SIGL
A Drawing Language

Phong Pham
Abelardo Gutierrez
Alketa Aliaj

May 7, 2007
Outline

• Introduction
  – What is SIGL?
  – Feature highlighting

• SIGL anatomy
  – Scanning and parsing
  – Overall design
  – Evaluation

• Testing
What is SIGL?

• Simple Image Generation Language: simple language for drawing 2D images

• Motivation
  – VRML language: standard 3D model specification
  – Lack of controlling flow
  – Repetition required
  – Only suitable for machine generation

• Introduce more control in form of C-like syntax
**Drawing in SIGL**

- Draw 3 vertically aligned boxes

```plaintext
for (i = 0; i < 3; ++i) {
  :translate(0, i * 2): {
    rectangle(0, 0, 1, 1);
  }
}
```
Features

• Drawing features
  – OpenGL-like drawing mechanism
  – Support commonly used primitives: lines, circle, ellipse, polygons
  – Transformations: translation, rotation, scale

• Language features
  – C-like language
  – Support nearly all C constructions (except for switch)
  – Data types: int, double, boolean, associative array
  – Dynamic type system, no type decoration
  – Static scoping
  – Applicative evaluation order
Grammar

• C-like operators / comments / ID
  – Three types of operational tokens: Integer, real number, logical

• C-like arithmetic precedent etc.
  – Mult, Div, and Mod precedence over addition and subtraction

• C-like function declaration and flow control statements
  – for, if, while, break, continue, return, empty statement (;)
**Parser - Walker**

- Build AST tree in 2 steps
  - Build default ANTLR tree (Parser)
    ```
    while_stmt : "while"~ LPAREN! expr RPAREN! stmt ;
    ```
  - Transform default AST tree into object tree (Walker)
    ```
    #("while" e1=expr s1=stmt { s = new While(e1, s1); } )
    ```

- Store location of the expressions for debugging purposes.
  ```
  #(LOR a=expr b=expr { e = new LogicalOperation("||", a, b); e.setLine(#LOR.getLine()); e.setColumn(#LOR.getColumn()); } )
  ```

- The object tree makes Walker simpler, allows language flexibility
Class Hierarchy

Stmt  
  If
  For
  While
  Block
  Break
  Continue
  Return
  Function

Expr
  BoolConstant
  IntConstant
  RealConstant
  BinaryOperation
  UnaryOperation
  RelationalOperation
  ArithmeticOperation
  LogicalOperation

Value
  IntValue
  RealValue
  ArrayValue
  BoolValue
  FunctionValue
  ThunkValue
Type checking

- Expressions are evaluated into Values
- Type-checking is done using Values
- Example: “%” operator
  - Evaluate left hand side to val1
  - Evaluate right hand side to val2
  - Check that both val1 and val2 are both of type IntValue
Environment

- Stored current states of the program

- Components:
  - Symbol table
  - Drawing canvas (this includes colors, etc.)
  - Current transformation
  - Break, continue, return flag
Symbol table

- Desired behavior

```plaintext
x = 1; // x is bound to 1
{
    x = 5; // x is bound to 5
    y = 6; // x is bound to 5, y is bound to 6
}
// x is bound to 5, y is unbound
```

Clone

Extend

Extend destructively
Functions

• Functions are first-order entities in SIGL
  – Can be passed as arguments to other functions

• Function declarations are evaluated into FunctionValues

• FunctionValue: tuple of 2 values \( fv = (f, env) \)
  – The function \( f \) itself
  – A cloned environment \( env \) of the environment at which the function is declared

• Handle recursive function: bind destructively \( f \) to \( fv \) in \( env \)
Function call evaluation

• Retrieve FunctionValue associated with the given name

• Execute the function (stored in FunctionValue)
  – Static scoping: using the environment stored in FunctionValue
  – Dynamic scoping: using the current environment

• Evaluation order
  – Applicative order: evaluate each argument expressions and pass to the function
  – Normal order: create a ThunkValue
    ∗ ThunkValue: tuple (expr, env)
Modified access in symbol table

- ThunkValue should only be evaluated once

- Access is called:
  - Get the value
  - If the value is ThunkValue
    - Evaluate \( \text{expr} \) in ThunkValue using \( \text{env} \) in ThunkValue
    - Replace ThunkValue in symbol table with new value
  - return value
Built-in functions

• Don’t need to change lexer/parser

• Implement as FunctionValue

• Automatically loaded
Testing

• Some unit testing using JUnit
• Peer-review
• Big-bang testing
Thank you

Questions?