ELMO

ELMO Loves Manipulating Objects

Jeffrey Cua
jmc2108

Erik Peterson
edp2002

Stephen Lee
sl2285

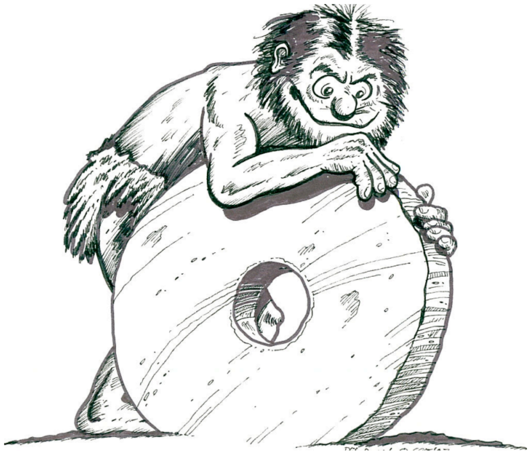
John Waugh
jrw2005

December 21, 2004

Language Goals

Accessibility ♿

- Comprehensible for non-programmer
  - Avoid direct matrix manipulation
  - Main commands (move, scale, etc) should be ‘human readable’
  - Still make it similar to popular programming languages (Java/C) so the wheel doesn’t have to be re-invented.



Funky Functions

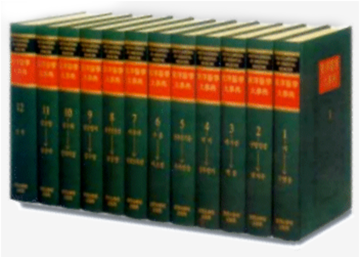
- In C++, one can have defaults, but only in limited way

```
void foo(int i=0, int j=0);
```

  - When calling foo, can’t give j a value without giving one to i as well

- In ELMO, any input can have a default, and you specify which are overridden.
  - Call foo like so: `foo(j=99);`
  - More verbose syntax, but defaults are more useful, and encourages good naming of function inputs.

References



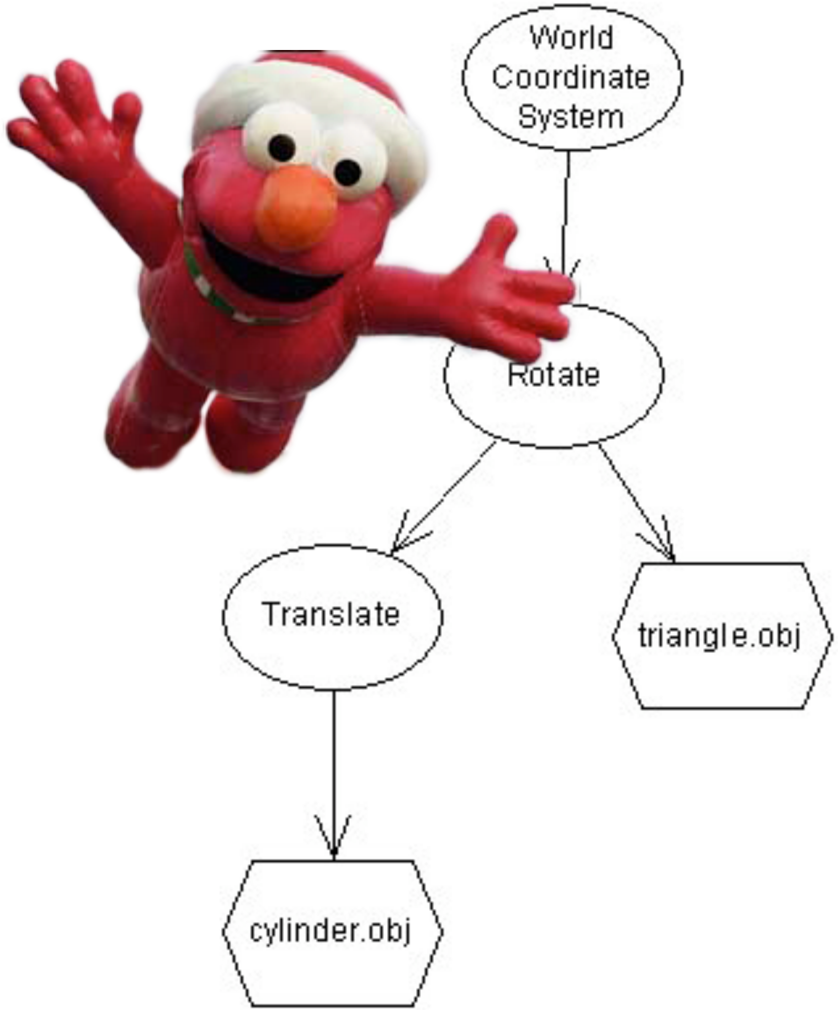
- Any variable can use referencing via the “=&” operator

```
int a =& b;
a+=5;               //changes b
foo(j=&a);          //pass a to foo by reference
a =& 22;            //a no longer refers to b
```

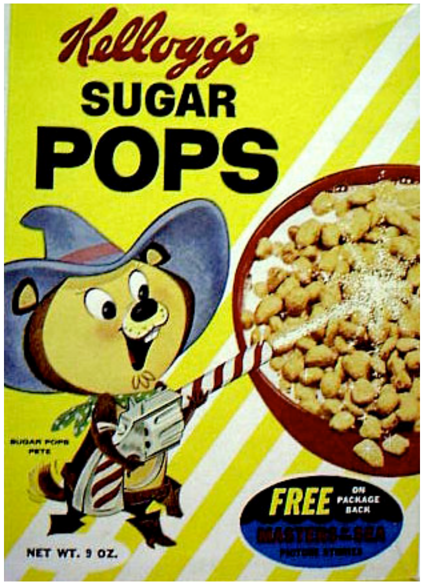
- The “=&” operator can be used anywhere ‘=’ would be

Scene Graphs

- Scene graphs allow organization of 3D transforms through hierarchical grouping.
- Easy to build up composite transforms using groups-within-groups



Sugary Syntax



- Vector syntax:

```
vector vec = <1,2,3>;
```

- Random number syntax:

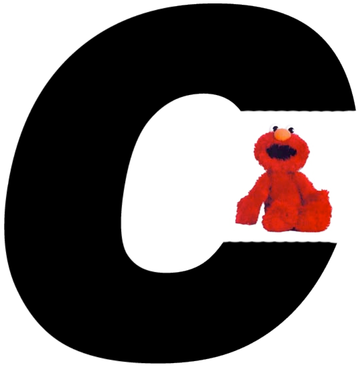
```
float r = [a..b/2];
```

- Typical transform commands:

```
rotate g around <1,0,0> by 15 deg;
move obj along obj.X by 5;
```

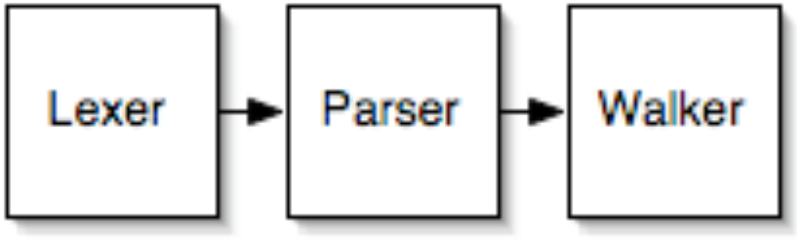
Not Quite C

- No switch statements
- for and foreach are the only iteration constructs
- Functions must be declared before they're used
- No custom data types (struct/union)
- No external definitions
  - all code must be in one .elmo file

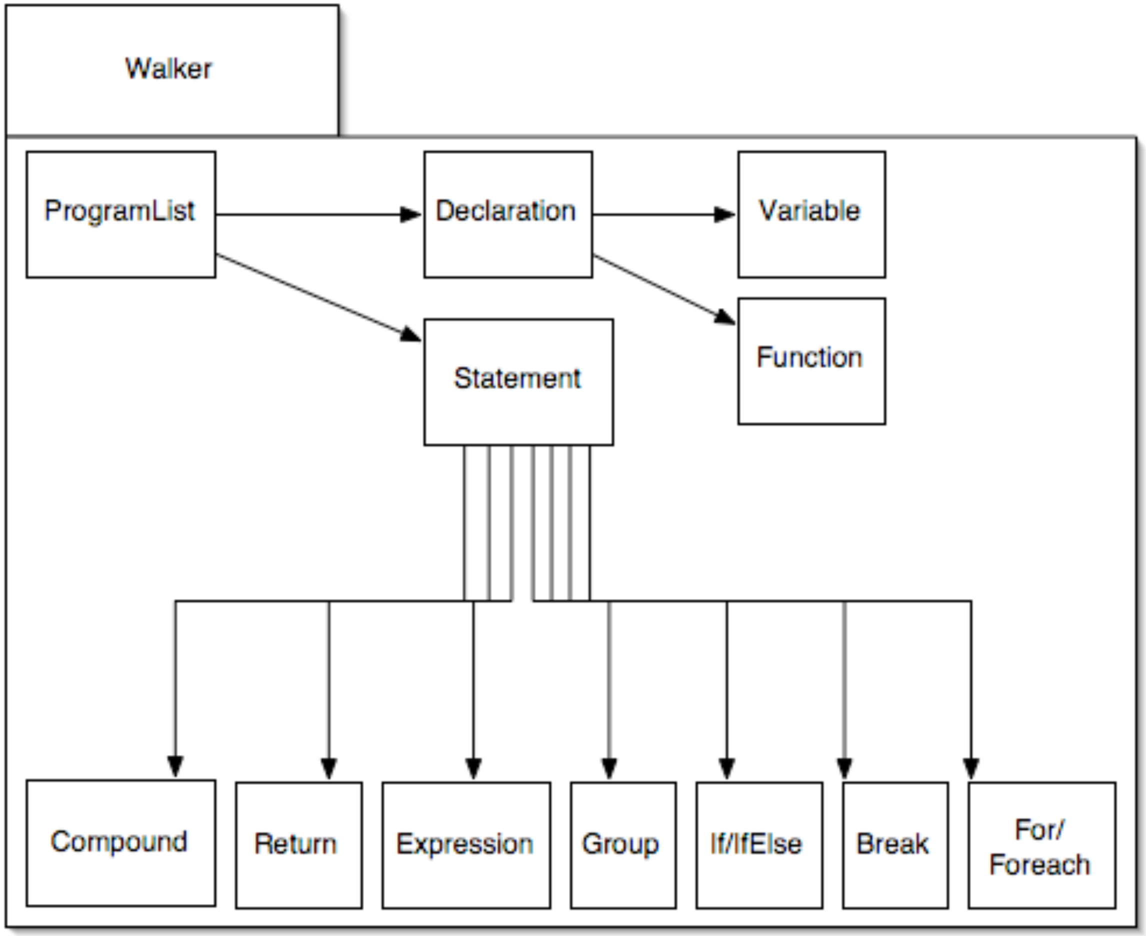


Language Implementation

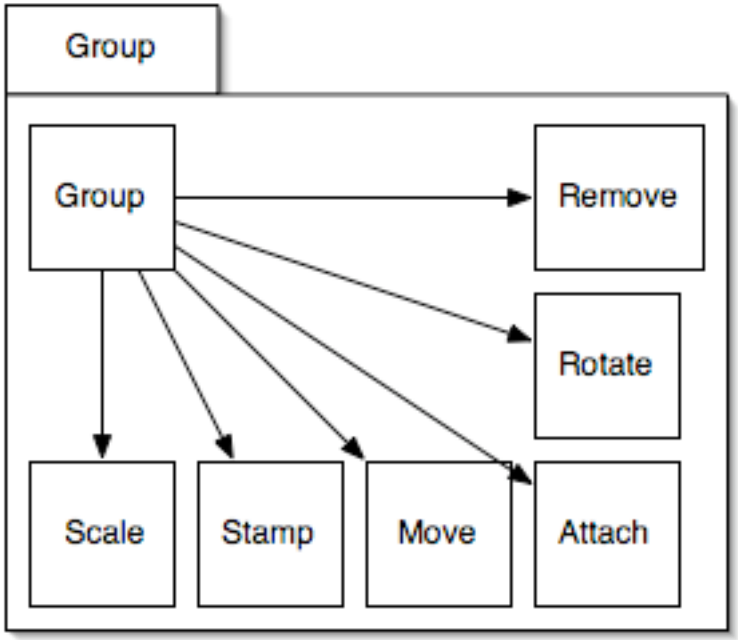
Top-Level



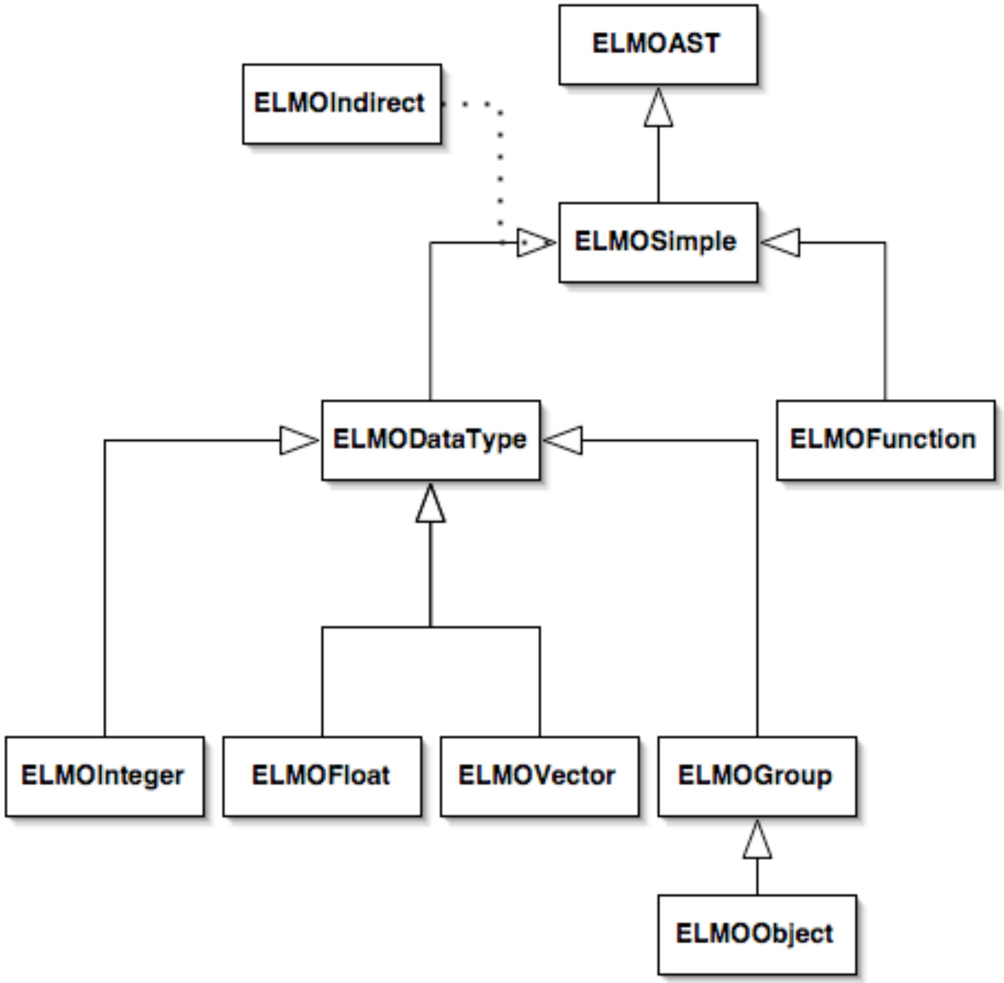
Walker


Walker
ProgramList
Declaration
Variable
Function
Statement
Compound
Return
Expression
Group
If/IfElse
Break
For/
Foreach

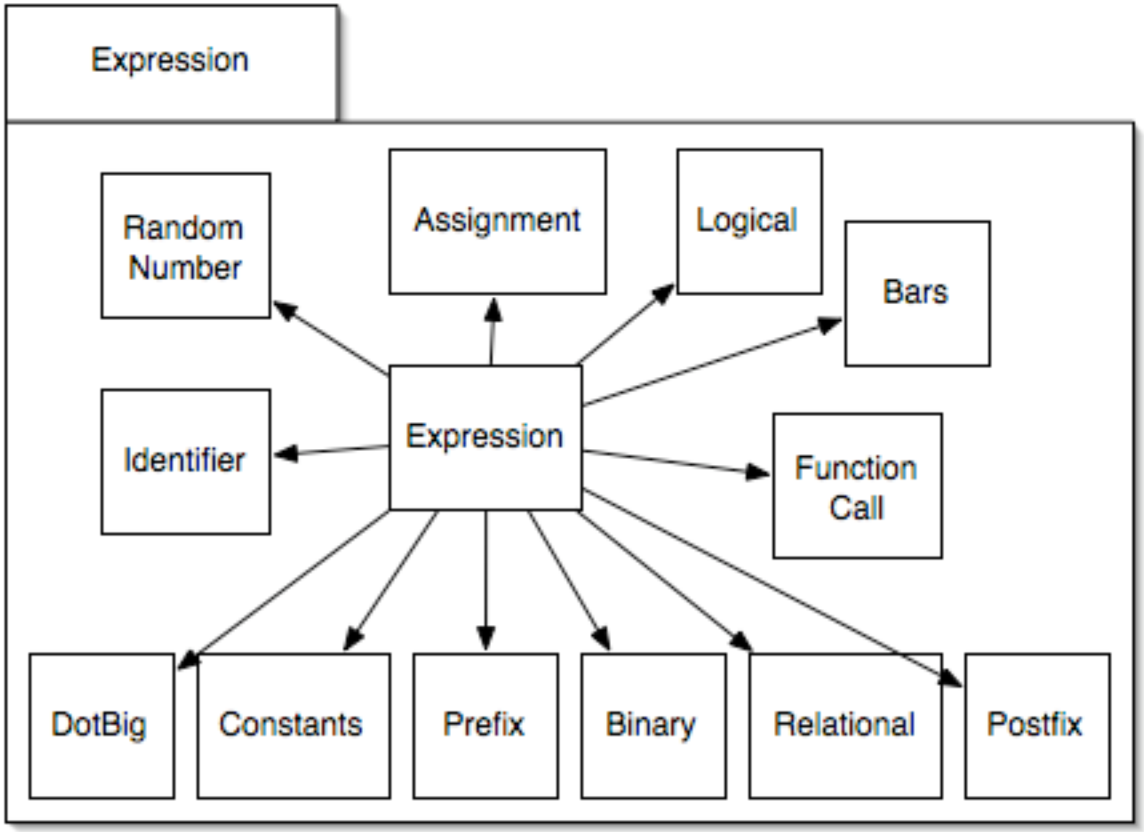
Group Statements



Class Structure

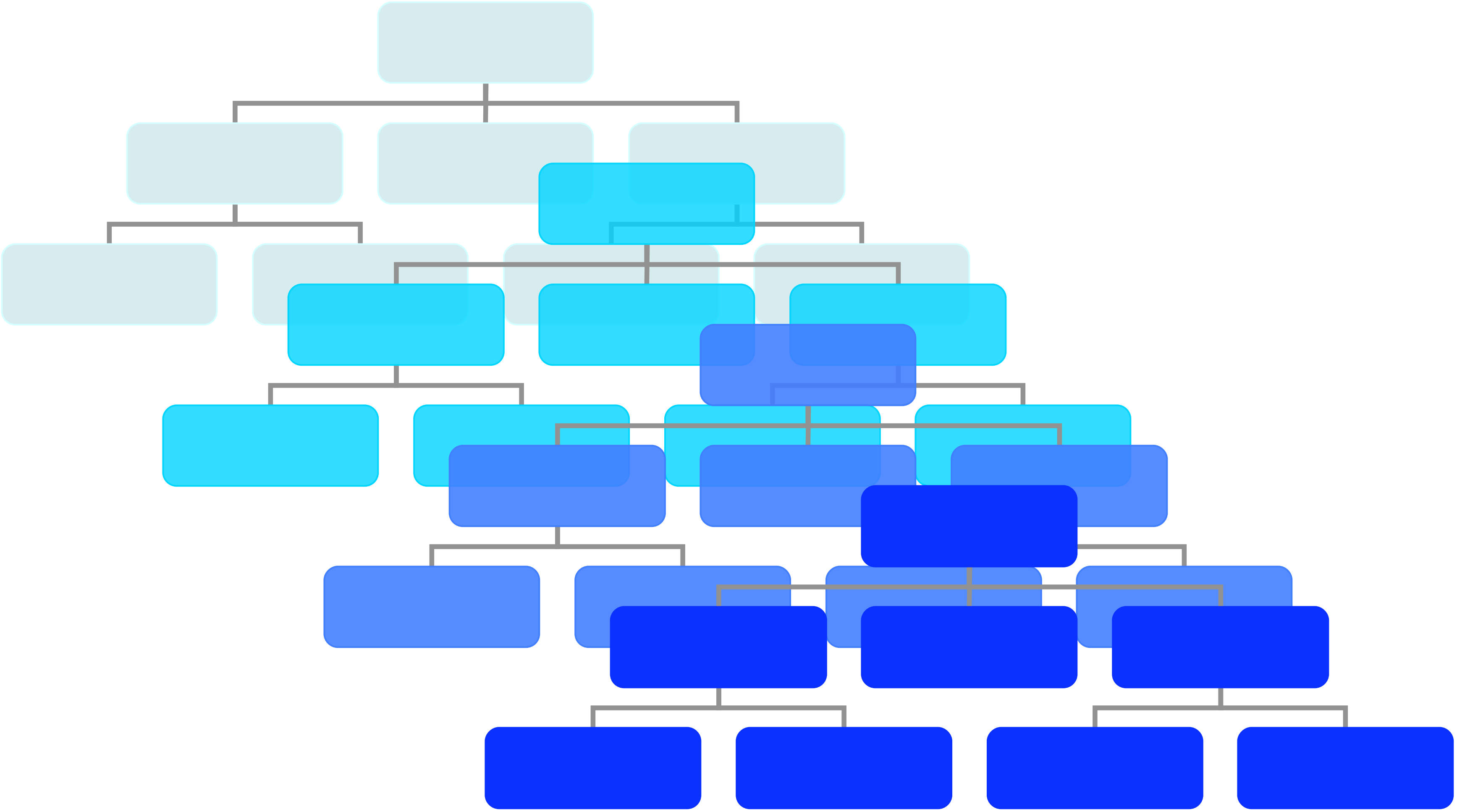


Expressions



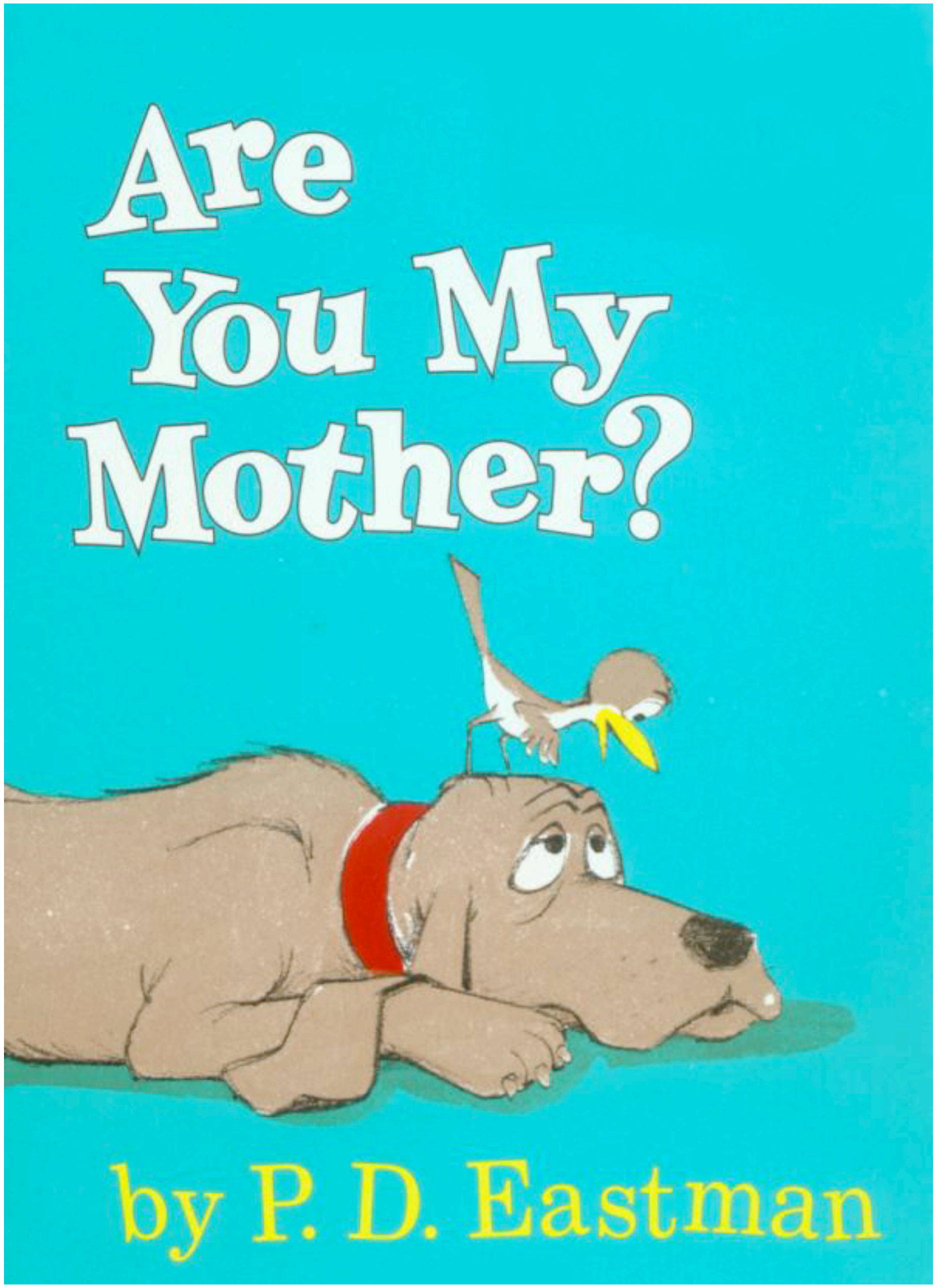
Grouper *ing*

Hierarchy & Tree Structure

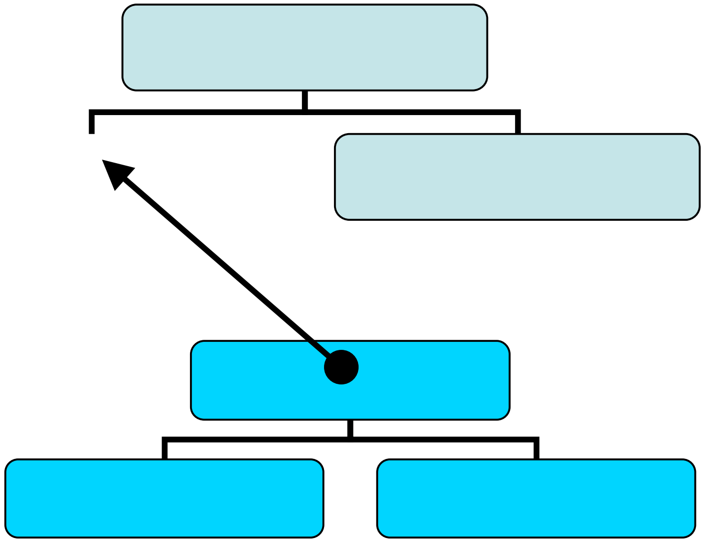


ELMOGroup nodes compose trees in the scene forest

```
// each group has a single parent
ELMOGroup _parent;

// default to orphan
ELMOGroup() {
        ...
        _parent = null;
}

// adoption by parent
setParent( ELMOGroup parent ) {
        _parent = parent;
}

// born to parent
ELMOGroup( ELMOGroup parent ) {
        ...
        _parent = parent;
}
```



```
attach( ELMOGroup a ) {
        if (a._parent==null && !isAncestor(a)) {
                ...
                a.setParent( this );
        }
}
```



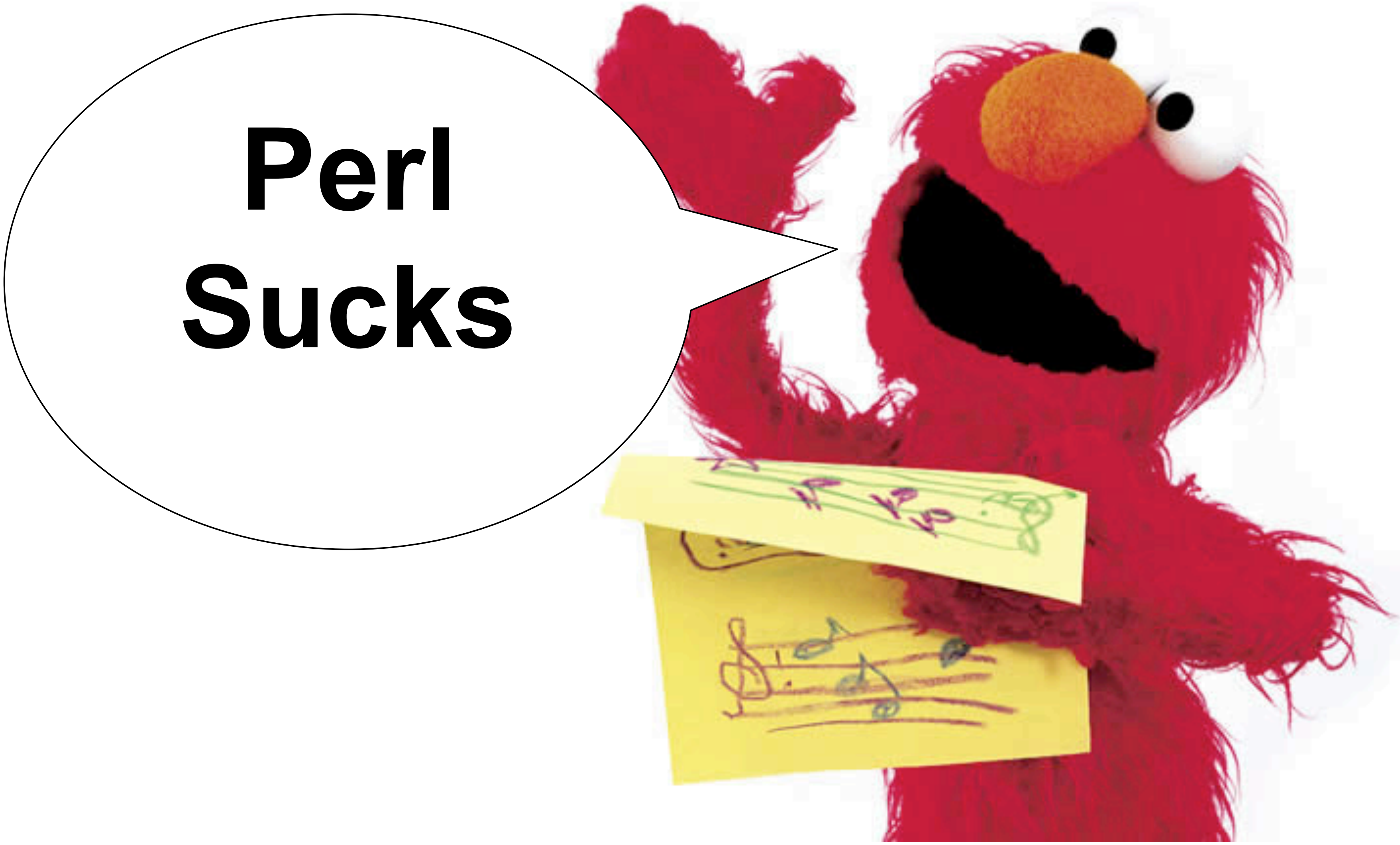
```
isAncestor( ELMOGroup a ) {
        if (this == a) {
                return true;
        } else if (_parent == null) {
                return false;
        } else {
                return _parent.isAncestor(a);
        }
}
```

```
// parent disowns you
remove( ELMOGroup a ) {
        ...
        a.setParent( null );
}

// you get a car
removeWithInheritance( ELMOGroup a ) {
        ...
        a.setParent( null );
        ELMOMatrix t = this.getInheritedTM();
        a.multiply( t );
}
```

```
getInheritedTM() {
        ELMOMatrix t = ELMOMatrix.ID();
        ELMOGroup g = _parent;
                while( g != null ) {
                t = ELMOMatrix.mult( t, g.getTransformationMatrix() );
                g = g.getParent();
        }
        return t;
}
```

Making ELMO Sing:

A Quick Tutorial



Importing OBJ files

```
//imports and assigns sphere.obj to object
  //sphere
object sphere = "tests/sphere.obj";
//prints filename
print sphere;

//copies sphere to sphere2
object sphere2 = sphere;
```

Transforming an Object

```
// moves sphere along x-axis by 3 units
move sphere along <1,0,0> by 3;
// rotates sphere around axis by PI/6 radians
rotate sphere around axis by PI/6;
// scales sphere around origin by 90%
scale sphere around <0,0,0> by 0.9;
```

Creating/Calling a function

```
// creates function named curl
void curl ( int counter, object sphere, vector axis ) {
//<insert body here>
}

// calls function curl setting the following args
curl( counter=10, sphere=sphere, axis=<0,1,0> );
```

Exporting

```
stamp sphere;
```

stamp sends *sphere* to an object buffer that will hold it until the program finishes and flushes the contents to a file.

Recursion

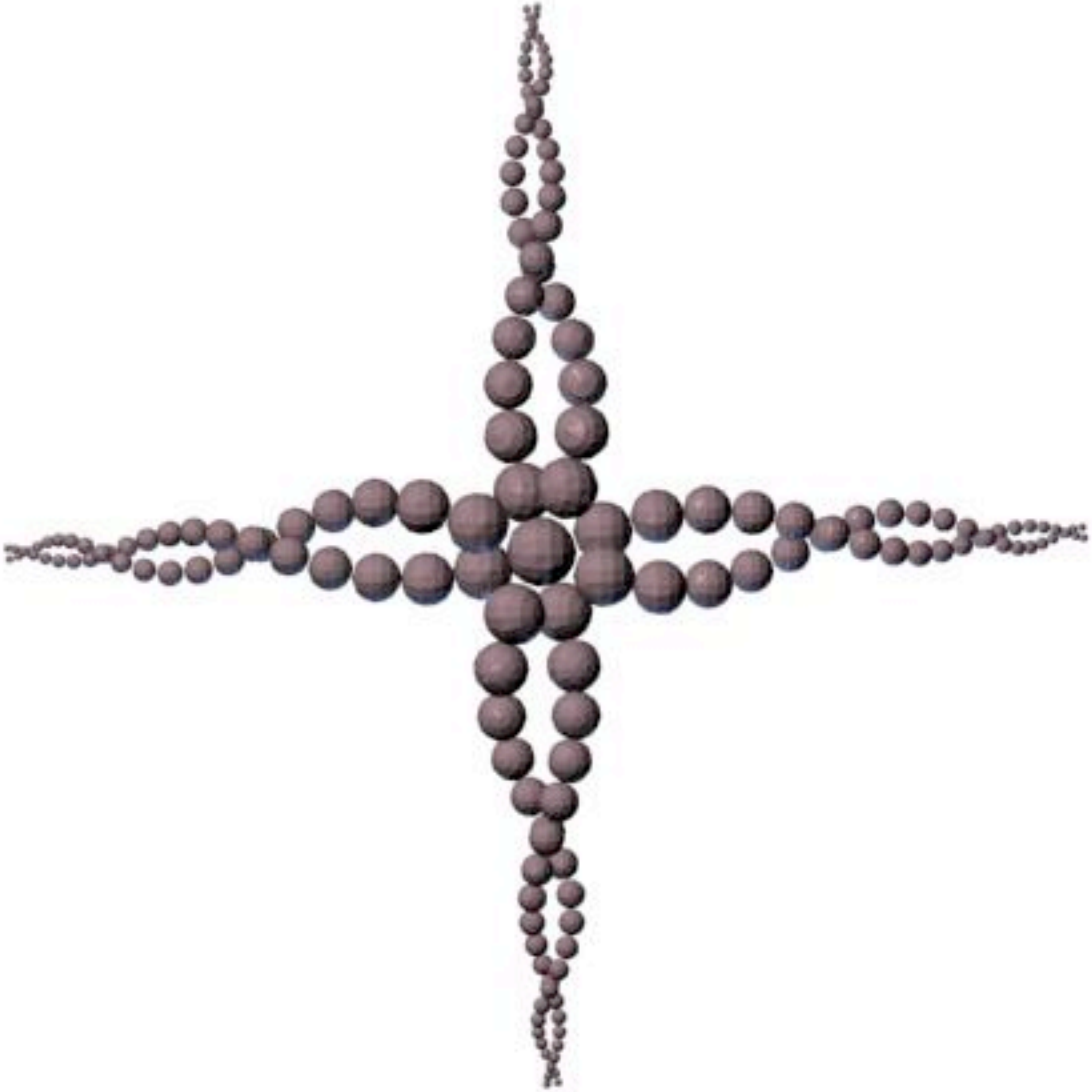
```
void curl ( int counter, object sphere, vector axis ) {
    stamp sphere;
    if ( counter != 0 ) {
        counter --;
        rotate sphere around axis by PI/6;
        move sphere along axis by 4;
        scale sphere around <0,0,0> by 0.9;
        curl( counter=counter, sphere=&sphere, axis=axis );
    }
}
```

Putting it all together

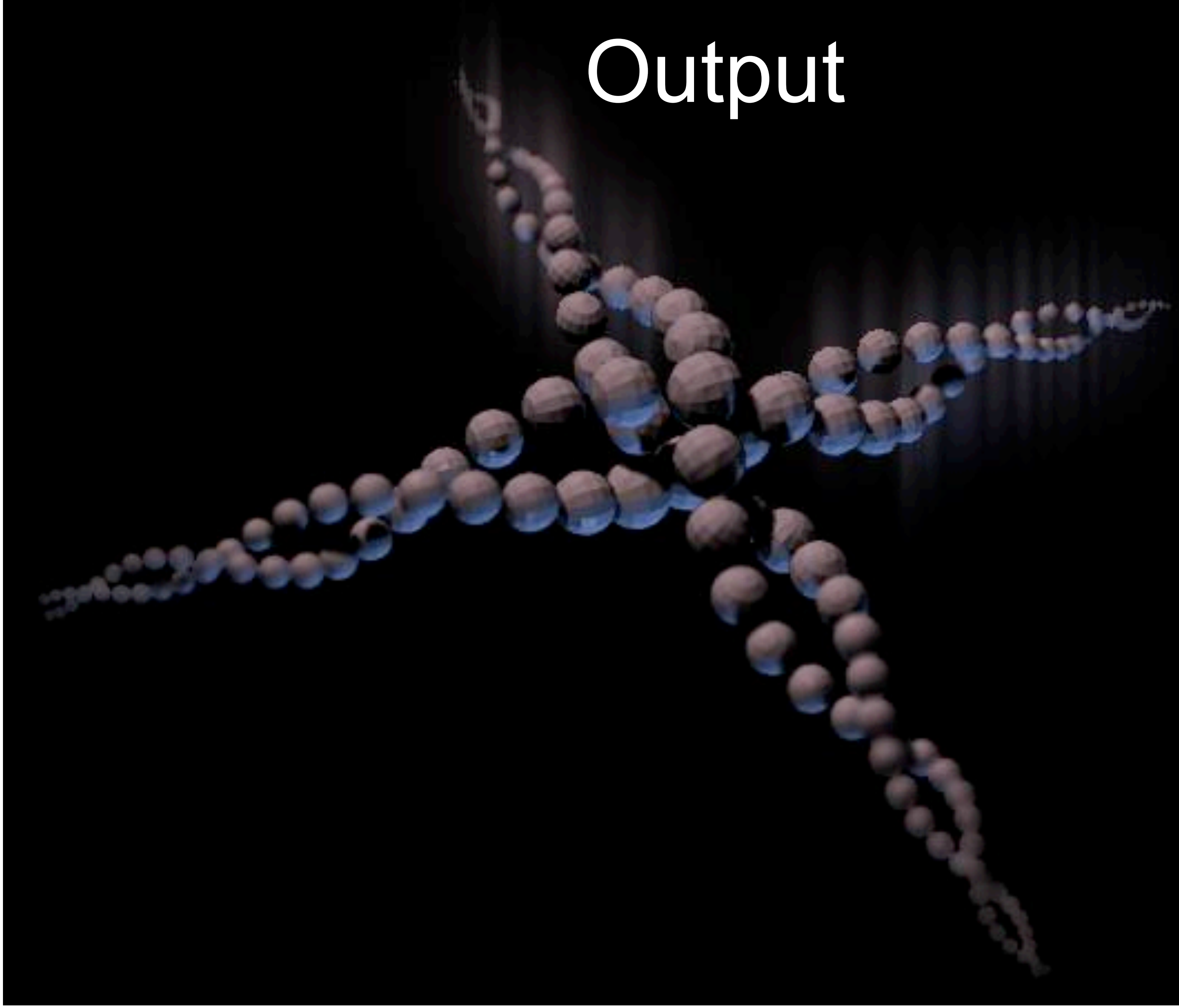
```
object sphere = "tests/sphere.obj";
print sphere;

object sphere2 = sphere;

move sphere along <1,0,0> by 3;
move sphere2 along <1,0,0> by -3;
int counter = 20;
```

Creating a quick compound Object

```
curl( counter=counter, sphere=sphere, axis= <0,1,0> );
curl( counter=counter, sphere=sphere2, axis= <0,1,0> );
curl( counter=counter, sphere=sphere, axis= <0,0,1> );
curl( counter=counter, sphere=sphere2, axis= <0,0,1> );
curl( counter=counter, sphere=sphere, axis= <0,-1,0> );
curl( counter=counter, sphere=sphere2, axis= <0,-1,0> );
curl( counter=counter, sphere=sphere, axis= <0,0,-1> );
curl( counter=counter, sphere=sphere2, axis= <0,0,-1> );
```



Output



Damn Straight!