FunkGL

John Gallagher, Mack Lu, Christos Savvopoulos, Oren Sivan
{*jmg2016,yl2194,cs2467,os2109*}@columbia.edu

May 7, 2007

Contents

Chapter 1

Introduction

FunkGL is a language designed for the purpose of 3D graphics manipulation. The language offers a simple interface to specify an environment, create objects within the environment, and algorithmically manipulate the objects.

1.1 Background

FunkGL follows the "update-render cycle" model of real-time graphics. The update part of the cycle determines what should be drawn and how, and the render part of the cycle actually draws the objects onto the screen. In designing our language, we aimed to abstract away redundancies and similarities in updating and rendering. FunkGL creates a large state machine that keeps track of the state of the environment and all objects. The state of each object is updated through functional programming and rendered by OpenGL libraries every cycle. This simple paradigm allows us to harness the power of OpenGL in a flexible way to create 3D graphics.

1.2 Motivation

Our main motivation for creating FunkGL is to be able to simplify basic 3D object manipulation and interactions, in the process reducing the amount of code necessary for rendering procedures. To create a scene in OpenGL requires setting up lights, cameras, and objects in a format that the OpenGL libraries can understand. Doing so takes a lot of meticulous yet tedious coding, eating up valuable hours of a graphics programmer's time. FunkGL will attempt to solve this problem by taking care of unnecessary configurations and make it easier for code to go straight to work. Rendering objects is also a cumbersome task when programming with OpenGL. Ffor example to draw a blue triangle, you would have to call three functions: one to change the current position of the desired triangle object, one to change the color attribute of the triangle object to blue, and one to render the triangle. The triangle is the basic working unit of OpenGL but for complex objects, it is hard to work with and error prone.

Chapter 2

Language Tutorial

Mandatory attributes such as the "file" attribute of a mesh must be declared before user-defined attributes. User-defined attributes describe the state of an object. To change the state of an object

2.1 Simple Example

In this example, we will explore how to create some simple objects with FunkGL and to manipulate these objects through attribute functions. We will draw a sphere object and have it increase gradually in size.

A FunkGL program consists of declarations and functions. Declarations defines a concrete object as well as all the attributes and initial values associated with that object. There three main types of objects: meshes, representing 3D models, materials, representing the texture of a mesh, and cameras, representing the perspective. Objects contain attributes that describe their internal state.

We can create an object using the follow FunkGL syntax:

```
mesh sphere {
  file = "sphere.obj";
  vector pos = (0.,0.,0.);
  vector scale =(0.5,0.5,0.5);
  float time = 0.;
}
```

This creates a mesh object named sphere with four attributes. The first attribute is named "file" and its initial value is "sphere.obj". A mesh declaration requires a mandatory "file" attribute and the value is loaded at runtime to create an internal representation of the sphere in memory. The second attribute is an OpenGL specifc attribute called pos with a static vector value and will place the object at the origin of the OpenGL space. The third attribute is also OpenGL specific and will scale all dimensions of the object by 1/2 of its default size specified in `sphere.obj`. The fourth attribute is a user-defined attribute counter for the time.

All attribute values of the sphere are static which makes for a pretty boring sphere. Let's spice it up by making the attibutes change with time. In order to change the value of an attribute, we must create a function and "attach" the function to the attribute we want to change.

In every update cycle, the program scans all the attributes of all the objects and calls any attached functions with their specified parameters. The returned value from these functions is then set to the attribute it is attached to. This allows attributes to change over time.

To create a ball that increases with size, we'll create two functions, timeUpdate and crazyScale, to attach to time and pos respectively.

We can create functions using the following syntax:

```
float timeUpdate():sphere.time+dt;
vector crazyScale():(sphere.time/10.,sphere.time/10.,sphere.time/10.);
```

The variable dt is a global implicit attribute that reports the difference in time since the last frame. Notice that the function crazyScale can access sphere.time directly. This is because the scope manager keeps track of both the local scope and the outer declarative scope.

Functions, like e-mails, are no fun without attachments. To attach these functions to attributes, we modify the sphere declaration to look like this:

```
mesh sphere{
file="sphere.obj";
vector position=(0.,15.,0.);
vector vel=(0.,0.,0.);
vector scale=crazyScale();
~scale=(0.5,0.5,0.5);
float time=timeUpdate();
~time=0.;
}
```

Notice that the syntax for attaching a function is the same for assigning a static value to an attribute. The next line after attaching the function is an initialization expression. If the initialization expression is omitted, default values will be used.

The object is now fully defined and will gradually grow in size as the program runs. All that's left to do is to specify a light, a camera, and we're ready for action!

See the next page for a complete code listing for the example.

simple.fgl

```
camera cam{
vector position=cam.position;
~position=(0.,0.,-50.);
vector center=cam.center;
~center=(0.,0.,0.);
vector up=cam.up;
~up=(0.,1.,0.);
}

//default lighting
light l{
}

mesh sphere{
file="sphere.obj";
vector position=(0.,15.,0.);
vector vel=(0.,0.,0.);
vector scale=crazyScale();
~scale=(0.5,0.5,0.5);
float time=timeUpdate();
~time=0.;
}

//update time
float timeUpdate():sphere.time+dt;

//scales with time
vector crazyScale():(sphere.time/10.,sphere.time/10.,sphere.time/10.);
```

2.2 Compiling and Running

The compilation of an fgl program takes three steps. The process is initialized by the funkglc shell script. This script takes three input parameters: the platforms (win, linux, or mac), the source .fgl file, and the ouput native executable file. The first step passes the source .fgl file to a java executable, FunkGLC, which compiles the .fgl file into a c++ file containing include directives to the c++ opengl and other support libraries. This compilation uses the antlr-generated L, P, and W classes which lex and parse the file and walk the resulting syntax tree. The second step compiles the c++ support libraries if necessary using a makefile target based on the platform argument. Thirdly, the temporary c++ file generated by FunkGLC is compiled and statically linked with the library and the output file is saved to the location specified by the third argument. The script makes sure the FunkGLC produced no errors before attempting a c++ compilation.
Format of the script is:
`/funkglc [win|linux|mac] sourcefile.fgl targetexecutablename`
For example to run gcd.fgl on a linux machine:
`funkglc linux edu/columbia/cs/vicecity/test/gcd.fgl ../funkgl/gcd`

2.3 Complex Examples

Please see Appendix A for more complex examples.

Chapter 3

Language Reference Manual

3.1 Lexical Conventions

A program consists of one or more series of definitions. There are two types of definitions: Declarative and Functional. The former specifies 3D objects and their attributes, whereas the latter specifies functions that can be used to update the attributes of objects.

3.2 Comments

The characters "/*" introduce a multi-line comment, which terminates with "*/". Multi-line comments do not nest. The characters "//" introduce a single-line comment, which terminates with the next new line character. LF and CR by themselves represent a single new line whereas CRLF represents two new lines.

3.3 Tokens

There are six classes of tokens: identifiers, keywords, floating constants, operators, string constants and other separators. White space is only meaningful in the context of separating otherwise adjacent tokens and is otherwise ignored.

3.4 Identifiers

An identifier is a sequence of letters and digits. The first character must be a letter. Upper and lower case letters are treated as different letters. Underscores are considered letters.

3.5 Keywords

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

mesh	material	light	camera
float	list	vector	file

In addition, any C++ keyword is also reserved an may not be used as an identifier

3.6 Constants

There are four types of constants: floating,vector, list, and string constants.

Constant → *FloatingConstant*
| *VectorConstant*
| *ListConstant*
| *StringConstant*

3.6.1 Floating Constants

Floating constants are the computer equivalent of scientific notation and serve as atomic elements of lists and vectors. The exact number of significant figures and the range of possible exponents are machine-specific; FunkGL always uses the C data type double to store floating-point numbers. Floating-point constants are defined in a very similar way to C, that is:

> "A floating constant consists of an integer part, a decimal part, a fraction part, an e or E, an optionally signed integer exponent" ... " the integer and fraction parts both consist of a sequence of digits. Either the integer part, or the fraction part (not both) may be missing; either the decimal point or the e and the exponent (not both) may be missing." - The C Programming Language

3.6.2 Vector Constants

VectorConstant → '(' *FloatConstant* ',' *FloatConstant* ',' *FloatConstant* ')'

A vector constant is a structure of three floating-point constants, representing magnitudes in the spatial x, y and z dimensions. Vectors are used to represent many object attributes, such as position, velocity, or even color. A vector is strictly of the form "(x,y,z)" where x,y,z represent three floating-point constants.

3.6.3 List Constants

ListConstant → '[' (*ListInner*)? ']'
ListInner → *Constant* (',' *Constant*)*

Lists are implemented as singly linked lists to allow for maximum flexibility. Linked lists are simple to modify and iterate through. A list may contain a floating-point constant, a vector constant or another list constant as an element. An empty list is declared as `[]`.

3.6.4 String Constants

A string constant is a sequence of characters surrounded by double quotes. String constants are only used to specify the pathname (relative to the directory of the executable program) of the file that represents an object. As such, they only appear in declarative definitions. Specifically, they are character strings surrounded by double quotes.

3.7 Basic Types

There are four data types: floating-point numbers, vectors, lists, and strings whose data type labels are 'float', 'vector', 'list', and 'string' respectively. A data type label indicates that the data representation of the memory held by the identifier should be interpreted as the type specified when

placed in front of an attribute or function parameter. It indicates that the data representation of the result of a function should be interpreted as the type specified when placed in front of a function definition.

3.8 Objects

Objects are the basic building blocks of rendered scenes. Each object has attributes describing various characteristics of the object. Objects are different from data types because they cannot be used as arguments. An important distinction between objects in FunkGL and objects in traditional Object-Oriented languages such as Java is that FunkGL objects are statically created based on declarative definitions. Thus, they cannot be dynamically created with functional definitions. There are five types of objects: meshes, materials, lights, cameras, and prints. See section below on declarative definitions for more information on specific objects.

3.9 Declarative Definitions

ObjectDecl → *Mesh*
| *Material*
| *OtherObject*

Declarative definitions are used to define objects and their attributes. An attribute may be linked to a functional definition through the assignment symbol =. The function that the attribute is attached to updates the value of the attribute on every update cycle. The following are templates for the declarative definitions of all five object types. Note that each object type has some special attributes that, when declared, have unique and predefined consequences on the rendering of that object (such as the position of certain objects). If a special attribute is not declared, OpenGL's default values will be used. All other attributes are extra attributes and are used exclusively to that particular object.

An attribute is defined as:

Attribute → *TypeSpecifier Identifier* '`=`' *Expression* '`;`'
'∼' *Identifier* '`=`' *Constant* '`;`'

3.9.1 Meshes

Mesh → '`mesh`' *Identifier* `{`
'`file`' '`=`' *StringConstant* '`;`'
('`material`' '`=`' *Identifier* '`;`')?
(*Attribute*)*
`}`

A mesh is a collection of triangles. Special attributes: file (string constant), position (vector), rotation (vector), scale (vector), and material (identifier for a material object).

3.9.2 Materials

Material → '`material`' *Identifier* {
'file' '=' *StringConstant* ';'
(*Attribute)**
}

A material is used as an attribute for a mesh and specifies the mesh's appearance. Special attributes: ambient (vector), diffuse (vector), specular (vector), and shininess (float).

3.9.3 Light

OtherObject → ('`light`' | '`camera`') *Identifier*
'`{`' (*Attribute*)* '`}`'

A light is necessary for meshes to appear three-dimensional. Without lighting, and the gradients light creates, it is impossible for humans to perceive depth. Special attributes: ambient (vector), diffuse (vector), specular (vector), position (vector), attenuation (vector), and w (float).

3.9.4 Camera

OtherObject → ('`light`' | '`camera`') *Identifier*
'`{`' (*Attribute*)* '`}`'

A camera is defined once. It allows the user to move through the scene. Special attributes: position (vector), direction (vector), and up (vector).

3.10 Functional Declarations

Functional definitions define the operations that update the attributes of each object. As the name suggests, these definitions are purely functional. Statements cannot be sequenced. Functions are evaluated concurrently based on the current state, and changes will only be manifest after the next update-render cycle. Functional definitions are of the form:

FunctionDecl → *TypeSpecifier Identifier* '`(`' (*Argument*)* '`)`' `:`'*FunctionBody* '`;`'
Argument → *TypeSpecifier Identifier*
FunctionBody → *Expression*

Where type specifies the return type, function specifies the function name, and parameter specifies the parameter name. Expression is any expression that evaluates to a data type consistent with the return type of the function and will be the return value of the function.

3.11 Expressions

Expression → *ConditionalExpression*

Expressions are constructed of floating point values, vectors, lists, or function calls, combined using conditional or mathematical operators which ultimately return the data type specified by the function definition which precedes it. Expressions types are listed here in order of precedence (greatest to least). Floats, vectors, and lists have already been discussed above under constants.

3.11.1 Conditional Expressions

ConditionalExpression → *OrExpression* ('`?`' *Expression* '`:`' *Expression*)?

The first expression is evaluated, including all side effects. If the result is true (1.0), the result is the value of the second expression, otherwise that of the third expression. Only one of the second and third operands is evaluated. Note that because there are only floating-point numbers in FunkGL, Boolean expressions should only be produced by either logical operators or built-in functions to ensure correct/exact "true" and "false" values.

3.11.2 Logical Operators

OrExpression → *AndExpression* ('`||`' *AndExpression*)*
AndExpression → *EqualityExpression* ('`&&`' *EqualityExpression*)*

These return 1.0 on evaluating true and 0.0 otherwise and are used in infix notation. The OR '`|`' operator returns 1.0 if either of its operands are unequal to 0.0. The AND `&` operator returns 1.0 if both of its operands are unequal to 0.0. AND takes precedence over OR.

3.11.3 Equality Operators

EqualityExpression → *ComparativeExpression* ('`==`' | '`!=`' *ComparativeExpression*)*

Vectors and lists can be element-wise compared for equality and inequality. Due to the imprecision associated with floating-point numbers, care should be taken when testing the equality of two floating-point numbers.

3.11.4 Comparative Operators

ComparativeExpression → *AdditiveExpression* (('`<`' | '`<=`' | '`>`' | '`>=`') *AdditiveExpression*)*

These return 1.0 on evaluating true and 0.0 otherwise and are used in infix notation. Floating point numbers can be compared as expected.

3.11.5 Additive Operators

AdditiveExpression → *MultiplicativeExpression* ('`+`' | '`-`' *MultiplicativeExpression*)*

The + and − operators are grouped from left-to-right. Addition and subtraction between floating numbers and between floating point numbers and vectors follow the standard rules of mathematics and are used in infix notation. That is, floating point values can be added and subtracted as expected. Vectors can be added or subtracted with other vectors according to vector addition rules. Mathematical operations can be grouped using parentheses.

3.11.6 Multiplicative Operators

MultiplicativeExpression → *UnaryExpression* ('`*`' | '`/`' *UnaryExpression*)*

The * and / operators are grouped from left-to-right. A vector multiplied by a vector results in a floating-point number representing the dot product. If a vector is multiplied by a scalar, the vector is element-wise multiplied by the scalar or its inverse, respectively. Vector-to-vector, vector-to-float, and float-to-vector division, as well as any mathematical operation on lists, is illegal.

3.11.7 Unary Operators

UnaryExpression → *PostfixExpression*
| '`!`'*UnaryExpression*
| '`-`'*UnaryExpression*

The logical inversion operator, written before a floating point number, that returns 1.0 if the floating point number is 0.0, and returns 0.0 otherwise.

The logical negation operator, written before a floating point number, returns the negative value of the floating point number.

3.11.8 Postfix Expressions

PostfixExpression → *PimaryExpression* ('`.`' *Identifier*)?

`objectname.attributename` accesses the attribute of a specific object.
`vectorname.x` or `vectorname.y` or `vectorname.z` accesses an element of a vector.
`listname.first` accesses the first element of a list

`listname.rest` access the rest of the list
`listname.mpty` checks if the list is empty

3.11.9 Primary Expressions

PrimaryExpression → *Identifier* '`(`' (*ExpressionList*)? '`)`'
| *FloatConstant*
| '`(`' *Expression* ('`)`' | '`,`' *Expression* '`,`' *Expression* '`)`')
| `[` *ExpressionList* `]`
ExpressionList → *Expression* (`,` *Expression*)*
Primary expressions consists of identifiders, constants, or parenthesized expressions.

Function Call Calls to functions are made by naming naming the function to be called, and passing an expression for each of the defined parameters of the function whose evaluation matches the data type specified for that parameter. For simplicity, evaluation is always strict. That is, f(g(x)) will always substitute the result of g(x) before evaluating f. This means that expressions passed as parameters must each be fully evaluated to a floating point number, a constant vector, or constant list before the function is actually called. A function call is a postfix expression, whose arguments are specified as an argument expression list.

3.12 Operator Precedence

Order of precedence is implicit in the context free grammar, with `()` the highest and `=` being the lowest

Operation	Description	Association
`()`	Nesting expressions	left-to-right
`.`	Inner element accessor	left-to-right
`!`	Logical Inversion	N/A
`-`	Unary Negation	
`*`	Muliplication	left-to-right
`/`	Division	
`+`	Addition	left-to-right
`-`	Subtraction	
`<`	Logical greater than	left-to-right
`<=`	Logical greater than or equal to	
`>`	Logical less than	
`>=`	Logical less than or equal to	
`==`	Logical is equal to	left-to-right
`!=`	Logical not equal to	
`&&`	Logical and	left-to-right
`||`	Logical or	left-to-right
`?:`	Conditional	right-to-left
`=`	Assignment	right-to-left

3.13 Global Attributes

Global attributes allow for user to probe relevant information about the environment in which they run. These attributes have internal functions attached to them and runs independently of any other functions. If declared in a normal object they would look like:

```
{
...
float dt=...;
float mousex=...;
float mousey=...;
float mouselpressed=...;
float mouserpressed=...;
...
}
```

dt is the actual elapsed time between this frame and the last
mouse x is the origin-relative x position of the mouse pointer
mousey is the origin-relative y position of the mouse pointer
mouselpressed is the left mouse button depressed during this frame with 0.0 representing not pressed and 1.0 representing pressed
mouserpressed is the right mouse button depressed during this frame with 0.0 representing not pressed and 1.0 representing pressed

3.13.1 Built-in Functions

There are a few built-in functions to help programmers write code for FunkGL.

Mathematical built-in functions:
`float sin(float x);`
Provides same functionality as C++ `sinf` function.
`float cos(float x);`
Provides same functionality as C++ `cosf` function.
`float exp(float x);`
Provides same functionality as C++ `exp` function.
List specific built-in functions:
`listconstant.mpty`
Return 1.0 if a list is empty, 0.0 otherwise.
`listconstant.first`
Returns the first element of a list. Could be a float, a vector, or another list.
`listconstant.rest`
Returns a copy of listconstant without the first element. Could be a float a vector or another list.
`list cons(expr inputfirst, list inputrest)`
Returns a new list with inputfirst as the first element and inputrest as the rest of the list. `inputfirst` can be a float, vector, or list.

3.14 Context Free Grammar

Program → (*FunctionDecl* | *ObjectDecl*)+
ObjectDecl → *Mesh*
| *Material*
| *OtherObject*
Mesh → '`mesh`' *Identifier* `{`
'`file`' '`=`' *StringConstant* '`;`'
('`material`' '`=`' *Identifier* '`;`')?
(*Attribute*)*
`}`
Material → '`material`' *Identifier* {
'file' '=' *StringConstant* ';'
(*Attribute)**
}
OtherObject → ('`light`' | '`camera`') *Identifier*
'`{`' (*Attribute*)* '`}`'
Attribute → *TypeSpecifier Identifier* '`=`' *Expression* '`;`'
'∼' *Identifier* '`=`' *Constant* '`;`'
Constant → *FloatConstant*
| *VectorConstant*
| *ListConstant*
VectorConstant → '`(`' *FloatConstant* '`,`' *FloatConstant* '`,`' *FloatConstant* '`)`'
ListConstant → '`[`' (*ListInner*)? '`]`'
ListInner → *Constant* ('`,`' *Constant*)*

FunctionDecl → *TypeSpecifier Identifier* '`(`' (*Argument*)* '`)`' :'*FunctionBody* '`;`'
Argument → *TypeSpecifier Identifier*
FunctionBody → *Expression*
Expression → *ConditionalExpression*
ConditionalExpression → *OrExpression* ('`?`' *Expression* '`:`' *Expression*)?
OrExpression → *AndExpression* ('`||`' *AndExpression*)*
AndExpression → *EqualityExpression* ('`&&`' *EqualityExpression*)*
EqualityExpression → *ComparativeExpression* ('`==`' | '`!=`' *ComparativeExpression*)*
ComparativeExpression → *AdditiveExpression* (('`<`' | '`<=`' | '`>`' | '`>=`') *AdditiveExpression*)*
AdditiveExpression → *MultiplicativeExpression* ('`+`' | '`-`' *MultiplicativeExpression*)*
MultiplicativeExpression → *UnaryExpression* ('`*`' | '`/`' *UnaryExpression*)*
UnaryExpression → *PostfixExpression*
| '`!`'*UnaryExpression*
| '`-`'*UnaryExpression*
PostfixExpression → *PimaryExpression* ('`.`' *Identifier*)?
PrimaryExpression → *Identifier* '`(`' (*ExpressionList*)? '`)`'
| *FloatConstant*
| '`(`' *Expression* ('`)`' | '`,`' *Expression* '`,`' *Expression* '`)`')
| `[` *ExpressionList* `]`
ExpressionList → *Expression* (`,` *Expression*)*

Chapter 4

Project Plan

Creating a language and compiler is a large and complicated project, with many components and tasks to complete along the way. The project tested our software engineering skills and ability to plan and work effectively as a team.

4.1 Development Process

4.1.1 Planning

Our group decided to have weekly meetings every Monday to discuss the status of the project. We mapped out timelines and set goals for ourselves during these meetings and it was an opportunity for team members to update each other on what they accomplished over the past week and to pose questions and concerns. From these meetings we were able to continuously update our timeline and stay on top of things.

4.1.2 Specification

We decided to break the project down into components and assign team members to tackle individual components first before combining. The main components in our system was the lexer, the parser, the tree walker, the scene libraries, and the OpenGL display. In the beginning all of us focused on getting the lexer and parser to work correctly. After the parser reached a stable state, we had a few team members start working on the scene libaries and OpenGL displays.

4.1.3 Development

The development process used depended on how many members were working on a particular component. For the parts that only one person was working on, developmenet was done and changes were commited to the repository. For parts that required more than one team member, the members would work individually and meet frequently to talk about new changes and concerns. During component integration, these ad hoc meetings were crucial in making sure common interfaces were honored and to eliminate problems early.

4.1.4 Testing

During individual component development, members were responsible for making sure their code compiled correctly and to test locally. When it came time to combine the components, one person

was responsible for integration and regression testing and provided instant feedback when something broke.

4.2 Programming Style Guide

The styles of programming are separated by language. There was one person mainly in charge of the development of the ANTLR, Java, C++ and FunkGL code. The design of our language allowed components to be very isolated from each other, requiring only a few lines of interface code to create the necessary interactions. The style for the IR classes, which make up the bulk of the code, was based on the example of the three-address assembly language provided on the PLT page. Scene.h and expr.h use their own style, one based on pointers and one on references, but the actual amount of interfacing necessary is quite small. The style for the antlr grammar was a based mainly on the PLT assembler example as well.

4.3 Project Timeline

2-5-07	Finish whitepaper
3-5-07	Finish LRM
3-7-07	Finish project architecture
4-16-07	Finish Lexer and Parser
4-23-07	Finish Tree Walker
4-25-07	Finish Scene Libraries and OpenGL display
5-1-07	All project features complete

4.4 Team Responsibilities

Everyone was responsible for a part of the project and needed to make sure all code was debugged and conformed to coding conventions. Christos was the team leader and was responsible for communicating with Professor Edwards and Joon Kim, our TA.

John Gallagher	Lexer, Parser, Tree Walker
Mack Lu	Lexer, Documentation
Christos Savvopoulos	Parser, Scene Libraries, OpenGL Display
Oren Sivan	Tree Walker, Testing

4.5 Software Development Environment

Lexer and Parser	ANTLR 3.0 (via Antlreclipse plugin)
Tree Walker	Java
Scene Libraries	C++
IDE	Eclipse, Visual Studios 2003
Source Control	Subversion
Internal Wiki	DokuWiki

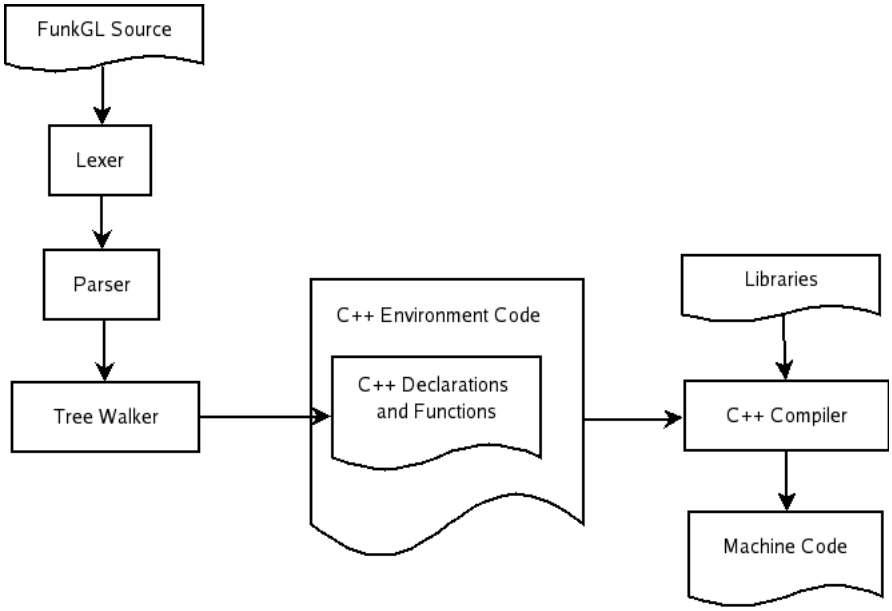
4.6 Project Log

1-29-07	Team assembled, selected Team Leader
2-7-07	Whitepaper complete, established primary features
2-17-07	Created project wiki
2-19-07	Created Subversion repository
2-24-07	Lexer, First Version
2-26-07	Finished CFG for Parser
3-5-07	LRM Complete
3-7-07	Project architecture complete
3-19-07	Began work on parser
3-26-07	Parser, First Version
3-27-07	C++ OpenGL prototype
3-14-07	Began testing of lexer and parser
4-2-07	Tree Walker, First Version
4-9-07	Began testing of tree walker
4-16-07	Error recovery, First Version
4-23-07	C++ scene libaries, First Version
4-27-07	Integrated testing begins
4-28-07	First source code to runtime prototype
4-30-07	Lexer and Parser, Final Version
5-3-07	Tree Walker, Final Version
5-4-07	Examples testing
5-7-07	Final Presentation

Chapter 5

Architectural Design

5.1 Overview



The architecture for FunkGL includes many components and they work together to transform a FunkGL source file into a machine specific executable file. After a FunkGL source code is created, it is read in by the FunkGL Java compiler. There, the code passes through the lexer, the parser, and the tree walker. The output of the tree walker is syntactially correct C++ code. This C++ code is then compiled by a machine specific C++ compiler, in the process gets linked to OpenGL libraries. The final product is executable machine code.

5.2 Lexer

The entire compiling process begins within the `main` method of FunkGLC.java which reads the command line argument and creates a lexer object. The lexer object is an instance of ANTLR's `L` lexer class. It takes as input a `DataInputStream` object, a stream that contains the FunkGL code. The lexer reads the input one character at a time and using the using rules specified in ANTLR syntax, returns tokens to the parser to process. The lexer lookahead was 2 to resolve ambiguity between assignment `=` and equality `==`.

The interface between the lexer and the parser is defined by the names of the tokens. The tokens are all uppercase and their names reflect their meaning. For example `ID` for the identifiers, `LPAREN` for left parentheses and so forth. The lexer and parser implemention teams worked together to establish a set of common names for tokens.

Mack and John worked on the Lexer.

5.3 Parser

FunkGLC.java also creates a parser object, an instance of ANTLR `P` parser class. The parser takes the lexer object as input and parses the stream of tokens that the lexer generated. The `program` rule is the rule in the FunkGL context-free grammar that characterizes the entire program so when `parser.program` was called, the token parsing begins.

The interfafce between the parser and the tree walker is define by the names of the nodes in the AST. The parser and tree walker teams worked together to estalbish a set of common names for the AST node names.

John and Christos worked on the Parser.

5.4 Tree Walker

The tree walker parses the AST created by the parser and through rules created using the ANTLR syntax, emits error free C++ code. The walker traverses the tree twice, the first time emitting object names, object attributes names, object attribute initial values, and function declarations. The symbol is table is also populated at this point. The second pass emits function bodies, update expressions, and mesh assignment of material identifiers.

Since real-time graphics require constant rendering, every FunkGL program will have a basic update-render loop. Thus, the underlying structure of every C++ program emitted by the tree walker will resemble:

```
//data structure emissions

init() {
  //initialization emissions
}

while(active)
{
  update() {
  //update emissions
  }
  render();
}
```

The init() function will be responsible for initially enabling OpenGL and loading all of the declared objects into the OpenGL engine while the update functions will update the attributes of the objects during iteration of the while loop, representing one frame. The rendering function finally draws the objects.

The interface between the tree walker and the emitted C++ code is defined by the infrastructure library functions and data structures. All rendering functions that the emitted C++ code has to

call is prefaced with `fgl` such as `fglScene`. The tree walker team and the infrastructure library team worked together to establish a common API for the function calls and a common structure for the data.

John and Oren worked on the Tree Walker.

5.5 Error Checking

Error checking happens in both the parser and the tree walker. The parser checks for syntax errors encountered while parsing the stream of tokens and throws an exception that is caught by `main`.

The tree walker checks for type errors, scope errors, and structural errors on the second pass through. It checks if the return type of function bodies matches the declared return type of the function, if the types of parameters in function calls match the declared types, and if the returned type of an attached function matches the type of the attribute it is attached to. On a mismatch, the tree walker displays the error and exists gracefully. The tree walker checks for scoping errors by looking up an identifier value in the symbol table created after the first pass. If the symbol doesn't exist, an error is outputted. Finally, the tree walker ensures structural integrity of the program by checking for things such as attributes are only declared in object declarations and keywords are used properly.

5.6 Infrastructure Libraries

The infrastructure libraries consists of many classes that mirror the IR represented in the AST. There is a wrapper float class that handles operation on floating point numbers, a vec class to handle vectors and a list class.

There also scene functions that wrap OpenGL functionality like `fglLoadMeshData` which loads a mesh representation from a file and `fglAddLight`, which as its name suggests, adds a light to the scene. These scene function are the ones that the emitted C++ code will call to use OpenGL to render objects.

With a functional language, new data structures are create when variables are manipulated. Garbage collection functions exist to clean up unused memory. Everytime a float, a vector, or a list is created, it is added to a garbage collection table. On every frame, all the float, vector, and list objects in the garbage collection table is freed for future use.

Christos worked on the Infrastructure Libraries

Chapter 6

Testing Plan

6.1 Methods

Testing FunkGL occurred within each component development. Phase I was during lexer and parser development. We created a graphical user input interface that allowed us to copy and paste chunks of code in FunkGL syntax and see the output. This ensured that each rule of the parser was working correctly before we combined rules.Phase II tested whether the tree walker would be able to generate the correct C++ code

Lexer testing: Mack Parser testing: John Tree Walker Testing: John and Oren Infrastucture Libraries testing: Christos Integration Testing: Oren

6.2 Test Cases

Below is a sample of the test cases we used for each phase and what they were used for.

6.2.1 Sample Lexer Test Cases

Testing identifiers

```
peanuts
l337h4xx0rs
peanut_sauce
```

Testing floating point numbers:

```
1337.0
5.0e-5
.25
```

Testing comments:

```
//This is a comment
/*This is a comment */
```

Sample Parser Test Cases

Testing vectors:

```
(5., 454., 5.)
(0.,0.,0.)
```

Testing lists:

```
[3]
[3, 4, 5, 6, 7]
[peanut, 3, (5, 5, 5), [3]]
[[[[[[[[3]]]]]]]]
```

Testing meshes

```
mesh sphere{
file="sphere.obj";
vector position=bouncepos(sphere.position,sphere.vel);
~position=(0.,30.,0.);
vector vel=bouncevel(sphere.position,sphere.vel);
~vel=(0.,0.,0.);
}

mesh sphereReflection{
file="sphere.obj";
vector position=reflectSpherePos();
~position=(0.,0.,0.);
vector vel=reflectSphereVel();
~vel=(0.,0.,0.);

}
mesh plane{
file="plane.obj";
vector position=plane.position;
~position=(0.,0.,0.);
}
```

Testing dot operator for vectors

```
float sumVec(vector a):a.x+a.y+a.z;
```

Testing dot operator for lists

```
float firstList(list a):a.first;
list restList(list a):a.rest;
```

Testing inversion

```
float testInv(float af):!af;
```

Sample Tree Walker Tests:

Testing casts:

```
float plus(float a, float b):a+b;
float plusl(list l, float b):plus(l.first,b);
float anyplusl(list l, float b):l.first + b;
vector plusV(vector a, vector b):a+b;
vector plusVl(list l, vector b):plusV(l.first,b);
vector anyplusVl(list l, vector b):l.first + b;
```

Testing for equality

```
float equalsTest(float a, float b):a==b;
float equalsTestl(list l, float b):equalsTest(l.first,b);
float anyequalsTestl(list l, float b):l.first == b;
float equalsTestV(vector a, vector b):a==b;
float equalsTestVl(list l, vector b):equalsTestV(l.first,b);
float anyequalsTestVl(list l, vector b):l.first == b;
```

6.3 Automation

Because the output for most components of the projects varied greatly from week to week, it was difficult to create an automatic test suite that could be accurately checked against. Furthermore, the final output of the project is graphical which also made it difficult to check with automatic testing tools.

6.4 Source and Target Code

mousetest.fgl

```
//ball follows x, y coordinates of mouse
//On mouse click, ball grows proportionally large to
//the amount of time the mouse was held down.

camera cam{
vector position=cam.position;
~position=(0.,0.,120.);
vector center=cam.center;
~center=(0.,0.,0.);
vector up=cam.up;
~up=(0.,-1.,0.);
}

//default lighting
light l{}

mesh sphere{
file="sphere.obj";
vector position=bouncepos();
vector scale=growme(sphere.scale);
~scale=(1.,1.,1.);
}
mesh plane{
file="torus.obj";
vector position=plane.position;

}

//position depends on x, y position of mouse
vector bouncepos(): (250.-mousex,mousey-250.,0.);

//size increases with how long the mouse is held down
vector growme(vector prev):
   mouselpressed?prev*1.025:mouserpressed?prev*.995:prev;
```

mouseTest.fgl C++ target output code

```
#include "FunkGL.h"
#include <iostream>

namespace{
floatt sin(floatt x);
floatt cos(floatt x);
floatt exp(floatt x);
vec mousepos();
vec growme(vec prev);


struct state {
struct t_obj_1{
}camera_c;
struct t_obj_2{
}light_l;
struct t_obj_3{
unsigned int __dataid;
unsigned int __meshid;
vec position;
vec scale;
}mesh_sphere;
};
state prev, cur;

void init(){
dt=0;

fglSetCamera(0, 0, 0);


fglAddLight(0, 0, 0, 0, 0, 0);



cur.mesh_sphere.position=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));

cur.mesh_sphere.scale=vec(floatt(1.0f),floatt(1.0f),floatt(1.0f));
cur.mesh_sphere.__dataid=fglLoadMeshData("sphere.obj");
cur.mesh_sphere.__meshid=fglAddMesh(cur.mesh_sphere.__dataid, 0,
      &cur.mesh_sphere.position, 0, &cur.mesh_sphere.scale);
}

void update(){
prev=cur;
collect();
```

```
cur.mesh_sphere.position=mousepos();

cur.mesh_sphere.scale=growme(prev.mesh_sphere.scale);
}

floatt sin(floatt x){

return floatt(sinf(x));
}
floatt cos(floatt x){

return floatt(cosf(x));
}
floatt exp(floatt x){

return floatt(expf(x));
}
vec mousepos(){

return vec(floatt(mouseX) - floatt(250.0f),floatt(250.0f)
      - floatt(mouseY),floatt(0.0f));
}
vec growme(vec prev){

return floatt(mouseLPressed) && ((list().rest().hasfirst()
        ?floatt(0.0):floatt(1.0)))?prev * floatt(1.0025f):floatt(mouseRPressed)
          ?prev * floatt(0.9995f):prev;
}
}

int main(int argc, char **argv) {
try{
init();
fglScene(update,&argc,argv);
}catch(char const *str){
std::cout<<str<<std::endl;
}
}
```

source of SmallSolarSystem.fgl

```
//The solar system declars four sphere objects, scaled differently
//to represent a static planet and three moons orbiting. All the planets
//change their shading gradually, from light to dark, then back to dark
//again. The y position of the blue planet follows the mouse's y position.
//On left mouse click, the planets are instantly scale back, giving the
//illusion that we're undergoing space travel. On right click, the planets
//go crazy.

camera cam{
// vector position=cam.position;
vector position=campos();
~position=(0.,0.,120.);
vector center=cam.center;
~center=(0.,0.,0.);
vector up=cam.up;
~up=(0.,-1.,0.);
}
light l{
}
material sunmat{
file="null.null";
vector diffuse=mouserpressed?crazyColor():(1.,0.,0.);
vector ambient=mouserpressed?crazyColor():sunChange();
vector specular=mouserpressed?crazyColor():(1.,0.,0.);
}
material planetmat1{
file="mat.bmp";
vector diffuse=mouserpressed?crazyColor():(0.,0.,1.);
vector ambient=mouserpressed?crazyColor():(0.,0.,sunmat.ambient.x);
vector specular=mouserpressed?crazyColor():(0.,0.,1.);
}
material planetmat2{
file="null.null";
vector diffuse=mouserpressed?crazyColor():(0.,1.,0.);
vector ambient=mouserpressed?crazyColor():(0.,sunmat.ambient.x,0.);
vector specular=mouserpressed?crazyColor():(0.,1.,0.);
}
material planetmat3{
file="null.null";
vector diffuse=mouserpressed?crazyColor():(0.,0.6,0.5);
vector ambient=mouserpressed?crazyColor():
      (0.,0.6*sunmat.ambient.x,0.5*sunmat.ambient.x);
vector specular=mouserpressed?crazyColor():(0.,0.6,0.5);
}
```

```
mesh sun{
file="sphere.obj";
material=sunmat;
vector position=(0.,15.,0.);
vector scale=sunScale();

}
mesh planet{
file="sphere.obj";
material=planetmat1;
vector position=planetPos();
~position=(15.,15.,0.);
vector scale=planetScale1();
~scale=(0.5,0.5,0.5);
float time=timeUpdate();
~time=0.;
}


mesh planet2{
file="sphere.obj";
material=planetmat2;
vector position=planet2Pos();
~position=(5.,15.,5.);
vector scale=planetScale2();
~scale=(0.3,0.3,0.3);
float time=timeUpdate2();
~time=2.;
}

mesh planet3{
file="sphere.obj";
material=planetmat3;
vector position=planet3Pos();
~position=(10.,15.,10.);
vector scale=planetScale3();
~scale=(0.1,0.1,0.1);
float time=timeUpdate3();
~time=4.;
}

vector crazyColor():( 1.+0.5*sin(2.*planet.time) ,1.+0.5*cos(3.*planet.time/2.)
  ,1.+0.5*sin(planet.time/2.));
float timeUpdate():planet.time+dt;
float timeUpdate2():planet2.time+dt;
float timeUpdate3():planet3.time+dt;
vector sunScale():mouserpressed?(2.,2.,2.)*cos(0.5*planet.time):(1.,1.,1.);
vector planetScale1():mouserpressed?sunScale()*1.*sin(planet.time)
```

```
  :sunScale()*0.5;
vector planetScale2():mouserpressed?sunScale()*0.6*cos(1.5*planet.time)
  :sunScale()*0.3;
vector planetScale3():mouserpressed?sunScale()*0.4*sin(2.*planet.time)
  :sunScale()*0.2;
//2-d orbit around sun
vector planetPos(): (35.*cos(planet.time),mousey-100.,35.*sin(planet.time));
vector planet2Pos(): (55.*cos(1.5*planet2.time),15.,55.*sin(1.5*planet2.time));
vector planet3Pos(): (70.*cos(2.*planet3.time),15.,70.*sin(2.*planet3.time));

vector campos():mouselpressed?(0.,0.,400.):(0.,0.,120.);

vector sunChange():((1.+0.5*sin(planet.time/2.)),0.,0.);
```

SmallSolarSystem.fgl

```
#include "FunkGL.h"
#include <iostream>

namespace{
floatt sin(floatt x);
floatt cos(floatt x);
vec crazyColor();
floatt timeUpdate();
floatt timeUpdate2();
floatt timeUpdate3();
vec sunScale();
vec planetScale1();
vec planetScale2();
vec planetScale3();
vec planetPos();
vec planet2Pos();
vec planet3Pos();
vec campos();
vec sunChange();


struct state {
struct t_obj_1{
vec position;
vec center;
vec up;
}camera_cam;
struct t_obj_2{
}light_l;
struct t_obj_3{
unsigned int __matid;
vec diffuse;
vec ambient;
vec specular;
}material_sunmat;
struct t_obj_4{
unsigned int __matid;
vec diffuse;
vec ambient;
vec specular;
}material_planetmat1;
struct t_obj_5{
unsigned int __matid;
vec diffuse;
vec ambient;
vec specular;
```

```
}material_planetmat2;
struct t_obj_6{
unsigned int __matid;
vec diffuse;
vec ambient;
vec specular;
}material_planetmat3;
struct t_obj_7{
unsigned int __dataid;
unsigned int __meshid;
vec position;
vec scale;
}mesh_sun;
struct t_obj_8{
unsigned int __dataid;
unsigned int __meshid;
vec position;
vec scale;
floatt time;
}mesh_planet;
struct t_obj_9{
unsigned int __dataid;
unsigned int __meshid;
vec position;
vec scale;
floatt time;
}mesh_planet2;
struct t_obj_10{
unsigned int __dataid;
unsigned int __meshid;
vec position;
vec scale;
floatt time;
}mesh_planet3;
};
state prev, cur;

void init(){
dt=0;


cur.camera_cam.position=vec(floatt(0.0f),floatt(0.0f),floatt(120.0f));

cur.camera_cam.center=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));

cur.camera_cam.up=vec(floatt(0.0f),(floatt(0.0f)
      - floatt(1.0f)),floatt(0.0f));
fglSetCamera(&cur.camera_cam.position, &cur.camera_cam.center,
```

```
      &cur.camera_cam.up);


fglAddLight(0, 0, 0, 0, 0, 0);



cur.material_sunmat.diffuse=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));

cur.material_sunmat.ambient=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));

cur.material_sunmat.specular=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));
cur.material_sunmat.__matid=fglLoadMaterial("",
       &cur.material_sunmat.ambient, &cur.material_sunmat.diffuse,
       &cur.material_sunmat.specular, 0);



cur.material_planetmat1.diffuse=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));

cur.material_planetmat1.ambient=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));

cur.material_planetmat1.specular=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));
cur.material_planetmat1.__matid=fglLoadMaterial("",
      &cur.material_planetmat1.ambient, &cur.material_planetmat1.diffuse,
      &cur.material_planetmat1.specular, 0);



cur.material_planetmat2.diffuse=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));

cur.material_planetmat2.ambient=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));

cur.material_planetmat2.specular=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));
cur.material_planetmat2.__matid=fglLoadMaterial("",
      &cur.material_planetmat2.ambient, &cur.material_planetmat2.diffuse,
      &cur.material_planetmat2.specular, 0);



cur.material_planetmat3.diffuse=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));

cur.material_planetmat3.ambient=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));

cur.material_planetmat3.specular=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));
cur.material_planetmat3.__matid=fglLoadMaterial("",
      &cur.material_planetmat3.ambient, &cur.material_planetmat3.diffuse,
      &cur.material_planetmat3.specular, 0);
```

```
cur.mesh_sun.position=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));

cur.mesh_sun.scale=vec(floatt(0.0f),floatt(0.0f),floatt(0.0f));
cur.mesh_sun.__dataid=fglLoadMeshData("sphere.obj");
cur.mesh_sun.__meshid=fglAddMesh(cur.mesh_sun.__dataid,
      cur.material_sunmat.__matid, &cur.mesh_sun.position, 0,
      &cur.mesh_sun.scale);



cur.mesh_planet.position=vec(floatt(15.0f),floatt(15.0f),floatt(0.0f));

cur.mesh_planet.scale=vec(floatt(0.5f),floatt(0.5f),floatt(0.5f));

cur.mesh_planet.time=floatt(0.0f);
cur.mesh_planet.__dataid=fglLoadMeshData("sphere.obj");
cur.mesh_planet.__meshid=fglAddMesh(cur.mesh_planet.__dataid,
      cur.material_planetmat1.__matid, &cur.mesh_planet.position, 0,
      &cur.mesh_planet.scale);



cur.mesh_planet2.position=vec(floatt(5.0f),floatt(15.0f),floatt(5.0f));

cur.mesh_planet2.scale=vec(floatt(0.3f),floatt(0.3f),floatt(0.3f));

cur.mesh_planet2.time=floatt(2.0f);
cur.mesh_planet2.__dataid=fglLoadMeshData("sphere.obj");
cur.mesh_planet2.__meshid=fglAddMesh(cur.mesh_planet2.__dataid,
      cur.material_planetmat2.__matid, &cur.mesh_planet2.position, 0,
      &cur.mesh_planet2.scale);



cur.mesh_planet3.position=vec(floatt(10.0f),floatt(15.0f),floatt(10.0f));

cur.mesh_planet3.scale=vec(floatt(0.1f),floatt(0.1f),floatt(0.1f));

cur.mesh_planet3.time=floatt(4.0f);
cur.mesh_planet3.__dataid=fglLoadMeshData("sphere.obj");
cur.mesh_planet3.__meshid=fglAddMesh(cur.mesh_planet3.__dataid,
      cur.material_planetmat3.__matid, &cur.mesh_planet3.position, 0,
      &cur.mesh_planet3.scale);
}
```

```
void update(){
prev=cur;
collect();


cur.camera_cam.position=campos();

cur.camera_cam.center=prev.camera_cam.center;

cur.camera_cam.up=prev.camera_cam.up;




cur.material_sunmat.diffuse=floatt(mouseRPressed)?crazyColor():
      vec(floatt(1.0f),floatt(0.0f),floatt(0.0f));

cur.material_sunmat.ambient=floatt(mouseRPressed)?crazyColor():sunChange();

cur.material_sunmat.specular=floatt(mouseRPressed)?crazyColor():
      vec(floatt(1.0f),floatt(0.0f),floatt(0.0f));



cur.material_planetmat1.diffuse=floatt(mouseRPressed)?crazyColor():
      vec(floatt(0.0f),floatt(0.0f),floatt(1.0f));

cur.material_planetmat1.ambient=floatt(mouseRPressed)?crazyColor():
      vec(floatt(0.0f),floatt(0.0f),prev.material_sunmat.ambient.x);

cur.material_planetmat1.specular=floatt(mouseRPressed)?crazyColor():
      vec(floatt(0.0f),floatt(0.0f),floatt(1.0f));



cur.material_planetmat2.diffuse=floatt(mouseRPressed)?crazyColor():
      vec(floatt(0.0f),floatt(1.0f),floatt(0.0f));

cur.material_planetmat2.ambient=floatt(mouseRPressed)?crazyColor():
      vec(floatt(0.0f),prev.material_sunmat.ambient.x,floatt(0.0f));

cur.material_planetmat2.specular=floatt(mouseRPressed)?crazyColor():
      vec(floatt(0.0f),floatt(1.0f),floatt(0.0f));



cur.material_planetmat3.diffuse=floatt(mouseRPressed)?crazyColor():
```

```
      vec(floatt(0.0f),floatt(0.6f),floatt(0.5f));

cur.material_planetmat3.ambient=floatt(mouseRPressed)?crazyColor():
      vec(floatt(0.0f),floatt(0.6f) * prev.material_sunmat.ambient.x,
       floatt(0.5f) * prev.material_sunmat.ambient.x);

cur.material_planetmat3.specular=floatt(mouseRPressed)?crazyColor():
      vec(floatt(0.0f),floatt(0.6f),floatt(0.5f));



cur.mesh_sun.position=vec(floatt(0.0f),floatt(15.0f),floatt(0.0f));

cur.mesh_sun.scale=sunScale();



cur.mesh_planet.position=planetPos();

cur.mesh_planet.scale=planetScale1();

cur.mesh_planet.time=timeUpdate();



cur.mesh_planet2.position=planet2Pos();

cur.mesh_planet2.scale=planetScale2();

cur.mesh_planet2.time=timeUpdate2();



cur.mesh_planet3.position=planet3Pos();

cur.mesh_planet3.scale=planetScale3();

cur.mesh_planet3.time=timeUpdate3();
}

floatt sin(floatt x){

return floatt(sinf(x));
}
floatt cos(floatt x){

return floatt(cosf(x));
}
```

```
vec crazyColor(){

return vec(floatt(1.0f) + floatt(0.5f) * sin(floatt(2.0f) *
      prev.mesh_planet.time),floatt(1.0f) + floatt(0.5f) *
      cos(floatt(3.0f) * prev.mesh_planet.time / floatt(2.0f)),
      floatt(1.0f) + floatt(0.5f) * sin(prev.mesh_planet.time /
     floatt(2.0f)));
}
floatt timeUpdate(){

return prev.mesh_planet.time + dt;
}
floatt timeUpdate2(){

return prev.mesh_planet2.time + dt;
}
floatt timeUpdate3(){

return prev.mesh_planet3.time + dt;
}
vec sunScale(){

return floatt(mouseRPressed)?vec(floatt(2.0f),floatt(2.0f),floatt(2.0f))
      * cos(floatt(0.5f) * prev.mesh_planet.time):vec(floatt(1.0f),
      floatt(1.0f),floatt(1.0f));
}
vec planetScale1(){

return floatt(mouseRPressed)?sunScale() * floatt(1.0f) * sin(
      prev.mesh_planet.time):sunScale() * floatt(0.5f);
}
vec planetScale2(){

return floatt(mouseRPressed)?sunScale() * floatt(0.6f) * cos(
      floatt(1.5f) * prev.mesh_planet.time):sunScale() * floatt(0.3f);
}
vec planetScale3(){

return floatt(mouseRPressed)?sunScale() * floatt(0.4f) * sin(
       floatt(2.0f) * prev.mesh_planet.time):sunScale() * floatt(0.2f);
}
vec planetPos(){

return vec(floatt(35.0f) * cos(prev.mesh_planet.time),floatt(mouseY)
      - floatt(100.0f),floatt(35.0f) * sin(prev.mesh_planet.time));
}
vec planet2Pos(){
```

```
return vec(floatt(55.0f) * cos(floatt(1.5f) * prev.mesh_planet2.time),
      floatt(15.0f),floatt(55.0f) * sin(floatt(1.5f) * prev.mesh_planet2.time));
}
vec planet3Pos(){

return vec(floatt(70.0f) * cos(floatt(2.0f) * prev.mesh_planet3.time),
      floatt(15.0f),floatt(70.0f) * sin(floatt(2.0f) * prev.mesh_planet3.time));
}
vec campos(){

return floatt(mouseLPressed)?vec(floatt(0.0f),floatt(0.0f),
      floatt(400.0f)):vec(floatt(0.0f),floatt(0.0f),floatt(120.0f));
}
vec sunChange(){

return vec((floatt(1.0f) + floatt(0.5f) * sin(prev.mesh_planet.time /
      floatt(2.0f))),floatt(0.0f),floatt(0.0f));
}
}

int main(int argc, char **argv) {
try{
init();
fglScene(update,&argc,argv);
}catch(char const *str){
std::cout<<str<<std::endl;
}
}
```

Chapter 7

Lessons Learned

7.1 John Gallagher

Working on a team is pretty difficult. We tried organizing the team structure in different ways. Our initial democratic approach didn't work well because descisions took longer than they should have, and there was a lot of backtracking. The dictatorial approach did not work because the dictator became overbearing and it was hard to get production flowing from more than one person. Despite team issues, I really enjoyed working on this project. I'm glad that we ended up working on a language designed for instant gratification, because it was exciting to see the results of our hard work. We were able to collaborate well for significant portions of the project, and Eclipse/SVN really helped out there. Again, I think this was a great experience and i'm glad we have a nifty program to show for it.

7.2 Mack Lu

The most important thing I learned is how valuable weekly meetings were. When we had regular meetings, the status was kept up to date, everyone knew what everyone else was working on, and the project hummed along at a nice pace. As soon as we stopped having weekly meetings, communication declined sharply between team members, there was confusion on who was working on which components, and progress slowed to a crawl. At one point, two members worked on the exact same classes at the exact same time and didn't even know! Regular group meetings makes sure everyone is on the same page and help keep people moving in a positive x direction.

A few tips for future teams:
1. Make sure everyone uses source control
2. Start early
3. Keep making progress
4. Find a caffeinated drink of choice
5. Prepare to be frustrated

7.3 Christos Savvopoulos

Sorry for sounding dramatic : −)

As the "dictator" of this team, my biggest lesson and surprise the response of people:work on the project basically started after I gave my teammates a goal and deadline.

Another lesson is that progress should have been monitored more closely, rather than assume that things were getting done in the way they should be and on time.

On the bright side, I also realized how important it was that we predefined the graphics library interface so well. It saved a lot of headaches and trouble with getting the compiler to produce code. In fact, when we got our first program to compile, it worked on the first try, exactly because of that reason. Finally, it is important to note how important it is that ALL of the members are using the same tools and environments. Even using different compilers to run the product created problems: while the project run from MSVC, it needed a lot of debugging on gcc.

7.4 Oren Sivan

This project really demonstrated the overhead costs associated with working in a team. In a team, the two main tasks, decision-making and coding, have to be done very differently than if you were to work alone so as to minimize these overhead costs.

For decision-making, it has become apparent that pure democracy does not work. There needs to be a tie-breaking dictator. Otherwise, too much time is wasted on making decisions.

As for coding, there needs to be a clear division of labor at all times, so as to prevent time wasted in idleness. Furthermore, there always needs to be a clear definition of input/output interfaces between every person's coding, so as to prevent time wasted trying to make everyone's code compatible. Finally, use SVN or die.

To summarize my advice:
1) Dictator
2) Division of labor
3) Well-defined input/output
4) SVN

Appendix A

Complex FunkGL Examples

mousetest.fgl

```
//ball follows x, y coordinates of mouse
//On mouse click, ball grows proportionally large to
//the amount of time the mouse was held down.

camera cam{
vector position=cam.position;
~position=(0.,0.,120.);
vector center=cam.center;
~center=(0.,0.,0.);
vector up=cam.up;
~up=(0.,-1.,0.);
}
//default lighting
light l{}

mesh sphere{
file="sphere.obj";
vector position=bouncepos();
vector scale=growme(sphere.scale);
~scale=(1.,1.,1.);
}
mesh plane{
file="torus.obj";
vector position=plane.position;

}
//position depends on x, y position of mouse
vector bouncepos(): (250.-mousex,mousey-250.,0.);

//size increases with how long the mouse is held down
vector growme(vector prev):
   mouselpressed?prev*1.025:mouserpressed?prev*.995:prev;
```

circus.fgl

```
//creates two spheres that rotates in a circular orbit through two static toruses
camera cam{
vector position=cam.position;
~position=(0.,0.,120.);
vector center=cam.center;
~center=(0.,0.,0.);
vector up=cam.up;
~up=(0.,-1.,0.);
}
light l{
}
mesh torus{
file="torus.obj";
vector position=(50.,15.,0.);

}

mesh torus2{
file="torus.obj";
vector position=(-50.,15.,0.);
}

mesh planet{
file="sphere.obj";
vector position=planetPos();
~position=(15.,15.,0.);
vector scale=(0.2,0.2,0.2);
float time=timeUpdate();
~time=0.;
}
mesh planet2{
file="sphere.obj";
vector position=planetPos2();
~position=(15.,15.,0.);
vector scale=(0.2,0.2,0.2);
float time=timeUpdate();
~time=0.;
}

float timeUpdate():planet.time+dt;
vector planetPos():(50.*cos(planet.time),50.*sin(planet.time),15.);
vector planetPos2():(-50.*cos(planet.time),-50.*sin(planet.time),15.);
```

gcd.fgl

```
//produces 4 spheres from left-to-right with scales 6, 4, gcd(6,4) , 2
//the 4th sphere is to prove correctness of gcd!

camera cam{
vector position=cam.position;
~position=(0.,0.,-600.);
vector center=cam.center;
~center=(0.,0.,0.);
vector up=cam.up;
~up=(0.,1.,0.);
}
light l{
}

mesh sphere6{
file="sphere.obj";
vector position=(300.,0.,0.);
vector vel=(0.,0.,0.);
vector scale=(6.,6.,6.);
}

mesh sphere4{
file="sphere.obj";
vector position=(0.,0.,0.);
vector vel=(0.,0.,0.);
vector scale=(4.,4.,4.);
}

mesh spheregcd{
file="sphere.obj";
vector position=(-200.,0.,0.);
vector vel=(0.,0.,0.);
vector scale=spheregcdscale();
}

mesh sphereactual{
file="sphere.obj";
vector position=(-300.,0.,0.);
vector vel=(0.,0.,0.);
vector scale=(2.,2.,2.);
}

vector spheregcdscale():(gcd(sphere6.scale.x,sphere4.scale.x),
  gcd(sphere6.scale.y,sphere4.scale.y),gcd(sphere6.scale.z,sphere4.scale.z));

//gcd   note: a<1 instead of a==0 as floats!
```

```
float gcd(float m, float n):(m<1.)?n:((n<1.)?m:gcd(n, remainder(m,n)));

//a%b.....assumes int-like floats
float remainder(float a, float b):(a<b)?a:remainder((a-b),b);
```

ListMouseTest.fgl

```
//Mouse test implementation using lists
camera c{}
light l{}
mesh sphere{
file="sphere.obj";
vector position=mousepos();
vector scale=timesme(sphere.scale,sphere.set); //growme(sphere.scale);
~scale=(1.,1.,1.);
list set=[1.001,2.];

}
vector mousepos(): (mousex-250.,250.-mousey,0.);
/*vector growme(vector prev):
mouselpressed
?prev*1.0025
:mouserpressed
?prev*.9995
:prev;
*/
float castval(float t):t;
vector timesme(vector s, list r): !(r.mpty)?timesme(s*castval(r.first),r.rest):s;
list resttest():[].rest;

source of ListScaleChange.fgl
camera cam{
        vector position=cam.position;
        ~position=(0.,0.,-50.);
        vector center=cam.center;
        ~center=(0.,0.,0.);
        vector up=cam.up;
        ~up=(0.,1.,0.);
}
light l{
}

mesh sphere{
        file="sphere.obj";
        vector position=(0.,15.,0.);
        vector vel=(0.,0.,0.);
        list history =historyUpdate(sphere.history);
        ~history=[];
        vector scale=crazyScale(sphere.history);
        ~scale=(0.5,0.5,0.5);
        float time=timeUpdate();
        ~time=0.;
}
```

```
float timeUpdate():sphere.time+dt;

//scales with time
//vector crazyScale():(sphere.time/10.,sphere.time/10.,sphere.time/10.);
vector crazyScale(list l):(listSum(l),listSum(l),listSum(l));
list historyUpdate(list l):cons(0.05,l);
float listSum(list l):l.mpty?0.:l.first+listSum(l.rest);
```

source of reflected.fgl

```
//two toruses bouncing off each other as the camera pans out
camera cam{
vector position=cam.position;
~position=(0.,0.,-50.);
vector center=cam.center;
~center=(0.,0.,0.);
vector up=cam.up;
~up=(0.,1.,0.);
}

light l{
}

mesh torus{
file="torus.obj";
vector position=bouncepos(torus.position,torus.vel);
~position=(0.,30.,0.);
vector vel=bouncevel(torus.position,torus.vel);
~vel=(0.,0.,0.);
}

mesh torusReflection{
file="torus.obj";
vector position=reflectTorusPos();
~position=(0.,0.,0.);
vector vel=reflectTorusVel();
~vel=(0.,0.,0.);

}
mesh sphere{
file="sphere.obj";
vector position=sphere.position;
~position=(0.,0.,0.);
}
vector bouncevel(vector pos, vector vel):
(vel.y<0. & pos.y <0.)
? (0.,0.,0.)-vel+(0.,-3.,0.)*dt
: vel+(0.,-3.,0.)*dt;

vector bouncepos(vector pos, vector vel): pos+vel*dt;

//the dot operator for vectors,lists, & referencing other objects' attributes
vector reflectTorusVel():(0.,0.,0.)-torus.vel;
vector reflectTorusPos():(0.,0.,0.)-torus.position;
```

source of SmallSolarSystem.fgl

```
//The solar system declars four sphere objects, scaled differently
//to represent a static planet and three moons orbiting. All the planets
//change their shading gradually, from light to dark, then back to dark
//again. The y position of the blue planet follows the mouse's y position.
//On left mouse click, the planets are instantly scale back, giving the
//illusion that we're undergoing space travel. On right click, the planets
//go crazy.

camera cam{
// vector position=cam.position;
vector position=campos();
~position=(0.,0.,120.);
vector center=cam.center;
~center=(0.,0.,0.);
vector up=cam.up;
~up=(0.,-1.,0.);
}
light l{
}
material sunmat{
file="null.null";
vector diffuse=mouserpressed?crazyColor():(1.,0.,0.);
vector ambient=mouserpressed?crazyColor():sunChange();
vector specular=mouserpressed?crazyColor():(1.,0.,0.);
}
material planetmat1{
file="mat.bmp";
vector diffuse=mouserpressed?crazyColor():(0.,0.,1.);
vector ambient=mouserpressed?crazyColor():(0.,0.,sunmat.ambient.x);
vector specular=mouserpressed?crazyColor():(0.,0.,1.);
}
material planetmat2{
file="null.null";
vector diffuse=mouserpressed?crazyColor():(0.,1.,0.);
vector ambient=mouserpressed?crazyColor():(0.,sunmat.ambient.x,0.);
vector specular=mouserpressed?crazyColor():(0.,1.,0.);
}
material planetmat3{
file="null.null";
vector diffuse=mouserpressed?crazyColor():(0.,0.6,0.5);
vector ambient=mouserpressed?crazyColor():
      (0.,0.6*sunmat.ambient.x,0.5*sunmat.ambient.x);
vector specular=mouserpressed?crazyColor():(0.,0.6,0.5);
}
```

```
mesh sun{
file="sphere.obj";
material=sunmat;
vector position=(0.,15.,0.);
vector scale=sunScale();

}
mesh planet{
file="sphere.obj";
material=planetmat1;
vector position=planetPos();
~position=(15.,15.,0.);
vector scale=planetScale1();
~scale=(0.5,0.5,0.5);
float time=timeUpdate();
~time=0.;
}


mesh planet2{
file="sphere.obj";
material=planetmat2;
vector position=planet2Pos();
~position=(5.,15.,5.);
vector scale=planetScale2();
~scale=(0.3,0.3,0.3);
float time=timeUpdate2();
~time=2.;
}

mesh planet3{
file="sphere.obj";
material=planetmat3;
vector position=planet3Pos();
~position=(10.,15.,10.);
vector scale=planetScale3();
~scale=(0.1,0.1,0.1);
float time=timeUpdate3();
~time=4.;
}

vector crazyColor():( 1.+0.5*sin(2.*planet.time) ,1.+0.5*cos(3.*planet.time/2.)
  ,1.+0.5*sin(planet.time/2.));
float timeUpdate():planet.time+dt;
float timeUpdate2():planet2.time+dt;
float timeUpdate3():planet3.time+dt;
vector sunScale():mouserpressed?(2.,2.,2.)*cos(0.5*planet.time):(1.,1.,1.);
vector planetScale1():mouserpressed?sunScale()*1.*sin(planet.time)
```

```
  :sunScale()*0.5;
vector planetScale2():mouserpressed?sunScale()*0.6*cos(1.5*planet.time)
  :sunScale()*0.3;
vector planetScale3():mouserpressed?sunScale()*0.4*sin(2.*planet.time)
  :sunScale()*0.2;
//2-d orbit around sun
vector planetPos(): (35.*cos(planet.time),mousey-100.,35.*sin(planet.time));
vector planet2Pos(): (55.*cos(1.5*planet2.time),15.,55.*sin(1.5*planet2.time));
vector planet3Pos(): (70.*cos(2.*planet3.time),15.,70.*sin(2.*planet3.time));

vector campos():mouselpressed?(0.,0.,400.):(0.,0.,120.);

vector sunChange():((1.+0.5*sin(planet.time/2.)),0.,0.);
```

Appendix B

IR Classes

```
source of Add.java
package edu.columbia.cs.vicecity.ir;

public class Add extends Op {
Expr lchild, rchild;

public Add(Expr lchild, Expr rchild) {
super(null, resultType(lchild, rchild));
this.lchild = lchild;
this.rchild = rchild;
}

public static Type resultType(Expr a, Expr b) {
Type rettype = null;
Type atype = a.type, btype = b.type;
if (atype == Type.Any && btype != Type.Any) {
warn("the actual type of " + a + " in addition of " + a + " and " + b
+ "\n may vary at runtime and operation may fail");
atype = btype;
a.setCast(btype);
}
if (btype == Type.Any && atype != Type.Any) {
warn("the actual type of " + b + " in addition of " + a + " and " + b
+ "\n may vary at runtime and operation may fail");
btype = atype;
b.setCast(atype);
}
if (atype == Type.Any && btype == Type.Any) {
warn("the actual type of both variables in addition of " + a + " and " + b
+ "\n may vary at runtime and operation may fail");
}
if (atype == btype)
rettype = a.type;
if (!(rettype == Type.Vec || rettype == Type.Float || rettype == Type.Any))
```

```
error("incompatible types: cannot add " + a.type.toString() + " to a "
+ b.type.toString());
return rettype;
}

public String toString() {
return lchild.castexpr() + " + " + rchild.castexpr();
}

public String decl() {
return lchild.decl() + rchild.decl();
}
}
```

```
source of ArgExprList.java
package edu.columbia.cs.vicecity.ir;

public class ArgExprList extends StuffList<Expr> {
private boolean first;

public ArgExprList() {
super(Type.None);
first = true;
}

public void add(Expr expr) {
super.add(expr);
if (first) {
sexpr += expr.toString();
first = false;
}
else {
sexpr += "," + expr.toString();
}
}

public String toString() {
boolean firste = true;
String retval = "(";
for (Expr expr : getList()) {
if (firste) {
retval += expr.castexpr();
firste = false;
}
else
retval += ", " + expr.castexpr();
}
```

```
retval += ")";
return retval;
}

}


source of Attr.java
package edu.columbia.cs.vicecity.ir;
//should extend Id?
public class Attr extends Expr {
private static int unique=0;
public static String temp_id_prefix="t_list_";
protected int tempnum;
private DeclObject parent;
private String name;
private Expr update_function;
private Expr initial_val;
public Attr(Type type, DeclObject parent, String name){
this(type,parent,name,null,null);
if(type==Type.List) tempnum=++unique;
}
public Attr(Type type, DeclObject parent, String name, Expr update_function){
this(type,parent,name,update_function,null);
}
public Attr(Type type, DeclObject parent, String name, Expr update_function,
Expr initial_val){
super(null,type);
if(update_function==null){
//needs to be attached later
//error("attribute "+parent.getName()+"."+name+" must have an associated
update function");
}
else if(update_function.type!=type){
error("type mismatch: attribute "+parent.getName()+"."+name+" expects type
"+type
+" but update function "+update_function+ " returns type "+
update_function.type);
}
this.parent=parent;
this.name=name;
this.update_function=update_function;
if(!(initial_val==null)){
if(initial_val.type!=type){
error("type mismatch: attribute "+parent.getName()+"."+name+"
expects type "+type
+" but initial value "+initial_val+ " returns type "+
initial_val.type);
```

```
}
this.initial_val=initial_val;
}
else{
this.initial_val=type.defaultVal();
}
}
public void addUpdateFunction(Expr update_function){
if(update_function==null){
error("attribute "+parent.getName()+"."+name+" must have an associated
update function");
}
else if(update_function.type!=type){
error("type mismatch: attribute "+parent.getName()+"."+name+" expects
type "+type
+" but update function "+update_function+ " returns type "+
update_function.type);
}
this.update_function=update_function;
}
public String decl(){
if(!(type==Type.List))
return type+" "+name+";";
else
return "permlist "+name+";";
}
public String ctor(){
if(!(type==Type.List))
return initial_val.decl()+"\n\tcur."+qname()+"="+initial_val+";";
else{
String retval=initial_val.decl()+"\n\tlist "+temp_id_prefix+tempnum+
"="+initial_val+";";
retval+="\n\tcur."+qname()+"="+temp_id_prefix+tempnum+";";
return retval;
}
}
public String update(){
if(!(type==Type.List))
return update_function.decl()+"\n\tcur."+qname()+"="+update_function+";";
else{
String retval=update_function.decl()+"\n\tlist "+temp_id_prefix+tempnum+
"="+update_function+";";
retval+="\n\tcur."+qname()+"="+temp_id_prefix+tempnum+";";
return retval;
}

}
public String qname(){return parent+"."+name;}
```

```
public String scopename(){return parent.getName()+"."+name;}
public String toString(){return name;}
}


source of AttrList.java
package edu.columbia.cs.vicecity.ir;

public class AttrList extends StuffList<Attr> {
private String ctorexpr;
private String updateexpr;

public AttrList() {
super(Type.None);
declexpr = "";
ctorexpr = "";
updateexpr = "";
}

public void add(Attr attr) {
super.add(attr);
declexpr += "\n\t" + attr.decl();
ctorexpr += "\n\t" + attr.ctor();
updateexpr += "\n\t" + attr.update();
}

public String ctor() {
return ctorexpr;
}

public String update() {
return updateexpr;
}

}


source of Cond.java
package edu.columbia.cs.vicecity.ir;

public class Cond extends Expr {
private Expr if_expr;
private Expr then_expr;
private Expr else_expr;

public Cond(Expr if_expr, Expr then_expr, Expr else_expr) {
super(null, resultType(if_expr, then_expr, else_expr));
this.if_expr = if_expr;
```

```
this.then_expr = then_expr;
this.else_expr = else_expr;
}

public static Type resultType(Expr a, Expr b, Expr c) {
Type rettype = null;
if (b.type == c.type && b.type != Type.Any)
rettype = b.type;
else {
warn("the actual return type of the conditional " + a + "?" + b + ":" + c
+ "\n may vary at runtime because the types do not agree:"
+ "\n then expression has type" + b.type + "\n else expression
has type"
+ c.type);
rettype = Type.Any;
}
// this could be much more robust
// think 'or'ing of types
return rettype;
}

public String decl() {
return if_expr.decl() + then_expr.decl() + else_expr.decl();
}

public String toString() {
return if_expr + "?" + then_expr + ":" + else_expr;
}

}
```

source of DeclCamera.java

```
package edu.columbia.cs.vicecity.ir;

public class DeclCamera extends DeclObject {
public static boolean havecamera = false;

public DeclCamera(String name) {
super(name, Type.Camera);
if (!havecamera)
havecamera = true;
else
error("only one camera may exist. camera " + name
+ " conflicts with an existing camera.");
}

public String decl() {
```

```
return "struct " + temp_id_prefix + tid + "{" + attrs.decl() + "\n}camera_"
+ name + ";";
}

public String ctor() {
return "\n\t" + attrs.ctor() + "\n\tfglSetCamera(" + attrToGLRef("position",
Type.Vec)
+ ", " + attrToGLRef("center", Type.Vec) + ", " + attrToGLRef("up",
Type.Vec)
+ ");";
}

public String update() {
return "\n\t" + attrs.update();
}

public String toString() {
return "camera_" + name;
}
}


source of DeclLight.java
package edu.columbia.cs.vicecity.ir;

public class DeclLight extends DeclObject {
public DeclLight(String name) {
super(name, Type.Light);
}

public String decl() {
return "struct " + temp_id_prefix + tid + "{" + attrs.decl() + "\n}light_" +
name + ";";
}

public String ctor() {
return "\n\t" + attrs.ctor() + "\n\tfglAddLight(" + attrToGLRef("position",
Type.Vec)
+ ", " + attrToGLRef("w", Type.Float) + ", " + attrToGLRef("ambient",
Type.Vec)
+ ", " + attrToGLRef("diffuse", Type.Vec) + ", "
+ attrToGLRef("specular", Type.Vec) + ", " + attrToGLRef(
"attenutation", Type.Vec)
+ ");";
}

public String update() {
return "\n\t" + attrs.update();
```

```
}

public String toString() {
return "light_" + name;
}
}
```

source of DeclMaterial.java

```
package edu.columbia.cs.vicecity.ir;

public class DeclMaterial extends DeclObject {
protected String filename;
protected Attr matid;
public DeclMaterial(String name, String filename){
super(name,Type.Material);
if(filename==null||filename.equals("")){
error("material "+name+" must supply a valid filename to load
material from");
}
this.filename=filename;
}
public void setAttrs(AttrList attrs){this.attrs = attrs; matid=new Attr
(Type.IR_uint,this,"__matid");}
public String decl(){return "struct "+temp_id_prefix+tid+"{\n\t"+matid.decl()
+attrs.decl()+"\n}material_"+name+";";}
public String ctor(){
return "\n\t"
+attrs.ctor()+"\n\tcur."+matid.qname()+"=fglLoadMaterial(\"\"," /
/currently broken, should be filename
+attrToGLRef("ambient",Type.Vec)+", "
+attrToGLRef("diffuse",Type.Vec)+", "
+attrToGLRef("specular",Type.Vec)+", "
+attrToGLRef("shininess",Type.Float)+");";
}
public String update(){return "\n\t"+attrs.update();}
public String toString(){return "material_"+name;}
public Attr matID(){return matid;}
}
```

source of DeclMesh.java

```
package edu.columbia.cs.vicecity.ir;

public class DeclMesh extends DeclObject {
protected String filename;
protected DeclMaterial material;
protected Attr dataid, meshid;
```

```
public DeclMesh(String name, String filename) {
this(name, filename, null);
}

public DeclMesh(String name, String filename, DeclMaterial material) {
super(name, Type.Mesh);
if (filename == null || filename.equals("")) {
error("mesh " + name
+ " file attribute must supply a valid filename to load
structure from");
}
/*
 * if(material==null||material.type!=Type.Material){ error("mesh
 * "+name+" material attribute must reference valid material"); }
 */
this.filename = filename;
this.material = material;
}

public void setMaterial(DeclMaterial mat) {
this.material = mat;
}

public void setAttrs(AttrList attrs) {
this.attrs = attrs;
dataid = new Attr(Type.IR_uint, this, "__dataid");
meshid = new Attr(Type.IR_uint, this, "__meshid");
}

public String decl() {
return "struct " + temp_id_prefix + tid + "{\n\t" + dataid.decl() +
"\n\t" + meshid.decl()
+ attrs.decl() + "\n}mesh_" + name + ";";
}

public String ctor() {
String retval = "\n\t" + attrs.ctor() + "\n\tcur." + dataid.qname()
+ "=fglLoadMeshData(\""
+ filename + "\");" + "\n\tcur." + meshid.qname() + "=fglAddMesh(cur."
+ dataid.qname() + ", ";
if (material != null)
retval += "cur." + material.matID().qname() + ", ";
else
retval += "0, ";
retval += attrToGLRef("position", Type.Vec) + ", " + attrToGLRef(
"rotation", Type.Vec)
+ ", " + attrToGLRef("scale", Type.Vec) + ");";
```

```
return retval;
}

public String update() {
return "\n\t" + attrs.update();
}

public String toString() {
return "mesh_" + name;
}

public Attr matID() {
return meshid;
}
}
```

source of DeclObject.java

```
package edu.columbia.cs.vicecity.ir;

public class DeclObject extends Expr {
private static int unique = 0;
public static String temp_id_prefix = "t_obj_";
protected int tid;
protected String name;
protected AttrList attrs;

public DeclObject(String name, Type t) {
super(null, t);
this.name = name;
attrs = null;
tid = ++unique;
}

public DeclObject(String name) {
this(name, Type.None);
}

public String getName() {
return name;
}

public String ctor() {
return null;
}

public String update() {
return null;
```

```
}

public void setAttrs(AttrList attrs) {
this.attrs = attrs;
}

protected String attrToGLRef(String attrname, Type expectedtype) {
Attr attr;
if ((attr = (Attr) (ScopeManager.instance().get_no_err(name + "."
+ attrname))) != null) {
if (attr.type != expectedtype)
c_error("initializing GL values of attribute " + name + "."
+ attrname + ".\n"
+ "GL subsystem expects this attribute to be "
+ expectedtype + "\n"
+ "but attribute declared as " + attr.type + ".");
return "&cur." + attr.qname();
}
else {
c_warn("using default value for " + name + "." + attrname + ".\n"
+ "you will not be able to reference this attribute.");
return "0";
}

}
}


source of Divide.java
package edu.columbia.cs.vicecity.ir;

public class Divide extends Multiply {
public Divide(Expr lchild, Expr rchild) {
super(lchild, rchild);
}

public String toString() {
return lchild.toString() + " / " + rchild.toString();
}

public static Type resultType(Expr a, Expr b) {

Type atype = a.type, btype = b.type;
if (atype == Type.Any && btype != Type.Any) {
warn("the actual type of " + a + " in division of " + a + " and " + b
+ "\n may vary at runtime and operation may fail");
atype = btype;
a.setCast(btype);
```

```
}
if (btype == Type.Any && atype != Type.Any) {
warn("the actual type of " + b + " in division of " + a + " and " + b
+ "\n may vary at runtime and operation may fail");
btype = atype;
b.setCast(atype);
}
if (atype == Type.Any && btype == Type.Any) {
warn("the actual type of both variables in division of " + a + " and " + b
+ "\n may vary at runtime and operation may fail");
// assume float as can only be float/float
atype = btype = Type.Float;
a.setCast(atype);
b.setCast(btype);
}
if (atype != btype || atype != Type.Float)
error("incompatible types: cannot divide " + a.type.toString() + " with a "
+ b.type.toString());
return Type.Float;
}

}
```

source of Dot.java

```
package edu.columbia.cs.vicecity.ir;

public class Dot extends Op {
Expr lchild, rchild;

public Dot(Expr lchild, Expr rchild) {
super(null, resultType(lchild, rchild));
this.lchild = lchild;
this.rchild = rchild;
}

public static Type resultType(Expr a, Expr b) {
Type rettype = null;
String field = b.toString();
Expr obj_attr = null;
Type atype = a.type;
if (atype == Type.Any) {
// essentially what we are doing here is setting up a cast
warn("the actual value of " + a + " in dot operation of " + a + "." + b
+ "\n may vary at runtime");
if (field.equals("x") || field.equals("y") || field.equals("z"))
atype = Type.Vec; // assume vec
if (field.equals("first") || field.equals("rest") || field.equals("mpty"))
```

```
atype = Type.List; // assume list1111111

}
if (atype == Type.Vec) {
if (field == null
|| (!(field.equals("x")) && !(field.equals("y")) && !
(field.equals("z"))))
error("illegal vector field name used with dot operator:
field name must be 'x', 'y', or 'z'");
else
rettype = Type.Float;
}
else if (atype == Type.List) {
if (field == null
|| !(field.equals("first") || field.equals("mpty") ||
field.equals("rest")))
error("illegal list field name used with dot operator:
field name must be 'first' or 'rest'");
else if ((field.equals("first")))
rettype = Type.Any;
else if ((field.equals("rest")))
rettype = Type.List;
else if ((field.equals("mpty")))
rettype = Type.Float;
}
else if (a instanceof DeclObject) {
// check if valid attribute name
obj_attr = ScopeManager.instance().get(((DeclObject) a).getName()
+ "." + field);
if (obj_attr == null)
error("illegal object field name used with dot operator: field
name must be an attribute of the "
+ a.type.toString() + " object");
else
rettype = obj_attr.type;
}
else
error("incompatible type: cannot use dot operator on " + a.type);
return rettype;
}

public String toString() {
Expr obj_attr = null;
String rexpr = rchild.toString();
Type ltype = lchild.type;
if (ltype == Type.Any) {
// essentially what we are doing here is setting up a cast
if (rexpr.equals("x") || rexpr.equals("y") || rexpr.equals("z"))
```

```
ltype = Type.Vec; // assume vec
if (rexpr.equals("first") || rexpr.equals("rest") || rexpr.equals("mpty"))
ltype = Type.List; // assume list
lchild.setCast(ltype);
}
if (ltype == Type.Vec) {
rexpr = rchild.toString();
}
else if (ltype == Type.List) {
if (rexpr.equals("mpty")) {
return "(" + lchild.castexpr() + ".hasfirst()?floatt(0.0):floatt(1.0))";
}
else
rexpr = rchild.toString() + "()";
}
else if (lchild instanceof DeclObject) {
rexpr = rchild.toString();
obj_attr = ScopeManager.instance().get(((DeclObject) lchild).getName()
+ "." + rexpr);
if (rexpr == null || obj_attr == null)
c_error("illegal object field name used with dot operator:
field name must be an attribute of the "
+ rchild.type.toString() + " object");
if (!(obj_attr instanceof Attr))
c_error("expected dot field to return an Attr type");
return "prev." + ((Attr) obj_attr).qname();
}
return lchild.castexpr() + "." + rexpr;
}

public String decl() {
if (!(lchild instanceof DeclObject))
return lchild.decl() + rchild.decl();
else
return rchild.decl();
}
}
```

source of EqualNotEqual.java

```
package edu.columbia.cs.vicecity.ir;

public class EqualNotEqual extends Op {
Expr lchild, rchild;

public EqualNotEqual(String actualOp, Expr lchild, Expr rchild) {
super(actualOp, resultType(actualOp, lchild, rchild));
this.lchild = lchild;
```

```
this.rchild = rchild;
}

public static Type resultType(String actualOp, Expr a, Expr b) {
Type atype = a.type, btype = b.type;
if (atype == Type.Any && btype != Type.Any) {
atype = btype;
warn("the actual type of " + a + " in comparison of " + a + " " +
actualOp + " " + b
+ "\n may vary at runtime and operation may fail");
a.setCast(btype);
}
if (atype != Type.Any && btype == Type.Any) {
btype = atype;
warn("the actual type of " + b + " in comparison of " + a + " " +
actualOp + " " + b
+ "\n may vary at runtime and operation may fail");
b.setCast(atype);
}
if (atype == Type.Any && btype == Type.Any) {
warn("the actual type of both variables in comparison of " + a + " " +
actualOp + " "
+ b + "\n may vary at runtime and operation may fail");
b.setCast(Type.Float);
a.setCast(Type.Float);
}
if (atype != btype || !(atype == Type.Float || atype == Type.Any))
error("incompatible types: cannot compare " + a.type + " and " +
b.type + " with '"
+ actualOp + "' operator");
return Type.Float;
}

public String toString() {
return lchild.castexpr() + " " + sexpr + " " + rchild.castexpr();
}

public String decl() {
return lchild.decl() + rchild.decl();
}
}
```

source of Expr.java

```
package edu.columbia.cs.vicecity.ir;

public class Expr extends Node {
public String sexpr;
public Type type;
```

```
public Type castto;

public Expr(String expression, Type returntype) {
sexpr = expression;
type = returntype;
castto = null;
}

public void setCast(Type castto) {
this.castto = castto;
}

public String decl() {
return "";
}

public String toString() {
return sexpr;
}// return "Expr:" + sexpr;}

public String castexpr() {
if (castto != null)
return "((" + castto + ") (" + this.toString() + "))";
else
return this.toString();
}

public void emitDecl() {
System.out.println(decl());
}

public void emit() {
emit(this.toString());
}
}


source of FieldAccessor.java
package edu.columbia.cs.vicecity.ir;


public class FieldAccessor extends Expr {

//should be one of [x,y,z,first,rest] for vecs & lists but can be anything
for objects like meshes etc.

public FieldAccessor(String sexpr){
super(sexpr, null);
```

```
}
public String toString(){
return sexpr;
}
}
```

source of Float.java

```
package edu.columbia.cs.vicecity.ir;

public class Float extends Expr {
public Float(double floatconst) {
super(Double.toString(floatconst) + "f", Type.Float);
}

public String toString() {
return "floatt(" + sexpr + ")";
}
}
```

source of FunCall.java

```
package edu.columbia.cs.vicecity.ir;

import java.util.Iterator;

public class FunCall extends Expr {
private ArgExprList argexprs;
private String name;
public FunCall(String name, ArgExprList argexprs){
super(null,ScopeManager.instance().get(name).type);
Function declaration = (Function) ScopeManager.instance().get(name);

//check that sizes match up
if(declaration.getArgs().getList().size()!=argexprs.getList().size())
error("incorrect number of function parameters for function call to "+name
+ ", expected "+declaration.getArgs().getList().size()
+ " arguments but called with "+argexprs.getList().size()
+ " arguments");
// hahah when you say it out loud it sounds like parameter, but its an
iterator. i love english
Iterator<Id> paramiter=declaration.getArgs().getList().iterator();
Iterator<Expr> expriter=argexprs.getList().iterator();
int count=0;
while(paramiter.hasNext()&& expriter.hasNext()){
count++;
Expr curexpr=expriter.next();
Type paramtype=paramiter.next().type;
```

```
Type exprtype=curexpr.type;
if(paramtype!=Type.Any){
if(exprtype==Type.Any){
warn("the actual type of parameter "+count+" to function call
 of "+name
+"\n may vary at runtime and operation may fail");
exprtype=paramtype;
curexpr.setCast(paramtype);
}
if(paramtype!=exprtype){
error("mismatched type for parameter "+count+" to function
 call of "+name
+ ", expected "+paramtype
+ " but called with "+exprtype);
}
}
}

this.argexprs=argexprs;
this.name=declaration.getName();
}
public String toString(){
//stupid hack if(name.equals("sin")) return "sinf"+argexprs;
return name+argexprs;} //all that for so little
}


source of Function.java
package edu.columbia.cs.vicecity.ir;

public class Function extends Expr {
private String name;
private Expr body;
private ParamList args;

public Function(Type returntype, String name, ParamList args, Expr body) {
super(null, returntype);
this.name = name;
this.body = body;
this.args = args;
if (body != null && returntype != body.type) // allow body to be
// filled in later
error("incompatible types: " + name + " declared return type " + returntype
+ "does not agree with " + name + " body return type " + body.type);
}

public String decl() {
return type + " " + name + args + ";";
```

```
}

public String toString() {

if (body != null)
return type + " " + name + args + "{\n\t" + body.decl() + "\n\treturn "
+ body.castexpr() + ";\n}";
else
return type + " " + name + args + "{BODY_NOT_DEFINED}";
}

public ParamList getArgs() {
return args;
}

public void addBody(Expr body) {
if (body == null) {
c_error("cannot supply a null body to function " + decl());
}
if (!(this.body == null)) {
c_error("in function " + decl() + "\n cannot replace exiting body: "
 + this.body
+ " with " + body);
}
if (body.type == Type.Any) {
c_warn("the actual return type of the body of the function " + decl()
+ "\n may vary at runtime and cause an error");
body.setCast(this.type);
}
if (body != null && this.type != body.type && !(body.type == Type.Any))
error("incompatible types: " + name + " declared return type " + this.type
+ "does not agree with " + name + " body return type " + body.type);
this.body = body;
}

public String getName() {
return name;
}
}


source of Group.java
package edu.columbia.cs.vicecity.ir;

public class Group extends Expr {
Expr expr;

public Group(Expr expr) {
```

```
super("(" + expr + ")", expr.type);
this.expr = expr;
}

public String decl() {
return expr.decl();
}

public String toString() {
return "(" + expr + ")";
}
}
```

source of Id.java

```
package edu.columbia.cs.vicecity.ir;

public class Id extends Expr {
// hash table ref?
public Id(String id, Type p) {
super(id, p);
}

public String paramDecl() {
return type.toString() + " " + sexpr;
}

public String decl() {
return /* type.toString()+" "+sexpr; */"";
}

public String toString() {
return sexpr;
}
}
```

source of Inversion.java

```
package edu.columbia.cs.vicecity.ir;

public class Inversion extends Op {
Expr child;

public Inversion(Expr child) {
super(null, resultType(child));
this.child = child;
}
```

```
public static Type resultType(Expr a) {
Type rettype = null;
a.setCast(Type.Float);
if (a.type == Type.Any) {
warn("the actual type of " + a + " when inverted"
+ "\n may vary at runtime and operation may fail");

}
if (a.type == Type.Float || a.type == Type.Any)
rettype = Type.Float;
else
error("incompatible type: only a float can be used with the '!' operator");
return rettype;
}

public String toString() {
return "(" + child.castexpr() + "==0.0f?1.0f:0.0f)";
}

public String decl() {
return child.decl();
}
}
```

```
source of ListList.java
package edu.columbia.cs.vicecity.ir;

public class ListList extends StuffList<Expr> {
//private static int unique=0;
//public static String temp_id_prefix="t_list_";
protected String endstring;
//protected int tempnum;
public ListList(){
super(Type.List);
sexpr="list()";
//tempnum=++unique;
//sexpr=temp_id_prefix + tempnum;
//declexpr="List "+temp_id_prefix+tempnum+";\n\t";
}
public void add(Expr expr){
super.add(expr);
sexpr= "list(" + expr.toString() + ", "+sexpr+")";
declexpr+= expr.decl();
}
}
```

```
source of Logical.java
package edu.columbia.cs.vicecity.ir;

public class Logical extends Op {
Expr lchild, rchild;

public Logical(String actualOp, Expr lchild, Expr rchild) {
super(actualOp, resultType(actualOp, lchild, rchild));
this.lchild = lchild;
this.rchild = rchild;
}

public static Type resultType(String actualOp, Expr a, Expr b) {
Type atype = a.type, btype = b.type;
if (atype == Type.Any && btype != Type.Any) {
atype = btype;
warn("the actual type of " + a + " in comparison of " + a + " " +
actualOp + " " + b
+ "\n may vary at runtime and operation may fail");
a.setCast(btype);
}
if (atype != Type.Any && btype == Type.Any) {
btype = atype;
warn("the actual type of " + b + " in comparison of " + a + " " +
actualOp + " " + b
+ "\n may vary at runtime and operation may fail");
b.setCast(atype);
}
if (atype == Type.Any && btype == Type.Any) {
warn("the actual type of both variables in comparison of " + a + " " +
 actualOp + " "
+ b + "\n may vary at runtime and operation may fail");
atype = btype = Type.Float;
a.setCast(atype);
b.setCast(btype);
}
if (atype != btype || atype != Type.Float)
error("incompatible types: cannot compare " + a.type + " and " +
b.type + " with '"
+ actualOp + "' operator");
return Type.Float;
}

public String toString() {
return lchild.castexpr() + " " + sexpr + " " + rchild.castexpr();
}

public String decl() {
```

```
return lchild.decl() + rchild.decl();
}
}


source of Multiply.java
package edu.columbia.cs.vicecity.ir;

public class Multiply extends Op {
Expr lchild, rchild;

public Multiply(Expr lchild, Expr rchild) {
super(null, resultType(lchild, rchild));
this.lchild = lchild;
this.rchild = rchild;
}

public static Type resultType(Expr a, Expr b) {
Type rettype = null;
Type atype = a.type, btype = b.type;
if (atype == Type.Any && btype != Type.Float) {
warn("the actual type of " + a + " in multiplication of " + a + "
and " + b
+ "\n may vary at runtime and operation may fail");
atype = btype = rettype = Type.Float;
a.setCast(Type.Float);
}
else if (atype == Type.Any && btype != Type.Any) {
error("the actual type of " + a + " in multiplication of " + a + "
and " + b
+ "\n may vary at runtime and operation may fail a casting
function like"
+ "\n float thinkitsafloat(float x):x; is required");
atype = btype = rettype = Type.Any;
}
else if (btype == Type.Any && atype != Type.Any) {
error("the actual type of " + b + " in multiplication of " + a + "
and " + b
+ "\n may vary at runtime and operation may fail a casting
function like"
+ "\n float thinkitsafloat(float x):x; is required");
atype = btype = rettype = Type.Any;
}
else if (atype == Type.Any && btype == Type.Any) {
error("the actual type of both variables in multiplication of " + a +
" and " + b
+ "\n may vary at runtime and operation may fail a casting
function like"
```

```
+ "\n float thinkitsafloat(float x):x; is required for at
least one variable");
rettype = Type.Any;
}
else if (atype == Type.Float && btype == Type.Float)
rettype = Type.Float;
else if (atype == Type.Vec && btype == Type.Vec)
rettype = Type.Float; // dot product
else if (atype == Type.Vec && btype == Type.Float)
rettype = Type.Vec;
else if (atype == Type.Float && btype == Type.Vec)
rettype = Type.None;
if (!(rettype == Type.Vec || rettype == Type.Float || rettype == Type.Any))
error("incompatible types: cannot multiply " + a.type.toString() +
" with a "
+ b.type.toString());
return rettype;
}

public String toString() {
return lchild.castexpr() + " * " + rchild.castexpr();
}

public String decl() {
return lchild.decl() + rchild.decl();
}
}
```

source of Node.java

```
package edu.columbia.cs.vicecity.ir;

public class Node {
public Node() {
}

void c_error(String s) {
System.err.println(getClass().getSimpleName() + " Error " + s + "\n");
throw new Error(getClass().getSimpleName() + " Error " + s + "\n");
}

static void error(String s) {
System.err.println("error " + s);
throw new Error(" Error " + s + "\n");
}

void c_warn(String s) {
System.err.println(getClass().getSimpleName() + " Warning " + s + "\n");
```

```
}

static void warn(String s) {
System.err.println("warning " + s);
}

static void emit(String s) {
System.out.println("\t" + s);
}
}
```

source of Op.java

```
package edu.columbia.cs.vicecity.ir;

public class Op extends Expr {
public Op(String tok, Type p) {
super(tok, p);
}
}
```

source of ParamList.java

```
package edu.columbia.cs.vicecity.ir;

public class ParamList extends StuffList<Id> {
private boolean first;

public ParamList() {
super(Type.None);
first = true;
}

public void add(Id id) {
super.add(id);
if (first) {
sexpr += id.paramDecl();
first = false;
}
else {
sexpr += "," + id.paramDecl();
}
}

public String toString() {
return "(" + sexpr + ")";
}
}
```

```
source of Program.java
package edu.columbia.cs.vicecity.ir;

import java.util.LinkedList;

public class Program extends Node {
private LinkedList<Function> functions;
private LinkedList<DeclObject> objects;
private static boolean initialized = false;
private static Program instance;

private Program() {
initialized = true;
functions = new LinkedList<Function>();
objects = new LinkedList<DeclObject>();
}

public static Program init() {
if (!initialized)
instance = new Program();
return instance;
}

public static Program instance() {
return init();
}

public static void addFunction(Function function) {
instance.functions.add(function);
}

public static void addObject(DeclObject object) {
instance.objects.add(object);
}

public String toString() {
String retval = "";
retval += "#include \"FunkGL.h\"\n";
retval += "#include <iostream>\n\n";
retval += "namespace{\n";
for (Function function : instance.functions) {
retval += function.decl() + "\n";
}
retval += "\n\n";
retval += "struct state {\n";
for (DeclObject object : instance.objects) {
```

```
retval += object.decl() + "\n";
}
retval += "};";
retval += "\nstate prev, cur;";
retval += "\n\n";
retval += "void init(){";
retval += "\n\tdt=0;";
for (DeclObject object : instance.objects) {
retval += object.ctor() + "\n";
}
retval += "}";
retval += "\n\n";
retval += "void update(){";
retval += "\n\tprev=cur;";
retval += "\n\tcollect();";
for (DeclObject object : instance.objects) {
retval += object.update() + "\n";
}
retval += "}";
retval += "\n\n";
for (Function function : instance.functions) {
retval += function.toString() + "\n";
}
retval += "}\n\n";
retval += "int main(int argc, char **argv) {\n" + "\ttry{\n" + "\t\tinit();\n"
+ "\t\tfglScene(update,&argc,argv);\n" + "\t}catch(char const *str){\n"
+ "\t\tstd::cout<<str<<std::endl;\n" + "\t}\n" + "}\n";
return retval;
}

public static String programString() {
return instance.toString();
}

}


source of Relational.java
package edu.columbia.cs.vicecity.ir;

public class Relational extends Op {
Expr lchild, rchild;
public Relational(String actualOp, Expr lchild, Expr rchild){
super(actualOp,resultType(actualOp,lchild,rchild));
this.lchild=lchild;
this.rchild=rchild;
}
public static Type resultType(String actualOp, Expr a, Expr b){
```

```
Type atype=a.type,btype=b.type;
if(atype==Type.Any && btype!=Type.Any){
atype=btype;
warn("the actual type of "+a+" in comparison of "+a+" "+actualOp+" "+b
+"\n may vary at runtime and operation may fail");
a.setCast(btype);
}
if(atype!=Type.Any && btype==Type.Any){
btype=atype;
warn("the actual type of "+b+" in comparison of "+a+" "+actualOp+" "+b
+"\n may vary at runtime and operation may fail");
b.setCast(atype);
}
if(atype==Type.Any && btype==Type.Any){
warn("the actual type of both variables in comparison of "+a+" "
+actualOp+" "+b
+"\n may vary at runtime and operation may fail");
a.setCast(Type.Float);
b.setCast(Type.Float);
}
if(atype!=btype || atype!=Type.Float)
error("incompatible types: cannot compare "+a.type+" and "+b.type+
" with '"+actualOp+"' operator");
return Type.Float;
}
public String toString(){
return lchild.castexpr()+" "+sexpr+" "+rchild.castexpr();
}
public String decl(){
return lchild.decl()+rchild.decl();
}
}
```

source of Scope.java

```
package edu.columbia.cs.vicecity.ir;

import java.util.Hashtable;

public class Scope {
private Hashtable<String, Expr> scopetable;

public Scope() {
scopetable = new Hashtable<String, Expr>();
}

public Scope(ParamList l) {
scopetable = new Hashtable<String, Expr>();
```

```
for (Id id : l.list)
put(id.toString(), id);
}

public void put(String sexpr, Expr expr) {
Expr e;
reservedCheck(sexpr);
if ((e = scopetable.put(sexpr, expr)) != null) {
throw new Error("error: identifier redefinition: " + expr
+ " \ncompetes with existing " + e + " \nfor rights to
" + sexpr + "\n");
}
}

public Expr get(String sexpr) {
return scopetable.get(sexpr);
}

public static void reservedCheck(String s) {
if (s.equals("expr") || s.equals("list") || s.equals("floatt") ||
s.equals("vec")
|| s.equals("light") || s.equals("camera") || s.equals("mesh")
|| s.equals("material") || s.equals("expr") || s.equals("asm")
|| s.equals("auto")
|| s.equals("break") || s.equals("case") || s.equals("switch")
|| s.equals("catch")
|| s.equals("char") || s.equals("class") || s.equals("const")
|| s.equals("continue") || s.equals("default") || s.equals("delete")
|| s.equals("do") || s.equals("double") || s.equals("else")
|| s.equals("enum")
|| s.equals("extern") || s.equals("float") || s.equals("for")
|| s.equals("friend")
|| s.equals("goto") || s.equals("if") || s.equals("inline")
|| s.equals("int")
|| s.equals("long") || s.equals("new") || s.equals("operator")
|| s.equals("private") || s.equals("protected") || s.equals("public")
|| s.equals("register") || s.equals("return") || s.equals("short")
|| s.equals("signed") || s.equals("sizeof") || s.equals("static")
|| s.equals("struct") || s.equals("switch") || s.equals("template")
|| s.equals("this") || s.equals("throw") || s.equals("try")
|| s.equals("typedef")
|| s.equals("union") || s.equals("unsigned") || s.equals("virtual")
|| s.equals("volatile") || s.equals("while"))
throw new Error("error: identifier redefinition: " + s
+ " \ncompetes with existing c++ reserved word\n");
}

}
```

```
source of ScopeManager.java
package edu.columbia.cs.vicecity.ir;

import java.util.Stack;

public class ScopeManager {
private static boolean initialized = false;
private Stack<Scope> scopestack;
private Stack<Scope> tempstack;
private static ScopeManager instance;

private ScopeManager() {
initialized = true;
scopestack = new Stack<Scope>();
tempstack = new Stack<Scope>();
}

public static ScopeManager init() {
if (!initialized)
instance = new ScopeManager();
return instance;
}

public static ScopeManager instance() {
return init();
}

public void pushScope(Scope s) {
scopestack.push(s);
}

public Scope popScope() {
return scopestack.pop();
}

public Expr get_no_err(String sexpr) {
Expr retval = null;
while (scopestack.size() > 0 && (retval = scopestack.peek().get(sexpr))
== null) {
tempstack.push(scopestack.pop());
}
while (tempstack.size() > 0) {
scopestack.push(tempstack.pop());
}
return retval;
}
```

```
public Expr get(String sexpr) {
Expr retval = get_no_err(sexpr);
if (retval == null) {
throw new Error("invalid reference: " + sexpr + " is not in the
current scope\n");
}
return retval;
}

public void put(String sexpr, Expr expr) {
scopestack.peek().put(sexpr, expr);
}

}
```

source of StuffList.java

```
package edu.columbia.cs.vicecity.ir;

import java.util.LinkedList;

public class StuffList<T> extends Expr {
protected LinkedList<T> list;
protected String sexpr, declexpr;
public StuffList(){
super(null,Type.List);
list=new LinkedList<T>();
sexpr=""; declexpr="";
}
public StuffList(Type type){
super(null,type);
list=new LinkedList<T>();
sexpr=""; declexpr="";
}
public String decl(){return declexpr;}
public void add(T expr){
list.add(expr);
}
public String toString(){return sexpr;}
public LinkedList<T> getList(){
return list;
}
}
```

source of Subtract.java

```
package edu.columbia.cs.vicecity.ir;
```

```
public class Subtract extends Add {
public Subtract(Expr lchild, Expr rchild) {
super(lchild, rchild);
}

public String toString() {
return lchild.castexpr() + " - " + rchild.castexpr();
}
}
```

source of Type.java

```
package edu.columbia.cs.vicecity.ir;

public class Type {
public static final Type Float = new Type("floatt", new Float(0.0)),
Vec = new Type("vec",
new Vec(new Float(0.0), new Float(0.0), new Float(0.0))), List =
new Type("list",
new ListList()), None = new Type("NO_TYPE"), Any = new Type("expr"),
Mesh = new Type(
"mesh"), Material = new Type("material"), Light = new Type("light"),
Camera = new Type(
"camera"), IR_uint = new Type("unsigned int");
private String name;
private Expr defaultval;

public Type(String s) {
name = s;
this.defaultval = null;
}

public Type(String s, Expr defaultval) {
name = s;
this.defaultval = defaultval;
}

public String toString() {
return name;
}

public Expr defaultVal() {
return defaultval;
}
}
```

```
source of Vec.java
package edu.columbia.cs.vicecity.ir;

public class Vec extends Expr {
Expr x, y, z;

public Vec(Expr x, Expr y, Expr z) {
super(null, Type.Vec);
if (!(x.type == Type.Float && y.type == Type.Float && z.type == Type.Float))
error("incompatible types: a vector can only be constructed of 3 floating"
+ "point expressions.\n  expected (float,float,float)\n" +
"but saw (" + x.type
+ "," + y.type + "," + z.type + ")");
this.x = x;
this.y = y;
this.z = z;

}

public String toString() {
return "vec(" + x.toString() + "," + y.toString() + "," + z.toString() + ")";
}

public String decl() {
return x.decl() + y.decl() + z.decl();
}
}
```

Appendix C

Non-IR Code

```
source of t.g
class P extends Parser;
options { k=1; buildAST=true; }
tokens {
ARGS; ATTRS; FUNCTION; PROGRAM; BODY;
FUNCALL; LIST; VEC; GROUP; NEGATE;
}

//Start of program
program
: (function | object)+ { #program=#([PROGRAM,"PROGRAM"],#program); }
;

//Common
type_specifier
: ("float" | "vector" | "list" | "meshes")
;

//Object Rules
object
: mesh | material | other_objs;

mesh
: ("abstract")? "mesh"^ ID
LBRACE!
"file"! ASSIGN! STRINGCONST SEMI!
("material"! ASSIGN! ID SEMI!)?
attr_list
RBRACE!
;

material
: "material"^ ID
LBRACE!
```

```
"file"! ASSIGN! STRINGCONST SEMI!
attr_list
RBRACE!
;

other_objs
: ("light"^ | "camera"^) ID
LBRACE!
attr_list
RBRACE!
;
attr
:type_specifier update_identifier:ID^ ASSIGN! expr SEMI!
(TILDE! initializer_identifier:ID! {update_identifier.getText().equals(
  initializer_identifier.getText())}? ASSIGN! const_expr SEMI!)?
;

attr_list
:
(attr)* { #attr_list=#([ATTRS,"ATTRS"],#attr_list); }
;

const_expr
: const_float
| const_vec { #const_expr=#([VEC,"VEC"],#const_expr); }
| LBRACK! (const_list)? RBRACK! { #const_expr=#([LIST,"LIST"],#const_expr); }
;

const_float
: (MINUS^ {#const_float.setType(NEGATE);})? FLOAT
;

const_vec
: LPAREN! const_float COMMA! const_float COMMA! const_float RPAREN!
;

const_list
: const_expr (COMMA! const_expr)*
;

//Function Rules

function: type_specifier i:ID args COLON! body SEMI! {
 #function=#([FUNCTION, i.getText()],#function); }
;

args
: LPAREN! (list_arg)? RPAREN! { #args=#([ARGS,"ARGS"],#args); }
```

```
;

list_arg
: arg ( COMMA! arg)*
;

arg :  type_specifier ID
;

body : expr { #body=#([BODY,"BODY"],#body); }
;

expr
: cond_expr
;

cond_expr
        :       or_expr
                ( QMARK^ expr COLON! expr)?

        ;

or_expr
        :       and_expr ( OR^ and_expr )*
        ;

and_expr
        :       eq_expr ( AND^ eq_expr )*
        ;

eq_expr
        :       cmp_expr
                ( ( EQ^ | NE^ ) cmp_expr )*
        ;

cmp_expr
        :       add_expr
                ( ( LT^ | LE^ | GT^ | GE^ ) add_expr )*
        ;

add_expr
        :       mult_expr
                ( ( PLUS^ | MINUS^ ) mult_expr )*
        ;

mult_expr
        :       unary_expr
                ( ( MULTIPLY^ | DIVIDE^) unary_expr )*
```

```
        ;

unary_expr
        :       postfix_expr
        |       EMARK^ unary_expr
        | MINUS^ unary_expr {#unary_expr.setType(NEGATE);}
        ;

postfix_expr
        :       primary_expr (( DOT^ ID)+)?
        ;

primary_expr
        :       i:ID (LPAREN! list_arg_expr RPAREN! {
  #primary_expr=#([FUNCALL,i.getText()],#primary_expr);})?
        | FLOAT
        | LPAREN! expr ( COMMA! expr COMMA! expr RPAREN! {
  #primary_expr=#([VEC,"VEC"],#primary_expr); }
         | RPAREN! { #primary_expr=#([GROUP,"( )"],
  #primary_expr); })
        | LBRACK! (list_expr)? RBRACK! { #primary_expr=#([LIST,"LIST"],
  #primary_expr); }
        ;

list_expr
        :       expr ( COMMA! expr)*
        ;
//sometimes you want to tree it (function call) sometimes you dont (list_expr:
  for lists)
list_arg_expr
:  (list_expr)? { #list_arg_expr=#([ARGS,"ARGS"],#list_arg_expr); }
;


class L extends Lexer;
options {k=2; }

//unary operators
AND  : '&';
OR : '|';
QMARK : '?';
EMARK : '!';
COLON : ':'  ; //parser note: can be either THEN (after a ?) or naming
  (after keyword mesh)
DOT : '.';

//binary operators
PLUS : '+';
```

```
MINUS : '-';
MULTIPLY : '*';
DIVIDE : '/';

//comparison tokens
LT  : '<';
LE  : '<' '=';
EQ  : '=' '=';
NE : '!' '=';
GT : '>';
GE  : '>' '=';

//assignment token
ASSIGN : '=';

//grouping tokens
LPAREN  : '(';
RPAREN : ')';
LBRACK  : '[';
RBRACK  : ']';
LBRACE  : '{';
RBRACE  : '}';

//additonal tokens
COMMA : ',';
SEMI : ';';
TILDE : '~';

//skipped over tokens
protected
RCOMMENT : '*' '/';

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

//onto more interesting stuff...
//number tokens
protected
DIGIT
: ('0'..'9')
;

protected
```

```
DIGITS
: (DIGIT)+
;

protected
EXPONENT
: ('e'|'E') ('+'|'-')? DIGITS
;

FLOAT
:(DIGITS //starts with a digit
(
('.' //has decimal part
(
(DIGITS (EXPONENT)?)
|(EXPONENT)?
)
)
|EXPONENT //no decimal part, but has exponent
)
)
|('.' //no starting digit, has to start with decimal
(DIGITS (EXPONENT)?)
)
;

//letter-related tokens
protected
LETTER
: ('a'..'z' |'A'..'Z')
;

protected
IDTAIL
: (LETTER | DIGIT ) //only chars allowed in an id name
;

ID
:LETTER (IDTAIL)* //id must start with letter
;

STRINGCONST
: '"'! (~('"') )* '"'!
;


COMMENT
: '/'
```

```
(
('*'
(
('\r' '\n') => '\r' '\n'{ newline(); }
| '\r'                  { newline(); }
| '\n'                  { newline(); }
|'*' ~('/')
| ~('*'|'\n'|'\r')
)* RCOMMENT
)
| '/' (~('\n'))* '\n'           { newline(); }
)
{$setType(Token.SKIP);}
;


{import edu.columbia.cs.vicecity.ir.*;}
class W extends TreeParser;
{
ScopeManager scope=ScopeManager.instance();
}

programdecl returns [Program p]
{  p=Program.instance(); scope.pushScope(new Scope());
scope.put("dt",new Id("dt",Type.Float));
scope.put("mousex",new Id("floatt(mouseX)",Type.Float));
scope.put("mousey",new Id("floatt(mouseY)",Type.Float));
scope.put("mouselpressed",new Id("floatt(mouseLPressed)",Type.Float));
scope.put("mouserpressed",new Id("floatt(mouseRPressed)",Type.Float));
ParamList pre_def = new ParamList();
pre_def.add(new Id("x",Type.Float));
ParamList listref = new ParamList();
listref.add(new Id("&list",Type.Any));
scope.put("sin",new Function(Type.Float, "sin", pre_def,
new Expr("floatt(sinf(x))",Type.Float)));
Program.addFunction(new Function(Type.Float, "sin", pre_def,
new Expr("floatt(sinf(x))",Type.Float)));
scope.put("cos",new Function(Type.Float, "cos", pre_def,
new Expr("floatt(cosf(x))",Type.Float))    );
Program.addFunction(new Function(Type.Float, "cos", pre_def,
new Expr("floatt(cosf(x))",Type.Float)));
scope.put("exp",new Function(Type.Float, "exp", pre_def,
new Expr("floatt(expf(x))",Type.Float))    );
Program.addFunction(new Function(Type.Float, "exp", pre_def,
new Expr("floatt(expf(x))",Type.Float)));
ParamList consargs=new ParamList();
consargs.add(new Id("&e",Type.Any));
consargs.add(new Id ("&l", Type.List));
```

```
Function cons=new Function(Type.List, "list", consargs,
new Expr("list(e, l)",Type.List));
scope.put("cons",cons);


}
:  #(PROGRAM
(primarydecl)*
//functions
)
;
program returns [Program p]
{ p=Program.instance();}
:  #(PROGRAM
(primaryeval)*
//functions
)
;
//Put the functions into the scope manager first, add the body later
primarydecl
{ Type t=null; ParamList params; Function func; DeclMaterial mat;
DeclLight light; DeclCamera cam; DeclMesh mesh;}
:  #(FUNCTION
t=type ID params=paramlist BODY
{
Program.addFunction(func=new Function(t,#ID.getText(),
params,null));
scope.put(#ID.getText(),func);
}
)
|
#("material"
ID STRINGCONST
{mat = new DeclMaterial(#ID.getText(),#STRINGCONST.getText());}
declattrs[mat]
{Program.addObject(mat);scope.put(#ID.getText(),mat); }
)
|
#("light"
ID
{light = new DeclLight(#ID.getText());}
declattrs[light]
{Program.addObject(light);scope.put(#ID.getText(),light); }
)
|
#("camera"
ID
{cam = new DeclCamera(#ID.getText());}
```

```
declattrs[cam]
{Program.addObject(cam);scope.put(#ID.getText(),cam); }
)
|
#("mesh"
i:ID STRINGCONST (ID)? //thar be the material (.)
{mesh = new DeclMesh(i.getText(),#STRINGCONST.getText());}
declattrs[mesh]
{Program.addObject(mesh);scope.put(i.getText(),mesh); }
)
;

paramlist returns [ParamList params]
{ params = new ParamList(); Type t=null;}
: #(ARGS
(t=type ID { params.add(new Id(#ID.getText(),t));})*)
;

//order of operations here, this must be called so we can add all attrs
to the scope first, w/o dealing
//with their update functions
declattrs [DeclObject p]
: #(ATTRS (declattr[p])* )
;

functionizedattrlist [DeclObject p] returns [AttrList attrs]
{attrs=new AttrList(); Type t=null; Expr init; Expr update; Attr attr=null;}
: #(ATTRS
(
#(i:ID t=type  update=expr (.)? //the dot means anything, not '.'
  {attr=((Attr)scope.get(p.getName()+"."+i.getText()));
attr.addUpdateFunction(update);attrs.add(attr);}
   )
   )*
  )
;

declattr [DeclObject p]
{Attr attr= null; Type t=null; Expr init; }
: #(i:ID t=type  .
(
(init=expr {attr=new Attr(t,p,i.getText(),null,init);})
|(/*noinit*/{attr=new Attr(t,p,i.getText(),null,null);})
)
{scope.put(attr.scopename(),attr);}
)
;
```

```
primaryeval
{ Type t=null; ParamList params; Function func; Expr ebody; DeclObject o;
DeclMesh mesh; AttrList attrs;}
:  #(FUNCTION
t=type ID params=paramlist {scope.pushScope(new Scope(params));}
ebody=body
{
func=(Function) scope.get(#ID.getText());
func.addBody(ebody);
scope.popScope();

}
)
|
#("material"
ID STRINGCONST
{o = (DeclObject)scope.get(#ID.getText());}
attrs=functionizedattrlist[o]
{o.setAttrs(attrs);}
)
|
#("light"
ID
{o = (DeclObject)scope.get(#ID.getText());}
attrs=functionizedattrlist[o]
{o.setAttrs(attrs);}
)
|
#("camera"
ID
{o = (DeclObject)scope.get(#ID.getText());}
attrs=functionizedattrlist[o]
{o.setAttrs(attrs);}
)
|
#("mesh"
n:ID STRINGCONST
{mesh=(DeclMesh)scope.get(n.getText());}
(m:ID {mesh.setMaterial((DeclMaterial)scope.get(m.getText()));})?
attrs=functionizedattrlist[mesh]
{mesh.setAttrs(attrs);}
)
;



type returns [Type t]
```

```
{t = null;}
: ("float"  {t = Type.Float;}
| "vector"  {t = Type.Vec;}
| "list"  {t = Type.List;}
| "meshes" {t = Type.None;}
  )
;

body returns [Expr e]
{e = null;}
:#(BODY e=expr)
;

expr returns [Expr e]
{e=null; Expr a,b,c; ListList l; ArgExprList args;}
:
  ( #(PLUS  a=expr b=expr {e=new Add(a,b);})
|  #(MINUS  a=expr b=expr {e=new Subtract(a,b);})
| #(NEGATE a=expr {e=new Group(new Subtract(
new edu.columbia.cs.vicecity.ir.Float(0),a));})
|  #(MULTIPLY  a=expr b=expr {e=new Multiply(a,b);})
|  #(DIVIDE  a=expr b=expr {e=new Divide(a,b);})
| #(VEC a=expr b=expr c=expr {e=new Vec(a,b,c);})
| #(LIST {l=new ListList();} (a=expr {l.add(a);})* {e=l;})
| #(FUNCALL ID args=arglist {e=new FunCall(#ID.getText(),args);})
| #(QMARK a=expr b=expr c=expr {e=new Cond(a,b,c);})
| #(GROUP a=expr {e=new Group(a);})
| #(DOT a=expr b=fieldaccessor {e = new Dot(a,b);})
| #(EQ a=expr b=expr {e=new EqualNotEqual("==",a,b);})
| #(NE a=expr b=expr {e=new EqualNotEqual("!=",a,b);})
| #(LT a=expr b=expr {e=new Relational("<",a,b);})
| #(LE a=expr b=expr {e=new Relational("<=",a,b);})
| #(GE a=expr b=expr {e=new Relational(">=",a,b);})
| #(GT a=expr b=expr {e=new Relational(">",a,b);})
| #(AND a=expr b=expr {e=new Logical("&&",a,b);})
| #(OR a=expr b=expr {e=new Logical("||",a,b);})
| #(EMARK a=expr {e=new Inversion(a);})
|  FLOAT {e = new edu.columbia.cs.vicecity.ir.Float(
Double.parseDouble(#FLOAT.getText()));}
|  ID {e = scope.get(#ID.getText());}
)

;
fieldaccessor returns [FieldAccessor fa]
{ fa=null;}
: #(ID {fa = new FieldAccessor(#ID.getText());})
;
arglist returns [ArgExprList args]
```

```
{ args = new ArgExprList(); Expr e=null;}
: #(ARGS
(e=expr { args.add(e);})*)
;


source of expr.h
#ifndef EXPR_H_
#define EXPR_H_
#include <iostream>
#include <cmath>
#include "garbage.h"
//void const printlist(const expr &l)const{printlistr(l);
enum type{ floatt_t, list_t, vec_t };

class expr {
public:
virtual enum type type() const=0;
virtual const expr &first() const{
throw "cannot get first of this expr";
}
virtual const list &rest() const{
throw "cannot get rest of this expr";
}
virtual const bool hasrest() const{
throw "cannot get rest of this expr";
}
virtual const bool hasfirst() const{
throw "cannot get rest of this expr";
}
virtual const float val() const{
throw "cannot get val of this expr";
}
virtual const bool operator==(const expr &o){
throw "comparison ony defined for real numbers";
}
virtual const bool operator!=(const expr &o){
throw "comparison ony defined for real numbers";
}
virtual const bool operator<(const expr &o){
throw "comparison ony defined for real numbers";
}
virtual const bool operator>(const expr &o){
throw "comparison ony defined for real numbers";
}
virtual const bool operator<=(const expr &o){
throw "comparison ony defined for real numbers";
}
```

```
virtual const bool operator>=(const expr &o){
throw "comparison ony defined for real numbers";
}
virtual const expr &operator+(const expr &o) const{
throw "cannot add this expr";
}
virtual const expr &operator-(const expr &o)const{
throw "cannot subtract this expr";
}
virtual const expr &operator*(const expr &o)const{
throw "cannot multiply this expr";
}
virtual const expr &operator/(const expr &o)const{
throw "cannot divide this expr";
}
virtual const expr &operator -() const {
throw "cannot negate this expr";
}
virtual const floatt &length() const {
throw "expr not a vector";
}
virtual const vec &unit() const {
throw "expr not a vector";
}
virtual ~expr(){};
};

class floatt : public expr{
private:
float value;
public:
floatt(){
this->value=0;
}
floatt(float value){
this->value=value;
}
floatt(const expr &e){
if(e.type()==floatt_t)
this->value=e.val();
}
const floatt &operator=(expr &e){
if(this == &e) return *this;
if(e.type()==floatt_t)
this->value=e.val();
return *this;
}
const floatt &operator=(float value){
```

```
this->value=value;
return *this;
}
const float val() const{
return this->value;
}
operator float () const{
return this->value;
}
enum type type() const{
return floatt_t;
}
const floatt &operator+(const expr &o) const{
if(o.type()!=floatt_t) throw " type error tried to add non-float to float";
return newf((float)*((floatt *)this) + (float)(floatt)o);
}
const floatt &operator-(const expr &o) const{
if(o.type()!=floatt_t) throw " type error tried to subtract
non-float from float";
return newf((float)*((floatt *)this) - (float)(floatt)o);
}
const floatt &operator*(const expr &o) const{
if(o.type()!=floatt_t) throw " type error tried to muliply
non-float to float";
return newf((float)*((floatt *)this) * (float)(floatt)o);
}
const floatt &operator/(const expr &o) const{
if(o.type()!=floatt_t) throw " type error tried to divide
non-float to float";
return newf((float)*((floatt *)this) / (float)(floatt)o);
}
bool operator==(const expr &o) const{
if(o.type()==floatt_t)
return this->value == o.val();
else
throw "comparison == of float to non-float";
}
bool operator!=(const expr &o) const{
if(o.type()==floatt_t)
return this->value != o.val();
else
throw "comparison != of float to non-float";
}
bool operator<(const expr &o) const{
if(o.type()==floatt_t)
return this->value < o.val();
else
throw "comparison < of float to non-float";
```

```
}
bool operator>(const expr &o) const{
if(o.type()==floatt_t)
return this->value > o.val();
else
throw "comparison > of float to non-float";
}
bool operator<=(const expr &o) const{
if(o.type()==floatt_t)
return this->value <= o.val();
else
throw "comparison <= of float to non-float";
}
bool operator>=(const expr &o) const{
if(o.type()==floatt_t)
return this->value >= o.val();
else
throw "comparison >= of float to non-float";
}

};

class vec : public expr{
public:
floatt x,y,z;
vec(){
this->x=floatt(0.0);
this->y=floatt(0.0);
this->z=floatt(0.0);
}
vec(float x, float y, float z){
this->x=floatt(x);
this->y=floatt(y);
this->z=floatt(z);
}
vec(floatt x, floatt y, floatt z){
this->x=x;
this->y=y;
this->z=z;
}
vec(const expr &e){
*this=dynamic_cast<const vec &> (e);
}
enum type type() const{
return vec_t;
}
const vec &operator=(const expr &e){
*this=(vec)e;
```

```
return *this;
}
const vec &operator +(const vec &v) const {
return newv(x+v.x, y+v.y, z+v.z);
}
const vec &operator -(const vec &v) const {
return newv(x-v.x, y-v.y, z-v.z);
}
const vec &operator *(const floatt &v) const {
return newv(x*v, y*v, z*v);
}
const floatt &operator *(const vec &v) const {
return newf(x*v.x + y*v.y + z*v.z);
}
const vec &operator /(const floatt &v) const {
return newv(x/v, y/v, z/v);
}
const vec &operator -() const {
return newv(-x, -y, -z);
}
const floatt &length() const {
return newf(sqrtf(x*x+y*y+z*z));
}
const vec &unit() const {
floatt l=length();
return newv(x/l, y/l, z/l);
}
};

class list : public expr{
protected:
const expr *data;
const list *next;

virtual const expr *copyExpr(const expr &e) {
if(&e)
if(e.type()==floatt_t) {
return &(newf(e));
}
else if(e.type()==vec_t){
return &(newv(e));
}
else {//assume it's a list
return &(newl(e));
}
else return 0;
}
```

```
public:
enum type type() const{
return list_t;
}
list() {
data=0;
next=0; //empty (null) list
}
list(const list &source) {
if(&source){
if(source.type()==list_t) {
if(source.hasfirst())
data=copyExpr(source.first());
else
data=0;
if(source.hasrest())
next=&newl(source.rest());
else
next=0;
}
else
throw "cannot copy construct a list from a non-list";
}
else{
data=0;
next=0;
}
}
list(const expr &source) {
if(&source){
if(source.type()==list_t) {
if(source.hasfirst())
data=copyExpr(source.first());
else
data=0;
if(source.hasrest())
next=&newl(source.rest());
else
next=0;
}
else
throw "cannot copy construct a list from a non-list";
}
else{
data=0;
next=0;
}
```

```
}
list(const expr &first, const list &rest) {
data=copyExpr(first);
next=&newl(rest);
}
const list &operator=(expr &source){
if(this == &source) return *this;
if(&source){
if(source.type()==list_t) {
if(source.hasfirst())
data=copyExpr(source.first());
else
data=0;
if(source.hasrest())
next=&newl(source.rest());
else
next=0;
}
else
throw "cannot copy construct a list from a non-list";
}
else{
data=0;
next=0;
}
return *this;
}
const expr &first() const{
if(!data) throw "the list is empty";
return *data;
}
const list &rest() const{
if(!next) return *(new list());
return *next;
}
const bool hasrest() const{
return next;
}
const bool hasfirst() const{
return data;
}
/*~list() {
if(data){
std::cerr<<data->type()<<std::endl;
delete data;
}
if(next){
std::cerr<<next->type()<<std::endl;
```

```
delete next;
}
}*/
};

class permlist : public list{
private:
const expr *copyExprP(const expr &e) {
if(&e)
if(e.type()==floatt_t) {
return new floatt(e.val());
}
else if(e.type()==vec_t){
return new vec(e);
}
else {//assume it's a list
return new permlist(e);
}
else return 0;
}
public:
enum type type() const{
return list_t;
}
permlist() {
data=0;
next=0; //empty (null) list
}
permlist(const expr &source) {
if(&source){
if(source.type()==list_t) {
if(data){
delete data;
}
if(next){
delete next;
}
if(source.hasfirst())
data=copyExprP(source.first());
else
data=0;
if(source.hasrest())
next=new permlist(source.rest());
else
next=0;
}
else
throw "cannot copy construct a list from a non-list";
```

```
}
else{
data=0;
next=0;
}

}
permlist(const expr &first, const list &rest) {
data=copyExprP(first);
next=new permlist(rest);
}
const list &operator=(expr &source){
if(this == &source) return *this;
if(&source){
if(source.type()==list_t) {
if(data){
//std::cerr<<"pdd &"<<data<<std::endl;
delete data;
}
if(next){
//std::cerr<<"pdn &"<<next<<std::endl;
delete next;
}
if(source.hasfirst())
data=copyExprP(source.first());
else
data=0;
if(source.hasrest())
next=new permlist(source.rest());
else
next=0;
}
else
throw "cannot copy construct a list from a non-list";
}
else{
data=0;
next=0;
}
return *this;
}
~permlist() {
if(data){

//std::cerr<<"pdd &"<<data<<std::endl;
delete data;
}
if(next){
```

```
//std::cerr<<"pdn &"<<next<<std::endl;
delete next;
}
}

};

#endif /*EXPR_H_*/


source of FunkGL.h
#include "expr.h"
#include "scene.h"
#include "garbage.h"



source of garbage.cpp
#include <vector>
#include "expr.h"

using namespace std;

vector<floatt *> fAlloc;
vector<vec *> vAlloc;
vector<list *> lAlloc;

const floatt &newf(float v) {
floatt *n = new floatt(v);
fAlloc.push_back(n);
return *n;
}

const floatt &newf(const floatt &v) {
return newf((float)v);
}

const floatt &newf(const expr &v) {
return newf((floatt)v);
}

const vec &newv(const floatt &x, const floatt &y, const floatt &z) {
vec *n = new vec(x,y,z);
vAlloc.push_back(n);
return *n;
}

const vec &newv(const vec &v) {
```

```
vec *n = new vec(v);
vAlloc.push_back(n);
return *n;
}

const vec &newv(const expr &v) {
return newv((vec)v);
}

const list &newl(const list &v) {
list *n = new list(v);
lAlloc.push_back(n);
return *n;
}

const list &newl(const expr &v) {
return newl((list)v);
}

void collect() {
for(vector<floatt *>::const_iterator i=fAlloc.begin(); i!=fAlloc.end(); i++)
delete *i;
fAlloc.clear();

for(vector<vec *>::const_iterator i=vAlloc.begin(); i!=vAlloc.end(); i++)
delete *i;
vAlloc.clear();

for(vector<list *>::const_iterator i=lAlloc.begin(); i!=lAlloc.end(); i++)
delete *i;
lAlloc.clear();
}


source of garbage.h
#ifndef GARBAGE_H
#define GARBAGE_H

class list;
class floatt;
class vec;
class expr;

void collect();

const floatt &newf(float v);
const floatt &newf(const floatt &v);
const floatt &newf(const expr &v);
```

```
const vec &newv(const floatt &x, const floatt &y, const floatt &z);
const vec &newv(const vec &v);
const vec &newv(const expr &v);

const list &newl(const list &v);
const list &newl(const expr &v);

#endif


source of meshLoader.cpp
#include <cstdio>
#include <sstream>
#include <vector>

#include "sceneContainers.h"

using namespace std;

typedef vector<float> Fvector;
typedef vector<unsigned int> UIvector;

mData loadMeshData(const char *filename, float normRatio) {
size_t dwBytesRead;
Fvector V;
Fvector N;
Fvector T;
UIvector I;

char  buf[1]={0};
char numbuf[100]={0};
int vcount=0;
int indx=0;
bool bIsNormal=false;
bool bIsVertex=false;
bool bIsVector=false;
bool bIsTexture=false;
float v[3];

bool  bReadFile=true;

FILE * pFile=fopen(filename,"rb");
if(!pFile){
throw "tried to open obj file that does not exist";
}
bool bWasLabel=false;
int ibyte=0;
```

```
bool vlast=true;

while (dwBytesRead=fread(buf,1,1,pFile)) {
if (dwBytesRead==0) {
//ShowThisError(GetLastError());
break;
}
if (buf[0]=='N') {
bIsNormal=true;
bIsVector=false;
bIsTexture=false;
bWasLabel=true;
}
else if (buf[0]=='V') {
bIsVector=true;
bIsNormal=false;
bIsTexture=false;
bWasLabel=true;
}
else if (buf[0]=='T') {
bIsTexture=true;
bIsNormal=false;
bIsVector=false;
bWasLabel=true;
}
else if (buf[0]=='\n') {
if (!bWasLabel) {
sscanf(numbuf, "%f", &v[vcount]);
vcount++;
if (vcount==3) {
if (bIsNormal) {
N.push_back(v[0]*normRatio);
N.push_back(v[1]*normRatio);
N.push_back(v[2]*normRatio);
vlast=false;
}
else if(bIsVector) {
if (vlast) {
float x,y,z;
size_t nsize=N.size();
x=N[nsize-3];
y=N[nsize-2];
z=N[nsize-1];
N.push_back(x);
N.push_back(y);
N.push_back(z);
}
V.push_back(v[0]);
```

```
V.push_back(v[1]);
V.push_back(v[2]);
I.push_back(ibyte);
vlast=true;
ibyte++;
}
vcount=0;
}
else if (vcount==2) {
if (bIsTexture) {
T.push_back(v[0]);
T.push_back(-v[1]);
vcount=0;
}
}
indx=0;
}
else bWasLabel=false;
}
else {
numbuf[indx]=buf[0];
indx++;
}
}
fclose(pFile);
int i=0;
unsigned int * ITmp=new unsigned int[I.size()];
float * VTmp=new float[V.size()];
float * NTmp=new float[N.size()];

for (i=0;i<(int)I.size();i++) {
ITmp[i]=I[i];
}
for (i=0;i<(int)V.size();i++) {
VTmp[i]=V[i];
}
for (i=0;i<(int)N.size();i++) {
NTmp[i]=N[i];
}

float *TTmp=0;
if (T.size()>0) {
TTmp=new float[T.size()];
for (i=0;i<(int)T.size();i++)
TTmp[i]=T[i];
}
return mData(ibyte,VTmp,NTmp,TTmp,ITmp);
}
```

```
source of sceneContainers.h
#ifndef SCENECONTAINERS_H
#define SCENECONTAINERS_H

#ifdef WIN32
#include <windows.h>
#include <GL/gl.h>
#endif
#ifdef LINUX
#include <GL/gl.h>
#endif
#ifdef MACOSX
#include <OpenGL/gl.h>
#endif
#include <string>
#include "expr.h"
using namespace std;



class mData
{
public:
mData(int n, float *V, float *N, float *T, unsigned int *I) {
this->n=n; this->V=V; this->N=N; this->T=T; this->I=I;
}
mData &operator =(const mData &D) {
return *this;
}

void render() const {
if (V==0 || N==0 || I==0)
return;

glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glVertexPointer(3,GL_FLOAT,0,V);
glNormalPointer(GL_FLOAT,0,N);

if (T)
{
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer(2, GL_FLOAT, 0, T);
}
else {
glDisableClientState(GL_TEXTURE_COORD_ARRAY);
```

```
}
glDrawElements(GL_TRIANGLES,n,GL_UNSIGNED_INT,I);
}

private:
float *V, *N, *T;
unsigned int *I, n;
};

//FIXME: Textures not supported yet
//FIXME: handle zero pointers as default value
class material {
public:
//default material
material() {
*this=material("",0,0,0,0);
}

//file is texture; if string is empty then it's a simple material
(i.e. no texture)
material(string file, const vec *iAmbient, const vec *iDiffuse,
const vec *iSpecular, const floatt *iShininess)
: ambient(iAmbient), diffuse(iDiffuse), specular(iSpecular),
shininess(iShininess)
{
if(!file.empty())
loadTex(file);
else
tex=0;
if(!ambient)
ambient=new vec(.5,.5,.5);
if(!diffuse)
diffuse=new vec(.5,.5,.5);
if(!specular)
specular=new vec(.5,.5,.5);
if(!shininess)
shininess=new floatt(1.);
}

void use() const {
if(tex) {
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, tex);
}
else
glDisable(GL_TEXTURE_2D);
float tmp[4]={0.,0.,0.,1.};
tmp[0]=ambient->x; tmp[1]=ambient->y; tmp[2]=ambient->z;
```

```
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, tmp);

tmp[0]=diffuse->x; tmp[1]=diffuse->y; tmp[2]=diffuse->z;
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, tmp);

tmp[0]=specular->x; tmp[1]=specular->y; tmp[2]=specular->z;
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);

glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, *shininess);
}

private:
void loadTex(string file) {
FILE *fp;
unsigned short t2;
unsigned int t4;
int width,height;

//Try opening the file; use "rb" mode to read this *binary* file.
if ((fp = fopen(file.c_str(), "rb")) == 0)
return;

//Read File Header
fread(&t2, 2, 1, fp);

//Check if BMP file
if ((char)t2!='M' && (char)(t2<<8)!='B') return;
fread(&t4, 4, 1, fp);
fread(&t4, 4, 1, fp);
fread(&t4, 4, 1, fp);

/*
* Read Info Header
*/
fread(&t4, 4, 1, fp); //infoheader size
if (t4!=40) return;
fread(&width, 4, 1, fp); //Width
if (width==0) return;
fread(&height, 4, 1, fp); //Height
if (height==0) return;
fread(&t2, 2, 1, fp); //Bitplanes
if (t2 != 1) return;
fread(&t2, 2, 1, fp); //Depth
if (t2!=24) return;
fread(&t4, 4, 1, fp); //Compression type
if (t4!=0) return;
fread(&t4, 4, 1, fp);
fread(&t4, 4, 1, fp);
```

```
fread(&t4, 4, 1, fp);
fread(&t4, 4, 1, fp);
fread(&t4, 4, 1, fp);

//Read Bitmap in inverse order, line per line,
//leave space for Alpha component, and make it RGB
unsigned char *buf= new unsigned char(width*height*3);

int pos=0;
for (int i=height-1; i>=0; i--)
for (int j=0; j<width; j++)
{
fread (buf+pos+2, 1, 1, fp); //Blue
fread (buf+pos+1, 1, 1, fp); //Green
fread (buf+pos+0, 1, 1, fp); //Red
pos+=3;
}
if (feof (fp)) return;

glGenTextures(1,&tex);
glBindTexture(GL_TEXTURE_2D,tex);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width,height, 0, GL_RGB,
GL_BYTE, buf);

delete[] buf;
}

GLuint tex;
const vec *ambient, *diffuse, *specular;
const floatt *shininess;
};

class mesh {
public:
mesh(const mData data_, const material mat_, const vec *position_,
const vec *rotation_, const vec *scale_)
: data(data_), mat(mat_), position(position_), rotation(rotation_),
scale(scale_)
{
if(!position)
this->position=new vec(.0f,.0f,.0f);
if(!rotation)
this->rotation=new vec(.0f,.0f,.0f);
if(!scale)
```

```
this->scale=new vec(1.f,1.f,1.f);
}

mesh(const mesh &M)
: data(M.data),mat(M.mat),position(M.position),rotation(M.rotation),
scale(M.scale)
{
}

mesh &operator = (const mesh &M) {
this->data=M.data;
this->mat=M.mat;
this->position=M.position;
this->rotation=M.rotation;
this->scale=M.scale;
return *this;
}

void render() {
mat.use();
//FIXME: Figure out order, maybe correct rotation
glPushMatrix();
glTranslatef(position->x,position->y,position->z);
//glRotatef(90.f, rotation->x,rotation->y,rotation->z);
glScalef(scale->x,scale->y,scale->z);
data.render();
glPopMatrix();
}

private:
mData data;
material mat;
const vec *position, *rotation, *scale;
};

class light {
public:
light(const vec *iPos, const floatt *iPosW, const vec *iA, const vec *iD,
const vec *iS, const vec *iAttn)
: Pos(iPos), PosW(iPosW), A(iA), D(iD), S(iS), Attn(iAttn)
{
if(!Pos)
Pos=new vec(-1.f,-1.f,-1.f);
if(!PosW)
PosW=new floatt(0.f);
if(!A)
A=new vec(.5f,.5f,.5f);
if(!D)
```

```
D=new vec(.5f,.5f,.5f);
if(!S)
S=new vec(.5f,.5f,.5f);
if(!Attn)
Attn=new vec(1.f,0.f,0.f);
}

void use(GLenum L) {
glEnable(L);
GLfloat v[4] = { 0.f,0.f,0.f,1.f };
v[0]=A->x; v[1]=A->y; v[2]=A->z;
glLightfv(L,GL_AMBIENT,v);
v[0]=D->x; v[1]=D->y; v[2]=D->z;
glLightfv(L,GL_DIFFUSE,v);
v[0]=S->x; v[1]=S->y; v[2]=S->z;
//glLightfv(L,GL_SPECULAR,v);
v[0]=Pos->x; v[1]=Pos->y; v[2]=Pos->z; v[3]=*PosW;
glLightfv(L,GL_POSITION,v);
//glLightf(L, GL_CONSTANT_ATTENUATION, Attn->x);
//glLightf(L, GL_LINEAR_ATTENUATION, Attn->y);
//glLightf(L, GL_QUADRATIC_ATTENUATION, Attn->z);
}

private:
const vec *Pos, *A, *D, *S, *Attn;
const floatt *PosW;
};

#endif //SCENECONTAINERS_H


source of scene.cpp
#ifdef WIN32
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#endif
#ifdef LINUX
#include <sys/time.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#endif
#ifdef MACOSX
#include <sys/time.h>
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
```

```
#include <GLUT/glut.h>
#endif
#define GLUT_DISABLE_ATEXIT_HACK


#include "sceneContainers.h"
#include <vector>
#include "scene.h"
using namespace std;

vector<mData> dataV;
vector<mesh> meshV;
vector<material> matV(1);
vector<light> lights;
vec *camPos,*camCen,*camUp;
void (*update)(void) = 0;
bool mouseRPressed, mouseLPressed;
int mouseREvent, mouseLEvent;
int mouseX, mouseY;
floatt dt;


mData loadMeshData(const char *filename, float normRatio);

/* ERROR REPORTING */
#define printOpenGLError() printOglError(__FILE__, __LINE__)

int printOglError(char *file, int line)
{
    // Returns 1 if an OpenGL error occurred, 0 otherwise.
    GLenum glErr;
    int    retCode = 0;

    glErr = glGetError();
    while (glErr != GL_NO_ERROR)
    {
        printf("glError in file %s @ line %d: %s\n", file, line,
gluErrorString(glErr));
        retCode = 1;
        glErr = glGetError();
    }
    return retCode;
}

dataID fglLoadMeshData(string path) {
dataV.push_back(loadMeshData(path.c_str(),1.));
return (dataID)dataV.size()-1;
}
```

```
meshID fglAddMesh(dataID meshData, matID material, vec *position, vec *rotation,
vec *scale) {
meshV.push_back(mesh(dataV[meshData],matV[material],position,rotation,scale));
return (meshID)meshV.size()-1;
}

matID fglLoadMaterial(string path, vec *ambient, vec *diffuse, vec *specular,
floatt *shininess) {
matV.push_back(material(path,ambient,diffuse,specular,shininess));
return (meshID)matV.size()-1;
}

void fglSetCamera(vec *position, vec *center, vec*up) {
if(!position)
camPos=new vec(0.f,0.f,100.f);
else
camPos=position;
if(!center)
camCen=new vec;
else
camCen=center;
if(!up)
camUp = new vec(floatt(0.f), floatt(1.0f), floatt(0.f));
else
camUp=up;
}

void fglAddLight(const vec *iPos, const floatt *iPosW, const vec *iA,
const vec *iD, const vec *iS, const vec *iAttn) {
lights.push_back(light(iPos,iPosW,iA,iD,iS,iAttn));
}

void reshape(int x,int y);
void keyboard(unsigned char key,int x,int y);
void mouseMove(int x, int y);
void mouseClick(int button, int state,int x, int y);
void cycle();

void fglScene(void (*func)(void), int *argc, char **argv) {
glutInit(argc,argv);
glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100, 100);
glutCreateWindow (argv[0]);

update=func;
glShadeModel(GL_SMOOTH);
```

```
glClearColor(0.0f, 0.0f, 0.0f, 1.f);
glClearDepth(1.0f);
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LEQUAL);
glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
glEnable(GL_LIGHTING);
glEnable(GL_CULL_FACE);
glShadeModel(GL_SMOOTH);

glutDisplayFunc(cycle);
glutIdleFunc(cycle);
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutMotionFunc(mouseMove);
glutMouseFunc(mouseClick);
glutPassiveMotionFunc(mouseMove);
glutMainLoop();
}

void reshape(int w, int h) {
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if(!h)
h=1;
gluPerspective(80,(float)w/(float)h,1.,1000.);
glMatrixMode(GL_MODELVIEW);
}

bool fullScreen=false;
void keyboard(unsigned char key, int x, int y)
{
if(key==27)
exit(0);
if(key=='f' || key=='F')
if(fullScreen) {
fullScreen=false;
glutReshapeWindow(500,500);
}
else {
fullScreen=true;
glutFullScreen();
}
}

void mouseMove(int x, int y) {
mouseX=x;
mouseY=y;
```

```
}

void mouseClick(int button, int state,int x, int y) {
bool *pressed=0;
int *event=0;
if(button==GLUT_LEFT_BUTTON) {
pressed=&mouseLPressed;
event=&mouseLEvent;
}
if(button==GLUT_RIGHT_BUTTON) {
pressed=&mouseRPressed;
event=&mouseREvent;
}
if(state==GLUT_DOWN) {
*pressed=true;
*event=BUTTON_DOWN;
}
if(state==GLUT_UP) {
*pressed=false;
*event=BUTTON_UP;
}
mouseX=x;
mouseY=y;
}

long int getTicks() {

#ifdef WIN32
return GetTickCount();
#endif
#ifdef LINUX
struct timeval tp;
gettimeofday(&tp,NULL);
return tp.tv_sec*1000+tp.tv_usec/1000;
#endif
#ifdef MACOSX
struct timeval tp;
gettimeofday(&tp,NULL);
return tp.tv_sec*1000+tp.tv_usec/1000;
#endif
}

int prevTick,curTick=getTicks();
#include <cstdio>
void cycle() {
vector<mesh>::iterator O;
vector<light>::iterator L;
int i;
```

```
//printf("%i\n",keyPressed['a']);
//memset(keyPressed,0,sizeof(bool)*256);
//UPDATE
update();
prevTick=curTick;
curTick=getTicks();
dt=(curTick-prevTick)/1000.f;

//RENDER
//camera
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
gluLookAt(camPos->x,camPos->y,camPos->z,camCen->x,camCen->y,camCen->z,
camUp->x,camUp->y,camUp->z);

//lights
glEnable(GL_LIGHTING);
for(i=0; i<8; i++)
glDisable(GL_LIGHT0+i);
for(L=lights.begin(),i=0; L!=lights.end(); L++, i++)
L->use(GL_LIGHT0+i);

//action!
for (O=meshV.begin(); O != meshV.end(); O++)
O->render();

glutSwapBuffers();
printOpenGLError();
}


source of scene.h
#ifndef SCENE_H
#define SCENE_H

#include "expr.h"

#include <string>
using namespace std;

/*
 * The IDs below work like pointers.
 * To display a static mesh first you have to load it
 * using fglLoadMeshData() and then pass the returned
 * handle to fglAddMesh (and similarly for materials).
 * meshIDs are useful only for dynamic meshes, where we
 * may want to clone a mesh or remove it from the scene.
```

```
 */

#define BUTTON_DOWN 2
#define BUTTON_UP 1

extern bool mouseRPressed, mouseLPressed;
extern int mouseREvent, mouseLEvent;
extern int mouseX, mouseY;
extern floatt dt;

typedef unsigned int dataID;
typedef unsigned int meshID;
typedef unsigned int matID;

void fglScene(void (*update)(void), int *argc, char **argv);
dataID fglLoadMeshData(string path);

meshID fglAddMesh(dataID meshData, matID mat, vec *position, vec *rotation,
vec *scale);
meshID fglCloneMesh(meshID mesh);
void fglRemoveMesh(meshID mesh);

matID fglLoadMaterial(string path, vec *ambient, vec *diffuse, vec *specular,
floatt *shininess);
void fglAddLight(const vec *iPos, const floatt *iPosW, const vec *iA,
const vec *iD, const vec *iS, const vec *iAttn);

void fglSetCamera(vec *position, vec *center, vec *up);

#endif


source of FunkGLC.java
import java.io.DataInputStream;
import java.io.FileInputStream;

import antlr.collections.AST;
import edu.columbia.cs.vicecity.ir.Program;
public class FunkGLC {
public static void main(String[] args) {
try {
FileInputStream fi = new FileInputStream(args[0]);
DataInputStream input=new DataInputStream(fi);
L lexer=new L(new DataInputStream(input));
P parser=new P(lexer);
parser.program();
AST ast= parser.getAST();
        W walker = new W();
```

```
        Program p = walker.programdecl(ast);
        p = walker.program(ast);
        System.out.println("\n"+p);
} catch(Exception e) {
System.err.println("there was a problem compiling your file: "+e);
System.exit(1);
    }
System.exit(0);
}
}
```

source of TestFrame.java

```
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.ByteArrayInputStream;
import java.io.DataInputStream;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import edu.columbia.cs.vicecity.ir.Program;

import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;
import javax.swing.JTextArea;


public class TestFrame extends JFrame {
/**
 *
 */
private static final long serialVersionUID = 1L;
private JTextArea input;
private JButton submit, quit;
private JTextArea output;
private JScrollPane spi;

public TestFrame(){
super();
init();
}
private void init(){
input = new JTextArea(10,50);
submit = new JButton("Submit");
quit = new JButton("Quit");
```

```
output = new JTextArea(10,50);
spi = new JScrollPane(input);
this.getContentPane().setLayout(new FlowLayout());
this.getContentPane().add(submit);
this.getContentPane().add(quit);
this.getContentPane().add(spi);
//this.getContentPane().add(spo);
//this.getContentPane().show();

submit.addActionListener(new ButtonListener());
quit.addActionListener(new ButtonListener());
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.pack();
//System.out.println("done init");
}
private class ButtonListener implements ActionListener {
  ButtonListener() {
  }

  public void actionPerformed(ActionEvent e) {
  if(e.getSource()==quit)System.exit(0);
L lexer = new L(new DataInputStream(new ByteArrayInputStream(
input.getText().getBytes())));
P parser = new P(lexer);
try {
            parser.program();
            AST ast= parser.getAST();
            /*ASTFrame frame = new ASTFrame("ASTBrowser",ast);
            frame.setSize(new Dimension(300,600));
            frame.setVisible(true);*/
            W walker = new W();
            Program p = walker.programdecl(ast);
            p = walker.program(ast);
            System.out.println("\n\n\n\n"+p);

 } catch(Error ex) {
            System.err.println("\n\nexception: "+ex);
 }
 catch(Exception ex) {
            System.err.println("\n\nexception: "+ex);
 }
System.exit(0);
  }
}

}
```

```
source of Makefile
CC=g++
DEBUG= -g
FLAGSL=-DLINUX  -lGL -lGLU -lglut
FLAGSW=-DWIN32 -lopengl32 -lglu32
FLAGSM=-DMACOSX -framework OpenGL -framework GLUT
FLAGS=SETFLAGS

COMPILE=$(CC) $(DEBUG)

win:libsw $(FILE)
$(COMPILE) $(FLAGSW) -o $(OUTPUT) $(FILE) scene.o meshLoader.o garbage.o

linux:libsl $(FILE)
$(COMPILE) $(FLAGSL) -o $(OUTPUT) $(FILE) scene.o meshLoader.o garbage.o

mac:libsm $(FILE)
$(COMPILE) $(FLAGSM) -o $(OUTPUT) $(FILE) scene.o meshLoader.o garbage.o

libsl: FunkGL.h expr.h scene.h garbage.h sceneContainers.h scene.cpp
meshLoader.cpp garbage.cpp
$(COMPILE) $(FLAGSL) -c scene.cpp
$(COMPILE) $(FLAGSL) -c meshLoader.cpp
$(COMPILE) $(FLAGSL) -c garbage.cpp
libsw: FunkGL.h expr.h scene.h garbage.h sceneContainers.h scene.cpp
meshLoader.cpp garbage.cpp
$(COMPILE) $(FLAGSW) -c scene.cpp
$(COMPILE) $(FLAGSW) -c meshLoader.cpp
$(COMPILE) $(FLAGSW) -c garbage.cpp
libsm: FunkGL.h expr.h scene.h garbage.h sceneContainers.h scene.cpp
meshLoader.cpp garbage.cpp
$(COMPILE) $(FLAGSM) -c scene.cpp
$(COMPILE) $(FLAGSM) -c meshLoader.cpp
$(COMPILE) $(FLAGSM) -c garbage.cpp

clean:
rm -f main-test main-fgl scene.o meshLoader.o garbage.o


source of funkglc
#!/bin/sh
# funkglc compiler
# arguments: target=[win|mac|linux] fglinputfile executableoutputfile
# calls antlr/java compiler, then builds cpp with make
# windows platform assumes MinGW or equivalent g++ compiler
# antlr.jar must be in this directory
SPATH=${0%/*}/.
if [ "$SPATH" != "./." ]; then
```

```
    echo "RUN FROM SAME DIR AS SCRIPT"
    exit 1
fi
ANTLR_HOME=${SPATH}/antlr.jar
TARGET=$1
FILEFGL=$2
export FILE="temp.cpp"
export OUTPUT=$3

java -cp $SPATH:$ANTLR_HOME FunkGLC ${FILEFGL} >${FILE}

if [ "$?" -ne "0" ]; then
    echo "cannot continue. compilation failed."
    exit 1
fi

make $TARGET
make clean
```