Making of "Quick Mask"

STAGE #1 - Simple version

The "Quick Mask" is MAXScript available at ScriptSpot.com :
http://www.scriptspot.com/3ds-max/scripts/quick-mask
that quickly render black and white mask from the selected objects.

"Quick Mask" is written as macroScript but to simplify the explanation in this tutorial, I'll focus on the entire code (the macroScript body). First we need a working code, packing it as macroScript is the easy job, right?

The goal is to render our mask while keeping the scene materials intact.
That means we'll need to restore the scene settings and materials. We can do this really quickly by two manners - using Undo record or using Hold/Fetch.
And both methods rid us of storing/restoring a lot of data to variables.

Lets demonstrate the Undo method with example "clay"-render snippet:

```
theClay = standardMaterial diffusecolor:(color 222 222 222)
with undo on
(
   -- Note about the syntax: I used 2 commands on a single line,
   -- the ";" sign tell the Max where the first command end.
   objects.material = theClay ; render()
)
max undo
```

Here need to say that the Undo method has 2 disadvantages:
1. The massive record to the Undo buffer slow down the performance.
2. In 3ds Max 2008 and higher Undo buffer is not completely stable.

So I mark it as not safety and prefer to use Hold/Fetch feature.

The Hold/Fetch method also has a disadvantage but it's trivial:
After fetching the Undo buffer is clean and all objects are deselected.

Well, the choice is yours, if you run 3ds Max 9 and below and don't bother about the performance, you can rely on Undo feature, but I'll rely on Hold/Fetch feature.

Lets sketch our code:

```
(
   -- create needed materials
   matB = standardMaterial diffusecolor:black selfIllumAmount:100
   matW = standardMaterial diffusecolor:white selfIllumAmount:100

   holdMaxFile()
   /*
  The next step go here: ---------------------------------------
     Assign material to the objects and tiny settings setup
  */
   render() ; fetchMaxFile quiet:on ; gc()
)
```

In all code after *holdMaxFile*() we'd not bother about the current scene settings and materials 'cause we'll restore them by fetching at the end.

Now is time to decide how we'll apply the materials to the objects.
For example we can do this...

```
for obj in objects do (
   if obj.isSelected then obj.material = matW
      else obj.material = matB
)
```

...but this is not optimized 'case here we loop through all objects and apply material to them one-by-one. Assign material to the object set is mapped, for example...

```
-- that simple mapped call...
objects.material = undefined
-- do the sane as this one...
for obj in objects do obj.material = undefined
-- however the mapped call is much more faster
```

...so lets use this to optimize our code:

```
selection.material = matW
max select invert
selection.material = matB
```

This works fine. However I prefer the next one:

```
geometry.material = matB
shapes.material = matB
selection.material = matW
```

Well, the code is almost ready, now remain 3 simple but important set's. It's safety just to set explicitly this settings to desired values.

```
useEnvironmentmap = off
backgroundColor = black
renderers.current = Default_Scanline_Renderer()
```

```
-- and here is a finished state of the Stage #1
(
   matB = standardMaterial diffusecolor:black selfIllumAmount:100
   matW = standardMaterial diffusecolor:white selfIllumAmount:100

   holdMaxFile()
   geometry.material = matB
   shapes.material = matB
   selection.material = matW

   useEnvironmentmap = off
   backgroundColor = black
   renderers.current = Default_Scanline_Renderer()

   render() ; fetchMaxFile quiet:on ; gc()
)
-- You can test how it works.
```

STAGE #2 -- Adding set to clipboard feature

At the Stage #1 we end with working code. Now we'll extend it by adding auto-copy to Windows clipboard feature for the rendered image.

In 3ds Max 9 SP 1 and higher VFB has a Copy button that copies buffered image.
May still it's worth to automatize this task.

I'll use built-in *setclipboardBitmap*() function that was added in 3ds Max 2008.
It need [AVG] Avguard Extensions (which is free) for 3ds Max 9 and below.
You can use *dotNetClass "System.Windows.Forms.Clipboard"* as well, but this will benefit only 3ds Max 9 users because versions below 9 not support dotNet.

If you not plane to use the script in 3ds Max 9 and below or has AVG installed, you can just assign the rendered image to a variable and call the function:

```
img = render()
setclipboardBitmap img
freeSceneBitmaps() -- clear from memory
```

At the final script I add error handle (check whether the function exist):

```
if setclipboardBitmap != undefined do -- check whether command exist
(
  setclipboardBitmap img
  freeSceneBitmaps() -- clear from memory
)

-- and here is a finished state of the Stage #2
(
  matB = standardMaterial diffusecolor:black selfIllumAmount:100
  matW = standardMaterial diffusecolor:white selfIllumAmount:100

  holdMaxFile()
  geometry.material = matB
  shapes.material = matB
  selection.material = matW

  useEnvironmentmap = off
  backgroundColor = black
  renderers.current = Default_Scanline_Renderer()

  img = render() ; fetchMaxFile quiet:on ; gc()

  if setclipboardBitmap != undefined do -- check whether command is exist
  (
    setclipboardBitmap img
    freeSceneBitmaps() -- clear from memory
  )
)
-- You can test how it works.
```

STAGE #3 - Adding XRef Scene support

No additional code needed for XRef Objects but for XRef Scene would.
Let's first make an exercise. Reset Max. Add XRef Scene. Now run this code:

```
aXref = xrefs.getXRefFile 1
theMat = standardMaterial diffusecolor:yellow
aXref.tree.children.material = theMat
render()
```

Our new material was assigned. Now reopen your original scene file that you XRefed before and you'll see that the original materials inside are intact.
Cool, ya :) Well, what's next... maybe something like this:

```
xCount = xrefs.getXRefFileCount()
for x = 1 to xCount do (
   aXref = xrefs.getXRefFile x
   aXref.tree.children.material = matB
) -- that affect only root objects
```

Simply, we need a recursive function to get the whole scene hierarchy:

```
mapped fn getXRefEntirely srcArray &outArray = (
   objs = for x in srcArray.children collect x
   join outArray objs
   getXRefEntirely objs outArray -- recursive call
) -- Note: &outArray is a reference argument (i.e. the array must exist)
```

And the function above we can implement like this:

```
theXrefs = #() -- where to collect all xrefed scene objects
xCount = xrefs.getXRefFileCount()
for x = 1 to xCount do (
   aXref = xrefs.getXRefFile x
   xRootObjs = aXref.tree.children
   xObjs = for x in xRootObjs collect x
   getXRefEntirely xObjs xObjs -- collect entirely hierarchy
   join theXrefs xObjs
)
theXrefs.material = matB -- Note here: XRef Scenes are not selectable
--  ( and unselected we matte in black, right? )
```

As you can see I use the same array *xObjs* for the both arguments of my function.
This way I put the result of each recursive call directly to the same array.
Also I collect all objects from all XRef Scenes and then assign the material at once.

```
-- and here is a finished state of the Stage #3
macroScript Quick_Mask_Sel category:"Toos" tooltip: "Quick Mask"
(
   mapped fn getXRefEntirely srcArray &outArray = (
      objs = for x in srcArray.children collect x
      join outArray objs
      getXRefEntirely objs outArray
   )

   matB = standardMaterial diffusecolor:black selfIllumAmount:100
   matW = standardMaterial diffusecolor:white selfIllumAmount:100

   holdMaxFile()
   geometry.material = matB
   shapes.material = matB
   selection.material = matW

   -- Add XRefScene support --
   theXrefs = #()
   xCount = xrefs.getXRefFileCount()
   for x = 1 to xCount do (
```

```
        aXref = xrefs.getXRefFile x
        xRootObjs = aXref.tree.children
        xObjs = for x in xRootObjs collect x
        getXRefEntirely xObjs xObjs
        join theXrefs xObjs
    )
    theXrefs.material = matB

    useEnvironmentmap = off
    backgroundColor = black
    renderers.current = Default_Scanline_Renderer()

    img = render() ; fetchMaxFile quiet:on ; gc()

    if setclipboardBitmap != undefined do
    (
        setclipboardBitmap img
        freeSceneBitmaps()
    )
)
```

Well done.

I hope you learn something from this tutorial or at least enjoy reading it.
The final script you can get from the ScriptSpot:
http://www.scriptspot.com/3ds-max/scripts/quick-mask

Cheers,

P. Karabakalov (Anubis)
anubiss@abv.bg