Today

• MEL Expressions and Expression Editor.

• MEL Scripts and Script Editor.
# Maya Architecture

<table>
<thead>
<tr>
<th>User Interface</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>(MEL: Menus &amp; Windows)</td>
<td>(C++: Objects, Attributes, Connections)</td>
</tr>
</tbody>
</table>
Expressions

• Executed on each time step.
• Compute attributes of objects or particles based on:
  – Current time or frame number.
  – Attributes of this object or particle.
  – Attributes of other objects.
• May include arbitrary MEL code.
Expression Editor

Filter by Expression Name or Object Name

Default Object

Expression Code

Create/Edit Button
MEL Expression Concepts

- Statements.
- Declaration and naming of variables.
- Initialization of variables.
- Access to attributes of objects.
- Default object.
- Assignment.
Expression-Controlled Scale

```plaintext
mySphere.scaleZ = 1/(mySphere.scaleX*mySphere.scaleY);
```

In the scene demos-09-expression-controlled-scale.mb, the `scaleZ` attribute is controlled by an expression. The `scaleExpression` enforces the constraint: `scaleX*scaleY*scaleZ = 1.0` so that the volume of the sphere remains constant.
Referencing Attributes

Note: In the Expression Editor, if `mySphere` were the default object, then we could reference this attribute with the name `scaleZ` alone, i.e., without naming the object.
Algebraic Expression

\[ \frac{1}{\text{mySphere.scaleX*mySphere.scaleY}} \]

Division Operator

Multiplication Operator
Assignment Statement

Storage Location (Attribute or Variable)

mySphere.scaleZ = 1/(mySphere.scaleX*mySphere.scaleY);

Algebraic Expression

Assignment Operator

Semicolon (Ends the Statement)
Expression-Controlled Translation

float $location = nurbsSphere1.velocity * time;
float $phase = (frame/nurbsSphere1.period)*360.0;
float $height = nurbsSphere1.amplitude * sin(deg_to_rad($phase));
nurbsSphere1.translateX = $location;
nurbsSphere1.translateY = $height;

In the scene demos-09-expression-controlled-translation.mb, a NURB sphere is given attributes velocity, amplitude and period. The *translationExpression* references these attributes to compute X and Y translation at each time step.
User-Defined Attributes
Standard Attributes

• User may define new attributes of an object by selecting the object, opening the Attribute Editor and invoking Attributes-AddAttributes.
  – E.g., `nurbsSphere1.amplitude`
  – E.g., `nurbsSphere1.period`
  – E.g., `nurbsSphere1.velocity`

• Some attributes are automatically defined in every expression:
  – E.g., `time`
  – E.g., `frame`
Declaring a Variable

```plaintext
float $height;
int $age;
string $name;
```

Data type:
- **float** (Decimal Number)
- **int** (Whole Number)
- **string** (Sequence of Characters)

Variable Name: Must begin with a $, followed by a letter or underscore. Then it may include any sequence of letters, numbers and underscores.
Declaring and Initializing Variable

float $height = 4.25;

int $age = 6;

string $name = "Polly";
In the scene demos-09-expression-controlled-color.mb, the color of the blinn material assigned to a sphere is computed based on the position of the sphere. The computed color is the weighted average of the colors of two other spheres. The weights are proportional to the distances to the other spheres.
Vectors

vector $ball1Pos = <<ball1.translateX,ball1.translateY,ball1.translateZ>>;  
vector $ball2Pos = <<ball2.translateX,ball2.translateY,ball2.translateZ>>;  
vector $ball3Pos = <<ball3.translateX,ball3.translateY,ball3.translateZ>>;

• A vector $\langle x, y, z \rangle$ is a grouping of 3 float numbers.
• A vector may represent:
  – The position of an object relative to the origin of a coordinate system.
  – The distance and direction from one object to another.
float $d13 = mag($ball3Pos-$ball1Pos);
float $d23 = mag($ball3Pos-$ball2Pos);
Modifying the Color Control Expression
(In Class Exercise)

• Make \textit{ball1} tall and thin.
• Make \textit{ball2} short and fat.
• Arrange for \textit{ball3} to be:
  – Tall and thin when near \textit{ball1}.
  – Short and fat when near \textit{ball2}.
Solution to Exercise

vector $ball1Pos = <<ball1.translateX,ball1.translateY,ball1.translateZ>>;
vector $ball2Pos = <<ball2.translateX,ball2.translateY,ball2.translateZ>>;
vector $ball3Pos = <<ball3.translateX,ball3.translateY,ball3.translateZ>>;
float $d13 = mag($ball3Pos-$ball1Pos);
float $d23 = mag($ball3Pos-$ball2Pos);
float $w1 = $d23 / ($d13 + $d23);
float $w2 = $d13 / ($d13 + $d23);
blinn3.colorR = $w1*blinn1.colorR + $w2*blinn2.colorR;
blinn3.colorG = $w1*blinn1.colorG + $w2*blinn2.colorG;
blinn3.colorB = $w1*blinn1.colorB + $w2*blinn2.colorB;

// New statement:
ball3.scaleX = $w1*ball1.scaleY + $w2*ball2.scaleY;
Expression-Controlled Aim

vector $target1Pos = <<target1.translateX, target1.translateY, target1.translateZ>>;
vector $target2Pos = <<target2.translateX, target2.translateY, target2.translateZ>>;

float $d1 = mag($target1Pos-$pointerPos);
float $d2 = mag($target2Pos-$pointerPos);

float $maxMag = 500.0;

if ($d1 <= $d2)
{
    newtonField1.magnitude = $maxMag;
    newtonField2.magnitude = 0.0;
}
else
{
    newtonField1.magnitude = 0.0;
    newtonField2.magnitude = $maxMag;
}

In the scene, demos-09-expression-controlled-dynamic-eyes.mb, the eyes are aim-constrained to a locator parented to an invisible rigid body. The body is influenced by two Newton fields, each of which is attached to a cone moving on a path. The expression turns on the field of the closest cone and turns off the other field. The eyes are drawn to the closest cone.
Expression Controlled Deformers

float $temp = deformerA1.progress/0.08-deformerA1.hOffset;
deformerA1.pathValue = deformerA1.vOffset + floor($temp)*0.08;
deformerA1.phase = $temp - floor($temp);
motionPath8.uValue = deformerA1.pathValue - floor(deformerA1.pathValue);

In the scene demos-09-expression-controlled-deformers.mb, a collection of eight lattice deformers are cyclically moved along a path into positions where they can widen an extrusion surface so that a geometric shape can appear to bass through.
In the Hypergraph’s input/output connections view, we can see how deformerA1 receives the progress attribute from the blobTubeGroup. We can also see how the expression node receives values from deformerA1 and sends values back to deformerA1 and to its motion path node. Finally we can see that deformerA1 sends its phase attribute to two animation curves that control latticeA1 (via Set-Driven-Key).
Attributes and Connections

In the scene demos-09-expression-controlled-deformers.mb, the `blobTubeGroup1` node has a `progress` attribute that is connected (by the Connection Editor) to `progress` attributes on numerous other objects, including the deformers on the extruded surfaces, as well as cluster deformers that move the blob surface along the path. This arrangement allows the rate of the whole system to be controlled by animating the `blobTubeGroup1.progress` attribute.
Particle Expressions

• Created / accessed via “Per Particle Attributes” of the particle shape object in the Attribute Editor.
• Evaluated once per particle.
  – Creation Expression: Evaluated when particle is created.
  – Runtime Expression: Evaluated on each time step.
• May slow computation.
• Use Solvers-CreateParticleDiskCache to speed up successive running of simulations.
vector $pos = particleShape1.position;
float $posY = ($pos.x)*($pos.x) + ($pos.z)*($pos.z);
particleShape1.position = <($pos.x, $posY, $pos.z)>;
vector $vel = particleShape1.velocity;
particleShape1.velocity = <($vel.x, 0.0, $vel.z)>;

In the scene demos-09-expression-particles-paraboloid.mb, a particle system is constrained to move on the surface of a paraboloid. The expression shown above computes a particles Y position as a function of its X and Z velocities.
MEL Commands in Expressions

• Particles following spiral path along a specified curve:
  
  demos-09-expression-particles-spirial1.mb
  demos-09-expression-particles-spirial2.mb

• Uses point on curve command to place particle at a location that depends on the curve and the particles age.
Script Editor

• **Echoing menu commands:** When the user invokes a Maya menu, the menu runs a script. The commands executed by the script appear in the script editor.

• **Return v. Enter:** When typing in the white entry area of the script editor, Return (on main keyboard) moves to a new line and Enter (on numeric keypad) causes Maya to read and evaluate the contents of the entry area.

• **Copy and Paste:** Highlight some text in the Script Editor. Then use Control-c to copy. Move the mouse to a location in the entry area of the Script Editor. Use Control-v to paste.

• **Put Commands on Shelf:** First highlight a command in the Script Editor. Then click and drag the command with LMB to the Shelf. Now you can execute the command with one click.
Script Editor

Maya menu commands are “echoed” here.

Results of processing MEL commands are shown here:

Enter MEL commands here and press Enter:
Issuing MEL Commands in Maya

- Script Editor.
- Command Line:
  - Lower left of screen.
  - One command at a time.
  - Results in Command Feedback area.
- Command Shell:
  - Invoked with Window-GeneralEditors-CommandShell.
  - Like Command Line with access to prior commands.
Simple MEL Script

sphere -name "sun" -radius 1;
duplicate -name "earth";
scale 0.3 0.3 0.3;
move 5 0 0;
duplicate -name "mars";
move 8 0 0;
scale 0.2 0.2 0.2;
select "earth" "mars" "sun";
parent;
select "sun";
group -name "solarSystem";
Generalizing the Script

• What if we want to create a slightly different solar system, e.g., with different sizes and positions of the planets?
• Most of the script will stay the same, but a few parts will change.
• How can we reuse our work in writing this script?
• Answer: *Procedural Abstraction.*
Simple MEL Procedure

global proc createSolarSystem(string $name,
    float $earthR, float $marsR,
    float $earthS, float $marsS)
{
    sphere -name "sun" -radius 1.0;
    duplicate -name "earth";
    scale $earthS $earthS $earthS;
    move $earthR 0 0;
    duplicate -name "mars";
    move $marsR 0 0;
    scale $marsS $marsS $marsS;
    select "earth" "mars" "sun";
    parent;
    select "sun";
    group -name $name;
}
Invoking the Procedure

• Save the procedure in a file called:
  “createSolarSystem.mel”
• Put the MEL file in the mel subfolder of your project folder.
• Possibly execute the command rehash;
• Execute the command:
  createSolarSystem("solarSystem",5,8,.5,.3);
Snowman Script

• Dialog box asks for name, size and overlap parameter of snowman.
• Create button executes functions to make the snowman.
• Functions:
  – createSnowmanUI(): Constructs/displays the dialog box.
  – createSnowmanAction(): Collects data from dialog box and passes it to createSnowman() procedure.
  – createSnowman(): Procedure that finally creates the snowman.
Path Replicate Script

• User selects an object and a curve.
• User enters number of copies, and indicates whether an how the path follow option should be used.
• Create button executes functions to distribute copies of the object along the path.

• Functions:
  – pathReplicateUI(): Constructs/displays the dialog box.
  – pathReplicateAction(): Collects data from dialog box and passes it to pathReplicate() procedure.
  – pathReplicate(): Procedure that finally replicates the selected object along the selected curve.