DDRL Reference Manual

Columbia University

COMS W4115 Programming Languages and Translators

Spring 2007

Prof. Stephen Edwards
TA: Abhilash Itharaju

Team Members:
Zhiyang Cao zc2109@columbia.edu
Yitao Wang yw2226@columbia.edu
Alex Ling Lee al2537@columbia.edu
Yan Zhang yz2197@columbia.edu
Kyung Kwan Kim kk2367@columbia.edu

1. Introduction

DDRL is a computer language based on the C language [1]. The DDRL language uses basic C language data type and expression and control structure, but it is much easy and simple than C language, because the DDRL is focused on generating the graphics shapes and control sequence for dynamic presentation of the data. Only basic flow controls and simple operators is borrowed from C language.

As we mentioned above, our language consists of three parts: basic flow control and expression, graphics defines and control, animation sequence define and control, we will specify all these following.

2. Lexical Conventions

2.1 Comments

Every line of comments should begin with "//" and end at the end of the line.

2.2 Identifiers

An identifier must start with a lower case letter, and only include lower case letter and a number. A number, if contained by an identifier, can only be placed at the end of it.

2.3 Keywords

The following words are reserved by the system as keywords and should not be used as identifiers. Note that all key words in DDRL starts with capital letters

Basic:

If Else While For Foreach Switch Case
Return Break Continue
Int Float Dataset String Array
Import

Graphics Define:

Color Pointsize Linewidth
Point EndPoint Line EndLine Polygon EndPolygon
Box Endbox Cylinder EndCylinder Slice EndSlice
Plane EndPlane Axis EndAxis AxisMarker EndAxisMarker
Text EndText DisplayModel Display

Time tag:

Loop Clear

Graphics Control:

Show Unshow Position Rotation Scaling

2.4 Types and Constants

3 types of data type are supported: Int, Float, String.
Int: 16 bit integers;
Float: 32 bit float point data type;
Array: a sequence of data objects;
String: a sequence of characters;
Dataset: A buffered data manipulator, used to manipulate data from a file.

Accordingly, available constants include: Int constants, Float constants, String constants.

Note that programmers can use the attributes and member function of Dataset type to manipulate the content of a file. It can retrieve the column size, row size, column name, row name and the data.

Example:

```
Dataset mydata = OpenFile(“./example.txt”);
Int size = mydata.columnsize;
String rowname = mydata.rowname[1];
Array Int data[size] = mydata.row[1];
```

3. Expressions

DDRL expressions are list by precedence form high to low:

3.1.1 Primary expressions

Primary expressions include identifiers, constants, function calls, contained within group left to right.

3.1.2 identifier

Identifier can be Int, Float, String, Dataset, DisplayModel;

3.1.3 Constant

Constants in DDRL include Integer constant, Float constant, and String constant;

3.1.4 ***(expression)***

The parenthesis is used to construct the precedence;

3.1.5 ***Primary-expression <expression-list>***

Such expression is used to clarify the point on time axis;

3.1.6 ***Primary-expression (expression-list)***

This is used to describe function call, which is primary expression. As a part of function call, parenthesis is required. Since DDRL doesn’t support pointer type, all arguments are passed by value. Recursively function calls are supported.

3.2 Operators

3.2.1 Unary Operator

-expression

The type of the operand here must be Int or Float. Operator – doesn’t change the type of the expression, it only gives the negative value of the original operand. It associates from right to left.

3.3 Additive Operators

expression +expression,
expression – expression

3.4 Multiplicative Operators

*expression * expression*
expression / expression

3.5 Exponential Operator

expression ^ expression

3.6 Relational Operators

expression < expression
expression > expression
expression <= expression
expression >= expression

3.7 Equality Operators

expression == expression
expression != expression

3.6 Logical Operators

expression && expression
expression || expression

3.8 Assignment operator

expression = expression

3.9 Other Operators

expression , expression

Comma is used to separate 2 or more expressions;

constant int. constant.int

"." Is used to construct float numbers;

expression;
";" is used to terminate a statement;

{expression;}
"{",")" are used to group expressions.

4. Delearation

4.1 Type specifiers

The type specifiers are:

Type-specifier:

Int
Float
String
Dataset
DisplayModel

4.2 Declarators

declarator-list:
declarator
declarator declarator-list

declarator:
identifier

declarator declarator-list:
identifier

4.3 Declaration of struct type

Struct identifier { typedecllist}

To standardize declaration, DDRL only support one way of declaring struct.

4.4 Declaration of Array type

Array type-specifier [size]

4.5 Display Model Dclaration

DisplayModel identifier (parameter-list)

This will be specified in §8

4.4 Function Defination

func definition:
typespecifier identifier(parameter-list)
{
vardeclaration-list
basic-statement-list
graphics-control-statement-list
}

parameter-list:
typespecifier identifier
typespecifier identifier parameter-list

vardeclaration-list:
vardeclaration
vardeclaration vardeclaration-list

basic-statement-list:
basic-statement
basic-statement statement-list

graphic-control-statement-list
graphic-control-statement
graphic-control-statement graphic-control-statement-list

Note that functions should be defined before declared.

5. Basic Statements

Every statement should be ended with a semicolon. Statement is executed in sequence.

5.1 Expression statement

Most statements are expression statements, which have the form

expression ;

Usually expression statements are assignments or function calls.

5.2 Compound statement

compoundstatement:

{ statementlist}

basic-statementlist:
basic-statement
basic-statement basic-statementlist

5.3 Conditional statement

If (expression) statement
If (expression) statement Else statement

5.4 While statement

While (expression) statement

5.5 For statement

F*or (expression; expression; expression) statement*

5.6 Foreach statement

Foreach (identifier In identifier) statement

5.7 Switch statement

Switch (expression) statement

5.8 Break statement

Break;

5.9 Continue statement

Continue;

5.10 Return statement

Return (expression);

6. Graphics Define Statement

6.1 Vertex Statement

Specify a 3D coordinate for a vertex, which is used to define a vertex in a 3D model. It has the form

Vertex-statement:
Vex expression, expression, expression;

Normally, the expression would be identifier, constant and other simple expression; There expressions are the 3D coordinate of the vertex in form of *x, y, z*;

6.2 Compound Graphics Statement

The compound graphics statement consists of two parts: one is the basic statement; another is the vertex-statement. It has following form.

Compound-graphics-statement:

Basic-statement
Vertex-statement-list
Basic-statement Compound-graphics-statement
Vertex-statement-list Compound-graphic-statement

Vertex-statement-list:

Vertex-statement
Vertex-statement vertex-statement-list

6.3 Color Statement

Specify a color for the following 3D model; by default, the system color is black. When a color statement specifies a kind of color for system, it will remain this color for following any 3D model unless another color statement specifies a different color.

The Statement

Color-statement:

Color expression, expression, expression, expression;

Four expressions are the color exponents of *red, green, blue, alpha*; all values of color component should in the range of 0 - 255.

6.4 Point Size Statement

Define the size of the point displayed in 3D. It has the form

Point-size-statement:

Pointsize expression;

The point size should be an integer which at least is 1.

6.5 Line Width Statement

This statement defines the width of the line displayed in 3D. It has the form

Line-width-statement:

Linewidth expression;

The line width should be an integer which at least is 1.

6.6 Point Statement

Point statement defines a serial of point that in the 3D. This statement should also define at least one point by using *vertex-statement*.

Point-statement:

Point Compound-graphics-statement EndPoint;

6.7 Line Statement

Line Statement defines a single line connected all the vertexes specified in the statement. This statement should define at least two points by vertex-statement. And the line will connect the point in order.

Line-statement:

Line Compound-graphics-statement EndLine;

6.8 Polygon statement

This statement defines the polygon in 3D by specifying the vertex of the polygon. The polygon should have at least 3 vertexes and all the vertexes should convex, otherwise the polygon will not be showed.

Polygon-statement:

Polygon Compound-graphics-statement EndPolygon;

6.9 Box statement

Box statement defines a box in 3D by specifying the *width, length* and *height*. The box normally is placed at the point defined. And the width is the value on *x-axis*, length is the value on *z-axis,* height will be the value on *y-axis*;

Box-statement:

Box Compound-graphics-statement expression, expression, expression; EndBox;

6.10 Cylinder statement

Cylinder statement defines a cylinder in 3D by specifying the *radius and height*. The cylinder will be placed at the point defined. And the height will be the value on *y-axis*.

Cylinder-statement:

Cylinder Compound-graphics-statement expression, expression; EndCylinder;

6.11 Slice statement

Slice statement will defines parts of the pie model in 3D. It should define the radius, height, startAngle and angle of the slice. And the height is the value on y-axis. And the startAngle on the positive direction of x-axis is 0.

Slice-statement:

Slice Compound-graphics-statement expression, expression, expression; EndSlice;

The four expressions are radius, height, startAngle and Angle in order.

6.12 Plane statement

This statement defines a plane in 3D by specifying its leftmost vertex and rightmost vertex.

Plane-statement:

Plane Compound-graphics-statement EndPlane;

6.13 Axis statement

This defines and displays the axis in 3D. Only three axes X, Y, Z can be defined.

Axis-statement:

Axis X expression, expression, expression; EndAxis;
Axis Y expression, expression, expression; EndAxis;
Axis Z expression, expression, expression; EndAxis;

The three expressions are the displayed *title* of the axis, the *length* of the axis and the *interval* of the axis for marker.

The title is string type;
The length is integer type;
The interval is integer type;

6.14 AxisMarker statement

AxisMarker statement defines the title for the interval marker on the axis.

AxisMarker-statement:

AxisMarker Marker-statement-list EndAxisMarker;

Marker-statement-list:

Marker-statement
Marker-statement Marker-statement-list

Marker-statement:

expression, expression;

The two expressions are the index of the interval marker and the title of this marker. The index should be integer and the title should be string.

6.15 text statement

The text statement defines the text displayed in both 3D and 2D. The text should be a string. It has the form

Text-statement:
Text expression vertex-statement

If the vertex-statement third expression equals to zero, then the text will be showed in 2D, and the first two values will be normalize to the screen size.

6.16 graphics define statement

For following specification, we state this statement which is one of the most important parts in our language. The graphics define statement is used to define the 3D elementary models that will be displayed.

Graphics-define-statement:
Color-statement
Point-size-statement
Line-width-statement
Point-statement
Line-statement
Polygon-statement
Box-statement
Cylinder-statement
Slice-statement
Plane-statement
Axis-statement
AxisMarker-statement
Text-statement

Graphics-define-statement-list:
Graphics-define-statement
Graphics-define-statement Graphics-define-statement-list

7. Time Tag Statement

The time tag statement controls the time slice of the animation of special graphics statement. The time tag statement specifies the start time and the duration of the animation. The time tag statement can only control (precedence of) the graphics statement, like graphics define statement and graphics control statement.

7.1 Time tag statement

Time-tag-statement:

```
<starttime, duration>:
<starttime, duration>:Loop
<starttime, duration>:Clear
```

The *starttime* controls the start time of the following graphics animation. The *duration* is the length of this animation.

Keyword *Loop* means redisplay the following animation after it finishes.

Keyword *Clear* means clear the following graphics elements after it finishes.

The *starttime* and *duration* are integer type, and should be positive.

Time tag statement only effects the graphics statement on the same line. And graphics statement includes *Graphics-define-statement* and *Graphics-control-statement.*

We will specify the *Graphics-control-statement* at §9.

7.2 time wildcard

For easy define the start time and duration of animation, the language provides two wildcard for time tag statement: '$' and '#'.

'$': the time that last animation ends. It can only used for starttime in time tag statement.

'#': the duration of the graphics statement animation. The *Graphics-define-statement* default length is 1 second. And duration of the *Graphics-control-statement* is the length of its controlled Display Model. Looped animation will only count once for its duration.

Example: <$, #>:*Loop*

The scope of the wildcard is only in the current display model.

8. Display Model Definition

In our language, the major display model is Display Model. The model is defines by *type-decl-list* and *Graphics-define-statement.* The *type-decl-list* defines the data member of the Model. This should be at the top of the model. The *Graphics-define-statement* will define the graphics and animation sequence of the model.

The Display Model is the basic element of program. It can only be controlled by graphics control statement.

Display-model-definition:

```
DisplayModel identifier( parameter-list )
{
    type-decl-list
    $<DISPLAY
```

```
                    Display-model-def
                $DISPLAY>
            }
```

parameter-list:

```
typespecifier identifier
typespecifier identifier parameter-list
```

Display-model-def:

```
Basic-statement
Time-tag-statement Graphics-define-statement
Basic-statement Display-model-def
Time-tag-statement Graphics-define-statement Display-model-def
```

9. Graphics Control Statement

The graphics control statement control the display or animation of the display model. It can only control the display model. Control includes show the model; unshow the model, position of the model, rotation of the model and scaling of the model.

9.1 show model statement

This statement will show the display model on the screen according to its position setting. By default the model position will be at the (0, 0, 0);

Show-statement:

```
Time-tag-statement Show DisplayModel;
```

9.2 Unshow model statement

This statement will hide the display model on the screen.

Unshow-statement:

```
Time-tag-statement Unshow DisplayModel;
```

9.3 Position model statement

Set the position of the model. x,y,z are the position.

Position-statement:

```
Time-tag-statement Position DisplayModel x,y,z;
```

9.4 Rotation model statement

Set the rotation of the model. x,y,z are the rotation angle.

Rotation-statement:

Time-tag-statement Rotation DisplayModel x,y,z;

9.5 Scaling model statement

Set the scaling of the model.

Scaling-statement:

Time-tag-statement Scaling DisplayModel;

9.6 Graphics Control Statement

All statements above compose the graphics control statement.

Graphics-control-statement:

Show-statement
Unshow-statement
Position-statement
Rotation-statement
Scaling-statement

10. Scope rules

DDRL supports the declaration of global variables.
For local variables which are declared inside specific functions, they belong only to those functions and models.

In Compound statement, the local variables are only effective between '{' and '}';

11. Compiler control lines

Keyword "Import" is used to control the compiler so that the compiler knows where to find modules or functions.

Example: Import "./chartmodel.drl"

10. Example

```
//Example
//draw a column chart according to the original data

//define the column display model
DisplayModel columnchart(int size)
{
    Array float data[size];           //define the data array
    int r = 255, g=0, b=0, a = 255;    //the color for our selection
    float threshold = 50.0;             //threshold

    $<Display
        int i;
        for(i=0;i<size;i++)             //loop the array for all the data
        {
            if(data[i]>threshold)       //if the data larger than threshold
            {
                Color r,g,b,a;          //use our defined color
                <$,#>:   Box            //draw the box, height as value
                        Vertex 1.0+i*2,0.0,0.0;//start at current time of this model
                        2.0,2.0,data[i];            //duration is default time for Box
                        EndBox;
            }
            else
            {
                Color 255,255,255,100;
                <$,#>:  Box
                        Vertex 1.0+i*2,0.0,0.0;
                        2.0,2.0,data[i];
                        EndBox;
            }
        }
    $Display>
}


//main function that display the model
int main()
{
    Dataset mydata = OpenFile("./datafile.txt");    //open data
    int size = mydata.columnsize;                   //get column size
    DisplayModel    columnchart    mychart(size);           //define model
    mychart.data = mydata.row[1];                           //get data
    <10,20>:        Show    mychart;                //show model in 2s start 1
    <$,10>:         Postion mychart 10.0,10.0,10.0;     //position model
    <$,#>:          Rotation 0.0,90.0,0.0;              //rotation model
    <$,#>:          Unshow mychart;                     //hide model

}
```

REFERENCES

1. *Dennis M. Ritchie* ‘‘C Reference Manual’’ *Bell Telephone Laboratories* 1972.