How and where to enter commands

MAXScript Listener

MAXScript>MAXScript Listener or F11

MAXScript Editor

MAXScript>New Script evaluate with CTRL-E or Tools>Evaluate All

Selecting objects, $ and wildcards

$ current selection

sBox01 object with the name “Box01”

select sBox01 select object with the name “Box01”

sTile* objects whose names start with “Tile”

sTile A* objects whose names start with “Tile A” (use this for objects with whitespace characters)

s* all objects

Access and Modification of Object Properties

$.name = “table” set name of current object to “table”

$sBox01.width = 10 set-width of object “Box01”

$sSphere01.radius = 5 set radius of object “Sphere01”

$.renderable = false turn off “renderable” property of current object

$.<property> = value general syntax for modifying properties

Transforming Objects

move $ [10, 0, 0] move relatively by 10 units in x

rotate $ (eulerangles 90 0 0) rotate relatively by 90 degrees around x

scale $ [2, 2, 1] scale twice in x and y, z is unchanged

$.position = [10, 10, 0] set new absolute position

$.position.z = $.position.z + 10 move object up by 10 units

$.rotation = (eulerangles 0 0 90 ) rotate by 90 degrees around z

$.scale = [1, 1, 2] set new scale value: twice as high in z

Creating Objects, Modifiers and Materials

Box () create regular box at the origin

Box pos:[-5, 5, 0] wirecolor:red create a red box at [-5, -5, 0]

addModifier $ ( BendD() ) apply bend modifier

addModifier $ ( Taper amount:1 primaryaxis: 1 ) apply taper modifier with amount 1 and y as primary axis (enter command as one line)

$.material = StandardMaterial() apply standard material

$.material = VRayMtl() apply VRay material

$.material = VRayMtl diffuse: green reflection:gray reflection_glossiness:0.7 apply green glossy reflective VRay material

$.material.diffuse = color 255 0 0 change diffuse color to red (for VRay the name of this property is diffuse, for standard materials it is diffuseColor !)

Variables and data types, strings, arrays

SCALAR VARIABLES (= numbers, standard mathematical operations: +, -, *, /, ^)

w = 10 assign the value 10 to w

h = 10 assign the value 10 to h

a = w * h calculate the area a

STRINGS (= characters/text/symbols)

strPrefix = “LIGHT_” assign string to a variable

$name = strPrefix + $.name apply prefix to object name

ARRAYS (= collection of values and objects of arbitrary types)

a = #($Box01, $Box02, $Box03 ) array definition with three objects

a.count returns number of elements in array

a [1].renderable = false access elements of array with brackets [ ]

Random Values

$.pos.x = random 0 10 random value between 0 and 10 (integer)

sBox01.height = random 2.0 5.0 random height between 2 and 5 (float)

$.rotation = eulerangles 0 0 ( random 0.0 360 ) random z rotation (enter as one line)

$.material.diffuseColor = color 0 0 ( random 0 360 ) random blue part of diffuse color

Outputting Values to the MAXScript Listener

print $ position print position of current object

format “% objects selected.” selection. count output number of selected objects

Hierarchies and Animation

$.children [1].name = “Tile_01” set name of first child object

animate on ( at time 100f ( move $ [0, 0, 5] ) ) with Autokey on, move object at frame 50

animate on ( at time 100f ( $.radius = 10 ) ) with Autokey on, set radius at frame 100

ObjectSets

ObjectSets are collections of the current scene objects divided into the main 3ds max object type categories / classes.

objects all objects in the scene

grouping primitives, meshes and other geometry objects

lights lights (includes VRay and other third party lights)

cameras cameras

helpers helpers

shapes lines, circles, rectangles, splines

systems systems (e.g. bones)

spacewarps forces, spacewarps, deflectors etc.

selection current selection (this is the same as $)

Nifty “one-liners”

Examples for using ObjectSets to apply a command to a bigger group of objects

delete cameras deletes all cameras in current scene

delete Slights/Fill* delete all lights starting with “Fill”

hide shapes hides linework

freeze geometry freeze all geometric objects

$*.material = undefined removes all materials of scene

Tile*,.material = medit.getCurMtl() apply currently selected material in editor to all objects whose name starts with “Tile”
MAXScript Cheat Sheet 2/2

MAXScript control structures

Loops
For-loops are useful for parts of code that need to be executed more than once or, for example, when you would like to change a certain property for many objects.

for i = 1 to 10 do
{
    Box pos: [i * 30, 0, 0];
}

create 10 boxes at different x positions

for obj in (selection as array) where classOf obj == Cylinder do
{
    obj.scale.z = 2;
}

set scale value for cylinders in current selection

Conditional Statements
Conditional statements are used to make decisions based on comparisons at runtime. They allow you to create different branches and to control the program flow.

if ( $pos.z < 0 ) then $wirecolor = red
else $wirecolor = green

set wirecolor to red if object is below 0, otherwise green

if ( superclassof $ == Light ) then delete$

delete current object if it is a light

Functions
This is an example function to calculate the volume value from width, length and height. It will return the result, in this case the volume calculation. Generally, functions can be used to encapsulate code of any type or purpose to prevent repetition and clutter within bigger scripts.

function calculateVolume w \ l \ h =
{
    w * l * h
}

Function definition: What are the expected parameters (e.g. width, length, height) and what does the function do with it (e.g. multiply them to calculate volume)? The result of the last expression in the function body will be returned as a value to the caller of the function.

calculateVolume 10 20 5
Function call with concrete values: what does the function do with it (e.g. multiply parameters (e.g. width, length, height) and what does the function do with it (e.g. multiply them to calculate volume)? The result of the last expression in the function body will be returned as a value to the caller of the function.

Mass changing properties
for obj in (selection as array) do
{
    turn off “visible to camera” property for selected objects
}

for obj in (selection as array) where hasProperty obj “radius” do
{
    set the radius property for objects which expose it (applies to spheres and cylinders but not to a box, for example)
}

for m in obj.modifiers where ( classof m ) == TurboSmooth do
{
    turn off turbosmooth modifiers on selected objects by iterating all objects and also all their modifiers. The classof command checks if m is a Turbosmooth modifier and then disables it.
}

Environment Settings

backgroundColor = color 0 0 0
set environment background color

useEnvironmentMap = false
toggle environment map (true/false)

environmentMap = Checker()
apply environment map (e.g. a checker map)

Basic Render Settings

rendStart = 100
set render start frame

rendEnd = 250
set render end frame

renderWidth = 1920
set render width (horizontal resolution)

rendHeight = 1080
set render height (vertical resolution)

rendSaveFile = true
activate writing render to disk

rendOutputFilename = "P:\viz01\beauty_v01 _.exr"
set render output file path

... for a full list of render parameters, please consult the MAXScript reference

max quick render
equivalent to pushing the “render” button

render camera: $Camera01 frame:100 outputfile: "P:\viz01\beauty_v01 _.exr"
additional function to kick off renders independent of the current render dialog settings

Renderer-Specific Settings (e.g. VRay)

renders.current = VRay()
set current renderer to VRay

renders.current.adaptive-Subdivision_minRate = 1
set adaptive sampling min rate

renders.current.adaptive-Subdivision_maxRate = 2
set adaptive sampling max rate

renders.current.gl_on = true
turn on VRay’s GI calculation

... for a full list of your renderers’ parameters, please consult its user manual

Rollout Floater Basic Framework

The following code is a basic framework for building your own tools with graphical user interfaces (such as rollouts, buttons, sliders etc.) to increase the degree of interactivity of your scripts and provide more advanced functionality.

{ global rofTool

    Rollout roTool "Tool"
    {
        spinner spSpinner “Value:"
        button btnButton “Do it!”
        on btnButton pressed do
        {
            format “value: %
            spSpinner.getValue
            }
        }
    }

    try
    {
        closeRolloutFloater rofTool
        catch ()
        rofTool = newRolloutFloater "Tool Floater" 300 100
        addRollout rofTool rofTool
    }

    -start a new local variable scope here and declare a global variable for the rollout floater
    -start a new rollout definition
    -add a spinner and a button control to the rollout
    -event handler for the button
    -output current spinner value
    -mechanism to prevent opening more than one floater window.
    -if the user executes the script without closing the old floater, it will automatically close it before bringing up the new floater.
    -add rollout to the floater

The MAXScript reference provides a lot of examples, FAQs and scripts to help you build better tools and improve your workflow. Furthermore, there are useful websites such as scriptspot.com, cgarchitect.com, cgtalk.com and many more with a big collection of existing scripts and friendly experts to ask. Have fun with MAXScript!