PSL: THE FINAL REPORT

John Hamer **hamer**

Danian Martinez **djm188**

Taino Ortiz **tjo22**

John Rodriguez **jr534**

Table of Contents

1 Introduction

1.1 Introduction:

Physical Simulation Language (whereby known as PSL) is a language intended for the convenient analysis of motion and objects in a multi-dimensional space. It has unlimited potential in analyzing, simulating, and solving mathematical, scientific, and engineering problems.

1.2 Background:

PSL was created for two reasons. First was to facilitate the creation of simple polygonal shapes using a simple language. The second was to feature interaction among these objects for the demonstration of physical principles. This lent itself to the development of PSL, a full-fledged framework enabling rapid model simulation environment development.

1.3 Goals:

PSL is an intuitive, object-oriented, graphical, productive, multi-purpose, accurate, robust, flexible language.

1.3.1 Intuitive:

Any programmer who has worked with graphical user interfaces will tell you that it is a daunting task, unless you choose a Windows-based platform. Even the simplest GUI design can take up to 50 lines of code and just under a megabyte in memory. Although memory is no longer a scarce entity in our technologically prosperous society, one can easily scale a more complex project to take up an enormous amount of space.

The runtime of a GUI can also be a factor. While Java proved a significant advancement in creating multi-platform applications, its Swing API is often considered “heavyweight”, since graphical components depend on system resources and inter-language communication between Java and languages like C impose a lot of overhead.

The language is designed with the developer in mind. Commonly used primitives, such as circles, and triangles are embedded into the PSL API. Using these pre-made objects and defining their parameters, users are able to simulate a wide variety of conditions immediately. Even novice programmers could create an environment with relative ease.

1.3.2 Object-Oriented:

By implementing data types as objects, users will find it easier to envision and construct a simulation template. The objects defined in a PSL environment are the variables defined in the programs. Attributes are assigned to each object to help define its dynamics within the environment.

Say you want to make a environment that explains Newton's laws of motion. One can use say a ball (a circle in PSL lingo) and an initial velocity and bingo, a simulation will depict the ball's motion through space. It's as simple as that.

1.3.3 Graphical:

Results are easily conveyed through a graphical interface allowing the naïve science student to draw his or her own conclusions from the presentation. Mathematical formulas can also be displayed for the more advanced users. Prior languages simply computed and displayed formulas as results, results which were often difficult to analyze, visualize, and interpret.

With PSL, users can enter all constraints, attributes, objects disregarding all the laws of motion and observe the final results. There are options to display the real-time values of the objects as well as the environmental constraints so that one may juxtapose various results against others, by simply varying environmental constraints such as gravity. This form of presentation is great for data analysis and regression testing.

1.3.4 Productive:

PSL significantly decreases the turnaround time for a developer in abstracting most of the mathematical calculations going on.   The language is designed without the necessary physics or engineering background that is required in other model simulation environments such as CAD; these calculations are done transparently by the environment during runtime. This increases the throughput as professors spend little time creating presentations, development teams devote more time to other projects, and program managers cut development costs.

Graphics programming also introduces a challenge. Some languages require an extremely intricate knowledge of the underlying hardware in order to optimize the performance of their applications. Other languages, which abstract the underlying hardware, incur a performance penalty such as Sun's Java Swing package. Even then, the Java component object model is cumbersome to learn, especially for small purposed applications.

PSL avoids this common pitfall by implementing OpenGL technology. This popular graphics toolkit enables efficient graphics processing with a flexible, easy-to-learn API capable of maximizing the full performance of one's video hardware. All desired graphic simulations are generated during complete compilation.

1.3.5 Multi-Purpose:

Users with different levels of engineering background will find this language useful. Experimentation with objects and attributes will provide an educational experience in itself. More advanced users can use the tools to explain principles and ideas to students and peers. Thus, PSL proves handy in labs, homework sessions, recitations, lectures, meetings, and presentations.

Envision the physics professor who is having trouble teaching his or her students the mechanics of projectile motion. A simple demonstration can be the difference between a struggling student and a successful one. It is not a secret that visual presentations often hold the key to grasping a challenging concept. PSL helps facilitate without costing endless hours and budget on the part of the faculty in manufacturing these presentations.

1.3.6 Accurate:

Reliable results are what make PSL so powerful. Multiple runs of a given simulation will always provide the exact same data that a real world example would.

Why is this important? Why not just settle for approximations? Scientists and professors often depend on accurate analysis for their research and students require reliable examples for their educational materials. Elementary round-off errors can lead to erroneous final results, misleading our end users who may depend on this data.

1.3.7 Robust:

The scenarios possible are endless. The language can represent any real world situation or challenge.

Some examples include but are not limited to multimedia presentations (a la PowerPoint), projectile motion analysis, and educational tutorials. Due to the simplistic design of PSL, the user can extend the capabilities of the language as they see fit.

1.3.8 Flexible:

What if you don't want to simply use circles, squares or triangles? Well, make your own then. For those who find that their needs extend beyond the primitive library available, PSL allows the users to design their own shapes given a template. Their shapes can then interact seamlessly with the others. Programmers can define their own polygons by specifying the coordinates and any other coordinates.

1.4 Conclusion:

PSL is a valuable asset to the graphics designer, executive, or engineer who must incorporate graphical displays into their everyday assignments. A multimedia presentation will come alive with the enhancements PSL provides. A lecture given by a high school physics professor will be much more effective with PSL-designed presentations in his or her arsenal. The flexibility and robustness of PSL make it a language unique in his own purpose.

2 Tutorial

2.1 Installation of PSL

2.1.1Requirements:

J2SDK 1.3.1
-needed to run first step compilation (PSL source)

C++ compiler (Borland/Visual Studio.NET)
-needed to run second step compilation(C++ backend)

OpenGL libs
- needed for graphical rendering of PSL environment

GLUT toolkit for Windows
- needed for windowing environment to display OpenGL

2.1.2 Installation:

Download PSL.zip and unzip

Place PSL java classes and pslc.exe in directory of your choosing
e.g. $SOMEDIR\PSL\bin

You must set this path as an environment variable named PSLBIN
You may also want to add this path to your PATH variables to compile in any directory

Place C++ source files in directory of your choosing
e.g. $SOMEDIR\PSL\src

You must set this path as an environment variable named PSLSRC
You may also want to add this path to your PATH variables to compile in any directory

Other notes:
"java" and the command necessary to run your C++ compiler must be executable from any location. You should set their locations in your PATH environment variable or equivalent accordingly

2.1.3 Running:

Create a sample PSL source file, e.g. foo.psl
Compile it using pslc

```
Syntax --> pslc foo.psl [outfile]
```

If not specified, output will be named out.pslx

If compiled successfully, just run the executable *.pslx file

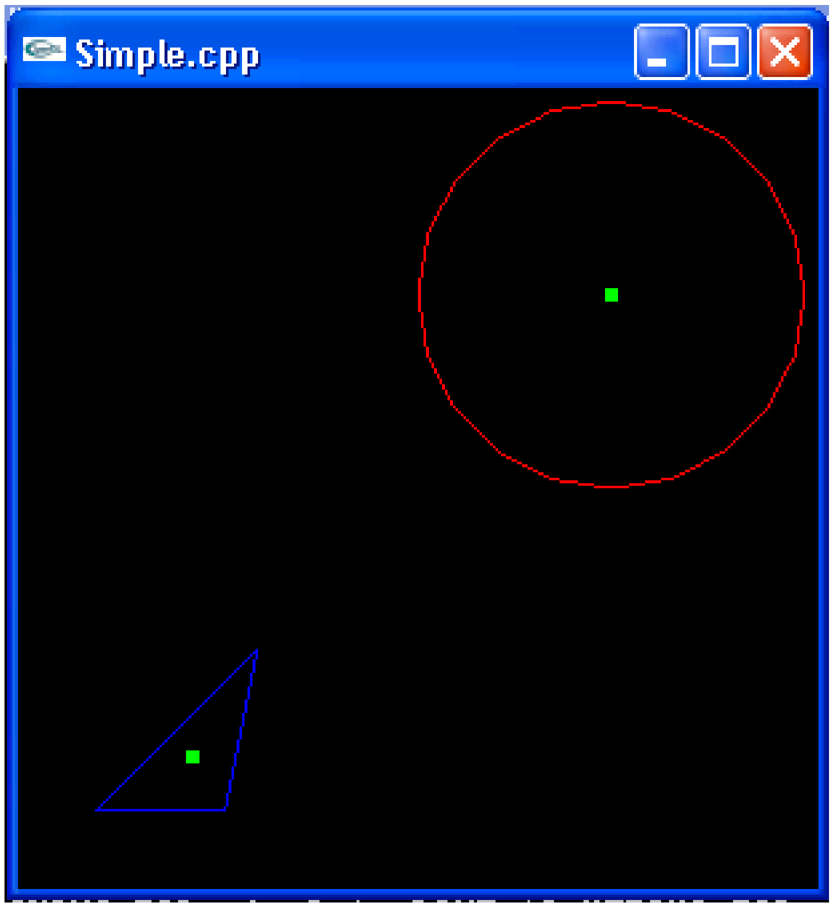
2.2 Simple.psl

Below is a very basic PSL file, whose filename is Simple.psl.

```
def circle1 Circle(3, 3, 3);//Creates a circle
circle1.color(255, 0, 0);//Sets color to red
circle1.points(20);

def poly1 Polygon(-5, -5, 3, -5);//Creates a polygon
Poly1.addpoint(-2.5, -2.5)

poly.color(0, 0, 255);//Sets color to blue

Global.zoom(15);//Set the camera zoom

/* Sets window size */
Global.windowSize(300, 300);
```

To compile this basic PSL file, you must first make sure that you have at least Java SDK 1.3.1. By typing on the command line "java Main Simple.psl", the PSL compiler will compile the code into a C++ file. To be able to compile the C++ file you must have GCC Installed and also you must have the OpenGL API, as well as GLUT for Windows or Mesa for Redhat Linux. If these requirements have been met then you can compile the C++ file by typing "g++ Simple.cpp". This will compile the C++ into an executable file, whose is default name is "a.out". Providing command line arguments to the g++ compiler can change the name of the executable.

The generated display when Simple.psl is compiled is:


Simple.cpp

3 Language Reference

3.1 Grammar Notation:

In this Reference Manual Grammars will be created as they are mentioned. Symbols that are in italics are none terminals, and quoted symbols are surrounded by single quotes if one character, or double quotes if more than one. In cases where the desired character cannot be described visual, an English sentence is used and a representation of the character is bolded. Regular expression notation is used to make the Grammars easier to read. The only time this is not true is when exempting a terminal or type. In these cases an English sentence will be written out. The symbol | stands for or such as 'a' | 'b' means the terminal a or the terminal b. The ? stands for one more of a given terminal or grammar as in 'a'? means either zero or one terminal a is expected. The * symbol stands for zero to infinite number of a given terminal or grammar such as 'a'* means there will be none or infinite terminal a. The + symbol represents that there is at least one, possibly infinite number of a given terminal or grammar such as 'a'+ means at least one terminal a possibly infinite.

3.2 Lexical Analysis:

PSL programs are written using the ASCII character set. Programs will be broken up into line, and than lines will be broken up into Input Elements which are defined as white space, comments, and tokens. A valid PSL program can be empty.

Input	->	*InputElements?*
InputElements	->	*InputElement* | *InputElement InputElements*
InputElement	->	*Whiespace* | *Comment* | *Token*

3.2.1 Line Terminators:

PSL breaks up the input to form lines and input characters. Lines are used in PSL to separate tokens, and to mark the end of a End of Line Comment. A line terminator is defined as the ASCII characters **cr** (“carriage return”) **lf** (“line feed”) and the combination of the two in the form **cr lf**. In the case of **cr lf** this is considered one line. All other characters are considered input characters.

Lineterminator	->	ASCII **cr** character | ASCII **lf** character | ASCII **cr** character ASCII **lf** character
Inputchar	->	Every ASCII character but (**cr** | **lf**) character

3.2.2 White Space:

White space in PSL is used to separate tokens from each other. White space is defined as the ASCII characters representing horzantial tab, space, form feeds as well as the previously defined line terminaters.

```
Whitespace    ->    ASCII sp character | ASCII ht character
                    | ASCII ff character | Lineterminator
```

3.2.3 Comments:

PSL supports both C++ style comments. There are two types of comments. The first starts with the input characters // and ends at a line terminator this is known as an End of Line comment. And, the second type of comment known as a block comment, starts with '/*' and ends when it arrives at a '*/'. It is important not to nest the second type of comments inside each other. The first type of comment can be nested inside the second type, but does not provide any special meaning.

```
Comment       ->    EOLComment | BlackComment

EOLComment
              ->   "//" (~('\n'|'\r'))* ('\n'|'\r'('\n')?)
BlockComment
              ->    "/*"
                    (
                      { LA(2)!='/' }? '*'
                      |  '\r' '\n'
                      |  '\r'
                      |  '\n'
                      | ~('*'|'\n'|'\r')
                    )*
                    "*/"
```

3.2.4 Tokens:

In PSL there are four classes of tokens: identifiers, keywords, separators and literals. Tokens are usually separated by white space, but that is not always necessary. Tokens are by nature greedy meaning the longest possible valid token will be formed even if that token would not necessarily make sense in the program.

```
Token         ->    Identifier | Keyword | Separator | Literal
```

3.2.5 Identifiers:

In PSL an identifier is a sequence of letters and numbers that is used to represent an object or variable name. The identifier must start with a letter or the underscore character, and can be followed by any number of letters or digits afterwards. Also an identifier cannot have the same letter combination as a keyword.

Identifier -> *letter* (*letter* | *digit*)*

digit -> ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’

letter -> ‘a’ | ‘A’ | ‘b’ | ‘B’ | ‘c’ | ‘C’ | ‘d’ | ‘e’ | ‘E’ | ‘f’ | ‘F’ | ‘g’
| ‘G’ | ‘h’ | ‘H’ | ‘i’ | ‘I’ | ‘j’ | ‘J’ | ‘k’ | ‘K’ | ‘l’ | ‘L’ | ‘m’
| ‘M’ | ‘n’ | ‘N’ | ‘o’ | ‘O’ | ‘p’ | ‘P’ | ‘q’ | ‘Q’ | ‘r’ | ‘R’ | ‘s’
| ‘S’ | ‘t’ | ‘T’ | ‘u’ | ‘U’ | ‘v’ | ‘V’ | ‘w’ | ‘W’ | ‘x’ | ‘X’ | ‘y’
| ‘Y’ | ‘z’ | ‘Z’ | ‘_’

3.2.6 Keywords:

Keywords have a special meaning in PSL and as such cannot be used as identifiers.

Keyword -> Any one of the following ASCII strings

PSL keywords:

def	Global
set	Circle
double	Line
Square	Rect
Poly	

3.2.7 Separators:

In addition to white space, tokens can be separators.

Separator -> '.' | ';' | '(' | ')' | ','

The '(' ')' separators will always appear in pairs in that order. Text is allowed between the two.

3.2.8 Literals:

In PSL a literal is input that represents the value of a primitive data type. In PSL there is only one type of literal of type double.

Literal -> *Double*

Digits -> *digit+*

3.2.8.1 Double Literals:

A double has the following parts: whole-number part, decimal point, a fractional part, and an exponent part. One digit has to be present in the whole-number part and then an optional fractional part can be created. The fractional part begins with a decimal point has one or more digits which can be followed by an optional exponential part.

```
Doubles      ->(Sign)? Number

Number       -> Digits
                ( ( '.' (Digit)+ (Expo)?) | /* Nothing */)

Expo         -> ( 'e' | 'E' ) SignedInt

SignedInt    -> ( '+' | '-' )? (DIGIT)+

Sign         -> ( '+' | '-' )
```

There are limits as to the values a double can represent. The largest double allowed is the number is 1.79769313486231570e+308 and the smallest non-zero number is 2.2250738585072014e–308. The number 0.0 is allowed.

3.3 PSL Specification:

A PSL specification is a file that describes the objects and variable of an environment. The specification will be translated into corresponding C++ code that will allow for a graphical representation of the described environment. A specification is made up of any number of statements.

```
PSLSpec      ->      Statement*
```

3.3.1 Statements:

Statements in PSL are the building blocks of any program. There are two distinct types of statements: Declaration and Expression. Each of these types of statements have a special role to play. All statements terminate with the separator ';' character.

```
Statement    ->      (Declaration | Expression) ';'
```

3.3.2 Names:

A name is the representation of a declaration, and will be valid for any part of the specification that follows the declaration. Also, a name must be unique to the program and not a keyword. A variable is defined as a name that has been declared.

3.3.3 Lists:

An important data type in PSL is the List. A List consists of one or my literals or variable those are surrounded by parentheses. There are two predefined lists in PSL that require a certain number of arguments to be inputted.

List	->	'(' *Param* (',' *Param*)* ')'
Param	->	(*Literal* | *Identifier)*

3.3.3.1 Three List

A three list is just a special case of list where three parameters have to be inputted.

ThreeList	->	'(' *Param* ',' *Param* ',' *Param* ')'

3.3.3.1 Four List

A four list is just a special case of list where four parameters have to be inputted.

FourList	->	'(' *Param* ',' *Param* ',' *Param* ',' *Param* ')'

3.3.4 Declaration:

In PSL a declaration is when you create a variable, and give it a value. There are two types of declarations in PSL primitive and complex.

Declaration	->	*simpledec* | *complexdec*

3.3.4.1 Simple Declaration:

A simple declaration in PSL is when you declare a variable of one of the two literals that are found in PSL. A simple declaration always starts with the word set, followed by the Identifier that is being declared. This is then followed by the literal type, which has the value you are setting the variable to in parentheses.

simpledec	->	*Set Identifier Literaltype '(' Literal ')'*
Set	->	*"set"*
Literaltype	->	"double"

3.3.4.2 Complex Declaration:

A complex declaration in PSL is used when describing one of the graphical objects that will be displayed in the environment. There are five types of Objects that can be made. The to declare one of these objects is to use the key word def followed by the name of the object followed by the type followed by a list of initial attributes the object has. The list of attributes is all Literals separated by the comma separator, and is surrounded by parentheses.

complexdec -> *Def Identifier ObjectType (ThreeList | FourList)*

Def -> "def"

ObjectType -> "Square" | "Rect" | "Line" | "Circle" | "Poly"

3.3.5 Expressions:

An expression in PSL is how different aspects of the objects that are created. There are two distinct types of expressions, those that deal with the global environment and those that deal with an object in the environment. The global environment uses the keyword Global in order that is can be referenced. To reference a created object the name of the object begins the expression. The basic structure of an expression starts with the variable that is being modified followed by the period separator followed by the variable to be changed, which is then followed by a List.

Expression -> (*Identifier* | "Global") ',' *Identifier List*

4 Project Plan

4.1 Team Responsibilities

Each person's responsibilities were made to fit what he and the group believed his strengths to be. Not everyone participated in the coding and testing phases of the project, but everyone had his own responsibility to do what he was assigned as well as possible.

Chris Hamer & Danian Martinez

They were responsible for the front end of the project. They used Antler to help build the parser and lexer for the PSL Language, and then they built their own Tree walker to retrieve the necessary information from the tree produced by the lexer file. They also built the Intermediate Code Generator, which produced from the Tree Walker information and the pre-defined PSL-specific C++ template the .cpp file.

John Rodriguez

He was responsible for the backend of the project. He handled the coding of the physics and also the GUI displayed by the program. He used C++ for the backend of the project. He used an API called OpenGL to handle the production of graphics, and he used GLUT to display on Windows Machines and Mesa to display on Redhat Linux Machines.

Taino Ortiz

He handled the organization of all the documents and was responsible for writing this Final Report. He also produced the presentation slides for the class presentation.

4.2 Project Timeline

2-17-2003	Language whitepaper complete
3-26-2003	Language Reference Manual complete
4-19-2003	Initial Parser Complete
5-07-2003	Code Generation complete
5-13-2003	Project Completed

4.3 Software Development

The project was developed using a combination of Java and C++/OpenGL. The PSLLexer and the PSLParser were developed using ANTLR, which was developed in Java. The PSLTreeWalker was also written in Java. The backend of the project was developed using C++/OpenGL. OpenGL is an API, whose libraries were written in C, for producing graphics, which can be displayed on Windows by using Glut or on Redhat Linux by using Mesa. Versioning of the project was controlled by RCS. When a file was checked out, it was not checked back in until it was proven to compile and produce the correct output.

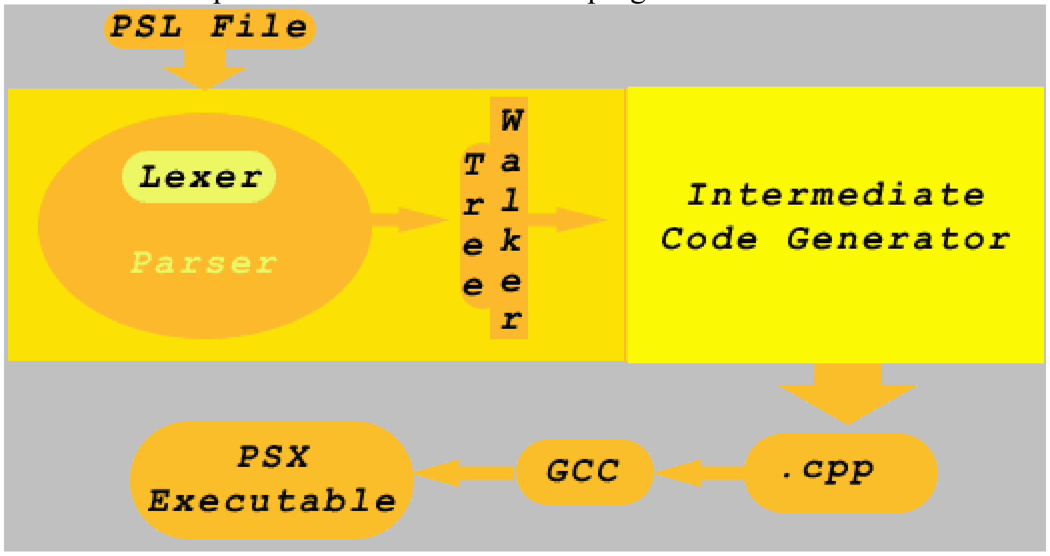
4.4 Project Log

02-02-2003	Formation of group
02-05-2003	Decision of PSL
02-18-2003	Creation of Whitepaper
03-11-2003	Creation of Initial Grammar
03-26-2003	Making of LRM
03-27-2003	Backend Object Oriented Design complete
03-29-2003	Backend Decision of Inheritance
04-13-2003	Making of Working Parser and Lexer
04-17-2003	Collision Detection Research
04-26-2003	Creation of Tree Walker
05-06-2003	Collision Detection Moment
05-09-2003	Finalization of grammars
05-10-2003	First successful .cpp file
05-12-2003	In-class Presentation
05-13-2003	Final Report

5 Architectural Design

5.1 Compiler

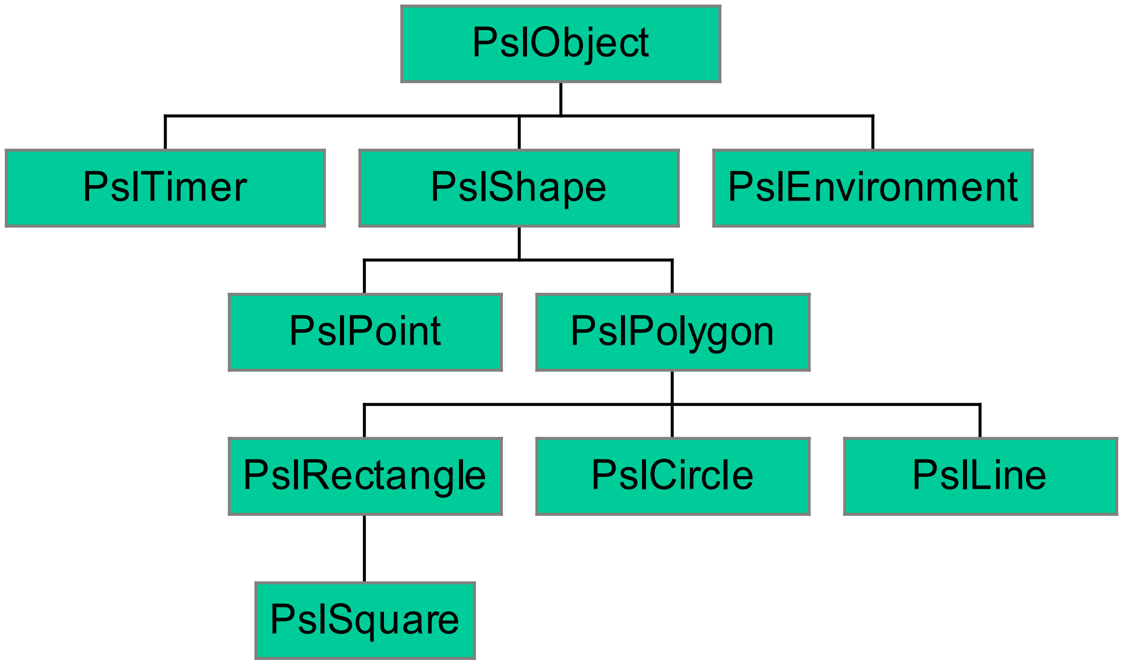
Below is a diagram of the interaction of the different components of the Compiler, and also the interaction of the compiler with the backend of the program.



Every program begins with the PSL file, which contains the PSL source code. The first even that occurs when a PSL file is compiled is the Lexer is called. The Lexer produces the tokens and then sends the tokens to the Parser. The Parsers role is to put the Tokens into an AST Tree for the TreeWalker to then step through. The TreeWalker performs all the semantic analysis and also setting of global variable. This information is then passed to the Intermediated Code Generator. Its purpose is to produce the C++ source file. The Intermediate Code abides by a PSL-specific C++ template to generate the appropriate environment and object specifications.

5.2 Backend

The backend of the PSL language implemented a carefully structured OO environment using C++/OpenGL. Below is the structure to which the Backend abided by.



Everything created is a descendant of PslObject, including the PslTimer and the PslEnvironment. Each child of the tree inherits characteristics from its parents that help define what it is. The PslObject was made to encompass everything just in case a container was need that would be able to hold all parts of the project. The PslShapes is a parent to all items below because of the nature of shapes. There are many overlapping attributes that making all the shapes inherit PslShape made sense. The same could also be said for the PslPolygon. All the shapes below it could be derived from PslPolygon, but their cases were special enough that they deserved their own class.

5.3 PSL

The two halves of the program communicate through the source file that is produced by the Intermediate Code Generator. This source file once compiled by g++ will display all the shapes and specifications that the user had defined. This display is the final product of the PSL Language. It is the sum of the compiler and the backend.

6 Test Plan

6.1 Example Programs with source that is edited

6.1.1 Example 1

```
/* Welcome to our testing program in here
are various problems that the compiler must
identify.  Let us begin with comments//ok */

def jack Poly(5, 6, 7, 8);
set psl double(5);
jack.addPoint(9,10);
set Dan double(30);
jack.color(Dan, 50, 70);
jack.addPoint(Dan, 30);

Global.windowSize(300,300);
Global.gravity(Dan);
```

The .cpp file produced:

```
#include <cstdlib>
using namespace std;

#include <GL/glut.h>
#include "PslEnvironment.h"
#include "PslCircle.h"
#include "PslPolygon.h"
#include "PslSquare.h"
#include "PslLine.h"
#include "PslPoint.h"
#include "PslRectangle.h"

PslEnvironment scene;

void init()
{
	glShadeModel(GL_FLAT);

	glClearColor(0.0, 0.0, 0.0, 0.0);
	glClearDepth(1.0);

	glEnable(GL_DEPTH_TEST);
	glDepthFunc(GL_LEQUAL);
	glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
	scene.zoom = 20;
}

void display() { scene.draw(); }

int main(int argc, char **argv)
{
	argc = 1;
	glutInit(&argc, argv);
	glutInitWindowPosition(100, 100);
	glutInitWindowSize(300, 300);
	glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );

	init();

	glutCreateWindow("hello.cpp");

	PslPolygon *PSL_jack = new PslPolygon("PSL_jack");
	PSL_jack->addPoint(5, 6, 0);
	PSL_jack->addPoint(7, 8, 0);
	PSL_jack->addPoint(9, 10, 0);
	PSL_jack->addPoint(30, 30, 0);
	PSL_jack->com[X] = 12.75;
	PSL_jack->com[Y] = 13.5;
	PSL_jack->com[Z] = 0.0;
	PSL_jack->velocity.x = 0;
	PSL_jack->velocity.y = 0;
	PSL_jack->velocity.z = 0;
	PSL_jack->acceleration.x = 0;
	PSL_jack->acceleration.y = 0;
	PSL_jack->acceleration.z = 0;
	PSL_jack->color[X] = 30;
	PSL_jack->color[Y] = 50;
	PSL_jack->color[Z] = 70;



	scene.addShape(PSL_jack);

	glutDisplayFunc(display);
	glutReshapeFunc(scene.resize);
	glutMouseFunc(scene.mouse);
	glutKeyboardFunc(scene.keyboard);
	glutMainLoop();
```

```
        if( PSL_jack ) delete PSL_jack;

        return 0;
}
```

6.1.2 Example 2

This is the PSL source file

```
set width double(300);
set height double(300);

def ball Circle(0, 0, 5);
def floor Rect(-145, -145, width, 3);

ball.vely(2.0);
ball.color(0, 0, 255);
ball.points(20);

//floor.color(0, 255, 0);
floor.accy(9.8);//Counter act gravity

Global.windowSize(width, height);
Global.gravity(-9.8);
Global.zoom(width);
```

This is the C++ source file produced:

```
#include <cstdlib>
using namespace std;

#include <GL/glut.h>
#include "PslEnvironment.h"
#include "PslCircle.h"
#include "PslPolygon.h"
#include "PslSquare.h"
#include "PslLine.h"
#include "PslPoint.h"
#include "PslRectangle.h"

PslEnvironment scene;

void init()
{
        glShadeModel(GL_FLAT);

        glClearColor(0.0, 0.0, 0.0, 0.0);
        glClearDepth(1.0);

        glEnable(GL_DEPTH_TEST);
        glDepthFunc(GL_LEQUAL);

        glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
        scene.zoom = 300;
}

void display() { scene.draw(); }

int main(int argc, char **argv)
{
        argc = 1;
        glutInit(&argc, argv);
        glutInitWindowPosition(100, 100);
        glutInitWindowSize(300, 300);
        glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );

        init();

        glutCreateWindow("ballbounce.cpp");

        PslCircle *PSL_ball = new PslCircle( "PSL_ball", 0, 0, 5, 20);
        PSL_ball->com[X] = 0;
        PSL_ball->com[Y] = 0;
        PSL_ball->com[Z] = 0.0;
        //PSL_ball->mass = 0;
        PSL_ball->velocity.x = 0;
        PSL_ball->velocity.y = 2.0;
        PSL_ball->velocity.z = 0;
        PSL_ball->acceleration.x = 0;
        PSL_ball->acceleration.y = -9.8;
        PSL_ball->acceleration.z = 0;
        PSL_ball->color[X] = 0;
        PSL_ball->color[Y] = 0;
        PSL_ball->color[Z] = 255;


        PslRect *PSL_floor = new PSLRect("PSL_floor"-145, -145, 300, 3);
        PSL_floor->com[X] = 5.0;
        PSL_floor->com[Y] = -143.5;
        PSL_floor->com[Z] = 0.0;
        //PSL_floor->mass = 0;
        PSL_floor->velocity.x = 0;
        PSL_floor->velocity.y = 0;
```

```
	PSL_floor->velocity.z = 0;
	PSL_floor->acceleration.x = 0;
	PSL_floor->acceleration.y = 0.0;
	PSL_floor->acceleration.z = 0;
	PSL_floor->color[X] = 0;
	PSL_floor->color[Y] = 0;
	PSL_floor->color[Z] = 0;



	scene.addShape(PSL_ball);
	scene.addShape(PSL_floor);

	glutDisplayFunc(display);
	glutReshapeFunc(scene.resize);
	glutMouseFunc(scene.mouse);
	glutKeyboardFunc(scene.keyboard);
	glutMainLoop();

	if( PSL_ball ) delete PSL_ball;
	if( PSL_floor ) delete PSL_floor;

	return 0;
}
```

6.2 Error Testing

To test how the errors in the language are detected one file was created that contained most of the mistakes of the typical compiler. One of the limitations of PSL is that only one error is detected at a time, so after each error was detected it was fixed and then it sought the next error. All of this was tested in the journal program in Linux to prevent having to copy and paste every line of code. The original source code is printed below:

```
/* Welcome to our testing program in here
are various problems that the compiler must
identify.  Let us begin with comments//ok

def Danian Poly(5, 6, 7, 8);
set Danian(5)
jack.addPoint(9,10);
danian.color(Dan, 50, 70);
jack.Dan(yes);

Global.windowSize(300,300);
Global.gravity(high);
```

All the errors that are present found here are:

1) Commenting error
2) Danian defined twice
3) Improper declaration of a double
4) Missing Semicolon
5) Undefined Shape
6) danian is undefined
7) Undefined symbol Dan
8) Undefined Variable Dan
9) expecting ID, found 'set'
10) Undefined symbol yes
11) Wrong number of arguments
12) Undefined symbol high

Below is the log of each attempted compilation:

```
Journal$ java Main hello.psl
Error:
exception: hello.psl:13:1: expecting '*', found '<EOF>'

Journal$ java Main hello.psl
Error:
hello.psl:6:11: expecting "double", found '('

Journal$ java Main hello.psl
Error:
hello.psl:7:1: expecting SEMICOLON, found 'jack'

Journal$ java Main hello.psl
Error:
Variable: Danian defined more then once

Journal$ java Main hello.psl
Error:
Undefined Shape: jack

Journal$ java Main hello.psl
Error:
Undefined Shape: danian

Journal$ java Main hello.psl
Error:
Undefined symbol: Dan

Journal$ java Main hello.psl
```

```
Error:
Undefined symbol: Dan

Journal$ java Main hello.psl
Error:
Undefined variable Dan

Journal$ java Main hello.psl
Error:
hello.psl:10:6: expecting ID, found 'set'

Journal$ java Main hello.psl
Error:
Undefined symbol: yes

Journal$ java Main hello.psl
Error:
Wrong number of arguments:
jack.addPoint( 30 )
Expected: 2 arguments
Found: 1 arguments

Journal$ java Main hello.psl
Error:
Undefined symbol: high
```

Here is what the program should look like:

```
/* Welcome to our testing program in here
are various problems that the compiler must
identify.  Let us begin with comments//ok */

def jack Poly(5, 6, 7, 8);
set psl double(5);
jack.addPoint(9,10);
set Dan double(30);
jack.color(Dan, 50, 70);
jack.addPoint(Dan, 30);

Global.windowSize(300,300);
Global.gravity(Dan);
```

7 Lessons Learned

7.1 Chris' Lessons

This project was my first experience working in a team environment for coding a program. I learned that my way wasn't always the right way, and it is important that you listen to the thoughts and ideas of your team members. Another major lesson I learned was to learn to trust people to do what they say they will do. I caught myself at many times butting in onto code wanting to force my ideas on how to do code something. It took effort not to take try to take control, and by the end of the project I was doing a much better job at this. The last thing, and the most important thing I learned was that learning new ideas through a project is a good idea, but you should have at least some background in the subject before hand. I had no background in either graphics or C++ prior to this project, and this limited me at the beginning because I didn't understand what was going on. Luckily by the end I had learned enough to be a major contributor on the backend side of the code.
Lessons Learned:

7.2 John's Lessons

1) Too much to handle
Our choice of language was too broad, involving more factors than originally expected
Numerical accuracy was a huge deterrent, as well as choosing collision detection
that was decently accurate while maintaining a feasible sense of realism mathematically
and perceptually.

2) Teamwork could have been more evenly divided
While 25% of every team member is often unrealistic to monitor or gauge,
it hindered the progress of our project to have 4 individual members work asynchronously.

3) Time management
Unlike past projects, we can say that work began on this from the get-go.
We designed our language earlier than many other groups, from compiler to backend.
We, unfortunately, made many errors, caught later on, sometimes requiring a revamped
architecture. Furthermore, indecision on ways to approach problems hindered overall progress.

4) Maintain goals
We stated this would be portable, but failed to benchmark on every platform regularly.

7.3 Danian's Lessons

Learning how to build a compiler was a very interesting task. Defining the grammars used and understanding how to program the compiler proved to be a very challenging task. First I had to understand how a compiler conceptually worked, and then the most difficult part was programming my concepts, so that the language would work. Sometimes the overall project proved to be overwhelming because I did not understand all the concepts, but it was a great experience to learn something new. Future groups should probably think of a language that would be most helpful to learning new concepts, but also beware that you do not over extend yourself and try to do things that are too difficult.

7.4 Taino’s Lessons

This project was one of the most difficult projects I ever had to deal with in terms of concepts and team works. Most of the time I felt that I hadn’t contributed enough, but I learned that support and listening could make up for a lack of knowledge and ability. Just listening to my partners explain their sections of the project helped me understand what was going on with the group, and it also helped see where we made errors. I definitely felt really behind on all the concepts of the project backend, and I would advise future groups to choose something everyone in the group could conceptually understand.

Appendix 1 Lexer and Parser Grammers:

A1.1 Lexer grammer:

```
options {
      mangleLiteralPrefix = "TK_";

}

class ProjectileLexer extends Lexer;
options {
      k=2;
      exportVocab=PSL;
      charVocabulary = '\3'..'\377';
}

tokens {
      "int"; "double"; "Square"; "Circle"; "Line"; "Rect"; "Poly";
      "Global"; "set"; "def";
}

WHITEY      :     (
            SL_COMMENT
      |     ML_COMMENT
      |     ' '
      |     '\t'
      |     '\n'  {newline();}
      |     '\r'  {newline();}
      |     "\r\n"      {newline();}
      )
            { $setType(Token.SKIP); }
      ;

LPAREN options {paraphrase = "'('";}
      :      '('
      ;

RPAREN options {paraphrase = "')'";}
      :      ')'
      ;

PERIOD
      :     '.'
      ;
SEMICOLON:
            ';'
      ;
COMMA:            ','
      ;

protected
POS:        '+'
      ;

protected
```

```
NEG:        '-'
      ;

ID options { testLiterals = true; }
      : LETTER ( DIGIT | LETTER )*;

protected
DIGIT
      :     '0'..'9'
      ;

protected
LETTER
      :     ( 'a'..'z' | 'A'..'Z' )
      ;

NUMBER
      : (SIGN)? NUMBER2
      ;

protected
NUMBER2
      : (DIGIT)+
         (( PERIOD (DIGIT)+ (EXPO)?)
         | )

      ;

protected
EXPO
      :  ( 'e' | 'E' ) SIGNEDINT
      ;

protected
SIGNEDINT
      : ( '+' | '-' )? (DIGIT)+
      ;
protected
SIGN
      : ( POS | NEG )
      ;

// Single-line comments
protected
SL_COMMENT
      :     "//"
            (~('\n'|'\r'))* ('\n'|'\r'('\n')?)
            {newline();}
      ;
protected
ML_COMMENT
      :     "/*"
            (
                  options {
                        generateAmbigWarnings=false;
                  }
            :
                  { LA(2)!='/' }? '*'
            |     '\r' '\n'         {newline();}
            |     '\r'              {newline();}
            |     '\n'              {newline();}
            |     ~('*'|'\n'|'\r')
            )*
            "*/"
      ;
```

A1.2 Parser Grammer:

```
options {
      language = "Java";
      mangleLiteralPrefix = "TK_";
}

class ProjectileParser extends Parser;
options {
      k = 2;
      buildAST = true;
      importVocab = PSL;
      defaultErrorHandler = false;
}

file
      :     (statement)+ EOF
      {
       //System.out.println("At the file grammar");
      }
      ;

statement
      : (declaration | expression) SEMICOLON!
      ;

declaration
      : (simpledec | complexdec)
      ;

simpledec
      : TK_set^ ID literalType LPAREN! literal RPAREN!
      {
       //System.out.println("Found a simple declartion");
      }
      ;

complexdec
      : TK_def^ ID objectType
      {
       //System.out.println("Found a complex dec");
      }
      ;

expression
      : (ID^ | TK_Global^) PERIOD! ID list
      {
       //System.out.println("Found a expression");
      }
      ;

list
      : LPAREN! (literal | ID) (COMMA! (literal | ID))* RPAREN!
      ;

literalType
      : ( TK_double )
      ;

literal
      : ( NUMBER )
      ;

objectType
      : TK_Square^ LPAREN! threeList RPAREN!
      | TK_Line^ LPAREN! fourList RPAREN!
      | TK_Circle^ LPAREN! threeList RPAREN!
      | TK_Rect^ LPAREN! fourList RPAREN!
      | TK_Poly^ LPAREN! fourList RPAREN!
      ;

protected
twoList
      : param COMMA! param
      ;

protected
threeList
      : param COMMA! param COMMA! param
      ;

protected
fourList
      : param COMMA! param COMMA! param COMMA! param
      ;

param
      : (literal | ID)
      ;
```

Appendix 2 C++ Output File Template:

```
#include <cstdlib>
using namespace std;

#include <GL/glut.h>
#include "PslEnvironment.h"
#include "PslCircle.h"
#include "PslPolygon.h"
#include "PslSquare.h"
#include "PslLine.h"
#include "PslPoint.h"
#include "PslRectangle.h"

PslEnvironment scene;

void init()
{
      glShadeModel(GL_FLAT);

      glClearColor(0.0, 0.0, 0.0, 0.0);
      glClearDepth(1.0);

      glEnable(GL_DEPTH_TEST);
      glDepthFunc(GL_LEQUAL);

      glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
      scene.zoom = 20;
}

void display() { scene.draw(); }

int main(int argc, char **argv)
{
      argc = 1;
      glutInit(&argc, argv);
      glutInitWindowPosition(100, 100);
      glutInitWindowSize(800, 800);
      glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );

      init();

      glutCreateWindow("Template.cpp");

      //Add object here.

      glutDisplayFunc(display);
      glutReshapeFunc(scene.resize);
      glutMouseFunc(scene.mouse);
      glutKeyboardFunc(scene.keyboard);
      glutMainLoop();


      return 0;
}
```

Appendix 3: Front End Code:

```
/*
  PslCompiler.java

  This runs our complier.
  It takes in a file name, attempts to open it, parse it, analyze then
  output as a .cpp file that will run the environment that the author
  creats in his .psl file.

  Author Chris Hamer and Danian Martinez

  @param args[0] -> the file name of the .psl file we are compling

*/

import java.io.*;
import java.util.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.DumpASTVisitor;
import antlr.RecognitionException;
import antlr.TokenStreamException;

public class PslCompiler
{

    static int errornum = 72;

    public static void main(String[] args)
    {
      if (args.length != 1){

          System.err.println("Error: ");
          System.err.println("Correct usage is :");
          System.err.println("java Main <filename.psl>");
          System.exit(1);
      }

      String filename = args[0];

      if (!filename.endsWith(".psl")){
          System.err.println("Error: ");
          System.err.println("Invalid File Name: " + filename);
          System.err.println("File should end with .psl");
          System.exit(errornum);
      }

      Vector vars = new Vector();
      Vector objs = new Vector();
      Global enviro = new Global();
      try {

          //Creates an object of the lexer
          ProjectileLexer lexer = new ProjectileLexer(new FileReader(filename));
```

```
//runs the lexer on the file
lexer.setFilename(filename);

//Creates the parser
ProjectileParser parser = new ProjectileParser(lexer);
parser.setFilename(filename);

// Parses the input
parser.file();

//Get the AST formed by the parser
AST t = parser.getAST();

String y = t.getText();

//This while loop is our treewalker
//It keeps getting children from the parent until
//it has iterated through the entire tree.
while (t.getText() != null){
  if (t == null)
      System.err.println(t.toStringList());
  y = t.getText();

  //Looks for the keyword "set" which describes
  //the head of a set primitive data type
  //statement.
  if (y.equals("set")){
      //x now equals the Tree that describes
      //a set statement
      AST x = t.getFirstChild();

      String a = x.getText();
      x = x.getNextSibling();
      if (alreadyDefined(a, objs, vars)){
        System.err.println("Error:");
        System.err.print("Variable: " + a);
        System.err.println(" defined more then once");
        System.exit(errornum);
      }
      String b = x.getText();
      x = x.getNextSibling();
      String c = x.getText();


      //Checks to see if variable already declared
      //for (int i=0; i<vars.size(); i++){
      //if (((Vari)vars.elementAt(i)).equals(a)){
      //System.err.println("Error:");
      //System.err.println("Multiple declartions of " +a);
      //System.exit(errornum);
      //}
      //}

      //Adds the new variable to the primitive
      //variable type list
      vars.addElement(new Vari(a, c));
  }
```

```
//Looks for the keyword "def" which describes
//the head of a define a Shape data type
//statement.
else if(y.equals("def")){
    AST x = t.getFirstChild();

    String a = x.getText();

    //Check if already defined
    if (alreadyDefined(a, objs, vars)){
      System.err.println("Error:");
      System.err.print("Variable: " + a);
      System.err.println(" defined more then once");
      System.exit(errornum);
    }

    x = x.getNextSibling();
    String b = x.getText();
    //Gets the type of Shape being created
    int type = x.getType();

    //This gets the subtree that is the list of arguements
    x = x.getFirstChild();
    String c = getValue(x, vars);
    x = x.getNextSibling();
    String d = getValue(x, vars);
    x = x.getNextSibling();
    String e = getValue(x, vars);
    if (type <8){
      if (type==6)
          objs.addElement(new Square(a, c, d, e, type));
      else
          objs.addElement(new Circle(a, c, d, e, type));
    }
    else{
      //These have four arguements so have to get
      //one more value
      x = x.getNextSibling();
      String f = getValue(x, vars);
      if (type==8)
          objs.addElement(new Line(a, c, d, e, f, type));
      else if (type==9)
          objs.addElement(new Rect(a, c, d, e, f, type));
      else if (type==10)
          objs.addElement(new Poly(a, c, d, e, f, type));
      else{

          if (exists(a, objs)){
            System.err.println("Error:");
            System.err.println("Unknown Shape type: " + b);
            System.exit(errornum);
          }
      }

    }
    }
    else {

        String a = t.getText();
        int type = t.getType();
        if (type == 22){

          if (!exists(a, objs)){
              System.err.println("Error:");
              System.err.println("Undefined Shape: " + a);
              System.exit(errornum);
          }

          int val = findObj(a, objs);
          AST x = t.getFirstChild();

          String b = x.getText();
          int num = getNum(b, val, objs);
          x = x.getNextSibling();
          Vector arg = new Vector();
          String tmp;
          //Gets as many arguments as needed
          //pushes them into the vector
          do{
              tmp = getValue(x, vars);
              arg.addElement(tmp);
              x = x.getNextSibling();
          }while (x != null);

          //Checks to make sure correct number of arguments
          //are inputted. Ouputs an error if the wrong number
          //have been inputted
          if (num != arg.size()){
              System.err.println("Error:");
              System.err.println("Wrong number of arguments:");
              System.err.print(a + "." +b + "( ");
              for (int i=0; i<arg.size(); i++)
                System.err.print(arg.elementAt(i) + " ");
              System.err.println(")");
              System.err.print("Expected: " + num);
              System.err.println(" arguments");
              System.err.print("Found: " + arg.size());
              System.err.println(" arguments");
              System.exit(errornum);
          }

          //Calls the setvariable function on the
          //object. with the variable being changed
          //and the arguments changing to.
          ((Shape) objs.elementAt(val)).setVar(b, arg);

        }
        else if(type == 11){

          AST x = t.getFirstChild();
          String b = x.getText();
          int num = getGNum(b, enviro);
```

```
            Vector arg = new Vector();
            String tmp;
            x = x.getNextSibling();
            do{
                tmp = getValue(x, vars);
                arg.addElement(tmp);
                x = x.getNextSibling();
            }while (x != null);

            if (num != arg.size()){
                System.err.println("Error:");
                System.err.println("Wrong number of arguments:");
                System.err.print(a + "." +b + "( ");
                for (int i=0; i<arg.size(); i++)
                  System.err.print(arg.elementAt(i) + " ");
                System.err.println(")");
                System.err.print("Expected: " + num);
                System.err.println(" arguments");
                System.err.print("Found: " + arg.size());
                System.err.println(" arguments");
                System.exit(errornum);
            }


            enviro.setVar(b, arg);

          }

      }
      t = t.getNextSibling();
    }

}
catch(TokenStreamException e) {
    System.err.println("Error:" );
    System.err.println("exception: " + e);
    System.exit(errornum);
}
catch(RecognitionException e) {
    System.err.println("Error:");
    System.err.println(e);
    System.exit(errornum);
}
catch(FileNotFoundException ex)
{
System.err.println("Error:");
System.err.println("File " + filename + " not found");
System.exit(errornum);
}

for (int i=0; i<objs.size(); i++){
    ((Shape) objs.elementAt(i)).applyGravity(enviro.gravity);
}


PslICG writer = new PslICG(filename, enviro, objs);
```

```
  writer.closeAll();
}

/*


*/
public static int getGNum(String b, Global enviro){
  Vector attri = enviro.getAtt();

  for (int i=0;i<attri.size(); i++){
      if (((Vari) attri.elementAt(i)).equals(b))
        return ((Vari)attri.elementAt(i)).getIntValue();
  }

  System.err.println("Error: ");
  System.err.println("Undefined Global variable " + b);
  System.exit(errornum);
  return -1;

}

public static int getNum(String b, int loc, Vector objs){
  Vector attri = ((Shape) objs.elementAt(loc)).getAtt();

  for (int i=0; i<attri.size(); i++){
      if (((Vari) attri.elementAt(i)).equals(b))
        return ((Vari)attri.elementAt(i)).getIntValue();
  }
  System.err.println("Error:");
  System.err.println("Undefined variable " + b);
  System.exit(errornum);
  return -1;
}

public static int findObj(String a, Vector objs){
  for (int i=0; i<objs.size(); i++){
      if (((Shape) objs.elementAt(i)).equals(a))
        return i;
  }
  return -1;
}

public static boolean alreadyDefined(String a, Vector objs, Vector vars){
  for (int i=0; i<objs.size(); i++){
      if (((Shape) objs.elementAt(i)).equals(a))
        return true;
  }

  for (int i=0; i<vars.size(); i++){
      if (((Vari) vars.elementAt(i)).equals(a))
        return true;
  }
  return false;
}
```

```
    public static boolean exists(String a, Vector objs){
      for (int i=0; i<objs.size(); i++){
          if (((Shape) objs.elementAt(i)).equals(a))
            return true;
      }
      return false;
    }

    /*
      getValue()
      This gets the value of a paramter. If it is a primitive then
      it the value of that primitive is returned else the value
      from the input is returned.
    */
    public static String getValue(AST x, Vector vars){

      int type = x.getType();
      String a = x.getText();
      //If it is an id Then we have to search through our
      //primitive variable table and find the value
      //if the primitive is not found then an Error will be outputted
      //and the complier will close.
      if (type == 22){
          for (int i=0; i<vars.size(); i++){
            if (((Vari)vars.elementAt(i)).equals(a)){
                return ((Vari) vars.elementAt(i)).getValue();
            }
          }
          System.err.println("Error:");
          System.err.println("Undefined symbol: " + a);
          System.exit(errornum);
      }

      return a;

    }
}//PslCompiler
```

```java
/**
 * PslICG.java
 *
 *
 * Created: Sat May 10 21:35:49 2003
 *
 * @author Danian J Martinez
 * @version
 */
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.IOException;
import java.util.Vector;

public class PslICG
{
    /*
     * Variables:
     *
     *  sInFilename       -> the name of the original psl file,
     *                         exact name is required (with path)
     *  sOutFilename      -> the name of the cpp output file
     *  bw                -> BufferedWriter object
     *  vObjNames         -> The vector that contains the names of
     *                         all the objects added. Names are used
     *                         for freeing memory when closing the file.
     *  global            -> the Global object created by front-end
     *  iNumberOfObjects  -> the current number of object in vector
     *
     */
    String sInFilename, sOutFilename;
    BufferedWriter bw;
    Vector vObjNames;
    Global global;
    int iNumberOfObjects;

    /*
     *  Constructor that begins the cpp file that uses openGL to
     *  create a scene for projectile simulation.  A file is created
     *  using a BufferedWriter.  A Vector is created that contains
     *  all the objects in a scene.
     *
     *  @param String fn - the original filename
     *  @see java.io.FileWriter
     *  @see java.io.BufferedWriter
     *  @see java.util.Vector
     */
    public PslICG(String fn, Global g, Vector v)
    {
      iNumberOfObjects = 0;
      vObjNames = v;

      global = g;

      sInFilename = fn;
      sOutFilename = fn.substring(0, fn.length() - 4) + ".cpp";
      System.out.print("The file " + sInFilename);
      System.out.println(" has been compiled into " + sOutFilename);

      try {
          bw = new BufferedWriter( new FileWriter(sOutFilename) );

          writeInit();
          writeDisplay();
          beginMain();
          readVector(v);
      }
      catch (IOException e) {
          System.err.println("Could not create writer becasue there was an I/O
problem: " + e);
          System.out.println("Closing program.");
          System.exit(1);
      }
    }

    /*
     * Writes the init() method for openGL
     * @exception IOException
     */
    private void writeInit() throws IOException
    {
      // try {
      String sInclude = "#include <cstdlib>\nusing namespace std;\n \n#include
<GL/glut.h>\n";
      sInclude += "#include \"PslEnvironment.h\" \n";
      sInclude += "#include \"PslCircle.h\"\n";
      //    sInclude += "#include \"PslEllipse.h\" \n";
      sInclude += "#include \"PslPolygon.h\"\n";
      sInclude += "#include \"PslSquare.h\"\n";
      sInclude += "#include \"PslLine.h\"\n";
      sInclude += "#include \"PslPoint.h\"\n";
      sInclude += "#include \"PslRectangle.h\"\n";

      bw.write(sInclude, 0, sInclude.length());
      bw.newLine();

      String sInit = "PslEnvironment scene;\n\nvoid
init()\n{\n\tglShadeModel(GL_FLAT);\n\n";
      sInit += global.getClearColor();
      sInit += "\tglClearDepth(1.0);\n\n";
      sInit += "\tglEnable(GL_DEPTH_TEST);\n\tglDepthFunc(GL_LEQUAL);\n\n";
      sInit += "\tglHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);\n";
      sInit += "\tscene.zoom = " + global.zoom + ";\n";
      sInit += "}\n";

      bw.write(sInit, 0, sInit.length());
      bw.newLine();
      /* }
         catch (IOException ex) {
         System.out.println("There was an I/O problem: " + ex);
         System.exit(1); }
      */
    }
    private void writeDisplay() throws IOException
    {
      //try {
      String sDisplay = "void display() { scene.draw(); }\n";
      bw.write(sDisplay, 0, sDisplay.length());
      bw.newLine();
      /* }
         catch (IOException exc) {
         System.out.println("Display: I/O -> " + exc);
         System.exit(1); }
      */
    }

    private void beginMain() throws IOException
    {
      // try {
      //these are obvious
      String sMain = "int main(int argc, char **argv)\n{\n\targc =
1;\n\tglutInit(&argc, argv);\n";
      sMain += "\tglutInitWindowPosition(100, 100);\n";
      sMain += global.getWindow();

      //use RGBA color scheme, double buffering and z-buffering
      sMain += "\tglutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH
);\n\n";

      sMain += "\tinit();\n\n";

      //creates window with a caption
      sMain += "\tglutCreateWindow(\"" + sOutFilename + "\");\n";

      bw.write(sMain, 0, sMain.length());
      bw.newLine();
      /* }
         catch (IOException exc) {
         System.out.println("Begin Main: I/O -> " + exc);
         System.exit(1);
         }
      */
    }

    /*
     * Creates a new object to be added into the scene.
     * The name of the object is added into the vector
     * that retains all names to be used for freeing of
     * memory when closing the file.
     *
     * @param String name - the name of the object to be add
     * @return boolean - true if object added correctly
     *                   false if IO problem occurred
     */
    public boolean addObj(String name)
    {
      try {
          vObjNames.addElement(name);
          String sInsertion = "";
```

```
            bw.write(sInsertion, 0, sInsertion.length());
            bw.newLine();

            return true;
        }
        catch (IOException exc) {
            System.out.println("Add Figure: I/O -> " + exc);
        }
        return false;
      }

      /*
       * Adds all the properties for an object.
       * Currently string must be as to be added exactly.
       *
       * @param String name - the name of the object
       * @param String mod - the modifications to be added
       * @exception IOException
       * @see java.io.IOException
       * @deprecated Preferred to have all objects in vector
       */
      public void addProperties(String name, String mod) throws IOException
      {
        bw.write(mod, 0, mod.length());
        bw.newLine();
      }

      /*
       * Given a vector of Shape objects this method will parse through and
       * add in all the objects and their modifications.
       *
       * @param Vector vObjs - the vector that the tree walker generates
       * @deprecated Shapes were enhanced to use toString()
       */
      private void addObjsFromVector(Vector vObjs)
      {
        String name = "", objsIni = "", objsCoor = "";

        for (int i = 0; i < vObjs.size(); i++)
        {
            name = ((Shape)vObjs.elementAt(i)).name;
            iNumberOfObjects++;

            switch( ((Shape)vObjs.elementAt(i)).type )
            {
              //polygon
              case 0:
                  objsIni += "\tPslPolygon *" + name + " = new PslPolygon(" + name
+ ");\n";
                  break;
              //circle
              case 1:
                  objsIni += "\tPslCircle  *" + name + "  = new PslCircle(0.5, " +
name + ");\n";
                  break;

                  /* //waiting value definitions
```

```
                    objsIni += "\tPslLine *" + name + " = new PslLine(" + name +
");\n";
                    objsIni += "\tPslRect *" + name + " = new PslRect(" + name +
");\n";
                    objsIni += "\tPslSquare *" + name + " = new PslSquare(" +
name + ");\n";
                    objsIni += "\tPslEllip *" + name + " = new PslSquare(" + name
+ ");\n";
                 */
           }

           objsCoor += "\t" + name + "->mass = " +
((Shape)vObjs.elementAt(i)).mass + ";\n";
           objsCoor += "\t" + name + "->velocity.x = " +
((Shape)vObjs.elementAt(i)).velx + ";\n";
           objsCoor += "\t" + name + "->velocity.y = " +
((Shape)vObjs.elementAt(i)).vely + ";\n";
           objsCoor += "\t" + name + "->velocity.z = 0.0;\n";
           objsCoor += "\t" + name + "->acceleration.x = " +
((Shape)vObjs.elementAt(i)).accx + ";\n";
           objsCoor += "\t" + name + "->acceleration.y = " +
((Shape)vObjs.elementAt(i)).accy + ";\n";
           //objsCoor += "\t" + name + "->color = " +
((Shape)vObjs.elementAt(i)).color + ";\n";
       }
     }

     /*
      * Given a vector of Shape objects this method will parse through and
      * add in all the objects and their modifications.
      * @param Vector v - the vector that the tree walker generates
      * @throws IOException
      */
     private void readVector(Vector v) throws IOException
     {
       String sVec = "";
       for (int i = 0; i < v.size(); i++)
       {
           iNumberOfObjects++;
           sVec += (Shape)v.elementAt(i) + "\n";
       }

       bw.write(sVec, 0, sVec.length());
       bw.newLine();
     }

     /*
      * Method called by closeAll that finishes all the necessary
      * lines for openGL and closes the main method.
      * @throws IOException
      */
     private void endMain() throws IOException
     {
       // try {
       String sMain = "";

       for (int i = 0; i < iNumberOfObjects; i++)
```

```
          sMain += "\tscene.addShape(" + ((Shape)vObjNames.elementAt(i)).name +
");\n";

      sMain += "\n";

      //this should never change, it just sets up the OpenGL environment
      //these are the callback functions
      sMain +=
"\tglutDisplayFunc(display);\n\tglutReshapeFunc(scene.resize);\n";
      sMain +=
"\tglutMouseFunc(scene.mouse);\n\tglutKeyboardFunc(scene.keyboard);\n";

      //these may not even be used
      /*
        glutPassiveMotionFunc(scene.passiveMotion);
        glutMotionFunc(scene.drag);
      */

      //the tight loop which keeps our program running
      sMain += "\tglutMainLoop();\n\n";

      //proper garbage collection
      //if( polygon1 ) delete polygon1;
      for (int i = 0; i < iNumberOfObjects; i++)
      {
          sMain += "\tif( " + ((Shape)vObjNames.elementAt(i)).name
            + " ) delete " + ((Shape)vObjNames.elementAt(i)).name + ";\n";
      }

      //int main() needs this
      sMain += "\n\treturn 0;\n}\n";

      bw.write(sMain, 0, sMain.length());
      bw.newLine();
      /*
         } catch (IOException exc) {
         System.out.println("End Main: I/O -> " + exc);
         System.exit(1);
         }
      */
    }

    /*
     *  An important method that MUST be called after a PSL object has been
     *  finished.  All connections to the file that was being written to
     *  are closed.
     */
    public boolean closeAll()
    {
      try {
          endMain();
          bw.close();
          return true; // completed successful
      }
      catch (IOException exce) {
          System.out.println("Could not close properly because there was an I/O
problem: " + exce);
```

```
          System.exit(1);
      }
      return false; // might not have closed due to an IO problem
    }

} // PSL
```

```java
import java.util.Vector;

/*************************************************************
 * Global.java                                               *
 *                                                           *
 * Created: Sat May 10 04:35:49 2003                         *
 * @author Danian J Martinez & Chris Hamer                   *
 * @version 1.0                                              *
 *************************************************************/

public class Global
{
    /*
     *  Environment Variables:
     *
     *  Vector attri -> all attributes of the world
     *  borderOn     -> encapsulating borders
     *  windowSizeX  -> size of window
     *  windowSizeY
     *  bgcolorR     -> color of world
     *  bgcolorG
     *  bgcolorB
     *  gravity      -> gravitational force
     *  zoom         -> window's zoom
     */
    boolean borderOn;
    int windowSizeX, windowSizeY, bgcolorR, bgcolorG, bgcolorB;
    String gravity, dragCoeff, zoom;
    Vector attri;
    static int errornum = 72;

    /***********************************************************
       Constructor:
         creates the global environment.
     ***********************************************************/
    public Global()
    {
      bgcolorR = 0;
      bgcolorB = 0;
      bgcolorG = 0;
      gravity = "-9.8";
      windowSizeX = 800;
      windowSizeY = 800;
      dragCoeff = "0";
      zoom = "20";
      borderOn = false;
      attri = new Vector();
      this.setAttri();
    }


    /*
     * setAttri:
     * Saves all the variables that pertain to the object.  The
     * vector list is designed to be used by the Tree Walker and
     * other methods to maintain a source of the values stored
     * by a Shape.  Each shape has special requirements and
 * therefore varying number of variables.  Also some
 * variables that store more than one value.
 */
private void setAttri()
{
  attri.addElement(new Vari("bgcolor", 3));
  attri.addElement(new Vari("bgcolorR", 1));
  attri.addElement(new Vari("bgcolorG", 1));
  attri.addElement(new Vari("bgcolorB", 1));
  attri.addElement(new Vari("gravity", 1));
  attri.addElement(new Vari("windowSizeX", 1));
  attri.addElement(new Vari("windowSizeY", 1));
  attri.addElement(new Vari("windowSize", 2));
  attri.addElement(new Vari("dragCoeff", 1));
  attri.addElement(new Vari("borderOn", 1));
  attri.addElement(new Vari("zoom", 1));
}

/*
 * setVar:
 *  A unique mutator method that sets the value(s) of any
 * variable in the object.  A Vector is passed for those
 * variables that require more that one stored value.
 *
 * @param String b - variable to be changed
 * @param Vector arg - value(s) to be entered
 */
public void setVar(String b, Vector arg)
{
  if (b.equals("bgcolor"))
  {
      bgcolorR = (int) Double.parseDouble((String) arg.elementAt(0));
      bgcolorG = (int) Double.parseDouble((String) arg.elementAt(1));
      bgcolorB = (int) Double.parseDouble((String) arg.elementAt(2));
  }
  else if (b.equals("bgcolorR")){
      bgcolorR = (int) Double.parseDouble((String) arg.elementAt(0));
  }
  else if (b.equals("bgcolorB")){
      bgcolorB = (int) Double.parseDouble((String) arg.elementAt(0));
  }
  else if (b.equals("bgcolorG")){
      bgcolorG = (int) Double.parseDouble((String) arg.elementAt(0));
  }
  else if (b.equals("gravity")){
      gravity = (String) arg.elementAt(0);
  }
  else if (b.equals("windowSizeX")){
      windowSizeX = (int) Double.parseDouble((String) arg.elementAt(0));
  }
  else if (b.equals("windowSizeY")){
      windowSizeY = (int) Double.parseDouble((String) arg.elementAt(0));
  }
  else if (b.equals("windowSize")){
      windowSizeX = (int) Double.parseDouble((String) arg.elementAt(0));
      windowSizeY = (int) Double.parseDouble((String) arg.elementAt(1));
  }
  else if (b.equals("dragCoeff")){
      dragCoeff = (String) arg.elementAt(0);
  }
  else if (b.equals("zoom")){
      zoom = (String) arg.elementAt(0);
  }

  if (!checkColors())
  {
      System.err.println("Error:");
      System.err.println("Illegal value for the global variable:");
      System.err.print("Global." + b + "( ");
      for (int i= 0; i< arg.size(); i++)
        System.err.print((String) arg.elementAt(0) + " ");
      System.err.println(")");
      System.exit(errornum);
  }
}

/*
  Check certain paramters and closes the program they
  are not valid
*/
private boolean checkColors()
{
  if (bgcolorR < 0 || bgcolorR > 255){
      return false;
  }
  else if (bgcolorG < 0 || bgcolorG > 255){
      return false;
  }
  else if (bgcolorB < 0 || bgcolorB > 255){
      return false;
  }
  else if (Integer.parseInt(dragCoeff) < 0)
      return false;
  else if (windowSizeX < 1)
      return false;
  else if (windowSizeY < 1)
      return false;
  else if (Double.parseDouble(zoom) < 0.0)
      return false;

  return true;
}

/** ACCESSOR METHODS **/
public Vector getAtt(){ return attri; }
public String getClearColor()
{
  String output = "\tglClearColor(";
  output += bgcolorR/(double)255;
  output += ", " + bgcolorG/(double)255;
  output += ", " + bgcolorB/(double)255;
  output += ", 0.0);\n";
  return output;
}
```

```
    public String getWindow()
    {
      String output = "";
      output += "\tglutInitWindowSize(" + windowSizeX +", "+windowSizeY;
      output += ");\n";
      return output;
    }



    /*
     * toString:
     *  this method is called everytime the object is attempted
     * to be printed to stream or file.
     *
     * @return String with information for the ICGenerator
     */
    public String toString()
    {
      String result = "Global Settings: \n";
      result += "gravity = " + gravity;
      result += "\n, window size: (" + windowSizeX + ", " + windowSizeY + ")\n";
      result += "dragCoeff: " + dragCoeff;
      return result;
    }
} // Global
```

```
/**
 * Shape.java
 *
 *
 * Created: Sat May 10 01:08:40 2003
 *
 * @author Danian J Martinez
 * @version
 */
import java.util.*;

public abstract class Shape {
    Vector attri;
    String mass;
    String velx;
    String vely;
    String name;
    int colorR, colorG, colorB;
    String accx;
    String accy;
    String cenx;
    String ceny;
    int type;
    static int errornum = 72;

    public Shape() {
      attri = new Vector();
      mass = "0";
      colorR = 0;
      colorB = 0;
      colorG = 0;
      accx = "0";
      accy = "0";
      velx = "0";
      vely = "0";
      setAttri();
    }

    private void setAttri(){
      attri.addElement(new Vari("color", 3));
      attri.addElement(new Vari("colorR", 1));
      attri.addElement(new Vari("colorG", 1));
      attri.addElement(new Vari("colorB", 1));
      attri.addElement(new Vari("color", 1));
      attri.addElement(new Vari("accx", 1));
      attri.addElement(new Vari("accy", 1));
      attri.addElement(new Vari("cenx", 1));
      attri.addElement(new Vari("ceny", 1));
      attri.addElement(new Vari("velx", 1));
      attri.addElement(new Vari("vely", 1));
      attri.addElement(new Vari("mass", 1));
      attri.addElement(new Vari("acc", 2));
    }

    public void setVar(String b, Vector arg){
      if (b.equals("acc")){
        accx = (String) arg.elementAt(0);
        accy = (String) arg.elementAt(1);
    }
    if (b.equals("accx")){
        accx = (String) arg.elementAt(0);
    }
    else if (b.equals("accy")){
        accy = (String) arg.elementAt(0);
    }
    else if (b.equals("velx")){
        velx = (String) arg.elementAt(0);
    }
    else if (b.equals("vely")){
        vely = (String) arg.elementAt(0);
    }
    else if (b.equals("cenx")){
        cenx = (String) arg.elementAt(0);
    }
    else if (b.equals("ceny")){
        ceny = (String) arg.elementAt(0);
    }
    else if (b.equals("mass")){
        mass = (String) arg.elementAt(0);
    }
    if (b.equals("color")){
        colorR = (int) Double.parseDouble((String) arg.elementAt(0));
        colorG = (int) Double.parseDouble((String) arg.elementAt(1));
        colorB = (int) Double.parseDouble((String) arg.elementAt(2));
    }
    else if (b.equals("colorR")){
        colorR = (int) Double.parseDouble((String) arg.elementAt(0));
    }
    else if (b.equals("colorB")){
        colorB = (int) Double.parseDouble((String) arg.elementAt(0));
    }
    else if (b.equals("colorG")){
        colorG = (int) Double.parseDouble((String) arg.elementAt(0));
    }



    if (!checkColors()){
        System.err.println("Error:");
        System.err.println("Illegal value for the global variable:");
        System.err.print("Global." + b + "( ");
        for (int i= 0; i< arg.size(); i++){
          System.err.print((String) arg.elementAt(0) + " ");
        }
        System.err.println(")");
        System.exit(errornum);
    }
  }

  private boolean checkColors(){

    if (colorR < 0 || colorR > 255){
        return false;
        }
        else if (colorG < 0 || colorG > 255){
            return false;
        }
        else if (colorB < 0 || colorB > 255){
            return false;
        }

        return true;

      }

      public boolean equals(String x){
        return this.name.equals(x);
      }

      public int getType(){
        return type;
      }

      public Vector getAtt(){
        return attri;
      }

      public String toString(){
        String output = "";
        output += "\t" + name + "->com[X] = " + cenx + ";\n";
        output += "\t" + name + "->com[Y] = " + ceny + ";\n";
        output += "\t" + name + "->com[Z] = 0.0;\n";
        output += "\t//" + name + "->mass = " +mass + ";\n";
        output += "\t" + name + "->velocity.x = " + velx + ";\n";
        output += "\t" + name + "->velocity.y = " + vely + ";\n";
        output += "\t" + name + "->velocity.z = 0;\n";
        output += "\t" + name + "->acceleration.x = " + accx + ";\n";
        output += "\t" + name + "->acceleration.y = " + accy + ";\n";
        output += "\t" + name + "->acceleration.z = 0;\n";
        output += "\t" + name + "->color[X] = " + colorR + ";\n";
        output += "\t" + name + "->color[Y] = " + colorG + ";\n";
        output += "\t" + name + "->color[Z] = " + colorB + ";\n";
        output += "\n";
        return output;
      }

      /*
        applyGravity
        This method applies the gravity to the objects accy variable.
        Gravity
      */
      public void applyGravity(String grav){
        accy = "" + (Double.parseDouble(accy) + Double.parseDouble(grav));
      }

  } // Shape
```

```
import java.lang.Math;
import java.util.Vector;

/****************************************************************
 * Line.java                                                    *
 *                                                              *
 *  A line consists of two points, each with an x and y             *
 * coordinate value.  It is an extension of the @see Shape      *
 * object.                                         *
 *                                                              *
 * Created: Sat May 10 01:54:47 2003                            *
 * @author Danian J Martinez & Chris Hamer                      *
 * @version 1.0                                                 *
 ****************************************************************/

public class Line extends Shape
{
    /*
     * Variables:
     *
     *  x1     -> x value of the first point
     *  x2     -> x value of the second point
     *  y1     -> y value of the first point
     *  y2     -> y value of the second point
     *  length -> distance between points
     *
     */
    String x1, x2, y1, y2;
    double length;

    /************************************************************
      Constructor:
        creates a Line object

      @param String a - name of object
      @param String b - x value of first point
      @param String c - y value of first point
      @param String d - x value of second point
      @param String e - y value of second point
      @param int f -  type integer
      @see Shape
      @see java.lang.String
     ************************************************************/
    public Line(String a, String b, String c, String d, String e, int f)
    {
      super();
      type = f;
      name = a;
      x1 = b;
      y1 = c;
      x2 = d;
      y2 = e;
      this.figureLength();
      this.setAttri();
      this.figureCenter();
    }
```

```
/*
 * setAttri:
 * Saves all the variables that pertain to the object.  The
 * vector list is designed to be used by the Tree Walker and
 * other methods to maintain a source of the values stored
 * by a Shape.  Each shape has special requirements and
 * therefore varying number of variables.  Also some
 * variables that store more than one value.
 */
private void setAttri()
{
  attri.addElement(new Vari("x1", 1));
  attri.addElement(new Vari("y1", 1));
  attri.addElement(new Vari("x2", 1));
  attri.addElement(new Vari("y2", 1));
  attri.addElement(new Vari("point1", 2));
  attri.addElement(new Vari("point2", 2));
}

/*
 * setVar:
 *  A unique mutator method that sets the value(s) of any
 * variable in the object.  A Vector is passed for those
 * variables that require more that one stored value.
 *
 * @param String b - variable to be changed
 * @param Vector arg - value(s) to be entered
 */
public void setVar(String b, Vector arg)
{
  if (b.equals("x1"))
  {
      x1 = (String) arg.elementAt(0);
      figureCenter();
  }
  else if (b.equals("x2"))
  {
      x2 = (String) arg.elementAt(0);
      figureCenter();
  }
  else if (b.equals("y1"))
  {
      y1 = (String) arg.elementAt(0);
      figureCenter();
  }
  else if (b.equals("y2"))
  {
      y2 = (String) arg.elementAt(0);
      figureCenter();
  }
  else if (b.equals("point1"))    //example of multivalue variable
  {
      x1 = (String) arg.elementAt(0);
      y1 = (String) arg.elementAt(1);
  }
  else if (b.equals("point2")){
      x2 = (String) arg.elementAt(0);
```

```
            y2 = (String) arg.elementAt(1);
        }
        else
            super.setVar(b, arg);
    }

    /*
     * figureCenter:
     *  Calculates the center of the object.  The center is a vital
     * source of information when dealing with graphics animation
     * and simulation.
     *
     */
    private void figureCenter()
    {
      cenx = "" + ((Double.parseDouble(x1) + Double.parseDouble(x2)) / 2);
      ceny = "" + ((Double.parseDouble(y1) + Double.parseDouble(y2)) / 2);
    }

    /*
     * figureLength:
     *  Determines the length of the shape by calculating the distance
     * formula between two points:
     *     SqaureRoot ( (x2 - x1)^2 + (y2 - y1)^ )
     */
    public void figureLength()
    {
      length = Math.sqrt(Math.pow(Double.parseDouble(x2) -
                             Double.parseDouble(x1), 2)
                      + Math.pow(Double.parseDouble(y2) -
                               Double.parseDouble(y1), 2));
    }

    /*
     * toString:
     *  this method is called everytime the object is attempted
     * to be printed to stream or file.
     *
     * @return String with information for the ICGenerator
     */
    public String toString()
    {
      String output = "";
      name = "PSL_" + name;
      output += "\tPslLine *" + name + " = new PSLLine(\"" + name + "\"";
      output += x1 +", " +y1 + ", " + x2 +", " + y2 +");\n";
      output += super.toString();

      return output;
    }

} // Line
```

```
/**
 * Square.java
 *
 *
 * Created: Sat May 10 01:27:56 2003
 *
 * @author Chris Hamer & Danian J Martinez
 * @version
 */
import java.util.*;

public class Square extends Shape{

    String x1;
    String x2;
    String y1;
    String y2;
    String length;

    public Square(String a, String b, String c, String d, int e) {
      super();
      type = e;
      name = a;
      length = d;
      if (Double.parseDouble(length) < 0){
          System.err.println("Error: ");
          System.err.println("Can not have a negative length");
          System.err.print("def " + name + " Square( " +b + " " +c);
          System.err.print(" " + d + " )");
          System.exit(Shape.errornum);
      }

      x1 = b;
      y1 = c;
      this.figureSides();
      this.setAttri();
    }

    /*
     * toString:
     *  this method is called everytime the object is attempted
     * to be printed to stream or file.
     *
     * @return String with information for the ICGenerator
     */
    public String toString(){
      String output = "";
      name = "PSL_" + name;
      output += "\tPslSquare *" + name + " = new PslSquare(\"" ;
      output += name + "\" " + x1 + ", " + y1 + ", " + length +");\n";
      output += super.toString();
      return output;
    }

    private void setAttri(){
      attri.addElement(new Vari("x", 1));
      attri.addElement(new Vari("y", 1));
```

```
        attri.addElement(new Vari("origin", 2));
        attri.addElement(new Vari("length", 1));
      }

      /*
       * setVar:
       *  A unique mutator method that sets the value(s) of any
       * variable in the object.  A Vector is passed for those
       * variables that require more that one stored value.
       */
      public void setVar(String b, Vector arg){
        if (b.equals("x")){
            x1 = (String) arg.elementAt(0);
            figureSides();
        }
        else if (b.equals("y")){
            y1 = (String) arg.elementAt(0);
            figureSides();
        }
        else if (b.equals("length")){
            length = (String) arg.elementAt(0);
            figureSides();
        }
        else if (b.equals("origin")){
            x1 = (String) arg.elementAt(0);
            y1 = (String) arg.elementAt(1);
            figureSides();
        }
        else
            super.setVar(b, arg);

      }

      private void figureCenter(){
        cenx = "" + (Double.parseDouble(x1) + Double.parseDouble(x2))/2;
        ceny = "" +(Double.parseDouble(y1) + Double.parseDouble(y2))/2;
      }
      private void figureSides(){
        double tempx1 = Double.parseDouble(x1);
        double tempy1 = Double.parseDouble(y1);
        x2 = "" + (tempx1 + Double.parseDouble(length));
        y2 = "" + (tempy1 + Double.parseDouble(length));
        this.figureCenter();
      }


  } // Square
```

```
/**
 * Circle.java
 *
 *
 * Created: Sat May 10 02:28:02 2003
 *
 * @author Chris Hamer & Danian J Martinez
 * @version
 */
import java.util.Vector;

public class Circle extends Shape {
    String radius; // radius of circle
    String x;      // the center's x value
    String y;      // the center's y value
    String points; // the point

    // Constructor takes in the name, center point, and radius
    public Circle(String a, String b, String c, String d, int e) {
      super();
      type = e;
      name = a;
      x = b;
      y = c;
      radius = d;
      points = "10";
      this.setAttri();
      this.figureCenter();
    }

   /*
     * setAttri:
     * Saves all the variables that pertain to the object.  The
     * vector list is designed to be used by the Tree Walker and
     * other methods to maintain a source of the values stored
     * by a Shape.  Each shape has special requirements and
     * therefore varying number of variables.  Also some
     * variables that store more than one value.
     */
    private void setAttri(){
      attri.addElement(new Vari("y", 1));
      attri.addElement(new Vari("x",1));
      attri.addElement(new Vari("origin",2));
      attri.addElement(new Vari("radius",1));
      attri.addElement(new Vari("points",1));
    }

    /*
     * setVar:
     *  A unique mutator method that sets the value(s) of any
     * variable in the object.  A Vector is passed for those
     * variables that require more that one stored value.
     *
     * @param String b - variable to be changed
     * @param Vector arg - value(s) to be entered
     */
    public void setVar(String b, Vector arg){
```

```
  if (b.equals("x")){
      x = (String) arg.elementAt(0);
      figureCenter();
  }
  else if (b.equals("y")){
      y = (String) arg.elementAt(0);
      figureCenter();
  }
  else if (b.equals("radius")){
      radius = (String) arg.elementAt(0);
  }
  else if (b.equals("origin")){
      x = (String) arg.elementAt(0);
      y = (String) arg.elementAt(1);
      figureCenter();
  }
  else if (b.equals("points")){
      points = (String) arg.elementAt(0);
      if (Integer.parseInt(points) < 5){
        System.err.println("Error:");
        System.err.println("A Circle must have at least 5 points");
        System.err.println(name + ".points(" + points +")");
        System.exit(Shape.errornum);
      }

  }
  else
      super.setVar(b, arg);
}

/*
 * figureCenter:
 *  Calculates the center of the object.  The center is a vital
 * source of information when dealing with graphics animation
 * and simulation.
 *
 */
private void figureCenter(){
  cenx = x;
  ceny = y;
}

/*
 * toString:
 *  this method is called everytime the object is attempted
 * to be printed to stream or file.
 *
 * @return String with information for the ICGenerator
 */
public String toString(){
  String output = "";
  name = "PSL_" + name;
  output += "\tPslCircle *" + name + " = new PslCircle( \"";
  output += name + "\", " + cenx + ", " +ceny + ", " +radius;
  output += ", " + points +");\n";
  output += super.toString();
```

```
      return output;
    }

} // Circle
```

```
import java.util.Vector;

/****************************************************************
 * Rect.java                                                    *
 *                                                              *
 *  A rectangle consists of two points, each with an x and y    *
 * coordinate value and a length and width for the size of the  *
 * sides int this shape.  It is an extension of the @see Shape  *
 * object.                                         *
 *                                                              *
 * Created: Sat May 10 02:15:25 2003                            *
 * @author Chris Hamer & Danian J Martinez                      *
 * @version 1.0                                                 *
 ****************************************************************/

public class Rect extends Shape
{
   /*
     * Variables:
     *
     *  x1     -> x value of the first point
     *  y1     -> y value of the first point
     *  length -> size of vertical size
     *  width  -> size of horizontal size
     *
     *  x2     -> x value of the second point
     *  y2     -> y value of the second point
     *
     */
    String x1, y1, length, width, x2, y2;

    /***********************************************************
       Constructor:
         creates a rectangle object

       @param String a - name of object
       @param String b - x value of first point
       @param String c - y value of first point
       @param String l - length
       @param String w - width
       @param int d -  type integer
       @see Shape
       @see java.lang.String
     ***********************************************************/
    public Rect(String a, String b, String c, String l, String w, int d)
    {
      super();
      type = d;
      name = a;
      x1 = b;
      y1 = c;
      length = l;
      width = w;
      this.figureSides();
      this.setAttri();
    }
```

```
/*
 * setAttri:
 * Saves all the variables that pertain to the object.  The
 * vector list is designed to be used by the Tree Walker and
 * other methods to maintain a source of the values stored
 * by a Shape.  Each shape has special requirements and
 * therefore varying number of variables.  Also some
 * variables that store more than one value.
 */
private void setAttri()
{
  attri.addElement(new Vari("x", 1));
  attri.addElement(new Vari("y", 1));
  attri.addElement(new Vari("length", 1));
  attri.addElement(new Vari("width", 1));
  attri.addElement(new Vari("origin", 2));
}

/*
 * figureSides:
 *  Determines the the opposite point from the point, length
 * and width provided by caller.
 *
 */
private void figureSides()
{
  x2 = "" + ((Double.parseDouble(x1)+Double.parseDouble(length)));
  y2 = "" + ((Double.parseDouble(y1)+Double.parseDouble(width)));
  figureCenter();
}

/*
 * figureCenter:
 *  Calculates the center of the object.  The center is a vital
 * source of information when dealing with graphics animation
 * and simulation.
 *
 */
private void figureCenter()
{
  cenx = "" + (Double.parseDouble(x1) + (Double.parseDouble(length) / 2));
  ceny = "" + (Double.parseDouble(y1) + (Double.parseDouble(width) / 2));
}

/*
 * setVar:
 *  A unique mutator method that sets the value(s) of any
 * variable in the object.  A Vector is passed for those
 * variables that require more that one stored value.
 *
 * @param String b - variable to be changed
 * @param Vector arg - value(s) to be entered
 */
public void setVar(String b, Vector arg)
{
  if (b.equals("x"))
  {
```

```
            x1 = (String) arg.elementAt(0);
            figureSides();
        }
        else if (b.equals("y"))
        {
            y1 = (String) arg.elementAt(0);
            figureSides();
        }
        else if (b.equals("length"))
        {
            length = (String) arg.elementAt(0);
            figureSides();
        }
        else if (b.equals("width"))
        {
            width = (String) arg.elementAt(0);
            figureSides();
        }
        else if (b.equals("origin"))
        {
            x1 = (String) arg.elementAt(0);
            y1 = (String) arg.elementAt(1);
            figureSides();
        }
        else
            super.setVar(b, arg);
    }

    /*
     * toString:
     *  this method is called everytime the object is attempted
     * to be printed to stream or file.
     *
     * @return String with information for the ICGenerator
     */
    public String toString()
    {
      String output = "";
      name = "PSL_" + name;
      output += "\tPslRect *" + name + " = new PSLRect(\"" + name + "\"";
      output += x1 +", " + y1 +", " + length + ", " + width + ");\n";
      output +=super.toString();

      return output;
    }

} // Rect
```

```
/**
 * Poly.java
 *
 *
 * Created: Sat May 10 01:27:56 2003
 *
 * @author Chris Hamer & Danian J Martinez
 * @version
 */
import java.util.*;

public class Poly extends Shape{

    Vector x; // x coordinate
    Vector y; // y coordinate

    public Poly(String a, String b, String c, String d, String e, int f) {
      super();
      type = f;
      name = a;
      x = new Vector();
      y = new Vector();
      x.addElement(b);
      y.addElement(c);
      x.addElement(d);
      y.addElement(e);
      this.figureCenter();
      this.setAttri();
    }

    /*
     * toString:
     *  this method is called everytime the object is attempted
     * to be printed to stream or file.
     *
     * @return String with information for the ICGenerator
     */
    public String toString(){
      String output = "";
      name = "PSL_" + name;
      output += "\tPslPolygon *" + name + " = new PslPolygon(\"" ;
      output += name + "\");\n";

      for (int i=0; i < x.size(); i++){
          output +="\t"+name+"->addPoint(" + x.elementAt(i);
          output +=", " + y.elementAt(i) + ", 0);\n";
      }
      output += super.toString();


      return output;
    }

    /*
     * setVar:
     *  A unique mutator method that sets the value(s) of any
     * variable in the object.  A Vector is passed for those
     * variables that require more that one stored value.
     *
     * @param String b - variable to be changed
     * @param Vector arg - value(s) to be entered
     */
    private void setAttri(){
      attri.addElement(new Vari("addPoint", 2));
    }

    public void setVar(String b, Vector arg){
      if (b.equals("addPoint")){
          x.addElement((String) arg.elementAt(0));
          y.addElement((String) arg.elementAt(1));
          this.figureCenter();
      }
      else
          super.setVar(b, arg);

    }

    private void figureCenter(){
      double sumx = 0;
      double sumy = 0;

      for (int i=0; i<x.size(); i++){
          sumx += Double.parseDouble((String)x.elementAt(i));
          sumy += Double.parseDouble((String)y.elementAt(i));
      }
      cenx = "" + (sumx/x.size());
      ceny = "" + (sumy/y.size());
    }
} // Poly
```

```
/**
 * Vari
 *
 *
 * Created: Sat May 10 00:13:37 2003
 *
 * @author Chris Hamer & Danian J Martinez
 * @version
 */

public class Vari  {

    String name;  // the name of the variables
    String value; // value represented as String
    int intvalue; // its value

    // Constructor
    public Vari(String x, String val) {
      name = x;
      value = val;
    }

    // Constructor
    public Vari(String x, int y){
      name = x;
      intvalue = y;
    }

    // Accessor - integer value
    public int getIntValue(){
      return intvalue;
    }

    // Accessor - String value
    public String getValue(){
      return value;
    }

    //Comparison method
    public boolean equals(String x){
      if (this.name.equals(x))
          return true;
      else
          return false;
    }

} // Vari
```

```
#!/bin/sh
ARGS=0
if test $# != $ARGS
then
      A1=`echo $1`
      # compile parser
      if [ "$A1" = "P" ]
      then
            echo "Creating parser"
            java antlr.Tool pslParser.g
            A1=`echo J`
      fi

      # compile java
      if [ "$A1" = "J" ]
      then
            echo "Compiling java files"
            ##javac *.java 2> COMPILATION
            A1=`echo "@`
            ARGS=2
            if test $# != $ARGS
            then
                  echo "USAGE xx.sh [OPTIONS] <filename>"
                  A1=`echo !`
            fi
      else
            NAME=`echo $1`
            # compile everything
            echo "Building project..."
            java antlr.Tool pslLexer.g
            java antlr.Tool pslParser.g
            echo "Compiling..."
            javac *.java 2> COMPILATION
            A1=`echo "@`
      fi

      if [ "$A1" = "@" ]
      then
            chgrp shim *
            chmod 770 *
            A1=`cat COMPILATION`
            echo "-"
            echo $A1
            echo "-"

            # run everything
            if [ "$A1" = "" ]
            then
                  echo "Lets Rock!"
                  java PslCompiler $NAME
            fi
      fi
else
      echo "USAGE xx.sh [OPTIONS] <filename>"
fi

rm -f COMPILATION
```

Appendix 4 Backend C++ files:

```
/***********************************************
 *  pslc
 *  @author John Rodriguez - jr534@columbia.edu
 ***********************************************/

#include <iostream>
#include <cstring>
using namespace std;

#ifdef WIN32
#define FILECHAR '\\'
#else
#define FILECHAR '//'
#endif

int main(int argc, char **argv)
{
      char *lastslash = strrchr(argv[0], FILECHAR);
      char *compileexecname = (!lastslash) ? argv[0] : lastslash + 1;

      if(argc == 1)
      {
            cout << "Usage: " << compileexecname << " infile <outfile>" << endl;
            return 1;
      }

      char *outfilename = (argc == 2) ? "out.pslx" : argv[2];

      int errorcode;

      string javacompilecommand = "java PslCompiler ";
      javacompilecommand += argv[1];
      errorcode = system((char*)javacompilecommand.c_str());
      if(errorcode)
      {
            cout << compileexecname << "("<<errorcode<<"): Error with java 
compilation stage" << endl;
            //cout << compileexecname << ": Please consult your administrator 
about your Java configuration" << endl;
            return 1;
      }

      string cppcompilecommand = "g++";
      cppcompilecommand += " PslCompiler -o ";
      cppcompilecommand += outfilename;
      errorcode = system((char *)cppcompilecommand.c_str());
      if(errorcode)
      {
            cout << compileexecname << "("<<errorcode<<"): Error with 
intermediary compilation stage" << endl;
            //cout << compileexecname << ": Please consult your administrator 
about your C++ compiler configuration" << endl;
            return 1;
      }
```

```
        return 0;
}
```

```
/*************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 *************************************************/

#ifndef __PSLOBJECT_H__
#define __PSLOBJECT_H__

class PslObject
{
      public:
            PslObject(char *n = "");
            virtual ~PslObject();
            virtual void draw();

    protected:
            char *name;
};
#endif
```

```
/***************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ***************************************************/

#include <iostream>
#include <cstdlib>
using namespace std;

#include "PslObject.h"

PslObject::PslObject(char *n):name(n) {}

PslObject::~PslObject()
{
      cout << "psl object destroyed" << endl;
}

void PslObject::draw()
{
}
```

```
/************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ************************************************/


#ifndef __PSLSHAPE_H__
#define __PSLSHAPE_H__

#include <GL/glut.h>

#include "PslObject.h"
#include "PslPoint.h"
#include "PslVector3d.h"

class PslShape : public PslObject
{
      public:
            PslShape(char *name = "");
            virtual ~PslShape();
            virtual void draw();
            virtual void update() {}

            GLubyte color[3];             //color of object

            PslPoint com;                       //center of mass

            PslVector3d velocity;         //velocity vector
            PslVector3d acceleration;     //acceleration vector

            GLdouble theta;                     //angular displacement
            GLdouble omega;                     //angular velocity
            GLdouble alpha;                     //angular acceleration

            GLdouble Kr;                        //coefficient of restitution
(elasticity of collisions)

            GLboolean drawVectors;        //draw vectors
            GLboolean hasCollided;        //collision check
};

#endif
```

```
/***********************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ***********************************************/

#include <iostream>
#include <cstdlib>
using namespace std;

#include <GL/glut.h>
#include "PslShape.h"

PslShape::PslShape(char *name):PslObject(name)
{
      color[0] = 255;
      color[1] = 255;
      color[2] = 255;

      theta = 0.0;
      omega = 0.0;
      alpha = 0.0;

      Kr    = 1.0;

      drawVectors = false;
      hasCollided = false;
}

PslShape::~PslShape()
{
      cout << "shape destroyed" << endl;
}

void PslShape::draw() { glColor3ub(color[0], color[1], color[2]); }
```

```
/*************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 *************************************************/

#ifndef __PSLTIMER_H__
#define __PSLTIMER_H__

class PslTimer
{
      public:
            PslTimer();
            ~PslTimer();
            void wait(double seconds);
            inline void startClock() {;}
            inline void stopClock() {;}
            inline long getElapsedTime() { return endTime - startTime; }
      private:
            clock_t startTime;
            clock_t endTime;
};

#endif
```

```
/***********************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ***********************************************/

/* clock example: countdown */
#include <iostream>
#include <ctime>
using namespace std;

#include "PslTimer.h"

PslTimer::PslTimer() { startTime = endTime = 0; }

void PslTimer::wait ( double seconds )
{
  /*** data type may need to be changed if seconds < 1/CLOCKS_PER_SEC ***/
  clock_t endwait;

  endwait = clock() + seconds * CLOCKS_PER_SEC;
  while (clock() < endwait) {}
}
```

```
/************************************************

 *

 *  @author John Rodriguez - jr534@columbia.edu

 ************************************************/



#ifndef __PSLVECTOR3D_H__
#define __PSLVECTOR3D_H__

#include <iostream>
#include <cstdlib>
using namespace std;

class PslVector3d
{
      public:
            PslVector3d(double=1.0, double=1.0, double=0.0);
            PslVector3d(const PslVector3d &rhs);
            void draw();
            void normalize();
            double x,y,z;

            double operator*(PslVector3d& rhs);
            void operator+=(PslVector3d& rhs);
            void operator-=(PslVector3d& rhs);
            void operator=(double* const values);
            void operator=(PslVector3d& rhs);
            PslVector3d operator+(PslVector3d& rhs);
            PslVector3d operator-(PslVector3d& rhs);
            PslVector3d operator%(PslVector3d& rhs);
            PslVector3d operator*(double scalar);

            friend ostream& operator<<(ostream& os, const PslVector3d& v);
            friend PslVector3d normalizeVector(PslVector3d v);
};

#endif
```

```
/*************************************************

 *

 *  @author John Rodriguez - jr534@columbia.edu

 *************************************************/



#include <iostream>
#include <cstdlib>
using namespace std;

#include <GL/glut.h>

#include "PslVector3d.h"
#include "PslMath3d.h"

PslVector3d::PslVector3d(double _x, double _y, double _z)
{
      x = _x;
      y = _y;
      z = _z;
}

PslVector3d::PslVector3d(const PslVector3d &rhs)
{
      x = rhs.x;
      y = rhs.y;
      z = rhs.z;
}

void PslVector3d::draw()
{
      PslVector3d newp = normalizeVector(*this);
      glBegin(GL_LINES);
            glVertex3f(0.0, 0.0, 0.0);
            glVertex3f(newp.x, newp.y, newp.z);
      glEnd();
}

void PslVector3d::normalize()
{
      double l = LENGTH(*this);
      x/=l;
      y/=l;
      z/=l;
}

double PslVector3d::operator*(PslVector3d& rhs)
{
      return this->x * rhs.x + this->y * rhs.y + this->z * rhs.z;
}

PslVector3d PslVector3d::operator*(double scalar)
{
	return PslVector3d(this->x * scalar, this->y * scalar, this->z * scalar);
}

PslVector3d PslVector3d::operator+(PslVector3d& rhs)
{
	return PslVector3d(this->x + rhs.x, this->y + rhs.y, this->z + rhs.z);
}

PslVector3d PslVector3d::operator-(PslVector3d& rhs)
{
	return PslVector3d(this->x - rhs.x, this->y - rhs.y, this->z - rhs.z);
}

void PslVector3d::operator+=(PslVector3d& rhs)
{
	this->x += rhs.x;
	this->y += rhs.y;
	this->z += rhs.z;
}

void PslVector3d::operator-=(PslVector3d& rhs)
{
	this->x -= rhs.x;
	this->y -= rhs.y;
	this->z -= rhs.z;
}

void PslVector3d::operator=(double* const values)
{
	this->x = values[0];
	this->y = values[1];
	this->z = values[2];
}

void PslVector3d::operator=(PslVector3d& rhs)
{
	this->x = rhs.x;
	this->y = rhs.y;
	this->z = rhs.z;
}

PslVector3d PslVector3d::operator%(PslVector3d& rhs)
{
	return PslVector3d(this->y * rhs.z - this->z * rhs.y,
	                        this->x * rhs.z - this->z * rhs.x,
	                        this->x * rhs.y - this->y * rhs.x);
}

ostream& operator<<(ostream& os, const PslVector3d& v)
{
	os << "(" << v.x << "i + " << v.y << "j + " << v.z << "k)";
	return os;
}

PslVector3d normalizeVector(PslVector3d v)
{
	PslVector3d p = v;
```

```
        p.normalize();
        return p;
}
```

```
/************************************************

 *

 *  @author John Rodriguez - jr534@columbia.edu

 ************************************************/


#ifndef __PSLMATH3D_H__
#define __PSLMATH3D_H__

#include <cmath>
using namespace std;

#define PI 3.1415926
#define DOT(u,v)  ((((u).x)*((v).x))+(((u).y)*((v).y)))
#define LENGTH(w) (sqrt(DOT(w,w)))

#endif
```

```
/**************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 **************************************************/

#include <iostream>
#include <cstdlib>
#include <cassert>
using namespace std;

#include <GL/glut.h>

#include "PslEnvironment.h"
#include "PslShape.h"
#include "PslPolygon.h"

PslEnvironment::PslEnvironment() { zoom = 20.0; }

PslEnvironment::~PslEnvironment()
{
      cout << "environment destroyed" << endl;
      for(int i = 0; i < (int)objects.size(); i++)
            if(objects[i])
                  delete objects[i];
}

void PslEnvironment::addShape(PslShape* shape) { objects.push_back(shape); }

void PslEnvironment::clear() { objects.clear(); }

void PslEnvironment::resize(int w, int h)
{
      if(!h) h = 1;
      glViewport(0,0,(GLsizei)w, (GLsizei)h);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      //gluOrtho2D(0, 5*w, 0, 5*h);
      gluPerspective(45.0, (GLfloat)w/(GLfloat)h, 0.1, 100.0);
    glMatrixMode(GL_MODELVIEW);
}

void PslEnvironment::draw()
{
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

      glLoadIdentity();
      gluLookAt(0.0,0.0,zoom,0.0,0.0,0.0,0.0,1.0,0.0);

      renderContents();
      glutPostRedisplay();

      glutSwapBuffers();
}

void PslEnvironment::renderContents()
{
      int no_polys = (int)objects.size();
	//this will store the new velocities of the shapes if collision
	vector<PslVector3d> Vrebound(no_polys);

	for(int i = 0; i < (int)Vrebound.size(); i++)
		assert(Vrebound[i].x != 0 || Vrebound[i].y != 0);

	vector<PslPoint *> collidepts;

	//check for collisions first
	for(int i = 0; i < no_polys - 1; i++)
	{
		for(int j = i + 1; j < no_polys; j++)
		{
			PslPolygon::collide((PslPolygon&)*objects[i],
(PslPolygon&)*objects[j], collidepts, Vrebound[i], Vrebound[j]);

			//display collision points between p1 and p2 (if any)
			for(int i = 0; i < (int)collidepts.size(); i++)
				collidepts[i]->draw();
		}
		collidepts.clear();
	}

	//now draw then update
	for(int i = 0; i < (int)objects.size(); i++)
	{
		objects[i]->draw();
		//cout << ((objects[i]->hasCollided) ? "true" : "false") << endl;
		if(objects[i]->hasCollided)
		{	//update to reflected velocity and remove marker of collision
			objects[i]->velocity = Vrebound[i];
			objects[i]->hasCollided = false;
		}
		else
			objects[i]->update();
	}
}

void PslEnvironment::mouse(int button, int state, int x, int y)
{
	cout << "test: environment mouse at ("<<x<<","<<y<<")" << endl;
}

void PslEnvironment::keyboard(unsigned char key, int x, int y)
{
	switch(key)
	{
		case 'q': case 'Q':
			exit(0);
		case 'r': case 'R':
			break;
		default: break;
	}
}
```

```
/*************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 *************************************************/

#ifndef __PSLPOINT_H__
#define __PSLPOINT_H__

#define X 0
#define Y 1
#define Z 2

#include <iostream>
using namespace std;

#include "PslVector3d.h"

class PslPoint
{
      public:
            PslPoint(double x = 0.0, double y = 0.0, double z = 0.0);
            PslPoint(PslPoint&);
            virtual ~PslPoint();
            virtual void draw();
            double& operator[] (int i);

            PslPoint operator+(PslPoint& rhs);
            PslPoint operator-(PslPoint& rhs);
            PslPoint operator/(double scalar);
            void operator+=(PslVector3d& rhs);
            void operator-=(PslVector3d& rhs);

            friend ostream& operator<< (ostream& o, PslPoint& p);

      private:
            double coords[3];
};

#endif
```

```
/**********************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 **********************************************/

#include <iostream>
#include <cstdlib>
#include <cassert>
using namespace std;

#include <GL/glut.h>

#include "PslPoint.h"
#include "PslVector3d.h"

PslPoint::PslPoint(double x, double y, double z)
{
      coords[X] = x;
      coords[Y] = y;
      coords[Z] = z;
}

PslPoint::PslPoint(PslPoint& p)
{
      coords[X] = p[X];
      coords[Y] = p[Y];
      coords[Z] = p[Z];
}

PslPoint::~PslPoint()
{
      cout << "PslPoint destroyed" << endl;
}

void PslPoint::draw()
{
      glBegin(GL_QUADS);
            glColor3ub(0,255,0);
            glVertex3d(coords[X]-0.1, coords[Y]-0.1, coords[Z]);
            glVertex3d(coords[X]+0.1, coords[Y]-0.1, coords[Z]);
            glVertex3d(coords[X]+0.1, coords[Y]+0.1, coords[Z]);
            glVertex3d(coords[X]-0.1, coords[Y]+0.1, coords[Z]);
      glEnd();
}

double& PslPoint::operator[] (int i)
{
      assert(i >= X && i <= Z);
      return coords[i];
}

PslPoint PslPoint::operator/(double scalar)
{
      return PslPoint(coords[X]/scalar, coords[Y]/scalar, coords[Z]/scalar);
}

PslPoint PslPoint::operator+(PslPoint& rhs)
```

```
{
	return PslPoint(coords[X]+rhs.coords[X], coords[Y]+rhs.coords[Y],
coords[Z]+rhs.coords[Z]);
}

PslPoint PslPoint::operator-(PslPoint& rhs)
{
	return PslPoint(coords[X]-rhs.coords[X], coords[Y]-rhs.coords[Y],
coords[Z]-rhs.coords[Z]);
}

void PslPoint::operator+=(PslVector3d& rhs)
{
	coords[X] += rhs.x;
	coords[Y] += rhs.y;
	coords[Z] += rhs.z;
}

void PslPoint::operator-=(PslVector3d& rhs)
{
	coords[X] -= rhs.x;
	coords[Y] -= rhs.y;
	coords[Z] -= rhs.z;
}

ostream& operator<< (ostream& o, PslPoint& p)
{
	o << "(" << p[X] << "," << p[Y] << "," << p[Z] << ")";
	return o;
}
```

```
/************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ************************************************/

#ifndef __PSLPOLYGON_H__
#define __PSLPOLYGON_H__

#include <vector>
using namespace std;

#include "PslShape.h"
#include "PslPoint.h"

class PslPolygon : public PslShape
{
      public:
            PslPolygon(char *name = "");
            PslPolygon(PslPolygon & c);
            virtual ~PslPolygon();
            virtual void draw();
            virtual void update();
            inline int size() { return (int)points.size(); }
            void addPoint(double x, double y, double z);
            PslPoint *operator[] (int i);

            friend ostream& operator<<(ostream& o, PslPolygon& p);

            static void collide(PslPolygon &a, PslPolygon &b, vector<PslPoint *>
&collidepts, PslVector3d &p1cv, PslVector3d &p2cv);


      protected:
            vector<PslPoint *> points;    //vector of points
};

#endif
```

```
/************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ************************************************/

#include <cstdlib>
using namespace std;

#include <GL/glut.h>

#define STEP           0.01
#define INFINITY  100000
#define E        0.001

#include "PslPolygon.h"
#include "PslShape.h"
#include "PslPoint.h"
#include "PslVector3d.h"

PslPolygon::PslPolygon(char *name):PslShape(name){}
PslPolygon::PslPolygon(PslPolygon & c){}

PslPolygon::~PslPolygon()
{
      for(int i = 0; i < (int)points.size(); i++)
            if(points[i])
                  delete points[i];
}

void PslPolygon::draw()
{
      //center of mass point
      glColor3ub(0,0,255);
      com.draw();

      PslShape::draw();
      glBegin(GL_LINE_LOOP);
            for(int i = 0; i < (int)points.size(); i++)
                  glVertex3d((*points[i])[X], (*points[i])[Y], (*points[i])[Z]);
      glEnd();

      cout<<this->name<<" has x="<<this->com<<", v="<<this->velocity<< ", a=" <<
this->acceleration<<endl;

      //draw velocity and acceleration vector
      glPushMatrix();
            glTranslated(com[X],com[Y],com[Z]);
            glColor3ub(255,0,0);
            velocity.draw();
            glColor3ub(0,255,0);
            acceleration.draw();
      glPopMatrix();
}

void PslPolygon::update()
{
      //update velocity component
	this->velocity += this->acceleration * STEP;

	//the amount by which all points in primitive are displaced
	PslVector3d displacement( this->velocity * STEP );

	com += displacement;

	//now update all points in polygon
	int size = (int)points.size();
	for(int i = 0; i < size; i++)
		*points[i] += displacement;
}

void PslPolygon::addPoint(double x, double y, double z)
{
	points.push_back(new PslPoint(x, y, z));
}

PslPoint* PslPolygon::operator[] (int i)
{
	int thissize = size();

	//lower bounds check
	while(i < 0) i += thissize;

	//upper bounds check
	while(i >= thissize) i -= thissize;

	return points[i];
}

void PslPolygon::collide(PslPolygon &p1, PslPolygon &p2, vector<PslPoint *>
&cpts, PslVector3d &p1cv, PslVector3d &p2cv)
{
	//for each pair [(x1a,y1a),(x2a,y2a)] in a
	//   for each pair [(x1b,y1b),(x2b,y2b)] in b
	//      check intersection between lines

	int p1size = p1.size();
	int p2size = p2.size();

	for(int i = 0; i < p1size; i++)
	{
		for(int j = 0; j < p2size; j++)
		{
			cout << "CHECKING "<<p1.name<<", LINE "<<(i + 1)<<" VERSUS
"<<p2.name<<", LINE "<<(j + 1)<<endl;

			//look at current points

			PslPoint *a = p1[i];
			PslPoint *b = p1[i+1];
			PslPoint *c = p2[j];
			PslPoint *d = p2[j+1];

			double mAB;
			double mCD;
```

```
                  if((*a)[X] == (*b)[X])
                        mAB = INFINITY;
                  else
                        mAB = ((*a)[Y]-(*b)[Y]) / ((*a)[X]-(*b)[X]);

                  if((*c)[X] == (*d)[X])
                        mCD = INFINITY;
                  else
                        mCD = ((*c)[Y]-(*d)[Y]) / ((*c)[X]-(*d)[X]);

                  double bAB = (*a)[Y] - (*a)[X] * mAB;
                  double bCD = (*c)[Y] - (*c)[X] * mCD;

                  //if lines are parallel, ignore, we wait til intersected
                  if(mCD == mAB) continue;

                  double collideX = (bAB - bCD)/(mCD - mAB);
                  double collideY = (mCD*bAB - mAB*bCD) / (mCD - mAB);

                  bool checkAB_X = false;
                  bool checkAB_Y = false;
                  bool checkCD_X = false;
                  bool checkCD_Y = false;

                  if((collideX >= (*a)[X] - E && collideX <= (*b)[X] + E) ||
(collideX >= (*b)[X] - E && collideX <= (*a)[X] + E))
                        checkAB_X = true;
                  if((collideY >= (*a)[Y] - E && collideY <= (*b)[Y] + E) ||
(collideY >= (*b)[Y] - E && collideY <= (*a)[Y] + E))
                        checkAB_Y = true;
                  if((collideX >= (*c)[X] - E && collideX <= (*d)[X] + E) ||
(collideX >= (*d)[X] - E && collideX <= (*c)[X] + E))
                        checkCD_X = true;
                  if((collideY >= (*c)[Y] - E && collideY <= (*d)[Y] + E) ||
(collideY >= (*d)[Y] - E && collideY <= (*c)[Y] + E))
                        checkCD_Y = true;

                  if(checkAB_X && checkAB_Y && checkCD_X && checkCD_Y)
                  {
//                      //a collision occurred, update the two polygons that
were affected
//
//                      //sides that collided
//                      cout << "line AB = " << *a << *b << endl;
//                      cout << "line CD = " << *c << *d << endl;
//
//                      //calculate normals
//                      PslVector3d Nab((*b)[Y]-(*a)[Y], (*a)[X]-(*b)[X]);
//                      PslVector3d Ncd((*d)[Y]-(*c)[Y], (*c)[X]-(*d)[X]);
//                      Nab.normalize();
//                      Ncd.normalize();
//
//                      cout << normalizeVector(p1.velocity) << endl;
//                      cout << p1.velocity << endl;
//                      cout << p1.velocity << endl;
//
```

```
//                      //for each polygon, find normal vector of side with
which it collided
//                      PslVector3d V_Nab = Ncd * (Ncd * p1.velocity);
//                      PslVector3d V_Ncd = Nab * (Nab * p2.velocity);
//
//                      PslVector3d V_Tab = p1.velocity - V_Nab;
//                      PslVector3d V_Tcd = p2.velocity - V_Ncd;
//
//                      PslVector3d V_Rab = V_Tab - V_Nab * p1.Kr;
//                      PslVector3d V_Rcd = V_Tcd - V_Ncd * p2.Kr;
//
//                      cout << p1.name << " redirected by " << p2.name << " to
" << V_Rab << endl;
//                      cout << p2.name << " redirected by " << p1.name << " to
" << V_Rcd << endl;
//
//glPushMatrix();
//
//    glTranslated(p1.com[X], p1.com[Y], 0);
//
//    glColor3ub(255,0,0);
//    p1.velocity.draw();
//
//    glColor3ub(0,255,0);
//    V_Nab.draw();
//
//    //glColor3ub(0,0,255);
//    //V_Tab.draw();
//
//    //glColor3ub(180,180,180);
//    //V_Rab.draw();
//
//    //glColor3ub(255,255,255);
//    //Nab.draw();
//
//glPopMatrix();
//
//glPushMatrix();
//    glTranslated(p2.com[X], p2.com[Y], 0);
//    glColor3ub(255,0,0);
//    p2.velocity.draw();
//    glColor3ub(0,255,0);
//    V_Ncd.draw();
//    glColor3ub(0,0,255);
//    V_Tcd.draw();
//    glColor3ub(255,255,255);
//    V_Rcd.draw();
//    glColor3ub(128,128,128);
//    Ncd.draw();
//glPopMatrix();
//
//                      /* ASSUMING REFLECTION CORRECTLY CALCULATED */
//                      //p1.velocity = V_Rab;
//                      //p2.velocity = V_Rcd;
//                      //cout << "p1 velocity " << p1.velocity << endl;
//                      //cout << "p2 velocity " << p2.velocity << endl;
//                      /* PLEASE WORK */
```

```
//
//                      p1cv += V_Rab;
//                      p2cv += V_Rcd;
//                      p1.hasCollided = true;
//                      p2.hasCollided = true;
//
                        cpts.push_back(new PslPoint(collideX, collideY, 0));

                        cout << "COLLISION: BETWEEN " << p1.name
                               << " and " << p2.name
                               << " at (" << collideX << "," << collideY << ")"
                               << endl;
                  }
            }
      }
}

ostream& operator<<(ostream& o, PslPolygon& p)
{
      o << p.name << " ";
      for(int i = 0; i < p.size(); i++) o << *(p[i]);
      return o;
}
```

```
/**************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 **************************************************/

#ifndef __PSLLINE_H__
#define __PSLLINE_H__

#include "PslPolygon.h"

class PslLine : public PslPolygon
{
      public:
            PslLine(char *name = "", double x1=0.0, double y1=0.0, double
x2=0.0, double y2=0.0);
            virtual ~PslLine();
};

#endif
```

```
/***********************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ***********************************************/

#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;

#define PIOVER180 0.01745329

#include <GL/glut.h>

#include "PslLine.h"
#include "PslPolygon.h"

PslLine::PslLine(char *name, double x1, double y1, double x2, double
y2):PslPolygon(name)
{
      PslPolygon::addPoint(x1,y1,0);
      PslPolygon::addPoint(x2,y2,0);
      PslPolygon::com[X] = (x1 + x2) / 2;
      PslPolygon::com[Y] = (y1 + y2) / 2;
}

PslLine::~PslLine()
{
      cout << "destroyed line" << endl;
}
```

```
/***************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ***************************************************/

#ifndef __PSLCIRCLE_H__
#define __PSLCIRCLE_H__

#include "PslPolygon.h"

class PslCircle : public PslPolygon
{
	public:
		PslCircle(char *name = "", double _cx = 0.0, double _cy = 0.0,
double _r = 1.0, int n = 10);
		virtual ~PslCircle();

	private:
		double r;
};

#endif
```

```
/**************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 **************************************************/

#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;

#define PIOVER180 0.01745329

#include <GL/glut.h>

#include "PslCircle.h"
#include "PslPolygon.h"

PslCircle::PslCircle(char *name, double _cx, double _cy, double _r, int
n):PslPolygon(name)
{
      PslPolygon::com[X] = _cx;
      PslPolygon::com[Y] = _cy;
      r = _r;
      double interval = 360.0 / n;
      for(double theta = 0; theta < 360.0; theta += interval)
            PslPolygon::addPoint(_cx + r*cos(theta*PIOVER180), _cy +
r*sin(theta*PIOVER180), 0.0);
}

PslCircle::~PslCircle()
{
      cout << "destroyed circle" << endl;
}
```

```
/***********************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ***********************************************/

#ifndef __PSLRECTANGLE_H__
#define __PSLRECTANGLE_H__

#include "PslPolygon.h"

class PslRectangle : public PslPolygon
{
	public:
		PslRectangle(char *name = "", double x = 0.0, double y = 0.0, double 
w = 1.0, double h = 1.0);
		virtual ~PslRectangle();
};

#endif
```

```
/************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ************************************************/

#include <iostream>
using namespace std;

#include <GL/glut.h>
#include "PslRectangle.h"

PslRectangle::PslRectangle(char *name, double x, double y, double w, double
h):PslPolygon(name)
{
      PslPolygon::addPoint(x  , y  , 0.0);
      PslPolygon::addPoint(x+w, y  , 0.0);
      PslPolygon::addPoint(x+w, y+h, 0.0);
      PslPolygon::addPoint(x  , y+h, 0.0);
}

PslRectangle::~PslRectangle() { cout << "destroyed PslRectangle" << endl; }
```

```
/************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ************************************************/

#ifndef __PSLSQUARE_H__
#define __PSLSQUARE_H__

#include "PslRectangle.h"

class PslSquare : public PslRectangle
{
      public:
            PslSquare(char *name = "", double x = 0.0, double y = 0.0, double s
= 1.0);
            virtual ~PslSquare();
};

#endif
```

```
/************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ************************************************/

#include <iostream>
using namespace std;

#include <GL/glut.h>
#include "PslSquare.h"

PslSquare::PslSquare(char *name, double x, double y, double
s):PslRectangle(name, x, y, s, s) {}
PslSquare::~PslSquare() { cout << "destroyed PslSquare" << endl; }
```

```
/************************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ************************************************/

#ifndef __PSLENVIRONMENT_H__
#define __PSLENVIRONMENT_H__

#include <vector>
using namespace std;

#include "PslObject.h"
#include "PslShape.h"

class PslEnvironment : public PslObject
{
	public:
		PslEnvironment();
		virtual ~PslEnvironment();

		void addShape(PslShape* shape);
		void clear();

		void draw();
		void renderContents();

		static void mouse(int button, int state, int x, int y);
		static void keyboard(unsigned char key, int x, int y);
		static void resize(int w, int h);
		GLdouble gravity;
		GLdouble zoom;

	protected:
		vector<PslShape *> objects;
};

#endif
```

```
/***********************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ***********************************************/

#ifndef __PSLCIRCLE_H__
#define __PSLCIRCLE_H__

#include "PslPolygon.h"

class PslCircle : public PslPolygon
{
      public:
            PslCircle(char *name = "", double _cx = 0.0, double _cy = 0.0,
double _r = 1.0, int n = 10);
            virtual ~PslCircle();

      private:
            double r;
};

#endif
/***********************************************
 *
 *  @author John Rodriguez - jr534@columbia.edu
 ***********************************************/

#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;

#define PIOVER180 0.01745329

#include <GL/glut.h>

#include "PslCircle.h"
#include "PslPolygon.h"

PslCircle::PslCircle(char *name, double _cx, double _cy, double _r, int
n):PslPolygon(name)
{
      PslPolygon::com[X] = _cx;
      PslPolygon::com[Y] = _cy;
      r = _r;
      double interval = 360.0 / n;
      for(double theta = 0; theta < 360.0; theta += interval)
            PslPolygon::addPoint(_cx + r*cos(theta*PIOVER180), _cy +
r*sin(theta*PIOVER180), 0.0);
}

PslCircle::~PslCircle()
{
      cout << "destroyed circle" << endl;
}
```

Global

- This is the environment that governs a scene during a simulation. This is where all the objects reside and interact with each other. The Global object contains all the default and user defined value for the variables that govern the scene. This object allows the user to modify the window in a restricted manner in order to ensure simplicity. A user is able to increase or decrease the size of a window, alter the gravitational force applied onto all objects, or simply select a color that is more applying to the eyes.

.bgcolor (double red, double blue, double green)

The user enters the value using the RGB scale with value ranging from 0 to 255

.gravity (double g)

Accepts any value for this acceleration but remember that a negative value would have a direction towards the world which is the bottom of the window.

.windowSize (double x, double y)
Sets the width of the output window to x and the height to y
.windowSizeX (double x)
Sets the width of the output window to x
.windowSizeY (double y)
Sets the height of the output window to y
.zoom
Sets the camera zoom

Shape

- This is the base object from which all PSL objects are defined. It is an abstract class that defines the bare variables such as acceleration and velocity. All Shapes have these variables.

.color(double red, double green, double blue)

The user enters the value using the RGB scale with value ranging from 0 to 255.

.color(double red)

Sets the red component of the color.

DEFAULT: red = 0

.color(double green)

Sets the green component of the color.

DEFAULT: green = 0.0

.color(double blue)

Sets the blue component of the color.

DEFAULT: blue = 0.0

.acc(double x, double y)

Sets the x component and y component of acceleration

.accx(double x)

Sets the x component of the acceleration

DEFAULT: accx = 0.0

.accy(double y)

Sets the y component of the acceleration

.velx(double x)

Sets the x component of velocity

.vely(double y)

Sets the x component of velocity

Shape

|

|→Line

- Line segment is composed of a point (x1, y1) and an endpoint (x2, y2). From this very simple shape, one can construct almost anything from text to more complex shapes and objects.
- Constructor: Line (double x1, double y1, double x2, double y2);
  def shape1 Line (0, 0, 4, 4); → a simple define statement that creates a line from the graphical origin to the point (4, 4).

.x1 (double x1)

Set the x value for first point

.x2 (double x2)

Set the x2 value for first point

.y1 (double .y1)

Set the y1 value for first point

.y2 (double y2)

Set the y2 value for first point

Shape

|

|→Square

- Square is composed of a point (x1, y1) and the length of a side. From this one length, the coordinates for the other corners are calculated, and the center is determined.
- Constructor: Square (double x1, double y1, double length);
  def shape2 Square (0, 0, 4); → a simple define statement that creates a square from the graphical origin to all points of distance 4.

.x (double x)

Set the x value of the point

.y (double y)

Set the y value of the point

.length (double s)

Set the length of the side.

Shape
|
|→ Rect

- Rectangles compose of a point (x1, y1), length, and width
- Constructor: Rect (double x, double y, double length, double width);
  def shape3 Rect (0, 0, 4, 2); → a simple define statement that creates a rectangle from the graphical origin to the length of four and width of two.

.x (double x)

Set the x value of the point

.y (double y)

Set the y value of the point

.length (double l)

Set the length of two sides

.width (double w)

Set the size of the two sides

Adds another point to the series of points that will compose of the final project.

Shape
|
|→ Poly

- Polygon is the most dynamic for its ability to get both concave and convex shapes and its seeming less endless size capacity.
- Constructor: Poly (double x1, double y1, double x2, double y2);
  def shape4 Poly (1, 1, 2, 2); → This statement that creates the beginning to almost any polygon the user desires. Beginning with a point at (1, 1) and another at (2, 2).

.addPoint (double x, double y)

Shape
|
|→ Circle

- In PSL a circle is just a collection of points at a given distance from the center rotated 360. The number of points is defined through the points variable.
- Constructor: Circle (double x, double y, double radius);
  def shape4 Circle (1, 1, 2, 2); → .

.points(double p)

Sets the number of points the number of points that make up the circle.

.x(double x)

Sets the x value of the center of the circle.

.y(double y)

Sets the y value of the center of the circle.

.origin(double x, double y)

Sets the x and y values of the center of the circle.

.radius(double x)

Sets the radius of the circle