COMS W4115 - Project White Paper
ELMO Loves Manipulating Objects (ELMO)

Jeffrey Cua, Stephen Lee, Erik Peterson, and John Waugh

(jmc2108,sl2285,edp2002,jrw2005)@columbia.edu

September 28, 2004

1 Introduction

Many programmers invest substantial effort into expressing three dimensional objects using API's like OpenGL in C/C++, Java3D, and Direct3D also in C/C++. Languages like C/C++ and Java are Swiss Army knives, which work well for a wide variety of applications, but can still become unwieldily.

OpenGL's enormously powerful, but it can become quite cumbersome for something as a simple as a rotation. The objective of ELMO is not to augment the functionality offered by OpenGL, but provide an additional way to express the same set of operations and primitives that OpenGL grafts on to C/C++. Likewise, Java3D, which may be easier to use than OpenGL, suffers from the same fundamental flaw; it grafted on to an existing general purpose language.

ELMO abstracts many of the details related to the matrix manipulation that OpenGL or other 3D API's reveal to the programmer. When a programmer wants to do a simple rotation, the programmer might call glRotate3f(), a function in OpenGL, which initializes a 4x4 matrix that is placed on one of the various transformation stacks that OpenGL employs to keep track of the various operations conducted on the 3D objects in the program.

ELMO aims to be intuitive to the graphics programmer and hopefully will allow graphics programmers to quickly express everything from simple 3D objects to immersive 3D environments.

Why did we call our language, ELMO? We could suggest that the acronym, ELMO, has some deeper meaning. Perhaps, we could even devise an elaborate story about how Elmo, the Sesame Street character, was really a carefree, sing-aalong creature only part of the time. In fact, we could embellish and even propose that Elmo designed vast 3D worlds using Maya in his spare time (when he was acting so darn happy!), but we would just be fooling ourselves. We really just wanted a recursive acronym. It was just "cool".

Though, you never know. Elmo might really like manipulating objects.

2 Key Aspects

2.1 3D Geometry

The core of ELMO is its manipulation of 3-dimensional graphical objects. While supporting a wealth of options and user-defined properties, ELMO aims to be easy to code and understand conceptually.

2.2 Basic Transformation

All the primative transformations you'd expect are supported within the language. Translation, rotation, scaling, and zoom are all handled under the surface with no need to learn matrix multiplication, fret over vectors, or understand homogenous coordinate systems.

2.3 Grouping and Cloning

On top of its primative operations, ELMO allows for convenient creation of more complex objects from simpler ones through an amalgation process more readily known as grouping. This allows basic cylinders and spheres to become more impressive graphical creations such as dendritic growths, ie trees. Having created these complex objects, ELMO also allows for their convenient cloning through an internal process of copying such that an inital conceptual model can be instantiated time and time again as fits the programmer's intentions.

2.4 Parameterized Objects

An important feature that differentiates ELMO from OpenGL or Java 3D is the ability to quickly script and generate complex and unique objects using OBJ input (either from OBJ files or other ELMO programs). By giving the user the ability to involve random numbers easily into object manipulation, projects like generating complex yet unique structures (a forest of trees) becomes much less of a hassle. Now all a modeler needs to do is create a tree script and include randomness. Run the script as many times as you need unique trees and viola! A diverse forest.

3 Implementation Ideas

3.1 Input/Output

Essentially, an ELMO program will take some "building block" geometry, perform various transformations on these objects or copies of these objects, and output the resultant 3D scene to a file. Tentatively, we will choose the Alias/Wavefront OBJ file format to describe both the inputs and the outputs of an ELMO program.

Thus, the operation of an ELMO program will look like this:

OBJ input(s) → ELMO execution → OBJ output

This lends itself to a particular language feature of ELMO - programs can be treated as objects. Consequently, anywhere that an ELMO program might want to reference a 3D object in an OBJ file, that program could also reference another ELMO program. This allows ELMO programs to be very flexible and encourage code re-use.

3.2 Types

ELMO will support several built-in types and automatic conversions between them, where applicable. The list of built-in types, with a short description of each, is as follows:

Type	Description
int	The same as an int in C - will be converted to a float where needed.
float	The same as a float in C - will be truncated (or possibly rounded) to an int where needed.
vector	A 3D vector, with "x", "y" and "z" attributes.
object	A piece of renderable 3D geometry. Will be converted to a group where needed (to access the coordinate axes of the object, for instance)
group	A set of objects with a common coordinate frame. If the group is moved/scaled/rotated then all the contained objects are as well. Has attributes including "right", "left" (+ or - X axis), "up", "down" (+ or - Y axis), "forward", and "backward" (+ or - Z axis).

For types with attributes, those attributes are referenced with C++/Java-style "dot" syntax. examples:

```
myvector.x
mygroup.up
```

Note that any operation performed on a group can also be performed on an object. If necessary, the object will be assigned to a group with a single member, that being the object itself, and operations will internally be performed on that group. Thus, it would be valid to write:

```
myobject.up
```

And internally, an implicit group, containing only myobject, would be used.

3.3 Functions

Applying functions to variables take the following general form:

```
<action> <target> <preposition> <input>
```

For instance,

```
rotate myobject around myobject.up
move myobject along myvector
```

Several built-in functions will enable the easy use of geometric transformations. These include:

```
rotate
move
scale
clone // create a duplicate of a given object
```

3.4 Random Numbers

In the automatic generation of fractals and interesting 3D objects in general, it is often desired to use random numbers for some inputs. In ELMO, writing

```
[5,10]
```

means "a random number between 5 and 10, inclusive".

3.5 Flow Control

Flow control happens much as it does in C/C++/Java, though the number of such structures available in ELMO is fewer. Braces define code blocks, as in C "if" and "for" control structures are identical in structure to C. There is no "switch" statement there is no try/catch exception handling. Additionally, there is a "foreach" statement available the syntax is as follows:

```
foreach myobject in mygroup {
    <perform some actions on myobject, etc>
}
```