }
 totalImgNum += iNum + 1;
 content += iNum + " " + tmp1;

 if(history.nodeID == IPLImageHistory.IMGLIST) history = history.animation;
 else history = history.next;
 }

 content = (totalImgNum - 1) + " " + content;

 try {
 File file = new File(filename);
 file.createNewFile();
 FileWriter write = new FileWriter(file);

 write.write(content);
 write.close();
 }
 catch (IOException e) {
 System.err.println("<Error>Recognition: Cannot Create File ("+filename+")");
 System.exit(0);
 }

}

private ImageINFO handleDEF(IPLImageHistory hs)
{
 IPLImageHistory animation = hs.animation;
 Affines ptr;
 double ratio;

 ptr = imgTail.anim = new Affines();
 while(animation != null)
 {
 ptr.duration = animation.duration * 1000;
 imgTail.duration += ptr.duration;
 ratio = ptr.duration/delay;
 animation.x /= ratio;
 animation.y /= ratio;
 animation.rotate = Math.toRadians(animation.rotate)/ratio;
 animation.alpha /= ratio;
 animation.scale = Math.pow(animation.scale, 1/ratio);

 ptr.concatAll(animation);
 ptr.updateAlpha(animation.alpha);

 animation = animation.animation;
 if(animation != null)
```

```
     {
     ptr.next = new Affines();
     ptr = ptr.next;
     }
     }

     return imgTail;
    }

    private ImageINFO handleCHILDREN(IPLImageHistory hs)
    {
     //System.out.println("in handleChild: " + hs.filename);
     if(hs == null) return null;

     switch (hs.nodeID)
     {
     case IPLImageHistory.DEFAULT:
     addImage(hs.filename);
hs.rotate = Math.toRadians(hs.rotate);
addTransform(imgTail, hs);

if(hs.animation != null)
{
handleDEF(hs);
}

if(hs.next != null)
{
handleCHILDREN(hs.next);
}

break;
     case IPLImageHistory.CHILDREN:
     handleCHILDREN(hs.children);

     if(hs.next != null)
     {
     handleCHILDREN(hs.next);
     }

     break;
     case IPLImageHistory.IMGLIST:
     handleIMGLIST(hs);
     double i;
     Affines af = new Affines();
     ImageINFO iptr = imgTail;
     Affines aptr;
     while(iptr != null)
     {
     addTransform(iptr, af);
     aptr = iptr.anim;
     while(aptr != null)
     {
     for(i = 0; i < aptr.duration; i += delay)
     {
     //af.concatAll(aptr);
     af.addTranslate(aptr.translate);
     af.addRotate(aptr.rotate);
     af.addScale(aptr.scale);
     af.updateAlpha(aptr.alpha);
     }
     aptr = aptr.next;
     }
     iptr = iptr.list;
     }

     for(i = 0; i <= hs.imgNum; i++)
     hs = hs.next;
     if(hs.next != null)
     handleCHILDREN(hs.next);
     break;
     default:
     break;
     }
     return imgTail;
    }

    private ImageINFO addImage2(ImageINFO nImg, String filename)
    {
     BufferedImage img = null;

     try {
         img = ImageIO.read(new File(filename));
     } catch (IOException e) {
     System.err.println("<Error>Recognition: File Not Found ("+filename+")");
     System.exit(0);
     }

     ImageINFO tmp = new ImageINFO(img);
     tmp.filename = filename;

     nImg.list = tmp;

     return tmp;
    }

    private void handleIMGLIST(IPLImageHistory hs)
    {
     int index;
IPLImageHistory iptr, aptr;// = hs.animation, temp;
//IPLImageHistory tmp, temp;
iptr = hs.next;
```

```
ImageINFO ipptr;

hs.rotate = Math.toRadians(hs.rotate);
if(hs.imgNum >= 1)
{
addImage(iptr.filename);
iptr.rotate = Math.toRadians(iptr.rotate);
addTransform(imgTail, iptr);
//addTransform(imgTail, hs);
iptr = iptr.next;
}

ipptr = imgTail;

for(index = 1; index < hs.imgNum; index++)
{
ipptr = addImage2(ipptr, iptr.filename);
iptr.rotate = Math.toRadians(iptr.rotate);
addTransform(ipptr, iptr);
//addTransform(ipptr, hs);
iptr = iptr.next;
}

Affines ptr = null;
     double ratio;

     aptr = hs.animation;
     if(aptr != null)
     {
     ptr = imgTail.anim = new Affines();
     }

     ImageINFO tmp;// = imgTail;
     Affines tmp2;
     while (aptr != null)
{
     tmp = imgTail;
     ptr.duration = aptr.duration * 1000.0 / hs.imgNum;
     imgTail.duration += ptr.duration;
     ratio = aptr.duration * 1000.0 / delay;
     aptr.x /= ratio;
     aptr.y /= ratio;
     aptr.rotate = Math.toRadians(aptr.rotate)/ratio;
     aptr.alpha /= ratio;
     aptr.scale = Math.pow(aptr.scale, 1/ratio);

     ptr.concatAll(aptr);
     ptr.updateAlpha(aptr.alpha);

     for(index = 1; index < hs.imgNum; index++)
     {
```

```
     tmp = tmp.list;
     if(tmp.anim == null)
     {
     tmp2 = tmp.anim = ptr.clone();
     tmp2.duration = ptr.duration;
     tmp.duration += tmp2.duration;
     //int i;
     //for(i = 0; i < index * ptr.duration; i = i + delay)
     //{
     // tmp.addTranslate(tmp2.translate);
     // tmp.addRotate(tmp2.rotate);
     // tmp.addScale(tmp2.scale);
     // tmp.updateAlpha(tmp2.alpha);
     //}
     }
     else
     {
     tmp2 = tmp.anim;
     while(tmp2.next != null)
     {
     tmp2 = tmp2.next;
     }
     tmp2.next = ptr.clone();
     tmp2.duration = ptr.duration;
     tmp.duration += tmp2.duration;
     }
     }

     aptr = aptr.animation;
     if(aptr != null)
     {
     ptr.next = new Affines();
     ptr = ptr.next;
     }
}
    }

    /**
     * This method is called by the thread that was created in
     * the start method. It does the main animation.
     */
    public void run() {
// Remember the starting time
     double tm, duration = 0;
     ImageINFO iptr;

     while (Thread.currentThread() == animator)
     {
     parseHistoryTree();

     tm = System.currentTimeMillis();
     iptr = imgHead;
     duration = 0;
     while(iptr != null)
     {
     if(iptr.list != null)
     {
     ImageINFO ptr = iptr;
     double tmp = 0;
     while(ptr != null)
     {
     tmp += ptr.duration;
     ptr = ptr.list;
     }
     if (duration < tmp) duration = tmp;
     }
     else
     {
     if(duration < iptr.duration)
     duration = iptr.duration;
     }

     iptr = iptr.next;
     }

     //System.out.println("DURATION : " + duration);
     while(duration > 0)
     {
repaint();

try {
     tm += delay;
     Thread.sleep((long)Math.max(0, tm - System.currentTimeMillis()));
     } catch (InterruptedException e) {
     break;
     }
     duration -= delay;
}

     if(history != null)
     if(history.nodeID == IPLImageHistory.IMGLIST) history = history.animation;
     else history = history.next;

     //repaint();
     }
    }

    public void paint(Graphics g)
    {
     update(g);
    }
```

```
public void update(Graphics g) {
 Dimension d = getPreferredSize();//new Dimension(100,100);

 // Create the offscreen graphics context
 if ((offGraphics == null)
 || (d.width != offDimension.width)
 || (d.height != offDimension.height)) {
 offDimension = d;
 offImage = createImage(d.width, d.height);
 offGraphics = offImage.getGraphics();
 }

 // Erase the previous image
 offGraphics.setColor(getBackground());
 offGraphics.fillRect(0, 0, d.width, d.height);
 offGraphics.setColor(Color.black);

 // Paint the frame into the image
 paintFrame(offGraphics);

 // Paint the image onto the screen
 g.drawImage(offImage, 0, 0, null);
 //offGraphics.dispose();
}

public void paintFrame(Graphics g)
{
 if (imgHead == null)
 {
 return;
 }

 Graphics2D g2 = (Graphics2D)g;
 ImageINFO tmp = imgHead;
 AlphaComposite myAlpha;

 while(tmp != null)
 {
 if(tmp.anim != null && tmp.anim.duration > 0)
 {
 addTransform(tmp, tmp.anim);
 tmp.anim.duration -= delay;
 if(tmp.anim.duration <= 0)
 tmp.anim = tmp.anim.next;
 }
 else if (tmp.list != null)
 {
 if(tmp.prev != null) tmp.prev.next = tmp.list;
 imgHead = tmp.list;
 tmp.list.prev = tmp.prev;
 tmp.list.next = tmp.next;
```

```
     if(tmp.next != null) tmp.next.prev = tmp.list;
     tmp = tmp.list;

     if(tmp != null && tmp.anim != null)
     {
     System.out.println("I'm here~!");
     addTransform(tmp, tmp.anim);
         tmp.anim.duration -= delay;
         if(tmp.anim.duration <= 0)
         tmp.anim = tmp.anim.next;
     }
     }
     myAlpha = AlphaComposite.getInstance(AlphaComposite.SRC_OVER, tmp.alpha);
     g2.setComposite(myAlpha);
     g2.drawImage(tmp.img, tmp.getTransform(), null);

     tmp = tmp.next;
     }
    }
}
```

B.3 IPLDataType and Main Module

B.3.1 src/IPL.java

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import antlr.CommonAST;
import antlr.NoViableAltException;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

public class IPL {

public static void main(String[] args) {
String a = args[0].toString();
FileInputStream i = null;
System.out.println("IPL version 0.3 : Jianning Yue, Wookyun Kho, Young Jin Yoon");
System.out.println("Since 2006 All Copyrights Reserved.");
System.out.println();
try {
i = new FileInputStream(a);
IPLLexer l = new IPLLexer(i);
IPLParser p = new IPLParser(l);
p.statement();
AST a1 =  p.getAST();
IPLTreeWalker walker = new IPLTreeWalker();
walker.stmts((CommonAST)p.getAST());
} catch(NoViableAltException e) {
System.err.println("<Error>UnproperUsage"+ e.line+":"+e.column+": You cannot use "
                    +e.node.getText()+"in this context.");
            System.exit(0);
} catch( RecognitionException e ) {
            System.err.println( "<Error>Recognition " + e.line+":"+e.column+":"
            +e.getMessage());
            System.exit(0);
        } catch( TokenStreamException e ) {
            System.err.println( "<Error>TokenStream: " + e );
            System.exit(0);
        } catch( IOException e ) {
            System.err.println( "<Error>File Error: " + e );
            System.exit(0);
        }
}
}
```

B.3.2 src/IPLArray.java

```
import java.util.LinkedList;
import antlr.RecognitionException;

public class IPLArray extends IPLDataType {
    protected LinkedList list;

    public IPLArray(String text) {
super(text);
debug("IPLArray::IPLArray("+text+")");
list = new LinkedList();
}

    @SuppressWarnings("unchecked")
public void listAdd(IPLRange range, IPLDataType target) throws RecognitionException {
debug("IPLArray::listAdd("+range+","+target+")");
int from = range.getFrom();
int to = range.getTo();
int opt = range.getOpt();
// Phase 1: targetCheck
if(target instanceof IPLArray){
if(!(target.getClass().equals(this.getClass())))
    error("Array type is different -> src:"+type+"target:"+target.type);
} else {
if(!canBeElement(target)) error("Cannot assign "+target.type+" into "+type);
}
// Phase 2: rangeCheck
if(from == -1){ // no range ex) imgA
if(!(target.getClass().equals(this.getClass())))
      error("Cannot replace"+target.type+" into "+type);
else this.list = ((IPLArray)target).list;
return;
}
if(to == 0) {
switch(opt)
{
case IPLRange.NONE: // single range: ex) imgA[3] ==> should be already in array.
if(list.size() <= from) error("Array is out of index :"+from);
else list.set(from, target);
return;
case IPLRange.PLUS: // inserting range: ex) imgA[3+] ==> can be started from size.
if(list.size() < from) error("Array is out of index :"+from+",size:"+list.size());
else {
if(target instanceof IPLArray) list.addAll(from, ((IPLArray)target).list);
else list.add(from,target);
}
return;
case IPLRange.MINUS:
error("Cannot execute pop-up same time");
}
} else {
switch(opt)
{
```

```
case IPLRange.NONE: // ex) imgA[3~5] ==> array should have those elements
if(!(target instanceof IPLArray) || (list.size() <= to))
        error("Cannot put value into Array");
if((to-from+1) != ((IPLArray)target).list.size())
    error("Range out of Array. from:"+from+" to:"+to);
else {
for(int i=from; i<=to; i++) list.remove(from);
list.addAll(from,((IPLArray)target).list);
return;
}
case IPLRange.PLUS: // ex) imgA[3~5+] ==> array can be started from size
if(!(target instanceof IPLArray) || (list.size() < from))
        error("Cannot put value into Array");
if((to-from+1) > ((IPLArray)target).list.size())
       error("Cannot put smaller array into larger range. from:"+from+" to:"+to);
else {
for(int i=from; i<list.size(); i++) list.remove(from);
list.addAll(from,((IPLArray)target).list);
return;
}
case IPLRange.MINUS:
error("Cannot pop-up and assign at the same time");
}
}
    }

private boolean canBeElement(IPLDataType target) {
debug("IPLArray::canBeElement("+target+")");
if((target instanceof IPLBool && this instanceof IPLBoolArray) ||
   (target instanceof IPLNumber && this instanceof IPLNumberArray) ||
   (target instanceof IPLImage && this instanceof IPLImageArray) ||
   (target instanceof IPLCoord && this instanceof IPLCoordArray)) return true;
else return false;
}

public void listAdd(IPLDataType target){
debug("IPLArray::listAdd("+target+")");
    list.add(target);
    }


    public IPLDataType listGet(IPLRange range) throws RecognitionException{
     debug("IPLArray::listGet("+range+")");
     int from = range.getFrom();
int to = range.getTo();
int opt = range.getOpt();
LinkedList temp;
if(from == -1){
return this;
}
if(to == 0){
```

```
switch(opt)
{
case IPLRange.NONE: // ex) imgA[3]
if(from >= list.size()) throw new ArrayStoreException("Array out of Range.");
return (IPLDataType) list.get(from);
case IPLRange.MINUS:
  // ex) imgA[3-] ==> should be smaller than size. if size is 3, should be less than 2
if(from >= list.size()) throw new ArrayStoreException("Array out of Range.");
list.remove(from);
return this;
default: // ex) imgA[3+]
error("Cannot assign while getting array.");
return this;
}
}else{
switch(opt)
{
case IPLRange.NONE: // ex) imgA[3~5] ==> should be smaller than size.
if(to >= list.size()) error("Array out of Range.");
temp = new LinkedList();
for(int i=from; i<=to;i++) temp.add(((IPLDataType)list.get(i)));
list = temp;
return this;
case IPLRange.MINUS: // ex) imgA[3~5-] ==> should be smaller than size.
if(to >= list.size()) error("Array out of Range.");
for(int i=from; i<=to;i++) list.remove(from);
return this;
default: // ex) imgA[3~5+]
error("Cannot assign while getting array.");
return this;
}

}
}
    @SuppressWarnings("unchecked")
public IPLArray convert() throws RecognitionException
    {
     debug("IPLArray::convert()");
     IPLArray newArr = null;
     IPLDataType value = (IPLDataType) list.getFirst();
     IPLDataType nextVal;
     if(value instanceof IPLImage) newArr = new IPLImageArray(this.getId());
     else if(value instanceof IPLBool) newArr = new IPLBoolArray(this.getId());
     else if(value instanceof IPLNumber) newArr = new IPLNumberArray(this.getId());
     else if(value instanceof IPLCoord) newArr = new IPLCoordArray(this.getId());
     else IPLDataType.error("unknown type in array.");
     newArr.list.add(value);
     for(int i=1; i < list.size(); i++)
     {
     nextVal = (IPLDataType) list.get(i);
     if(!value.getClass().equals(nextVal.getClass()))
     IPLDataType.error("unknown type in array.");
     else newArr.list.add(nextVal);
     }
     return newArr;
    }

    @SuppressWarnings("unchecked")
public Object clone() throws CloneNotSupportedException
    {
     debug("IPLArray::Clone()");
     IPLArray newClone = (IPLArray) super.clone();
     IPLDataType target;
     LinkedList newList = new LinkedList();
     int size = list.size();
     for(int i=0; i < size; i++){
     target = (IPLDataType) list.get(i);
     if(target instanceof IPLImage) target = (IPLImage) ((IPLImage)target).clone();
     else target = (IPLDataType) target.clone();
     newList.add(target);
     }
     newClone.list = newList;
     return newClone;
    }

}
```

B.3.3 src/IPLBool.java

```
public class IPLBool extends IPLDataType {
public static final String type = "bool";
private boolean flag;


    public IPLBool(String text) {
     super(text);
     debug("IPLBool::IPLBool("+text+")");
     flag = false;
}

public IPLBool(boolean b) {
debug("IPLBool::IPLBool("+b+")");
flag = b;
}

public IPLBool and(IPLBool arg2){
     debug("IPLBool::and("+arg2+")");
        flag = flag && arg2.getValue();
        return this;
    }

    public IPLBool or(IPLBool arg2){
     debug("IPLBool::or("+arg2+")");
        flag = flag || arg2.getValue();
        return this;
    }

    public IPLBool not(){
     debug("IPLBool::not()");
        flag = !flag;
        return this;
    }
    public boolean getValue()
    {
       debug("IPLBool::getValue()");
           return flag;
    }

public IPLBool neq(IPLBool bool) {
       debug("IPLBool::neq("+bool+")");
flag = (bool.flag != flag);
return this;
}

public IPLBool eq(IPLBool bool) {
       debug("IPLBool::eq("+bool+")");
flag = (bool.flag == flag);
return this;
}
}
```

B.3.4 src/IPLBoolArray.java

```
public class IPLBoolArray extends IPLArray {
public static final String type = "bool[]";
public IPLBoolArray(String text) {
super(text);
       debug("IPLBoolArray::IPLBoolArray("+text+")");
}
}
```

B.3.5 src/IPLCoord.java

```
public class IPLCoord extends IPLDataType {
public static final String type = "coord";
private int x;
    private int y;

    public IPLCoord(String text) {
     super(text);
       debug("IPLCoord::IPLCoord("+text+")");
     x = 0;
     y = 0;
    }

public IPLCoord(IPLNumber a, IPLNumber b) {
       debug("IPLCoord::IPLCoord("+a+","+b+")");
x = a.getInt();
y = b.getInt();
}

public IPLCoord(int a, int b) {
       debug("IPLCoord::IPLCoord("+a+","+b+")");
x = a;
y = b;
}
    public IPLCoord setx(IPLNumber arg2){
       debug("IPLCoord::setx("+arg2+")");
     x = arg2.getInt();
     return this;
    }

    public IPLCoord sety(IPLNumber arg2){
       debug("IPLCoord::setx("+arg2+")");
     y = arg2.getInt();
        return this;
    }

    public IPLNumber xof(){
       debug("IPLCoord::xof()");
        return new IPLNumber(x);
    }

    public IPLNumber yof(){
       debug("IPLCoord::yof()");
        return new IPLNumber(y);
    }

public int getY() {
       debug("IPLCoord::getY()");
return y;
}
public int getX() {
       debug("IPLCoord::getX()");
return x;
}

}
```

B.3.6 src/IPLCoordArray.java

```
public class IPLCoordArray extends IPLArray {
public static final String type = "coord[]";
```

```
    public IPLCoordArray(String text) {
 super(text);
    debug("IPLCoordArray::IPLCoordArray("+text+")");
}
}
```

B.3.7 src/IPLDataType.java

```
import antlr.RecognitionException;

public class IPLDataType implements Cloneable{
protected static boolean isAnimated = false;
public static final String type ="rawType";
public static final boolean debug = false;
private String id;

    public  IPLDataType(String name){
     debug("IPLDataType::IPLDataType("+name+")");
     id = name;
    }

    public  IPLDataType(){
     debug("IPLDataType::IPLDataType()");
    }

    public static void setAnimate(boolean value)
    {
     debug("IPLDataType::setAnimate("+value+")");
     isAnimated = value;
    }
    public static boolean isAnimate()
    {
     debug("IPLDataType::isAnimate()");
     return isAnimated;
    }

    public String getId(){
     debug("IPLDataType::getId() ->"+id);
        return id;
    }

    public void setId(String newId){
     debug("IPLDataType::setId("+newId+")");
     id = newId;
    }

    public static void error(String msg) throws RecognitionException
    {
throw new RecognitionException(msg);
    }
    public Object clone() throws CloneNotSupportedException {
     return super.clone();
    }

    public static void debug(String target) {
     if(debug) System.out.println(target);
    }
}
```

B.3.8 src/IPLFunc.java

```
import java.util.Vector;

import antlr.RecognitionException;
import antlr.collections.AST;

public class IPLFunc extends IPLDataType {
    private AST body;
    private Vector args;
    private Vector ret;

    public IPLFunc(String name, Vector args, Vector retVal, AST body){
     super(name);
     debug("IPLFunc::IPLFunc("+name+","+args+","+retVal+","+body+")");
     this.args = args;
     ret = retVal;
     this.body = body;
    }
@SuppressWarnings("static-access")
public IPLDataType invoke(IPLTreeWalker walker, Vector argp) throws RecognitionException {
     debug("IPLFunc::invoke("+walker+","+argp+")");
IPLDataType asgnp, retVal = null;
String[] target = new String[2];
//Phase 1: enter scope
walker.ist.newScope(true);
//Phase 2: Assign variable in symboltable
if(argp != null){
for(int i=0; i < argp.size();i++) {
asgnp = (IPLDataType) argp.get(i);
target = (String[]) args.get(i);
if((target[1].equals("number") && asgnp instanceof IPLNumber) ||
   (target[1].equals("bool") && asgnp instanceof IPLBool) ||
   (target[1].equals("image") && asgnp instanceof IPLImage)||
   (target[1].equals("coord") && asgnp instanceof IPLCoord) ||
   (target[1].equals("IMAGE_ARRAY") && asgnp instanceof IPLImageArray)||
   (target[1].equals("BOOL_ARRAY") && asgnp instanceof IPLBoolArray) ||
   (target[1].equals("COORD_ARRAY") && asgnp instanceof IPLCoordArray) ||
   (target[1].equals("NUMBER_ARRAY") && asgnp instanceof IPLNumberArray))
{
asgnp.setId(target[0]);
walker.ist.asgnVar(asgnp);
}
else asgnp.error("cannot assign "+asgnp.type+" into "+target[1]);
}
}
if(ret != null){
for(int i=0; i < ret.size();i++) {
target = (String[]) ret.get(i);
if(target[1].equals("number")) retVal = new IPLNumber(target[0]);
else if(target[1].equals("bool")) retVal = new IPLBool(target[0]);
else if(target[1].equals("image")) retVal = new IPLImage(target[0],"<NULL>");
else if(target[1].equals("coord")) retVal = new IPLCoord(target[0]);
else if(target[1].equals("IMAGE_ARRAY"))retVal = new IPLImageArray(target[0]);
else if(target[1].equals("BOOL_ARRAY"))retVal = new IPLBoolArray(target[0]);
else if(target[1].equals("COORD_ARRAY"))retVal = new IPLCoordArray(target[0]);
else if(target[1].equals("NUMBER_ARRAY"))retVal = new IPLNumberArray(target[0]);
else IPLDataType.error("cannot assign return value"+target[1]);
walker.ist.asgnVar(retVal);
}
} else retVal = null;
//Phase 3: Invoke function!
try {
walker.invoke(body);
} catch (RecognitionException e) {
// TODO Auto-generated catch block
e.printStackTrace();
System.exit(1);
}
if(retVal != null) retVal = walker.ist.getVar(retVal.getId(),
new IPLRange(retVal.getId()));
//Phase 4: exit scope
walker.ist.exitScope();
return retVal;
}

}
```

B.3.9 src/IPLImage.java

```
import antlr.RecognitionException;

public class IPLImage extends IPLDataType {
public static final String type = "image";

private IPLImageHistory history;

public IPLImage(String var, String con) {
super(var);
 debug("IPLImage::IPLImage("+var+","+con+")");
 history = new IPLImageHistory(con);
}
```

```
public IPLImage concat(IPLImage arg2) throws RecognitionException{
debug("IPLImage::concat("+arg2+")");
     IPLImageHistory modify = new IPLImageHistory(IPLImageHistory.CHILDREN);
     IPLImageHistory temp;
if(isAnimated) error("cannot use concat operator in animate()");
else {
modify.children = history;
if(history.nodeID == IPLImageHistory.IMGLIST)
{
temp = history.recentNext();
temp.next = arg2.history;
}
else history.next = arg2.history;
     history = modify;
  }
        return this;
    }

    public IPLImage scale(IPLNumber arg2){
     debug("IPLImage::scale("+arg2+")");
     if(isAnimated) history.recentTimeInfo().scale *= arg2.getDouble();
     else history.scale *= arg2.getDouble();
        return this;
    }

    public IPLImage rotate(IPLNumber arg2){
     debug("IPLImage::rotate("+arg2+")");
     if(isAnimated) history.recentTimeInfo().rotate += arg2.getDouble();
     else history.rotate += arg2.getDouble();
        return this;
    }

    public IPLImage set(IPLCoord arg2){
     debug("IPLImage::set("+arg2+")");
     if(isAnimated){
     history.setAnimatedX(arg2.getX());
     history.setAnimatedY(arg2.getY());
     } else {
     history.x = arg2.xof().getInt();
     history.y = arg2.yof().getInt();
     }
        return this;
    }


public IPLImage animate(IPLNumber time) throws RecognitionException{
debug("IPLImage::animate("+time+")");
if(!isAnimated) error("FIXME!: Unknown Error");
if(history.children != null)
    error("Cannot use animate() with concatenated image.");
history.recentTimeInfo().duration = time.getInt();
return this;
    }

public IPLImage alpha(IPLNumber d) {
     debug("IPLImage::alpha("+d+")");
     double a = d.getDouble();
     if(a > 100) a = 100;
     else if (a < -100) a = -100;
     a = a * 0.01f;
     if(isAnimated) history.recentTimeInfo().alpha = (float)a;
     else history.alpha = (float)a;
return this;
}

public Object clone() throws CloneNotSupportedException{
     debug("IPLImage::clone()");
IPLImage a = (IPLImage) super.clone();
a.history = (IPLImageHistory) a.history.clone();
return a;
}

public void createTimeInfo() throws RecognitionException
{
     debug("IPLImage::createTimeInfo()");
history.createTimeInfo();
}

public IPLImageHistory getHistory()
{
     debug("IPLImage::getHistory()");
return history;
}
public void setHistory(IPLImageHistory incoming)
{
     debug("IPLImage::setHistory("+incoming+")");
history = incoming;
}
}
```

B.3.10 src/IPLImageArray.java

```
import antlr.RecognitionException;

public class IPLImageArray extends IPLArray {
public static final String type = "image[]";

private IPLImageHistory history;

    public IPLImageArray(String text) {
     super(text);
     history = new IPLImageHistory(IPLImageHistory.TIMEINFO);
     debug("IPLImageArray::IPLImageArray("+text+")");
}
    public IPLImage animate(IPLNumber time) throws RecognitionException{
     debug("IPLImageArray::animate("+time+")");
     if (list == null || list.size() == 0)
            error("Cannot use animate() with empty array.");
     IPLImageHistory old, temp;
     IPLImage init = (IPLImage) list.getFirst();
     if(!(init instanceof IPLImage))
            error("elements in array are not image type");

     old = init.getHistory();
     temp = new IPLImageHistory(IPLImageHistory.IMGLIST);
     temp.next = old;
     history.duration = time.getInt();
     temp.animation = history;
temp.imgNum = list.size(); // including itself!
//     history.next = old;
     if(old.children != null)
            error("Cannot use Concatenated image into frame animation");
if(isComplexImage(old))
       error("Cannot convert complex image element into frame animation.");
     for(int i=1; i< list.size();i++)
     {
     old.next = ((IPLImage)list.get(i)).getHistory();
     if(isComplexImage(old.next))
        error("Cannot convert complex image element into frame animation");
     old = old.next;
     }
     init.setHistory(temp);
//     init.setHistory(history);
        return init;
    }

    private boolean isComplexImage(IPLImageHistory incoming)
    {
     if( (incoming.nodeID != IPLImageHistory.DEFAULT) ||
     (incoming.animation != null) ||
     (incoming.children != null) ||
     (incoming.next != null)) return true;
     else return false;
    }
public IPLDataType rotate(IPLNumber d) throws RecognitionException {
if(!isAnimated) error("FIXME: cannot be invoked!");
history.rotate += d.getDouble();
return this;
}
public IPLDataType set(IPLCoord c) throws RecognitionException {
if(!isAnimated) error("FIXME: cannot be invoked!");
history.x = c.getX();
history.y = c.getY();
```

```
return this;
}
public IPLDataType scale(IPLNumber d) throws RecognitionException {
if(!isAnimated) error("FIXME: cannot be invoked!");
history.scale *= d.getDouble();
return this;
}
public IPLDataType alpha(IPLNumber d) throws RecognitionException {
     double a = d.getDouble();
     if(a > 100) a = 100;
     else if (a < -100) a = -100;
     a = a * 0.01f;
     if(isAnimated) history.alpha = (float)a;
     else error("FIXME: cannot be invoked!");
return this;
}
}
```

B.3.11 src/IPLImageHistory.java

```
public class IPLImageHistory implements Cloneable{
static final int DEFAULT = 0;
static final int CHILDREN = 1;
static final int TIMEINFO = 2;
static final int IMGLIST = 3;

int nodeID;
int imgNum;
IPLImageHistory children = null;
IPLImageHistory next = null;
// IPLImageHistory below = null;
IPLImageHistory animation = null;

String filename;
double x;
double y;
double scale;
double rotate;
float alpha;
double duration;

public IPLImageHistory(int nodeID)
{
this.nodeID = nodeID;
filename = null;
imgNum = 0;
x = y = rotate = 0;
if(nodeID == TIMEINFO) alpha = 0;
else alpha = 1;
duration = 0;
```

```
scale = 1;
}

public IPLImageHistory(String filename)
{
nodeID = DEFAULT;
this.filename = filename;
imgNum = 0;
x = y = rotate = 0;
alpha = 1;
duration = 0;
scale = 1;
}
public Object clone() throws CloneNotSupportedException
{
IPLImageHistory newClone = (IPLImageHistory) super.clone();
if(newClone.children != null)
newClone.children = (IPLImageHistory) newClone.children.clone();
if(newClone.next != null)
newClone.next = (IPLImageHistory) newClone.next.clone();
if(newClone.animation != null)
newClone.animation = (IPLImageHistory) newClone.animation.clone();
return newClone;
}
public IPLImageHistory recentTimeInfo()
{
if(animation == null) return this;
else return animation.recentTimeInfo();
}
public IPLImageHistory createTimeInfo()
{
if(animation == null)
{
animation = new IPLImageHistory(TIMEINFO);
return animation;
}
else return animation.createTimeInfo();
}
public IPLImageHistory recentNext()
{
if(next == null) return this;
else return next.recentNext();
}
public void setAnimatedX(double x)
{
if(animation == null) this.x = x;
else animation.setAnimatedX(x-this.x);
}
public void setAnimatedY(double y)
{
if(animation == null) this.y = y;
```

```
else animation.setAnimatedY(y-this.y);
}

}
```

B.3.12 src/IPLNumber.java

```
public class IPLNumber extends IPLDataType {
public static final String type = "number";
private double value;

    public IPLNumber(String text) {
     super(text);
     debug("IPLNumber::IPLNumber("+text+")");
     value = 0;
    }

public IPLNumber(double d) {
     debug("IPLNumber::IPLNumber("+d+")");
value = d;
}

public IPLNumber add(IPLNumber arg2){
     debug("IPLNumber::add("+arg2+")");
       value +=arg2.getDouble();
       return this;
    }

    public IPLNumber subtract(IPLNumber arg2){
     debug("IPLNumber::subtract("+arg2+")");
       value -=arg2.getDouble();
        return this;
    }

    public IPLNumber multiply(IPLNumber arg2){
     debug("IPLNumber::multiply("+arg2+")");
       value *=arg2.getDouble();
        return this;
    }

    public IPLNumber divide(IPLNumber arg2){
        debug("IPLNumber::divide("+arg2+")");
        value /=arg2.getDouble();
        return this;
    }

    public IPLNumber modulo(IPLNumber arg2){
        debug("IPLNumber::modulo("+arg2+")");
        value %=arg2.getDouble();
        return this;
    }
```

```
public IPLBool le(IPLNumber arg2){
    debug("IPLNumber::le("+arg2+")");
 if(value <= arg2.getDouble()) return new IPLBool(true);
 else return new IPLBool(false);
}

public IPLBool ge(IPLNumber arg2){
    debug("IPLNumber::ge("+arg2+")");
 if(value >= arg2.getDouble()) return new IPLBool(true);
 else return new IPLBool(false);
}

public IPLBool lt(IPLNumber arg2){
    debug("IPLNumber::lt("+arg2+")");
 if(value < arg2.getDouble()) return new IPLBool(true);
 else return new IPLBool(false);
}

public IPLBool gt(IPLNumber arg2){
    debug("IPLNumber::gt("+arg2+")");
 if(value > arg2.getDouble()) return new IPLBool(true);
 else return new IPLBool(false);
}

public IPLBool eq(IPLNumber arg2){
    debug("IPLNumber::eq("+arg2+")");
 if(value == arg2.getDouble()) return new IPLBool(true);
 else return new IPLBool(false);
}

public IPLBool neq(IPLNumber arg2){
    debug("IPLNumber::neq("+arg2+")");
 if(value != arg2.getDouble()) return new IPLBool(true);
 else return new IPLBool(false);
}

public IPLNumber negative(){
    debug("IPLNumber::negative()");
    value = -value;
    return this;
}
public double getDouble()
{
    debug("IPLNumber::getDouble()");
 return value;
}
public int getInt()
{
    debug("IPLNumber::getInt()");
 return (int)value;
```

```
    }
}
```

B.3.13 src/IPLNumberArray.java

```
public class IPLNumberArray extends IPLArray {
public static final String type = "number[]";

    public IPLNumberArray(String text) {
     super(text);
        debug("IPLNumberArray::IPLNumberArray("+text+")");
}

}
```

B.3.14 src/IPLRange.java

```
public class IPLRange extends IPLDataType {

public static final int NONE = 0;
public static final int PLUS = 1;
    public static final int MINUS = 2;
    private int from;
    private int to;
    private int option;

public IPLRange(String string) {
super(string);
debug("IPLRange::IPLRange("+string+")");
from = -1;
to = -1;
option = NONE;
}
public IPLRange(IPLNumber a) {
debug("IPLRange::IPLRange("+a.getInt()+")");
from = a.getInt();
to = 0;
option = 0;
}
public IPLRange(IPLNumber a, IPLNumber b) {
debug("IPLRange::IPLRange("+a.getInt()+","+b.getInt()+")");
from = a.getInt();
to = b.getInt();
option = NONE;
}
    public void setOpt(int opt) {
debug("IPLRange::setOpt("+opt+")");
     if(opt >= NONE && opt <=MINUS) option = opt;
     else option = 0;
}
public int getFrom() {
```

```
debug("IPLRange::getFrom()");
return from;
}
public int getTo() {
debug("IPLRange::getTo()");
return to;
}
public int getOpt() {
debug("IPLRange::getOpt()");
return option;
}
}
```

B.3.15 src/IPLSymbolTable.java

```
import java.util.Hashtable;
import java.util.LinkedList;

import antlr.RecognitionException;
import antlr.collections.AST;

public class IPLSymbolTable {

class Scope {
public Hashtable thisScope;
boolean isFunc;
boolean inWhile;
}
private Hashtable globalScope;
private Hashtable func;
private LinkedList scopeList;
private Scope currentScope;

public IPLSymbolTable()
{
     IPLDataType.debug("IPLSymbolTable::IPLSymbolTable(no args)");
     globalScope = new Hashtable();
     func = new Hashtable ();
     scopeList = new LinkedList();
     currentScope = new Scope();
     currentScope.thisScope = globalScope;
     currentScope.isFunc = false;
     currentScope.inWhile = false;
}
    public void asgnVar(IPLDataType value)
     throws RecognitionException{
     IPLDataType.debug("IPLSymbolTable::asgnVar+("
     +value.getId()+")");
     if(value.getId() == null)
try {
throw new Exception("target variable is null");
```

```
} catch (Exception e) {
e.printStackTrace();
System.exit(1);
}
if(currentScope.thisScope.get(value.getId()) != null)
IPLDataType.error("redefined variable "+value.getId()
+" as "+value.type);
     currentScope.thisScope.put(value.getId(), value);
    }

    public boolean isGlobalScope()
    {
     return currentScope.thisScope == globalScope;
    }
    public boolean isWhileLoop()
    {
     if(currentScope.thisScope == globalScope)
     {
     if(!currentScope.inWhile) return false;
     else return true;
     }
     if(currentScope.isFunc) return false;
     else if(!currentScope.inWhile) {
     for(int i=0; i<scopeList.size(); i++)
     {
     Scope s = (Scope) scopeList.get(i);
     if(s.isFunc) return false;
     else if(s.inWhile) return true;
     }
     return false;
     }
     return currentScope.inWhile;
    }
    public void setWhileLoop(boolean value)
    {
     currentScope.inWhile = value;
    }
    public IPLDataType getVar(String name)
     throws RecognitionException{
     IPLDataType.debug("IPLSymbolTable::getVar("+name+")");
     IPLDataType retVal = (IPLDataType) currentScope.thisScope.get(name);
     if(retVal == null)
     {
     if(currentScope.thisScope == globalScope)
     IPLDataType.error("cannot find variable "+name);
     if(currentScope.isFunc)
     {
     retVal = (IPLDataType) globalScope.get(name);
     if(retVal == null)
     IPLDataType.error("cannot find variable "+name);
     }
     for(int i=0; i<scopeList.size(); i++)
     {
     Scope s = (Scope) scopeList.get(i);
     retVal = (IPLDataType) s.thisScope.get(name);
     if(retVal != null) break;
     if(s.isFunc)
     {
     retVal = (IPLDataType) globalScope.get(name);
         if(retVal == null)
         IPLDataType.error("cannot find variable "+name);
     }
     }
        if(retVal == null)
        IPLDataType.error("cannot find variable "+name);
    }
     try {
     if(retVal instanceof IPLArray)
     {
     retVal = (IPLDataType)((IPLArray)retVal).clone();
     }
     else if(retVal instanceof IPLImage){
     retVal = (IPLDataType) ((IPLImage) retVal).clone();
     if(IPLDataType.isAnimated)
     ((IPLImage) retVal).createTimeInfo();
     }
     else retVal = (IPLDataType) retVal.clone();
} catch (CloneNotSupportedException e) {
e.printStackTrace();
}
     return retVal;
    }

    public IPLDataType getVar(String text, IPLRange target)
     throws RecognitionException {
     IPLDataType.debug("IPLSymbolTable::getVar("+text+","+target+")");
IPLDataType array = getVar(text);
if(array instanceof IPLArray)
array = ((IPLArray) array).listGet(target);
if(array == null)
IPLDataType.error("cannot get array instance from array "
+array.type);
     return array;
}

    public void asgnFunc(String name, IPLFunc value){
     IPLDataType.debug("IPLSymbolTable::asgnFunc("+name+","+value+")");
     func.put(name, value);
    }

    public IPLFunc getFunc(String name){
     IPLDataType.debug("IPLSymbolTable::getFunc("+name+")");
```

```
        return (IPLFunc) func.get(name);
    }

    @SuppressWarnings("unchecked")
public boolean setVar(String name, IPLDataType value)
throws RecognitionException{
     IPLDataType.debug("IPLSymbolTable::setVar("+name+","
     +value.toString()+")");
     if(name == null)
IPLDataType.error("target variable is null");
value.setId(name);
     IPLDataType retVal
     = (IPLDataType) currentScope.thisScope.get(name);
     if(retVal == null)
     {
     if(currentScope.thisScope == globalScope)
     IPLDataType.error("cannot find variable "+name);
     if(currentScope.isFunc)
     {
     retVal = (IPLDataType) globalScope.get(name);
     if(retVal == null)
     IPLDataType.error("cannot find variable "+name);
     else globalScope.put(name, value);
     return true;
     }
     for(int i=0; i<scopeList.size(); i++)
     {
     Scope s = (Scope) scopeList.get(i);
     retVal = (IPLDataType) s.thisScope.get(name);
     if(retVal != null)
     {
     s.thisScope.put(name, value);
     return true;
     }
     if(s.isFunc)
     {
     retVal = (IPLDataType) globalScope.get(name);
         if(retVal == null)
         IPLDataType.error("cannot find variable "+name);
     else globalScope.put(name, value);
         return true;
     }
     }
        if(retVal == null)
        IPLDataType.error("cannot find variable "+name);
        return false;
    }
     currentScope.thisScope.put(name, value);
        return true;
    }
    @SuppressWarnings("unchecked")
public void newScope(boolean isFunc) // for if
{
IPLDataType.debug("IPLSymbolTable::newScope("+isFunc+")");
scopeList.addFirst(currentScope);
currentScope = new Scope();
currentScope.thisScope = new Hashtable();
currentScope.isFunc = isFunc;
}

    public void exitScope(){
     IPLDataType.debug("IPLSymbolTable::exitScope()");
     currentScope = (Scope)scopeList.getFirst();
    }


    public void export(IPLDataType target)
     throws RecognitionException{
     IPLDataType.debug("IPLSymbolTable::export()");
     if(target instanceof IPLImage)
     IPLImageDisplay.saveImage(((IPLImage)target).getHistory());
     else IPLDataType.error("Cannot export if it is not an image type.");
    }

    public void display(IPLDataType target) throws RecognitionException{
     IPLDataType.debug("IPLSymbolTable::display()");
     if(target instanceof IPLImage)
     IPLImageDisplay.displayImage(((IPLImage)target).getHistory());
     if(target instanceof IPLBool)
     System.out.println(((IPLBool)target).getValue());
     if(target instanceof IPLNumber)
     System.out.println(((IPLNumber)target).getDouble());
     if(target instanceof IPLCoord)
     System.out.println("("+((IPLCoord)target).getX()+","
     +((IPLCoord)target).getY()+")");
     if(target instanceof IPLArray)
     IPLDataType.error("Cannot display an array.");
    }


}
```

B.4 JAVA 2D Applet Module

B.4.1 src/IPLApplet.java

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import java.util.Scanner;

import javax.imageio.*;
import javax.swing.*;

import java.awt.geom.*;

public class IPLApplet extends java.applet.Applet implements Runnable{
IPLImageHistory history = null;
ImageINFO imgHead, imgTail;

    int frame;
    int delay = 50;
    Thread animator;

    AffineTransform trans = new AffineTransform();
    AffineTransform scale = new AffineTransform();
    AffineTransform rotate = new AffineTransform();
    //AffineTransform at4 = new AffineTransform();

    Dimension offDimension;
    Image offImage;
    Graphics offGraphics;

    public void init()
    {
     //String str = getParameter("fps");
     //int fps = (str != null) ? Integer.parseInt(str) : 10;
     //delay = (fps > 0) ? (1000 / fps) : 100;
     delay = 50;

     imgHead = imgTail = null;

     //String iplFile = getParameter("ipl");
     String iplFile = "IPLoutput.ipl";
     loadImage(iplFile);

     resize(1000, 600);

     trans.setToIdentity();
     scale.setToIdentity();
     rotate.setToIdentity();
    }
```

```
public void start()
{
 animator = new Thread(this);
 animator.start();
}

private void loadImage(String filename)
{
 try {
 //Scanner scan = new Scanner(new File(getCodeBase().toExternalForm() + filename), " ");
 Scanner scan = new Scanner(new File(filename));

 int totalImgNum = scan.nextInt();
 ImageINFO iptr;
     Affines aptr = null;
     int iNum = 0, aNum = 0;
     String name;

     while(totalImgNum > 0)
     {
     iNum = scan.nextInt();
     totalImgNum -= iNum + 1;
     while(iNum > 0)
     {
     iNum--;
     name = scan.next();
         addImage(name);
         iptr = imgTail;
         iptr.alpha = scan.nextFloat();
         iptr.duration = scan.nextDouble();
         iptr.affine.addTranslate(scan.nextDouble(), scan.nextDouble());
         iptr.affine.addScale(scan.nextDouble());
         iptr.affine.rotate.setTransform(scan.nextDouble(), scan.nextDouble(),
         scan.nextDouble(), scan.nextDouble(), scan.nextDouble(), scan.nextDouble());

         aNum = scan.nextInt();

         if(aNum > 0) aptr = iptr.anim = new Affines();
         while(aNum > 0)
         {
         aNum--;
         aptr.alpha = scan.nextFloat();
         aptr.duration = scan.nextDouble();
         aptr.addTranslate(scan.nextDouble(), scan.nextDouble());
         aptr.addScale(scan.nextDouble());
         aptr.rotate.setTransform(scan.nextDouble(), scan.nextDouble(), scan.nextDouble(),
         scan.nextDouble(), scan.nextDouble(), scan.nextDouble());
         if(aNum > 0)
         {
         aptr.next = new Affines();
```

```
        aptr = aptr.next;
        }
        }
    }

    if(totalImgNum <= 0) break;
    addImage(null);
    }
 }
 catch (IOException e) {
 System.err.println("<Error>Recognition: File Not Found ("+filename+")");
 System.exit(0);
 }
}

public Dimension getPreferredSize()
{
 return new Dimension(1000,600);
}

public void stop()
{
 animator = null;
 offImage = null;
 offGraphics = null;
}

private void addImage(String filename)
{
 BufferedImage img = null;
 ImageINFO tmp = null;

 if (filename != null)
 {
    //Alpha alpha = null;
    try {
        img = ImageIO.read(new File(filename));
        //alpha = new Alpha();
    } catch (IOException e) {
    System.err.println("<Error>Recognition: File Not Found ("+filename+")");
    System.exit(0);
    }

    tmp = new ImageINFO(img);
 }
 else tmp = new ImageINFO(null);

 tmp.filename = filename;
 if(imgHead == null)
 {
 imgHead = imgTail = tmp;
 //alphaHead = alphaTail = alpha;
 }
 else
 {
 imgTail.next = tmp;
 tmp.prev = imgTail;
 imgTail = tmp;

 //alphaTail.next = alpha;
 //alphaTail = alpha;
 }
}

private void addTransform(ImageINFO img, Affines animation)
{
 img.addTranslate(animation.getTranslate());
 img.addScale(animation.getScale());
 img.addRotate(animation.getRotate());
}

/**
 * This method is called by the thread that was created in
 * the start method. It does the main animation.
 */
public void run()
{
 // Remember the starting time
 long tm = System.currentTimeMillis();
 while (Thread.currentThread() == animator) {
     // Display the next frame of animation.
     repaint();

     // Delay depending on how far we are behind.
     try {
      tm += delay;
      Thread.sleep(Math.max(0, tm - System.currentTimeMillis()));
     } catch (InterruptedException e) {
      break;
     }

     // Advance the frame
     frame++;
 }
}

public void paint(Graphics g)
{
 update(g);
}

public void update(Graphics g)
```

```
{
 Dimension d = getPreferredSize();//new Dimension(100,100);

 // Create the offscreen graphics context
 if ((offGraphics == null)
 || (d.width != offDimension.width)
 || (d.height != offDimension.height)) {
 offDimension = d;
 offImage = createImage(d.width, d.height);
 offGraphics = offImage.getGraphics();
 }

 // Erase the previous image
 offGraphics.setColor(getBackground());
 offGraphics.fillRect(0, 0, d.width, d.height);
 offGraphics.setColor(Color.black);

 // Paint the frame into the image
 paintFrame(offGraphics);

 // Paint the image onto the screen
 g.drawImage(offImage, 0, 0, null);
 //offGraphics.dispose();
}

public void paintFrame(Graphics g)
{
 if (imgHead == null)
 {
 //if (Thread.currentThread() == animator)
 //System.out.println("No Image");

 return;
 }

 Graphics2D g2 = (Graphics2D)g;
 ImageINFO tmp = imgHead;
 AlphaComposite myAlpha;
 boolean flag = true;

 while(tmp != null && tmp.filename != null)// && tmp.duration > 0)
 {
 if(tmp.anim != null && tmp.anim.duration > 0)
 {
 flag = false;
 addTransform(tmp, tmp.anim);
 tmp.updateAlpha(tmp.anim.alpha);
 tmp.anim.duration -= delay;
 if(tmp.anim.duration <= 0)
 tmp.anim = tmp.anim.next;
 }
```

```
      //System.out.println("ALPHA : " + tmp.alpha);
      myAlpha = AlphaComposite.getInstance(AlphaComposite.SRC_OVER, tmp.alpha);
      g2.setComposite(myAlpha);
      g2.drawImage(tmp.img, tmp.getTransform(), null);

      tmp = tmp.next;
      }

      if(flag && tmp != null)
      {
      imgHead = tmp.next;
      }
     }
}
```

B.5 Testing cases and samples

B.5.1 testingcase/animate.ipl

```
image[] imgListA , imgListB;
image imgA;
imgListA[0+] = "strawberry.jpg"^2;
imgListA[1+] = "apple.jpg";
imgListA[2~5+] = {"pear_red.jpg","tomato.jpg","melon.jpg","orange.jpg"};
imgA = animate(imgListA.(600,400),50);
display(imgA);
```

B.5.2 testingcase/burger.ipl

```
image a = "sshield.jpg" ,a1;
image b = "cucs.jpg", b1;
image c = "edwards.jpg", c1;
image d = "burger.jpg",d1;

a1 = a.(512,300):-100;
b1 = b.(600,300);
c1 = c.(100,600);
d1 = d.(800,600);


a1 = animate(a1^5@20:+100,10);
b1 = animate(b1^3:-300,10);
c1 = animate(c1.(600,100):-300,10);
d1 = animate(d1.(100,100):100,10);

display(a1 $ b1 $ c1 $ d1);
```

B.5.3 testingcase/coordinate.ipl

```
number a = 2, b = 2;
coord c;
c = (foo(a,b),bar(a,b));
display(c);
```

B.5.4 testingcase/defination1.ipl

```
image imgA, imgB;

imgA = "sshield.jpg";

image imgC = "sshield.jpg";
```

B.5.5 testingcase/defination2.ipl

```
image[] imgListA;

image[] imgListC = {"apple.jpg"}, imgListB;
```

```
imgListA = {"pear_red.jpg","apple.jpg","orange.jpg"};
```

B.5.6 testingcase/display.ipl

```
image imgA, imgB;
imgA = sshield.jpg.(400,200);
imgB =(imgA$imgA@90 )#(imgA@270$imgA@180);
display(imgB);
```

B.5.7 testingcase/fading.ipl

```
image imgA = "sshield.jpg".(400,300);

imgA = animate(imgA:-300, 5);

display(imgA:-40);
```

B.5.8 testingcase/function.ipl

```
image imgA = "pear_red.jpg";
image imgB = "apple.jpg";

defunc showall ( image imgS, image imgQ ) image imgR
{
    bool b = true;
    number p=100;
    number i=0;
    image[] imgC;
    while (i<5)
    {
        if(i%2 ==1) {imgC[i+] = imgS.(p,p);}
        else {imgC[i+] = imgQ.(p,p);}
        p=p+100;
        i = i+1;
    }
    image imgD = animate(imgC.(800,400):-100,18);
    display(imgD);
}
showall( imgA, imgB );
```

B.5.9 testingcase/functiondefinition.ipl

```
defunc foo(number a, number b) number c { c = bar(a,b) + baz(a,b); }
defunc bar(number a, number b) number c { c = a * b; }
defunc baz(number a, number b) number c { if(a == b) {c = a*b;} else {c = b;} }
```

B.5.10 testingcase/imageassignment.ipl

```
image[] imgListA;

imgListA = {"pear_red.jpg","apple.jpg","orange.jpg"};
```

```
imgListA[0] = "strawberry.jpg";
```

B.5.11 testingcase/imageinserting.ipl

```
image[] imgListA;

imgListA = {"pear_red.jpg","apple.jpg","orange.jpg"};

imgListA[2~4+] = {"pear_red.jpg","apple2.gif","orange.jpg"};
```

B.5.12 testingcase/imagelistadding2.ipl

```
image[] imgListA;

imgListA = {"pear_red.jpg","apple.jpg","orange.jpg"};

imgListA[1~3] = { "apple.jpg","pear_red.jpg","orange.jpg" };

imgListA[1~3]={"apple.jpg","pearred.jpg","orange.jpg","apple.jpg" };
```

B.5.13 testingcase/imagelistadding.ipl

```
image[] imgListA;

imgListA = {"pear_red.jpg","apple.jpg","orange.jpg"};

imgListA[1~3] = { "apple.jpg","pear_red.jpg","orange.jpg" };

imgListA[2+] ={"strawberry","apple.jpg"};
```

B.5.14 testingcase/joining2.ipl

```
imgA = (imgA$imgA@90 )#(imgA@270$imgA@180);

imgB[3] = imgA.(0,1):30 @ 120 ;
```

B.5.15 testingcase/joining.ipl

```
image imgA, imgB;

imgA = "sshield.jpg".(400,200);

imgB=(imgA$imgA@90 )#(imgA@270$imgA@180);
```

B.5.16 testingcase/positioning.ipl

```
image imgA = "sshield.jpg".(300,200);

display(imgA);
display(imgA.(700,400);
```

B.5.17 testingcase/rotating.ipl

```
image imgA = "sshield.jpg".(400,300);

display(imgA);

display(imgA@90);
```

B.5.18 testingcase/scaling.ipl

```
image imgA = "sshield.jpg".(400,300);

display(imgA);

display(imgA^2);
```

B.5.19 testingcase/strawberry.ipl

```
image a = "strawberry.jpg";
image b = "strawberry.jpg";
image c, d;
a = a.(512,300):-100;
b = b.(600,300)^10;
c = animate(a^5@20:+100,2);
d = animate(b^0.001@30:-300,3);
display(c $ d);
```

B.5.20 testingcase/animate2.ipl

```
image[] imgListA , imgListB;
number a = 3;
image imgA = "strawberry.jpg";
imgA = imgA.(300,100);
image imgB = "car.gif";
image imgC = "strawberry.jpg";
imgB = imgB.(300,100);
imgListA[0+] = imgA;
imgListA[1+] = imgB;
imgListA[2+] = imgA;
imgListA[3+] = imgB;
imgListA[4+] = imgA;
imgListA[5+] = imgB;
imgListA[6+] = imgA;
imgListA[7+] = imgB;
imgA = animate(imgListA.(400,300)@960^10,1);
imgB = imgB:-100;
imgC = imgC:-80;
display(imgA);
```

B.5.21 testingcase/while.ipl

```
    bool bu = true;
    number i,a;
    i=0;
    while (i<3)
    {
        i = i+1;
    }
    a = a * 4;
    while(bu==true)
    {
      if(a>0) {a = a - 1; return; a = a - 2;}
      else {bu = false;}
    }
```

B.5.22 testingcase/scope1.ipl

```
number a = 1;
number b = 2;


defunc showall ( number numA ) number[] b
{
    bool bu = true;
    number a= 3;
    display(a);
}

showall(a);
```

B.5.23 testingcase/scope2.ipl

```
number a = 1;
number b = 2;

defunc showall ( number numlist ) number[] b
{
    bool bu = true;
    display(a);
}

showall(a);
```

B.5.24 testingcase/demo1.ipl

```

defunc add(number a, number b) number c {
c = a + b;
while(true) {
if(c == 4)
{
```

```
c = 0;
continue;
}
else
{
c = 7;
break;
}
}
}

number numA, numB;
number[] numC = { 0, 1, 2, 3, 4 }, numD;
coord cooA;
numA = 1;
numD = numC[1~2-];  // numD = {0, 3, 4}
numB = numD[1];  // numB = 3
cooA = (numB, numA);  // (3,1)
cooA = (yof(cooA), xof(cooA));  // (1,3)
display(cooA);
display(add(numA,numB));
```

B.5.25 testingcase/demo2.ipl

```
image imgA, imgB, imgC, imgD;

imgA = "sshield.jpg"‘(100,100);
imgB = imgA‘(800,100) @ 90:100;
imgC = imgA‘(800,500) @ 180:-100;
imgD = imgA‘(100,500) @ 270:50;

display(imgA $ imgB $ imgC $ imgD);
```

B.5.26 testingcase/demo3.ipl

```
defunc rotate_animation(image src, number time, number rotate_amount) image target
{
target = animate(src @ rotate_amount * time, time);
}
image imgA = "strawberry.jpg";
coord cooA;
number time = 3, rotate = 360;

cooA = (500,300);
imgA = imgA‘cooA;
imgA = rotate_animation(imgA, time, rotate);
display(imgA);
```

B.5.27 testingcase/demo4.ipl

```
image dis;
image[] imgTar,imgSrc = {  "T1.gif","T2.gif","T3.gif","T4.gif","T5.gif",
"T6.gif","T7.gif","T8.gif","T9.gif","T10.gif" };
number counter = 0;

while(counter < 20) {
imgTar[0+] = imgSrc;
counter = counter + 1;
break;
}
break;
dis = animate(imgTar‘(900,430)^5,10);
//dis = dis‘(900,430)^5;

display(dis);
```