INICIACIÓN A MAXSCRIPT

JAIME3D.INFO

MAXSCRIPT

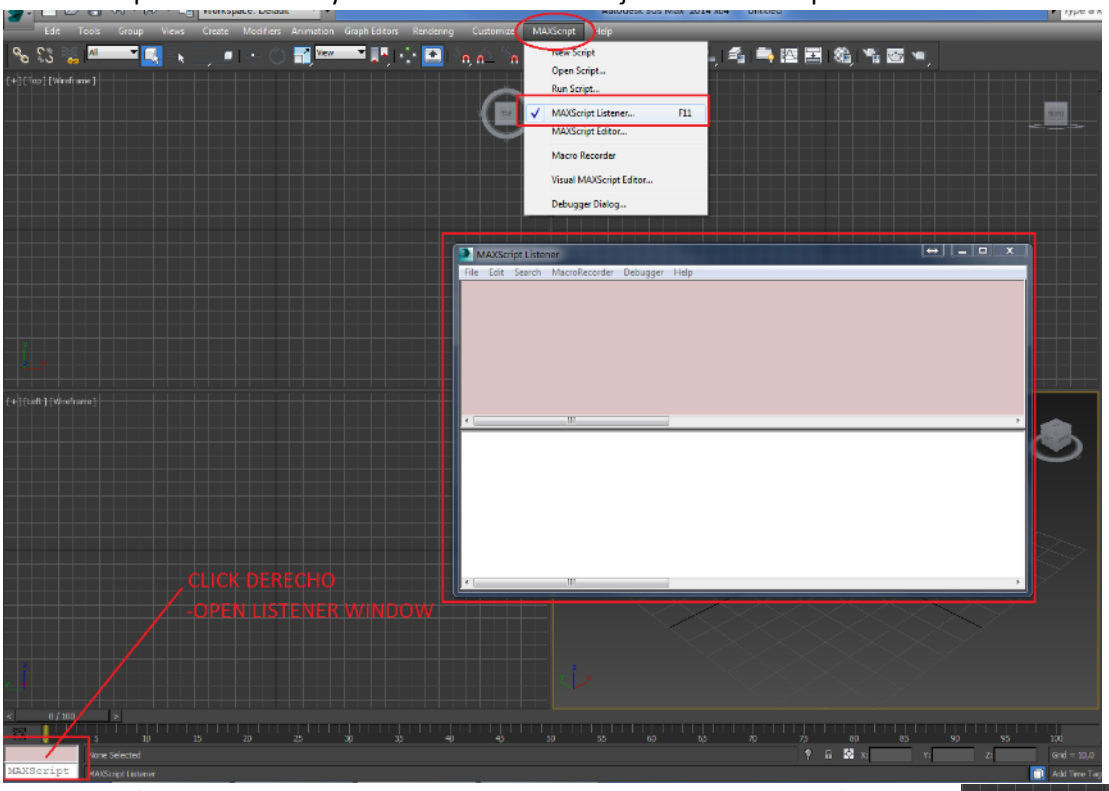
CONCEPTOS BÁSICOS

Maxscript es el lenguaje de programación de 3Dstudio Max.

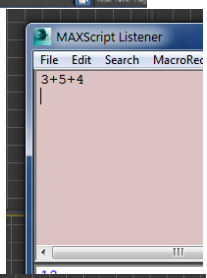
Un script son cadenas de texto que dan instrucciones al programa.

Hay mas de una zona donde se puede escribir el script .

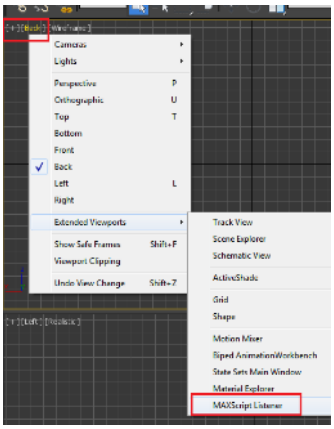
El Listener permite escribir y ver el resultado de la ejecución del script.



Hay varias formas de ver el listener. Puedes hacer clic derecho del ratón sobre la pantalla rosacea y clicar sobre Open listener Window, apretar F11, o irte al menú Maxscript de la barra de arriba y clicar sobre MAXscript Listener.



En la parte rosa podemos escribir el comando , clicamos enter, y abajo nos aparece el resultado.

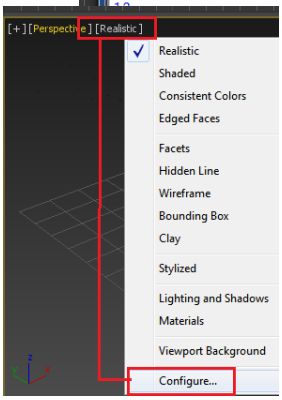


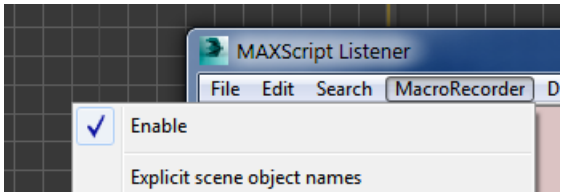
También podemos incluir el listener en uno de los visores de max.

Para volver a tener el visor como antes , basta con ir a otro , y clicar sobre la opcion de visualizacion de la ventana y configure.

En Layout encontraremos las diferentes conbinaciones de visores.

Clicamos el que nos interese y le damos a aplicar.

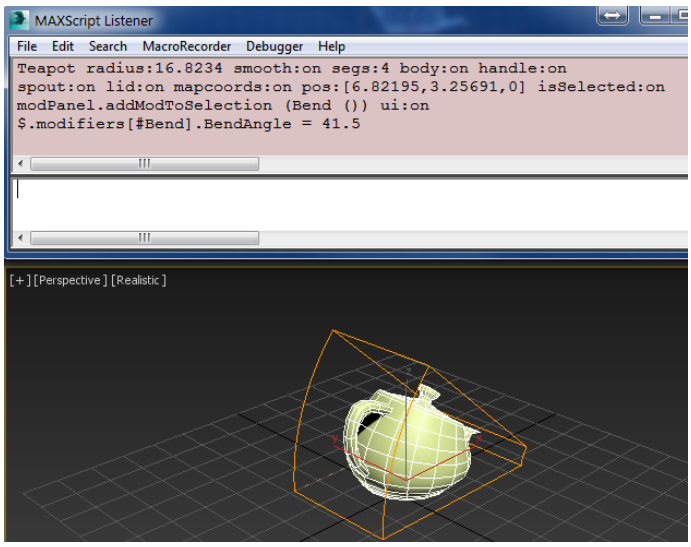




Dentro del Maxscript Listener tenemos el MacroRecorder.

Si lo activamos quedaran grabadas las acciones que hagamos en Max.

(Va bien para saber que comandos usa max al utilizar el programa).



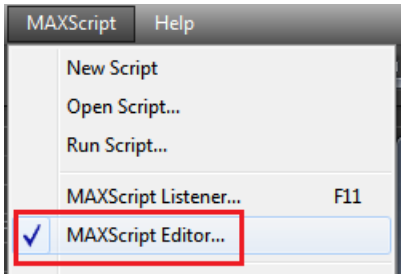
Por ejemplo creamos una tetera y le aplicamos un modificador.

Pues irá apareciendo el codigo en la pantalla rosa como podemos ver en la captura de pantalla.

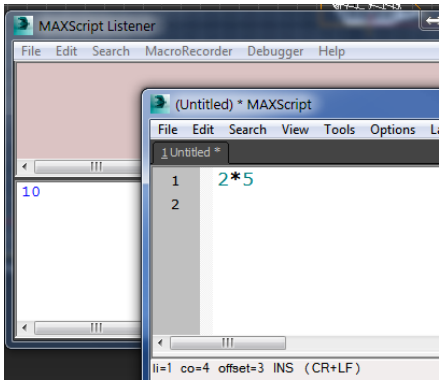
Para borrar el contenido del Maxscript le damos a Ctrl+D.

El panel en blanco es donde aparecen los resultados de la ejecución del codigo y donde podemos detectar los errores de los scripts.

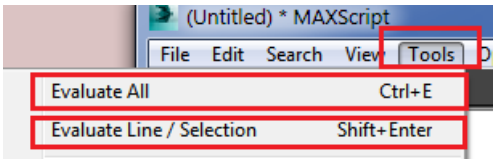
Otra forma de picar codigo , es en el Maxscript Editor.



Es como el bloc de notas, de hecho podríamos crear el código en el bloc de notas de windows y ejecutarlo en el programa,pero el editor de max es más cómodo de usar porque podemos ir abriendo ventanas y tenemos varias herramientas que nos pueden ser muy útiles, como evaluar todo el codigo o solo parte de el. O hacer comentarios de lo que vamos haciendo.



Existe una interactividad entre el Maxscript Editor y el Listener.Cuando picamos codigo en el Editor, el Listener nos irá devolviendo el resultado y nos avisará de los errores.



Para evaluar todo el codigo daremos a Ctrl+E, y para evaluar solo la parte seleccionada o la linea sobre la que esté el cursor, Shift+Enter.

Hay que ser ordenado al programar, y una de las maneras es indentar el código. Es decir, usar

La tecla Tab para que se vea todo más claro , separando unas líneas de otras y ordenarlas por jerarquía.

Una forma de ser ordenado, también es comentando el código. Hay varias maneras.

```
1 Untitled *
1  -- Esto es una multiplicación
2  2*5
3
4  /* Ahora viene el codigo de creación de la esfera */
5  Sphere radius:8.87015 smooth:on segs:32
6
```

Se puede realizar con dos guiones seguidos antes de cada comentario.

O si se quiere hacer un comentario muy largo añadimos barra y asterisco antes y despues de la linea.

```
/* Ahora viene el codigo de creación de la esfera */
Sphere radius:8.87015 \
smooth:on \
segs:32
```

Cuando existen lineas muy largas que se hacen incomodas de leer, las podemos partir con barra invertida y Max interpreta que sigue siendo la misma linea.

LAS VARIABLES

Las variables son contenedores y los datos asociados a estas se llaman valor.

Si escribimos A =10. La variable es A y el valor es 10.

Si a X le damos el valor de la creación de una esfera, cada vez que escribamos x , vamos a llamar a esa esfera.

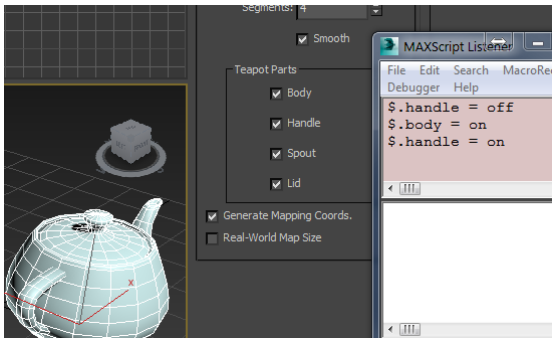
```
x = Box lengthsegs:1 widthsegs:1 heightsegs:1 length:10.1077 width:35.7194 height:7.65491
$Box:Box002 @ [0.000000,0.000000,0.000000]

x
$Box:Box002 @ [0.000000,0.000000,0.000000]
```

Podemos hacer operaciones con las variables que contienen valores numéricos.

También le podemos decir que una variable es mayor que otra, y nos contesta verdad o falso según los valores de estas variables.

```
x=1
1
R=2
2
x+R
3
x>R
false
X<R
true
```

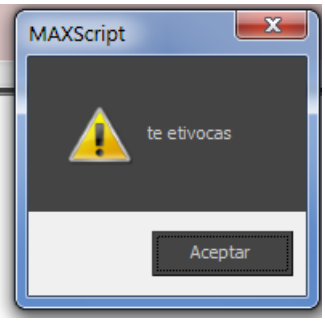
A las operaciones que nos dan valores false,true,on,off se llaman booleanas.



Muchas opciones de max se basan en estas operaciones. Por ejemplo en una tetera cuando activamos o desactivamos partes de esta en el maxscript con el macrorecorder activado, vemos que aparecen con el on y off según estén activadas o desactivadas.

CADENAS DE TEXTO O STRINGS

```
texto="te etivocas"
"te etivocas"
texto
"te etivocas"
messageBox "te etivocas"
```



En lugar de numeros como valores para las variables, se pueden asignar cadenas de texto.

Tambien podemos crear mensajes en ventanas de Max usando messageBox.

O hacer preguntas con yesnocancelbox"te enteras de algo?" -- aparece una ventana preguntando.

```
Y=1
1
T="2"
"2"
Y+T
-- Incompatible types: 1, and "2"
```

Se puede usar una cadena de texto para introducir números, pero luego no pueden ser usados como números para realizar operaciones.

En este caso debemos decirle a max, que esa cadena de texto debe interpretarse como un numero.

```
Y
1
T
"2"
Y+ (T as integer)
3
```

Entre parentesis debemos poner la variable seguido de .. as integer →

LAS FUNCIONES

Son como pequeños programas dentro de Max.

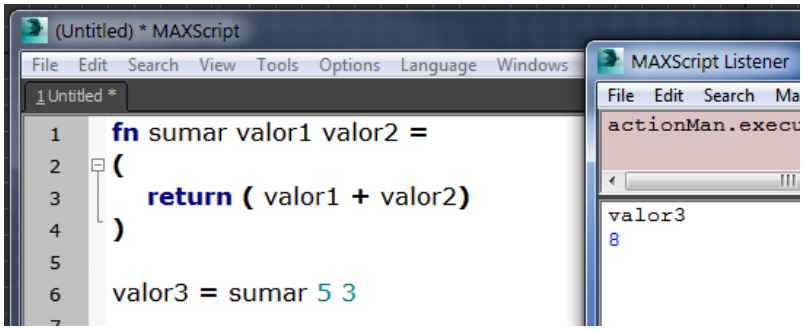
Los argumentos son cualidades necesarias en algunas funciones para poder ejecutarse.

Sphere() - - no necesita argumentos para ejecutarse. Podríamos ponerle argumentos como radio,longitud o número de segmentos, pero si no ponemos nada ,simplemente nos crea una esfera aleatoria en el 0,0,0 de los ejes de coordenadas.

Sin embargo messagebox() necesita argumentos en forma de cadenas de strings para poder ejecutarse.

En la siguiente función vamos a crear una variable llamada sumar , una llamada valor1 y otra llamada valor 2.

La función va a hacer que al ejecutar la variable sumar ,junto a las variables valor1 y valor2 el resultado será la suma de esos 2 números.



Si ejecutamos .... sumar 5 3 nos dará el valor 8.

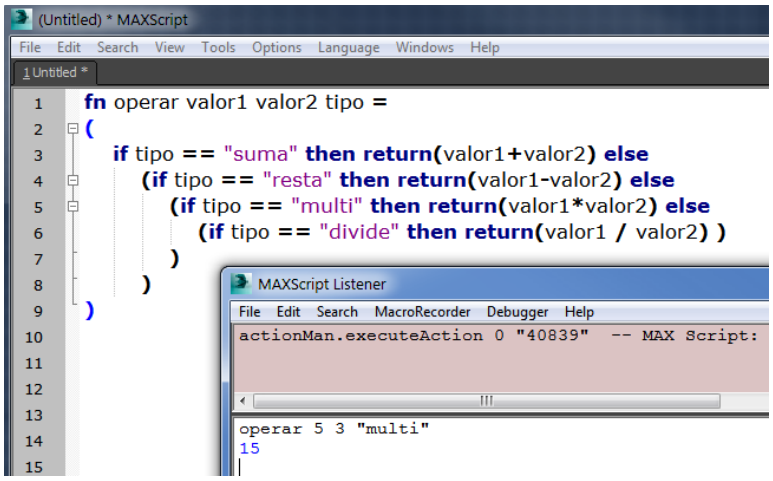
A su vez podemos crear otra variable que acumule la función con dos valores dados.

Cuando llamemos a Valor3 nos va a sumar 5+3.

Las funciones guardan en memoria variables que se ejecutan cuando les asignamos valores y las llamamos.

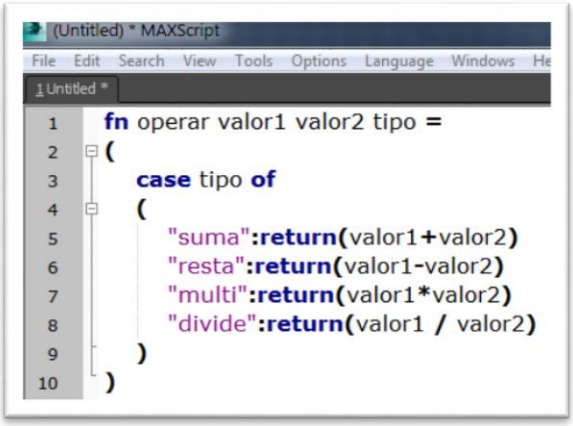
LOS CONDICIONALES

Añade condiciones, en este caso a una función.



Esta función crea variables y una de ellas se llama tipo.

Si tipo es suma entonces sumaremos los valor1 y valor2, y sino lo que miraremos es , si tipo es igual a resta , que entonces restaremos valor1-valor2 y sino entonces iremos comprobando las condiciones que van a continuación.



Al final en el listener llamaremos a la funcion diciendo que vamos a multiplicar valor 5 y 3.

Otra forma de hacer lo mismo es el case of.

Evitamos tener que escribir tanto y acabamos obteniendo el mismo resultado.

Tipo de condicionales

If x>5 do ()

If x>5 then ()

If x>5 then () else ()

Case x of ()

ARRAYS

```
lista=#(1,"casa",true,5)
#(1, "casa", true, 5)


lista[2]
"casa"

lista.count
4
```

Un array es una lista de elementos. Y esta se puede asignar a una variable. Para vaciarlo hacemos lista=#()

Podemos acceder al contenido del array poniendo la posición que ocupa este elemento entre corchetes.

En la captura vemos como la posición 2 de la lista , es “casa”.

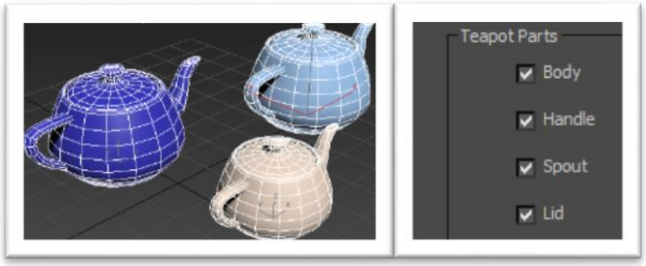
Podemos saber el número de elementos del array agregando .count a la variable que forma el array.

BUCLES O LOOPS

Nos sirve para repetir acciones

Si seleccionamos varias teteras, y escribimos:

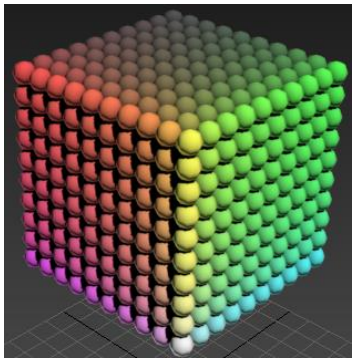
```
for x in selection do
    (
        if x.handle == off then
        ( x.handle= on) else
        ( x.handle= off)
    )
```



Este script quiere decir, que asignamos a x la selección actual, que son las 3 teteras.

Si el asa de las teteras esta quitada o desactivada entonces on (la ponemos), y sino la quitamos.

```
objecs=#()
colors=#()

for x=1 to 10 do
( for y= 1 to 10 do
    ( for z=1 to 10 do
        (
            sfera= sphere pos:[10*x,10*y,10*z] radius:5 segs:8
            sfera.material=standard diffuse:(color(255/x)(255/y)(255/z))
            append objecs sfera
            append colors sfera.material.diffuse
        )
    )
)
```



Vamos a analizar el script.

Al principio creamos las variables que contengan los arrays, luego creamos unos bucles en los cuales creamos esferas en la que la primera está en la posición 10,10,10, la siguiente en 10,10,20 la siguiente 10,10,30 hasta llegar a 10,10,100. Luego se crearía la siguiente en 10,20,100 hasta 10,100,100 y más adelante se incrementaría el eje x para acabar formando un cubo.

Mientras vamos creando el cubo, añadimos un material diferente a cada esfera.

Append añade a la variable objecs ,cada esfera ,para generar un array.

El siguiente array se crea añadiendo los materiales dentro de la variable colors.

Añadimos a colors sfera.material.diffuse , para añadir en la variable colors , los numeros que formarian cada color.

```
(
    sfera= sphere pos:[10*x,10*y,10*z
    sfera.material=standard diffuse:(col
    append objecs sfera
    append colors sfera.material
)
)
```

De esta manera podemos llamar a colors[1] y nos saldrian los valores de ese color.

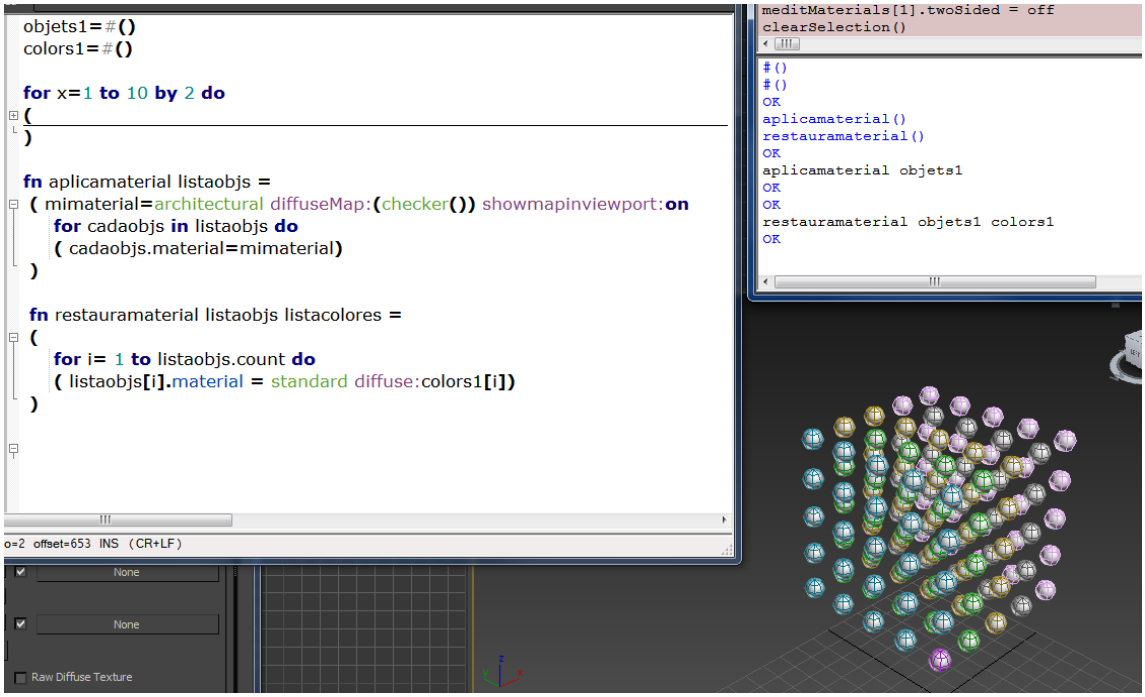
```
objecs
#($Sphere:Sphere001 @ [10.000000,10.00
colors
#((color 255 255 255), (color 255 255

colors[1]
(color 255 255 255)
colors[2]
(color 255 255 127)
```

Si no ponemos .diffuse al final , lo que pasa es que los valores que nos da en los colores no son numéricos sino el material standard de cada esfera.

```
colors[1]
Standardmaterial:Standard
```

```
objets1=#()
colors1=#()

for x=1 to 10 by 2 do
(
)

fn aplicamaterial listaobjs =
( mimaterial=architectural diffuseMap:(checker()) showmapinviewport:on
    for cadaobjs in listaobjs do
    ( cadaobjs.material=mimaterial)
)

fn restauramaterial listaobjs listacolores =
(
    for i= 1 to listaobjs.count do
    ( listaobjs[i].material = standard diffuse:colors1[i])
)
```

```
meditMaterials[1].twoSided = off
clearSelection()

#()
#()
OK
aplicamaterial()
restauramaterial()
OK
aplicamaterial objets1
OK
OK
restauramaterial objets1 colors1
OK
```



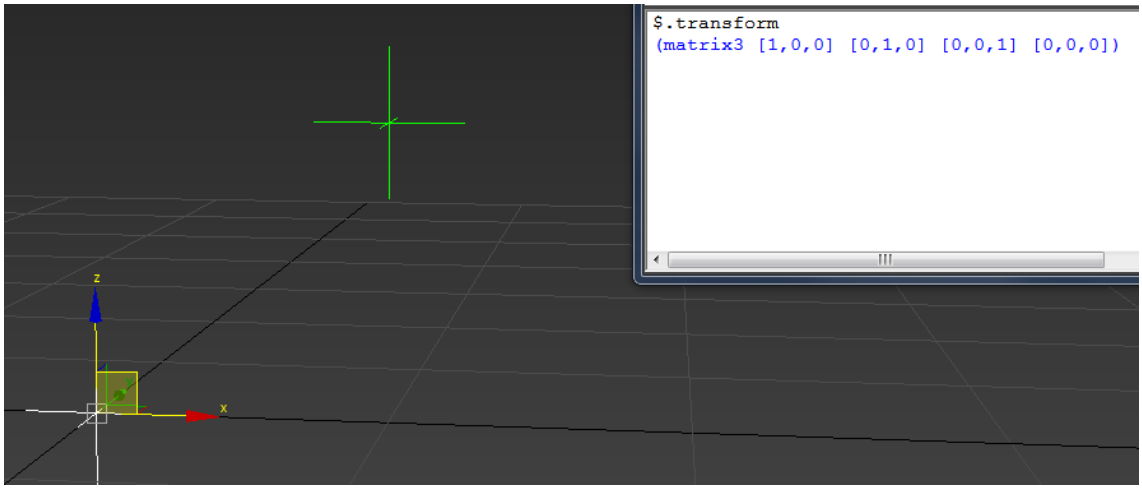
En esta última captura, creamos una función que aplica el mismo material checker a todas las esferas .Por otro lado tenemos otra función que vuelve a dar materiales difuse a cada esfera.

Dejo el texto para poder copiar pegar dentro del maxscript Editor de Max.

```
objets1=#()
colors1=#()

for x=1 to 10 by 2 do
(               for y=1 to 10 by 2 do
                (               for z=1 to 10 by 2 do
                                (               myesfer= sphere pos:[10*x,10*y,10*z] radius:5 segs:8
                                                myesfer.material = standard diffuse:(color(255/x)(255/y)(255/z))
                                                                append objets1 myesfer
                                                                append colors1 myesfer.material.diffuse
                                )
                )
)
fn aplicamaterial listaobjs =
 ( mimaterial=architectural diffuseMap:(checker()) showmapinviewport:on
                 for cadaobjs in listaobjs do
                 ( cadaobjs.material=mimaterial)
 )
 fn restauramaterial listaobjs listacolores =
 (              for i= 1 to listaobjs.count do
                ( listaobjs[i].material = standard diffuse:colors1[i])
 )
```

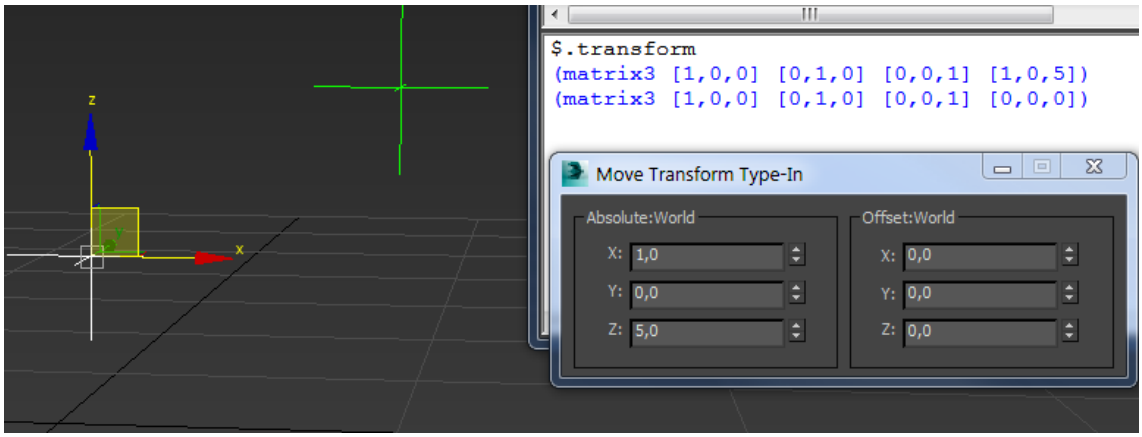
LOS VECTORES

Seleccionamos un objeto en el eje de coordenadas y en el listener window escribimos **$.transform** el dólar se refiere al eje seleccionado.



Nos devuelve una matriz de 4 x 3 elementos.

Los 3 primeros entre corchetes muestran las transformaciones en rotación y el ultimo muestra la posición.



Si lo movemos 5 en Z y 1 en X, veremos como en la matriz se han cambiado los valores a los de la nueva posición.

Las rotaciones y el escalado se escriben en los mismos elementos de la matriz, por lo que hay que tener mucho cuidado con esto.

En la siguiente captura muestro (de abajo a arriba) un cambio de posición, luego un cambio de rotación y luego dos cambios de rotación y escalado a la vez.

Al escribirse uno sobre el otro, no se entienden los cambios ya que unos chafan a otros.

```
$.transform
(matrix3 [2,1.19209e-007,0] [0,0.984808,0.173648] [0,-0.347296,1.96962] [1,0,5])
(matrix3 [2,1.19209e-007,0] [0,0.984808,0.173648] [0,-0.173648,0.984808] [1,0,5])
(matrix3 [1,0,0] [0,0.984808,0.173648] [0,-0.173648,0.984808] [1,0,5])
(matrix3 [1,0,0] [0,1,0] [0,0,1] [1,0,5])
```

Conceptos Básicos II

```
$bOX005
$Box:Box005 @ [20.759338,-3.389093,0.000000]

$bOX0011
undefined
```

Si escribimos dólar seguido del nombre de un objeto de la escena, nos dará la posición relativa de ese objeto con respecto al origen.

Si escribimos dólar seguido del nombre de un objeto que no existe, nos dirá undefined.

```
$Box*
$$objects/.../Box*
```

Si escribimos dólar, el nombre Box y luego asterisco(*) habremos llamado a todos los objetos que contengan Box como inicio del nombre.

Todos estos objetos pueden ser incluidos en una variable.

```
cajas=$Box*
$$objects/.../Box*
cajas
$$objects/.../Box*
cajas.count
10
```

La variable por ejemplo la llamamos cajas y ahora hacemos un cajas.count para saber cuántos objetos contiene.

Snapshot $ → crea una copia del objeto seleccionado

point pos:[0,0,0] → Crea un punto en la posición 0,0,0

$.transform → el dólar se refiere al objeto seleccionado.

Showproperties $ → Nos da las propiedades del objeto seleccionado.

$.position → devuelve la posición del objeto seleccionado.**$.position.X** da la X por ejemplo.

$.parent → nos devuelve el valor del padre del objeto seleccionado

Select $.parent → Selecciona el padre del objeto seleccionado

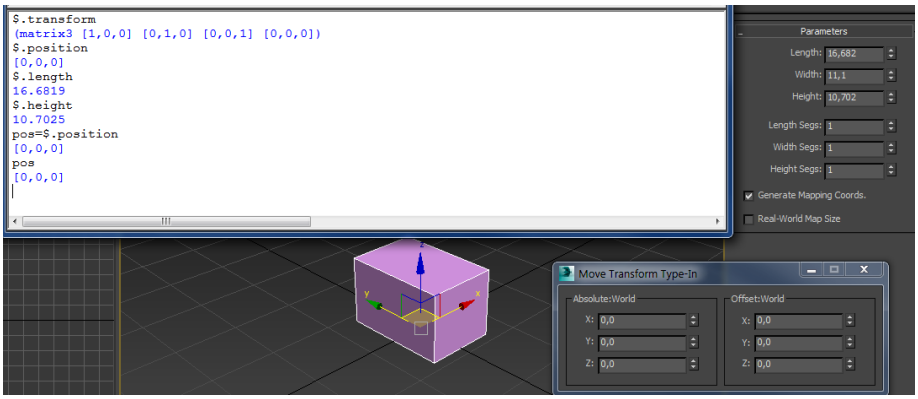
$.children → Nos devuelve el valor del hijo del objeto seleccionado.

$.children[2] → si el objeto tiene dos hijos, nos devuelve el valor del segundo hijo en la jerarquía.

_hlpBones = getCurrentSelection() → Añade a una variable la selección actual de objetos.

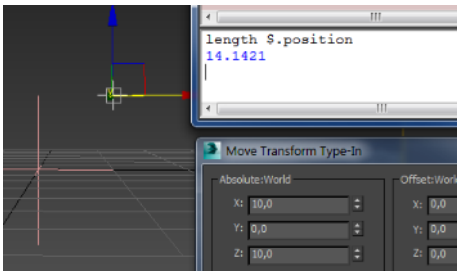
Podemos hacer lo mismo con **_hlpBones=$**

_hlpBones.count → Nos daría el número de elementos dentro de la variable.



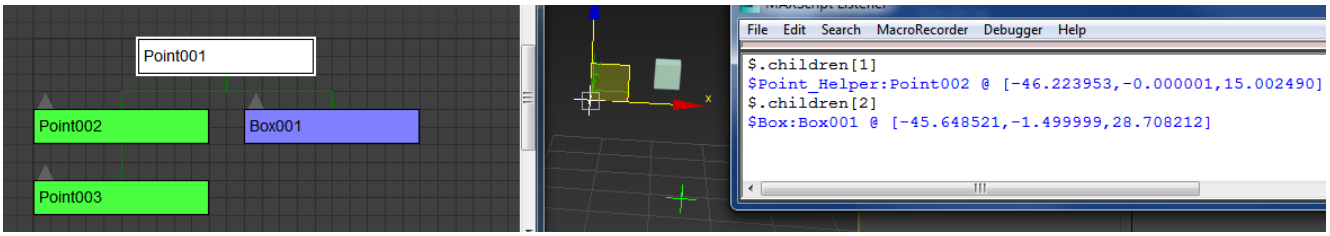
En esta captura vemos como ver las propiedades de la selección.

También creamos una variable llamada pos y le asignamos el vector posición del objeto seleccionado.



En esta captura hemos hecho un length ,pero a la posición del objeto seleccionado.

El valor que nos da es la longitud desde el origen hasta el objeto seleccionado.



He creado un point002 y un Box001 hijos de Point001.

$.children[1] nos da el valor del hijo número 1 en la jerarquía.

$.children[2] nos da el valor del hijo número 2, que es la caja.

Gc() liberamos memoria, hay que tener cuidado porque elimina los Undo y Redo.

CREAR HUESOS DESDE HELPERS (POINTS) y emparentarlos

```
point pos:[0,10,10] size:4 name:"1"  -- Creamos el punto 1
point pos:[0,0,0] size:4 name:"2"  -- Creamos el punto 2
point pos:[0,-10,-10] size:4 name:"3"  -- Creamos el punto 3
select $2                -- seleccionamos el punto 2
$.parent = $1            -- el padre del punto 2 es el punto 1
select $3                -- seleccionamos el punto 3
$.parent = $2            --el padre del punto 3 es el punto 2
select #($1, $2, $3)     -- Seleccionamos los 3 puntos en el orden 1,2,3
(
                         _hlpBones = getCurrentSelection() – metemos en una variable los 3 puntos seleccionados
                         _bones = #() -- inicializamos array para guardar los huesos nuevos.
                         -- Recorremos los puntos, y generamos cada hueso desde el point padre, hasta el hijo
                         for i = 1 to _hlpBones.count do  -- Bucle desde 1 hasta el numero de puntos ,que es 3.
                         (
                                  if _hlpBones[i].children.count != 0 then   -- Si el hueso seleccionado no tiene hijos entonces...
                                                                             --Esto se hace para no crear huesos en un
hueso sin hijos.
                                  (                  --Metemos en una variable un hueso que creamos desde el punto padre hasta el hijo.
                                           _newBone     =     BoneSys.createBone     _hlpBones[i].pos     _hlpBones[i].children[1].pos
_hlpBones[i].pos
                                           append _bones _newBone        --Añadimos al array del principio, el Nuevo hueso creado.
                                  )
                         )
                         --Ahora creamos un bucle en el que contamos desde el hueso final hasta 1 ,diciendo que si es diferente de 1 entonces
                                  el padre de _bones[i]    es    _bones[i-1]
                         for i = _bones.count to 1 by -1 do            ( if i != 1 then       _bones[i].parent = _bones[i-1] )
)
```

TRANSFORMACIONES ALEATORIAS

```
cajas=$Box*

lista = #()
for X in $Box* do
(
    append lista X
)
```

Siguiendo con lo que se dice en el apartado de conceptos básicos, podemos añadir a una lista de objetos o también llamada ARRAY, los objetos acumulados en una variable.

En este caso hemos creado 10 cajas llamadas Box001,Box002,etc

Las hemos metido en la variable cajas, hemos creado un array vacio llamado lista y luego un bucle para agregar con append todas las cajas.

De esta manera tenemos una variable con todas las cajas(cajas) y un array con todas las cajas(lista). Son varias maneras de guardar los elementos que queremos de la escena.

Random

```
random 1 100
24
random 1.0 100
76.0217
```

Devuelve un numero aleatorio entre dos números dados.

Cada vez que lo ejecutamos no s dará un valor diferente.

Si le damos un primer valor con decimales, el resultado es con decimales.

En la siguiente captura vemos que podemos decirle la posición a la que tiene que ir el objeto seleccionado.

La segunda linea nos da el valor de posicion en X del objeto seleccionado.

La tercera linea es igual que $.position

En la cuarta linea decimos que añadimos 10 unidades en X.

La quinta linea es igual que la cuarta, pero nos ahorramos escribir un poco mas.

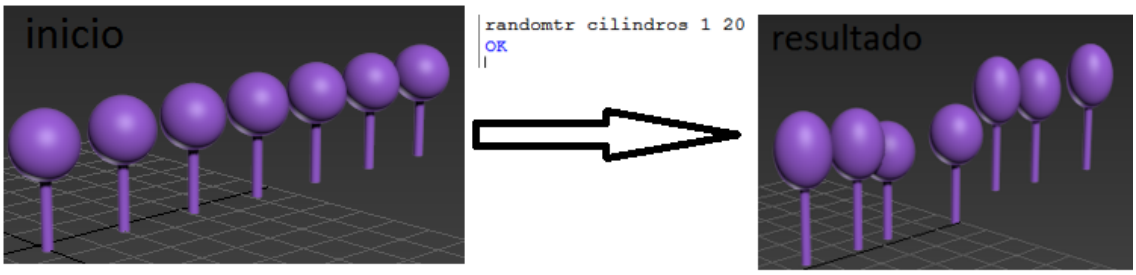
```
$.position =[-3.48064,0.214539,0]
[-3.48064,0.214539,0]
$.position.controller [#X_Position].controller.value
-3.48064
$.position.controller.value
[-3.48064,0.214539,0]
$.position.controller.value =$.position.controller.value + [10,0,0]
[6.51936,0.214539,0]
$.position.controller.value +=[10,0,0]
[16.5194,0.214539,0]
```

La posición se define con un vector respecto al origen, pero la rotación se define como la torsión en quaternions. Aunque podemos acceder a los valores por separado con value.z para solo el eje z por ejemplo y luego sumarle solo el primer valor, es decir X , de la nueva variable llamada variar , con variar[1].

```
cilindros = $cylinder*

fn randomtr listobj valormin valormax =
(
    for i in listobj do
    (
        variar=[random valormin valormax,random valormin valormax,0]

        i.position.controller.value +=variar
        i.rotation.controller.value.z +=variar[1]
        i.scale.controller.value.z +=variar[1]/50
    )
)
```

En la captura creamos una función en la que recorremos la lista de objetos , creamos una variable llamada variar con valor **X** random entre un minimo y un máximo, **Y** igual, y **Z** que vale 0.

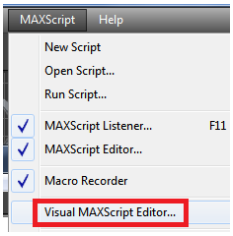
Luego al vector de posición de cada objeto le sumamos el vector variar, en rotación sumamos el valor X random al valor Z de rotación de cada objeto, y luego el escalado en Z sumado al valor X random partido por 50.



Al ejecutar la función **randomtr cilindros 1 20** vemos el resultado.

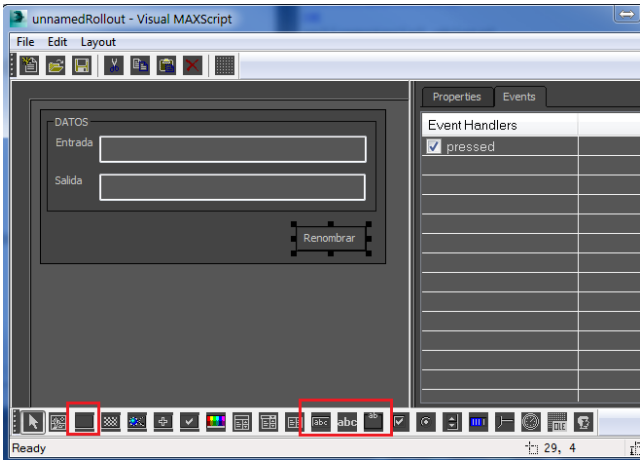
CREAR VENTANA SUSTITUIR NOMBRES

Vamos a activar el Visual Maxscript Editor.



Se nos abre la ventana

Y vamos diseñando la interfaz que necesitaremos para crear la ventana para sustituir nombres en el programa.



Lo haremos con los botones que aparecen abajo.

Una vez terminado , lo grabaremos como .vms y tambien como .ms , este último para ver el codigo dentro del Maxscript Editor.

Ahora vamos ver cómo funciona la función substitutestring.

Si seleccionamos un objeto cualquiera de la escena, y escribimos $.name nos va a devolver su nombre entre comillas.

Si aplicamos:

$.name
"Sptetera025"
substitutestring $.name "tetera" "here"
"Sphere025"
Vemos cómo funciona. Primero le damos el objeto a cambiar de nombre, luego la parte de texto que queremos cambiar de ese objeto y luego el texto que lo sustituirá.

Esto lo podemos meter en una variable, de manera que lo podamos ejecutar en el maxscript editor.

mystr= substitutestring $.name "tetera" "here" – aqui asignamos el string a la variable
$.name = mystr – aquí nombramos al objeto actual seleccionado como la variable.

Esto solo funciona cuando tenemos un objeto seleccionado. Para que funcione con más de un objeto, deberemos crear un loop. También creamos una función a la cual llamamos para ejecutar el script.

```
fn renombra texto1 texto2 =

if selection.count >0 then
(

	for cadaobjeto in selection do
		(
			mystr= substitutestring cadaobjeto.name texto1 texto2
			cadaobjeto.name = mystr
		)
)
else
(
	messagebox " selecciona uno o varios objetos"
)
```

Parte de este código es el que se ha creado automáticamente dentro del Visual MaxScript Editor.Aquí podemos ver el script al completo.

```
if miRenombrador != undefined do destroydialog miRenombrador

rollout miRenombrador "renombrador de objetos" width:379 height:251
(
	-- variables


	-- funciones

	fn renombra texto1 texto2 =
	(
		if selection.count > 0 then
		(
			for cadaobjeto in selection do
			(
				mystr = substituteString cadaobjeto.name texto1 texto2
				cadaobjeto.name = mystr
			)
		)
		else
		(
			messagebox " selecciona uno o varios objetos"
		)
	)

	-- elementos de la interfaz

	groupBox datos_grp "Datos" pos:[11,15] width:350 height:130
	editText input_txt "Entrada   " pos:[33,42] width:309 height:20
	editText output_txt "Salida     " pos:[33,89] width:309 height:20
	button renombra_btn "Renombra" pos:[202,190] width:158 height:39

	-- eventos

	on renombra_btn pressed  do   (		renombra  input_txt.text output_txt.text   )

)
createDialog miRenombrador 380 260
```

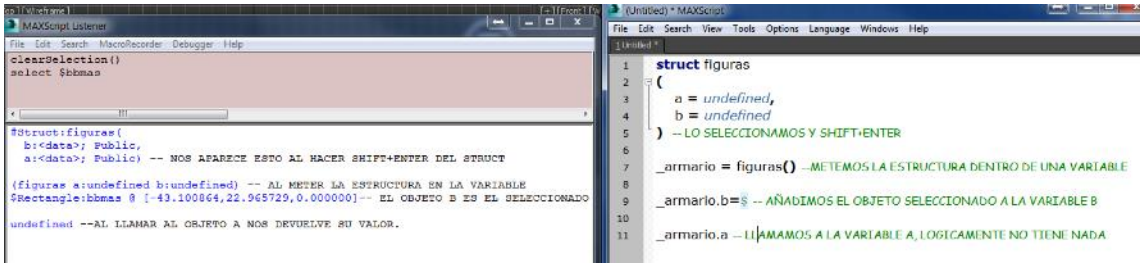
STRUCTS

Es como un armario donde podemos introducir variables,arrays o cualquier otro tipo de datos.

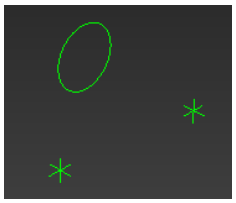
Luego podemos acceder a las propiedades de todos los objetos que se metan ahí.

Se usa para organizar objetos de una escena.

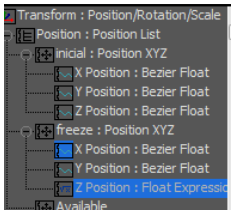
Podemos organizar por ejemplo todos los huesos en bloques.



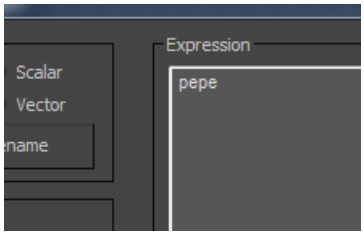
FLOAT EXPRESSION (+ freeze transform) POR SCRIPT

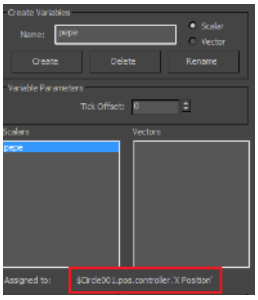


```
A=Point pos:[4.1832,-6.23926,0] size:10 wirecolor:green --  creamos un punto
B=Point pos:[-55.3652,0,0] size:10 wirecolor:green -- creamos un punto
C=Circle radius:13.1872 pos:[-16.3088,28.0928,0] wirecolor:green -- creamos un circulo
B.parent = A  -- El padre de B es A
rotate C (angleaxis 90 [1,0,0]) -- rotamos el circulo llamado C

-- añadimos position list al controlador de posición de uno de los puntos.
A.pos.controller = position_list ()
```




```
A.pos.controller.Available.controller = Position_XYZ ()
A.pos.controller.Setname 1 "Frozen Position"
A.pos.controller.Setname 2 "Zero Pos XYZ"
A.pos.controller.SetActive 2
A.pos.controller[2].Z_Position.controller = Float_Expression ()

-- Ahora con lo siguiente ponemos los valores a 0 del segundo controlador de posición
--para conseguimos lo mismo que con el Freeze transform deberíamos también freezear la rotación.En el script de Sticky points se ve mejor.
--- Aquí ponemos a 0 el Zero Pos XYZ como lo haría el Freeze Transform
A.pos.controller[2].X_Position=0
A.pos.controller[2].Y_Position=0
A.pos.controller[2].Z_Position=0

-- añadimos una variable escalar llamada pepe y la asignamos al controlador de posición del circulo en X.
A.pos.controller[2].Z_Position.controller.addscalartarget "pepe"
C.pos.controller.X_Position.controller
-- dentro del float expression metemos pepe en la expresión
A.pos.controller[2].Z_Position.controller.setexpression "pepe"
-- ahora evaluamos la expresion con la siguiente línea de código
A.pos.controller[2].Z_Position.controller.update()
```





CREAR UN PUNTO Y CAMBIARLO DE TAMAÑO Y COLOR

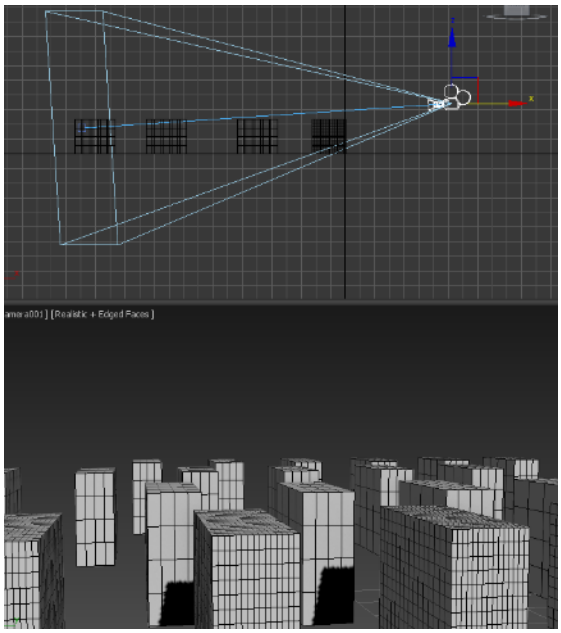
```
if wiki != undefined do destroydialog wiki
                                        poin=point()
                                        poin.size=10
                                        poin.position=[5.00049,194.406,0]
                                        poin.wirecolor=[0,0,25]
                                        hide poin   -- ocultamos el punto, hay que acordarse de borrarlo cuando acabe el script
rollout wiki "Untitled" width:250 height:183
(
        groupBox grp1 "CREAR POINT" pos:[7,5] width:234 height:158
        button btn1 "CREA EL POINT" pos:[24,36] width:196 height:32
        spinner spn1 "SIZE" pos:[28,87] width:81 height:16 range:[0,100,1] enabled:true
        colorPicker cp1 "" pos:[132,85] width:82 height:19 enabled:true color:[0,0,25]
        on btn1 pressed do
                (
                        poine=point()           poine.size=poin.size
                        poine.wirecolor=poin.wirecolor
                )
        on cp1 changed newcol do
                (
                        poin.wirecolor=newcol
                        if poine !=undefinded do (poine.wirecolor=poin.wirecolor)
                        for cadaobj in selection do (cadaobj.wirecolor=newcol)
                )
on spn1 changed val do
                (
                        poin.size=val
                        if poine !=undefinded do (poine.size=poin.size)
                        for cadaobj in selection do (cadaobj.size=val)
                )

)
createdialog wiki
```

PARAMETRO DISTANCE

Voy a mostrar un pequeño ejercicio para cambiar la definición de ciertos objetos según la distancia a la que estén de la cámara.

Esto puede servir para consumir menos memoria en las escenas que haya elementos lejos ,y que nos dé igual que se vean menos definidos porque no se va a notar.

```
camara=$  -- asignamos la variable camara al objeto seleccionado, en este caso la camara de la escena

for obj in geometry do   -- para toda la geometria de la escena hacemos..
                (
                if distance obj camara >120 then   -- si la distancia de la geometria a la camara es mayor de 120 entonces
                (obj.lengthsegs =4    -- segmentos de longitud a 4
                                obj.widthsegs =4    -- segmentos de profundidad a 4
                                obj.heightsegs =4) -- segmentos de altura a 4
                                else  -- sino
                                (obj.lengthsegs = 15
                                obj.widthsegs =15
                                obj.heightsegs =15
                )
```



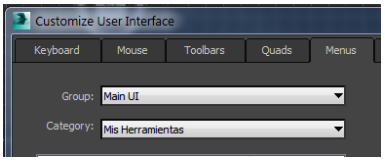
En la imagen podemos observar como los primeros bloques tienen una resolución de 15 segmentos , mientras que los de detrás tienen 4 segmentos.

Los primeros están a una distancia menor de 120 y los de detrás a una distancia superior.

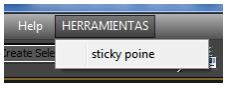
AÑADIR NUESTRAS PROPIAS HERRAMIENTAS EN MAX

Debemos ejecutar un script como este:

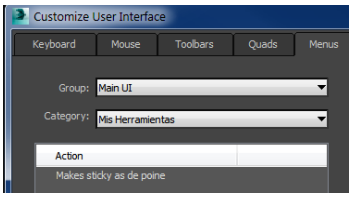
```
macroscript STICKYPOINTS
		category:"Mis Herramientas"
		buttonText:"sticky poine"
		tooltip:"Makes sticky as de poine"
(
		global stick
		if stick == undefined or not stick.open do
			filein "ventanasticky.ms"
)
```



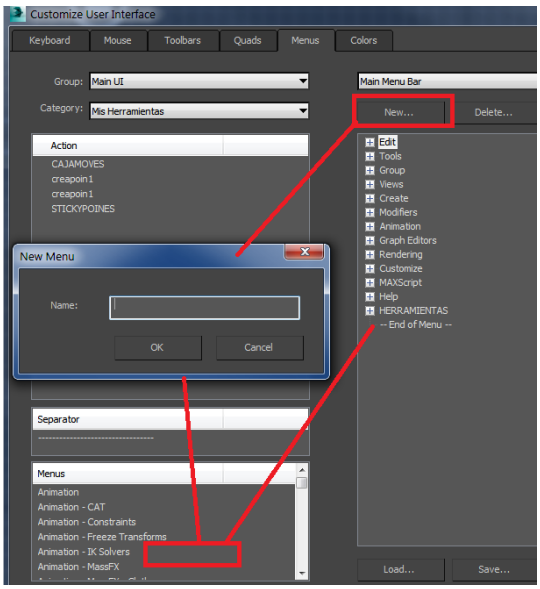
La categoría es lo que luego aparece dentro del customize user interface →Menú→Category



El buttontext es lo que aparece en el desplegable.



Tooltip es lo que aparece en el Custom User Interface , en la acción del Menú.en este caso (Makes sticky as de poine).



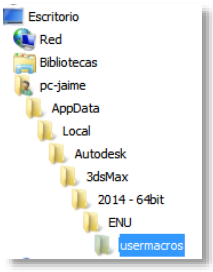
La acción a seguir es la siguiente:

Abrimos el Customize User Interface, vamos a la pestaña Menus, luego buscamos la categoría creada con el script dentro del desplegable Category.

Luego creamos una nueva barra de menú clicando sobre New...

Al crearla aparecerá en el desplegable de abajo , en Menus. Y debemos arrastrarla por debajo del Help.

Ahora arrastramos desde Action, el script creado (por ejemplo STICKYPOINES) por debajo del nuevo menú(en este caso HERRAMIENTAS).



Es importante saber, que las herramientas que la categoría que creamos en el Customize User interface, se guardan en la carpeta usermacros.

Se puede ver la ruta en la captura de la izquierda.

Para mas información sobre maxscript podemos usar el Help del MaxscriptEditor o también podemos ver videotutoriales de internet como los que se muestran aquí:

https://vimeo.com/album/1514565/page:1/sort:preset/format:thumbnail

Se puede llegar a estos videos también, buscando en google : **MAXScript 101**