Project Proposal

Dynamic Data Represent Language (DDRL)

Columbia University
COMS W4115 Programming Languages and Translators

Spring 2007
Prof. Stephen Edwards

Team Members:

Alex Ling Lee	**al2537@columbia.edu**
Yan Zhang	**yz2197@columbia.edu**
Yitao Wang	**yw2226@columbia.edu**
Zhiyang Cao	**zc2109@columbia.edu**

Part I

Overview:

The objective of our project is to develop a simple but efficient and powerful language for data representation, which can be used to display, manipulation and analysis the data in 3D graphical objects.

Our language can be easily mastered by the beginners who don't have much experience in 3D graphics programming, as well as powerful flexibilities for advanced user to create complex and sophisticate data presentation.

The programmer of our language not only can create different 3D models to represent raw data separately or in combination, but also define the data control and animation sequence to facilitate the presentation.

Introduction:

The Dynamic Data Representation Language is based on OpenGL graphics library. It is used to create the specific 3D graphics models according to original raw data, and control the model and define the sequence of animation.

The language designed to be similar to nature English language, so that is easy to use. The compiler provide the library to render 3D model, graphics effects and animation, the programmer only need to organize the model and control the sequence. The programmers don't have to take care of complicate 3D graphics programming.

The language provides a large flexibility to programmer, who can use it to create complex model and explicit scenario animation.

The programming consists of two parts.
First the programmer defines and creates the models based on raw data. The language has a few of primitive graphics model data type "PIE, COLUMN, LINE, AREA, RADAR". All these data type has some of properties. Programmers can create the simple model using the raw data, they also can create the combination models by organize the simple models.
Second, Programmers control the model, change their properties and define the display and animation sequence to generate clear and vivid dynamic data presentation.

Following is some basic feature of our Language.
The language can generate dynamic, visible report, easily and efficiently.

- Provide a better way to process formed data
- Various graphics models to represent data;
- Support model combination and hierarchy.
- Generate dynamic and interactive chart
- Merging of charts, which allows the new charts to be created using existing charts as components;
- Select and highlight the model, regroup the models.
- The concept "macro" can be used here to record datasets and their activities of generated components on the chart;
- A "console" mode will be supported. This allows user to interactive with chart components and facilitates debugging.

Part II Language Features

Basic Data Types

"INT", "FLOAT" and "STRING" are primary types to store data.
"DATASET" is used to read in data from formatted files.
"PIE","LINE", "COLUMN", "AREA" are built-in chart 3D objects. By creating instances for these chart objects, the models that are going to be drawn will be pre-defined. User can invoke the functions and use attributes of the objects.
“ARRAY” will be supported to store and manipulated large amount of data of the in array.

Key Words

Type I: Module Construction.

MODEL
ENDMODEL
ADD ... INTO
REMOVE … FROM
POSITION
ROTATION
SCALE

Type II: Module Manipulation

COMPARE
SPLIT
WAIT
ROTATE
DISPLAY
REMOVE
CLEAR

Arithmetic and Comparative Operators:

Plus:"+"
Subtract:"-"
Multiply:"*"
Divide:"/"
Exponential: "^",for example, x^5
Self-plus:"++"
Self-subtraction:"--" (note, to avoid unnecessary confusion, self-plus and self-subtraction can only be used separately as a single expression.)
Assignment:"="
Equal:"=="
Large than:">"
Less than:"<"
Not equal:"!="
No less than:">="
No larger than:"<="
Scope Operator:"{","}" and "(",")"
Comment: "##"

Control Statement

"if-else" condition:

```
IF(/*condition*/)
{
##expression
}
ELSE
{
##expression
}
```

"for" loop:

```
FOR 1 TO 10
{
##expression
}
```

"while" loop:

```
WHILE (i<=0)
{
##expression
}
```

"foreach" loop (used to walk through the objects in array):

```
FOREACH x IN array0
{
##expression
}
```

Part III Simple Examples

Formatted file (.txt)

form1.txt

```
w4115   w4118   w4187
A       4.0     3.0     3.3
B       3.6     3.3     3.3
C       3.8     3.3     2.7
```

form2.txt

```
w4115   w4118   w4187
Spring  30      40      50
Fall    40      50      40
```

Sample Code 1: Generate 3 Pie charts and display all them on the screen.

```
##Sample Code 1 starts
```

```
DATASET myDataSet=DATAREAD("form1.txt");

MODEL M1;
      PIE myPie1=myDataSet[1][1].value, myDataSet[2][1].value, myDataSet[3][1];
ENDMODEL M1;   ##End of model initialization

DISPLAY M1.myPie1;
##Draw the first pie chart

WAIT;
## Sample Code 1 ends
```

Chart 1

Sample Code 2: Generate 2 Column charts and merge them into a new one

```
## Sample Code 2 starts

DATASET myDataSet=DATAREAD("form2.txt");
##Read in data from formatted file

MODEL M2
      INT row = myDataSet.row;                 ##get the number of rows in dataset
      INT colum = myDataSet.col;               ##get the number of columns in dataset
      ARRAY COLUMN col[row*colum];             ##an array of COLUMN with row*col items

      INT colIndex = 1;
      INT rowIndex = 1;
      FOR 1 to row
      {
            FOR 1 to colum
            {
                  col[rowIndex*colum + colIndex] = myDataSet[rowIndex][colIndex].value;
                  colIndex++;
            }
            rowIndex++;
      }

      ADD myColumn1, myColumn2 INTO myColumn3;
      ##Merge myColumn1 and myColumn2 into a new chart object : myColumn3
ENDMODEL M2
```

```
SPLIT M2.myColumn3;
##Change the display mode to SPLIT

FOREACH COLUMN columnx in col
{
        DISPLAY columnx;                    ##display all the column using loop.
}
##Output to screen

WAIT 30;
## Hold the output onto the screen

CLEAR;
## Clear screen.
##Sample Code 2 ends
```

Chart 2