Forum: Questions & Réponses

Topic: Convertir les matériaux du Blender Render vers Cycles

Subject: Re: Convertir les matériaux du Blender Render vers Cycles

PostÃ© par: Tyman

Contribution le : 5/4/2017 13:13:08

Voici la dernière mise à jour du script à intégrer :

```
# convert_materials_to_cycles.py
#
# Copyright (C) 5-mar-2012, Silvio Falcinelli. Fixes by others.
#
# special thanks to user blenderartists.org cmomoney
#
# ***** BEGIN GPL LICENSE BLOCK *****
#
# This program is free software; you can redistribute it and/or
# modify it under the terms of the GNU General Public License
# as published by the Free Software Foundation; either version 2
# of the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software Foundation,
# Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
#
# ***** END GPL LICENCE BLOCK *****

bl_info = {
    "name": "Convert Materials to Cycles",
    "author": "Silvio Falcinelli, updates by community",
    "version": (0, 11, 1),
    "blender": (2, 71, 0),
    "location": "Properties > Material > Convert to Cycles",
    "description": "Convert non-nodes materials to Cycles",
    "warning": "beta",
    "wiki_url": "
http://wiki.blender.org/index.php/Extensions:2.6/Py/Scripts/Material/Blender_Cycles_Materials_Converter&amp;quot;,
    "category": "Material"}
```

```
import bpy
import math
from math import log
from math import pow
from math import exp
import os.path

def AutoNodeOff():
    mats = bpy.data.materials
    for cmat in mats:
        cmat.use_nodes = False
    bpy.context.scene.render.engine = &#039;BLENDER_RENDER&#039;

def BakingText(tex, mode):
    print(&#039;_____&#039;)
    print(&#039;INFO start bake texture &#039; + tex.name)
    bpy.ops.object.mode_set(mode=&#039;OBJECT&#039;)
    sc = bpy.context.scene
    tmat = &#039;&#039;
    img = &#039;&#039;
    Robj = bpy.context.active_object
    for n in bpy.data.materials:
        if n.name == &#039;TMP_BAKING&#039;:
            tmat = n
    if not tmat:
        tmat = bpy.data.materials.new(&#039;TMP_BAKING&#039;)
        tmat.name = "TMP_BAKING"

    bpy.ops.mesh.primitive_plane_add()
    tm = bpy.context.active_object
    tm.name = "TMP_BAKING"
    tm.data.name = "TMP_BAKING"
    bpy.ops.object.select_pattern(extend=False, pattern="TMP_BAKING", case_sensitive=False)
    sc.objects.active = tm
    bpy.context.scene.render.engine = &#039;BLENDER_RENDER&#039;
    tm.data.materials.append(tmat)
    if len(tmat.texture_slots.items()) == 0:
        tmat.texture_slots.add()
    tmat.texture_slots[0].texture_coords = &#039;UV&#039;
    tmat.texture_slots[0].use_map_alpha = True
    tmat.texture_slots[0].texture = tex.texture
    tmat.texture_slots[0].use_map_alpha = True
    tmat.texture_slots[0].use_map_color_diffuse = False
    tmat.use_transparency = True
    tmat.alpha = 0
    tmat.use_nodes = False
    tmat.diffuse_color = 1, 1, 1
    bpy.ops.object.mode_set(mode=&#039;EDIT&#039;)
    bpy.ops.uv.unwrap()
```

```
    for n in bpy.data.images:
        if n.name == &#039;TMP_BAKING&#039;:
            n.user_clear()
            bpy.data.images.remove(n)

    if mode == "ALPHA" and tex.texture.type == &#039;IMAGE&#039;:
        sizeX = tex.texture.image.size[0]
        sizeY = tex.texture.image.size[1]
    else:
        sizeX = 600
        sizeY = 600
    bpy.ops.image.new(name="TMP_BAKING", width=sizeX, height=sizeY, color=(0.0, 0.0, 0.0, 1.0),
alpha=True, float=False)
    bpy.data.screens[&#039;UV Editing&#039;].areas[1].spaces[0].image =
bpy.data.images["TMP_BAKING"]
    sc.render.engine = &#039;BLENDER_RENDER&#039;
    img = bpy.data.images["TMP_BAKING"]
    img = bpy.data.images.get("TMP_BAKING")
    img.file_format = "JPEG"
    if mode == "ALPHA" and tex.texture.type == &#039;IMAGE&#039;:
        img.filepath_raw = tex.texture.image.filepath + "_BAKING.jpg"

    else:
        img.filepath_raw = tex.texture.name + "_PTEXT.jpg"

    sc.render.bake_type = &#039;ALPHA&#039;
    sc.render.use_bake_selected_to_active = True
    sc.render.use_bake_clear = True
    bpy.ops.object.bake_image()
    img.save()
    bpy.ops.object.mode_set(mode=&#039;OBJECT&#039;)
    bpy.ops.object.delete()
    bpy.ops.object.select_pattern(extend=False, pattern=Robj.name, case_sensitive=False)
    sc.objects.active = Robj
    img.user_clear()
    bpy.data.images.remove(img)
    bpy.data.materials.remove(tmat)

    # print(&#039;INFO : end Bake &#039; + img.filepath_raw )
    print(&#039;_____&#039;)

def AutoNode(active=False):

    sc = bpy.context.scene

    if active:

        mats = bpy.context.active_object.data.materials
```

```
    else:

        mats = bpy.data.materials


    for cmat in mats:
        cmat.use_nodes = True
        TreeNodes = cmat.node_tree
        links = TreeNodes.links

        # Don&#039;t alter nodes of locked materials
        locked = False
        for n in TreeNodes.nodes:
            if n.type == &#039;ShaderNodeOutputMaterial&#039;:
                if n.label == &#039;Locked&#039;:
                    locked = True
                    break

        if not locked:
            # Convert this material from non-nodes to Cycles nodes

            shader = &#039;&#039;
            shmix = &#039;&#039;
            shtsl = &#039;&#039;
            Add_Emission = &#039;&#039;
            Add_Translucent = &#039;&#039;
            Mix_Alpha = &#039;&#039;
            sT = False

            for n in TreeNodes.nodes:
                TreeNodes.nodes.remove(n)

            # Starting point is diffuse BSDF and output material
            shader = TreeNodes.nodes.new(&#039;ShaderNodeBsdfDiffuse&#039;)
            shader.location = 0, 470
            shout = TreeNodes.nodes.new(&#039;ShaderNodeOutputMaterial&#039;)
            shout.location = 200, 400
            links.new(shader.outputs[0], shout.inputs[0])

            sM = True
            for tex in cmat.texture_slots:
                if tex:
                    if tex.use:
                        if tex.use_map_alpha:
                            sM = False
                            if sc.EXTRACT_ALPHA:
                                if tex.texture.type == &#039;IMAGE&#039; and tex.texture.use_alpha:
                                    if not os.path.exists(bpy.path.abspath(tex.texture.image.filepath +
"_BAKING.jpg")) or sc.EXTRACT_OW:
```

```
                        BakingText(tex, &amp;#039;ALPHA&amp;#039;)
                else:
                    if not tex.texture.type == &amp;#039;IMAGE&amp;#039;:
                        if not os.path.exists(bpy.path.abspath(tex.texture.name + "_PTEXT.jpg")) or
sc.EXTRACT_OW:
                            BakingText(tex, &amp;#039;PTEXT&amp;#039;)

    cmat_is_transp = cmat.use_transparency and cmat.alpha < 1

    if cmat_is_transp and cmat.raytrace_transparency.ior == 1 and not cmat.raytrace_mirror.use
and sM:
        if not shader.type == &amp;#039;ShaderNodeBsdfTransparent&amp;#039;:
            print("INFO:  Make TRANSPARENT shader node " + cmat.name)
            TreeNodes.nodes.remove(shader)
            shader = TreeNodes.nodes.new(&amp;#039;ShaderNodeBsdfTransparent&amp;#039;)
            shader.location = 0, 470
            links.new(shader.outputs[0], shout.inputs[0])

    if not cmat.raytrace_mirror.use and not cmat_is_transp:
        if not shader.type == &amp;#039;ShaderNodeBsdfDiffuse&amp;#039;:
            print("INFO:  Make DIFFUSE shader node" + cmat.name)
            TreeNodes.nodes.remove(shader)
            shader = TreeNodes.nodes.new(&amp;#039;ShaderNodeBsdfDiffuse&amp;#039;)
            shader.location = 0, 470
            links.new(shader.outputs[0], shout.inputs[0])

    if cmat.raytrace_mirror.use and cmat.raytrace_mirror.reflect_factor > 0.001 and
cmat_is_transp:
        if not shader.type == &amp;#039;ShaderNodeBsdfGlass&amp;#039;:
            print("INFO:  Make GLASS shader node" + cmat.name)
            TreeNodes.nodes.remove(shader)
            shader = TreeNodes.nodes.new(&amp;#039;ShaderNodeBsdfGlass&amp;#039;)
            shader.location = 0, 470
            links.new(shader.outputs[0], shout.inputs[0])

    if cmat.raytrace_mirror.use and not cmat_is_transp and cmat.raytrace_mirror.reflect_factor >
0.001 :
        if not shader.type == &amp;#039;ShaderNodeBsdfGlossy&amp;#039;:
            print("INFO:  Make MIRROR shader node" + cmat.name)
            TreeNodes.nodes.remove(shader)
            shader = TreeNodes.nodes.new(&amp;#039;ShaderNodeBsdfGlossy&amp;#039;)
            shader.location = 0, 520
            links.new(shader.outputs[0], shout.inputs[0])

    if cmat.emit > 0.001 :
        if not shader.type == &amp;#039;ShaderNodeEmission&amp;#039; and not
cmat.raytrace_mirror.reflect_factor > 0.001 and not cmat_is_transp:
            print("INFO:  Mix EMISSION shader node" + cmat.name)
            TreeNodes.nodes.remove(shader)
```

```
                shader = TreeNodes.nodes.new(&#039;ShaderNodeEmission&#039;)
                shader.location = 0, 450
                links.new(shader.outputs[0], shout.inputs[0])
            else:
                if not Add_Emission:
                    print("INFO:  Add EMISSION shader node" + cmat.name)
                    shout.location = 550, 330
                    Add_Emission = TreeNodes.nodes.new(&#039;ShaderNodeAddShader&#039;)
                    Add_Emission.location = 370, 490

                    shem = TreeNodes.nodes.new(&#039;ShaderNodeEmission&#039;)
                    shem.location = 180, 380

                    links.new(Add_Emission.outputs[0], shout.inputs[0])
                    links.new(shem.outputs[0], Add_Emission.inputs[1])
                    links.new(shader.outputs[0], Add_Emission.inputs[0])

                    shem.inputs[&#039;Color&#039;].default_value = cmat.diffuse_color.r,
cmat.diffuse_color.g, cmat.diffuse_color.b, 1
                    shem.inputs[&#039;Strength&#039;].default_value = cmat.emit

        if cmat.translucency > 0.001 :
            print("INFO:  Add BSDF_TRANSLUCENT shader node" + cmat.name)
            shout.location = 770, 330
            Add_Translucent = TreeNodes.nodes.new(&#039;ShaderNodeAddShader&#039;)
            Add_Translucent.location = 580, 490

            shtsl = TreeNodes.nodes.new(&#039;ShaderNodeBsdfTranslucent&#039;)
            shtsl.location = 400, 350

            links.new(Add_Translucent.outputs[0], shout.inputs[0])
            links.new(shtsl.outputs[0], Add_Translucent.inputs[1])

            if Add_Emission:
                links.new(Add_Emission.outputs[0], Add_Translucent.inputs[0])
                pass
            else:
                links.new(shader.outputs[0], Add_Translucent.inputs[0])
                pass
            shtsl.inputs[&#039;Color&#039;].default_value = cmat.translucency, cmat.translucency,
cmat.translucency, 1

        shader.inputs[&#039;Color&#039;].default_value = cmat.diffuse_color.r, cmat.diffuse_color.g,
cmat.diffuse_color.b, 1

        if shader.type == &#039;ShaderNodeBsdfDiffuse&#039;:
            shader.inputs[&#039;Roughness&#039;].default_value = cmat.specular_intensity

        if shader.type == &#039;ShaderNodeBsdfGlossy&#039;:
```

```
            shader.inputs[&#039;Roughness&#039;].default_value = 1 -
cmat.raytrace_mirror.gloss_factor

        if shader.type == &#039;ShaderNodeBsdfGlass&#039;:
            shader.inputs[&#039;Roughness&#039;].default_value = 1 -
cmat.raytrace_mirror.gloss_factor
            shader.inputs[&#039;IOR&#039;].default_value = cmat.raytrace_transparency.ior

        if shader.type == &#039;ShaderNodeEmission&#039;:
            shader.inputs[&#039;Strength&#039;].default_value = cmat.emit

        for tex in cmat.texture_slots:
            sT = False
            pText = &#039;&#039;
            if tex:
                if tex.use:
                    if tex.texture.type == &#039;IMAGE&#039;:
                        img = tex.texture.image
                        shtext = TreeNodes.nodes.new(&#039;ShaderNodeTexImage&#039;)
                        shtext.location = -200, 400
                        shtext.image = img
                        sT = True

                    if not tex.texture.type == &#039;IMAGE&#039;:
                        if sc.EXTRACT_PTEX:
                            print(&#039;INFO : Extract Procedural Texture  &#039;)
                            if not os.path.exists(bpy.path.abspath(tex.texture.name + "_PTEXT.jpg")) or
sc.EXTRACT_OW:
                                BakingText(tex, &#039;PTEX&#039;)

                            img = bpy.data.images.load(tex.texture.name + "_PTEXT.jpg")
                            shtext = TreeNodes.nodes.new(&#039;ShaderNodeTexImage&#039;)
                            shtext.location = -200, 400
                            shtext.image = img
                            sT = True
            if sT:
                    if tex.use_map_color_diffuse :
                        links.new(shtext.outputs[0], shader.inputs[0])

                    if tex.use_map_emit:
                        if not Add_Emission:
                            print("INFO:  Mix EMISSION + Texture shader node " + cmat.name)

                            intensity = 0.5 + (tex.emit_factor / 2)

                            shout.location = 550, 330
                            Add_Emission = TreeNodes.nodes.new(&#039;ShaderNodeAddShader&#039;)
                            Add_Emission.name = "Add_Emission"
                            Add_Emission.location = 370, 490
```

```
                        shem = TreeNodes.nodes.new(&#039;ShaderNodeEmission&#039;)
                        shem.location = 180, 380

                        links.new(Add_Emission.outputs[0], shout.inputs[0])
                        links.new(shem.outputs[0], Add_Emission.inputs[1])
                        links.new(shader.outputs[0], Add_Emission.inputs[0])

                        shem.inputs[&#039;Color&#039;].default_value = cmat.diffuse_color.r,
cmat.diffuse_color.g, cmat.diffuse_color.b, 1
                        shem.inputs[&#039;Strength&#039;].default_value = intensity * 2

                    links.new(shtext.outputs[0], shem.inputs[0])

                if tex.use_map_mirror:
                    links.new(shader.inputs[0], shtext.outputs[0])

                if tex.use_map_translucency:
                    if not Add_Translucent:
                        print("INFO:  Add Translucency + Texture shader node " + cmat.name)

                        intensity = 0.5 + (tex.emit_factor / 2)

                        shout.location = 550, 330
                        Add_Translucent =
TreeNodes.nodes.new(&#039;ShaderNodeAddShader&#039;)
                        Add_Translucent.name = "Add_Translucent"
                        Add_Translucent.location = 370, 290

                        shtsl = TreeNodes.nodes.new(&#039;ShaderNodeBsdfTranslucent&#039;)
                        shtsl.location = 180, 240

                        links.new(shtsl.outputs[0], Add_Translucent.inputs[1])

                        if Add_Emission:
                            links.new(Add_Translucent.outputs[0], shout.inputs[0])
                            links.new(Add_Emission.outputs[0], Add_Translucent.inputs[0])
                            pass
                        else:
                            links.new(Add_Translucent.outputs[0], shout.inputs[0])
                            links.new(shader.outputs[0], Add_Translucent.inputs[0])

                    links.new(shtext.outputs[0], shtsl.inputs[0])

                if tex.use_map_alpha:
                    if not Mix_Alpha:
                        print("INFO:  Mix Alpha + Texture shader node " + cmat.name)

                        shout.location = 750, 330
                        Mix_Alpha = TreeNodes.nodes.new(&#039;ShaderNodeMixShader&#039;)
```

```
                        Mix_Alpha.name = "Add_Alpha"
                        Mix_Alpha.location = 570, 290
                        sMask = TreeNodes.nodes.new(&amp;#039;ShaderNodeBsdfTransparent&amp;#039;)
                        sMask.location = 250, 180
                        tMask = TreeNodes.nodes.new(&amp;#039;ShaderNodeTexImage&amp;#039;)
                        tMask.location = -200, 250

                        if tex.texture.type == &amp;#039;IMAGE&amp;#039;:
                            # if tex.texture.use_alpha:
                            #    imask=bpy.data.images.load(img.filepath+"_BAKING.jpg")
                            # else:
                            imask = bpy.data.images.load(img.filepath)
                        else:
                            imask = bpy.data.images.load(img.name)

                        tMask.image = imask
                        links.new(Mix_Alpha.inputs[0], tMask.outputs[1])
                        links.new(shout.inputs[0], Mix_Alpha.outputs[0])
                        links.new(sMask.outputs[0], Mix_Alpha.inputs[1])

                        if not Add_Emission and not Add_Translucent:
                            links.new(Mix_Alpha.inputs[2], shader.outputs[0])

                        if Add_Emission and not Add_Translucent:
                            links.new(Mix_Alpha.inputs[2], Add_Emission.outputs[0])

                        if Add_Translucent:
                            links.new(Mix_Alpha.inputs[2], Add_Translucent.outputs[0])

                if tex.use_map_normal:
                    t = TreeNodes.nodes.new(&amp;#039;ShaderNodeRGBToBW&amp;#039;)
                    t.location = -0, 300
                    links.new(t.outputs[0], shout.inputs[2])
                    links.new(shtext.outputs[0], t.inputs[0])
    bpy.context.scene.render.engine = &amp;#039;CYCLES&amp;#039;

class mllock(bpy.types.Operator):
    bl_idname = "ml.lock"
    bl_label = "Lock"
    bl_description = "Lock/unlock this material against modification by conversions"
    bl_register = True
    bl_undo = True

    @classmethod
    def poll(cls, context):
        return True

    def execute(self, context):
        cmat = bpy.context.selected_objects[0].active_material
```

```
        TreeNodes = cmat.node_tree
        for n in TreeNodes.nodes:
            if n.type == &#039;ShaderNodeOutputMaterial&#039;:
                if n.label == &#039;Locked&#039;:
                    n.label = &#039;&#039;
                else:
                    n.label = &#039;Locked&#039;
        return {&#039;FINISHED&#039;}

class mlrefresh(bpy.types.Operator):
    bl_idname = "ml.refresh"
    bl_label = "Convert All Materials"
    bl_description = "Convert all materials in the scene from non-nodes to Cycles"
    bl_register = True
    bl_undo = True

    @classmethod
    def poll(cls, context):
        return True

    def execute(self, context):
        AutoNode()
        bpy.ops.object.editmode_toggle()
        bpy.ops.uv.unwrap(method=&#039;ANGLE_BASED&#039;, margin=0.001)
        bpy.ops.object.editmode_toggle()

        return {&#039;FINISHED&#039;}

class mlrefresh_active(bpy.types.Operator):
    bl_idname = "ml.refresh_active"
    bl_label = "Convert All Materials From Active Object"
    bl_description = "Convert all materials from actice object from non-nodes to Cycles"
    bl_register = True
    bl_undo = True

    @classmethod
    def poll(cls, context):
        return True

    def execute(self, context):
        AutoNode(True)
        bpy.ops.object.editmode_toggle()
        bpy.ops.uv.unwrap(method=&#039;ANGLE_BASED&#039;, margin=0.001)
        bpy.ops.object.editmode_toggle()
        return {&#039;FINISHED&#039;}

class mlrestore(bpy.types.Operator):
    bl_idname = "ml.restore"
    bl_label = "Restore"
```

```
    bl_description = "Switch Back to Blender Internal Nodes Off"
    bl_register = True
    bl_undo = True
    @classmethod
    def poll(cls, context):
        return True
    def execute(self, context):
        AutoNodeOff()
        return {&#039;FINISHED&#039;}

from bpy.props import *
sc = bpy.types.Scene
sc.EXTRACT_ALPHA = BoolProperty(attr="EXTRACT_ALPHA", default=False)
sc.EXTRACT_PTEX = BoolProperty(attr="EXTRACT_PTEX", default=False)
sc.EXTRACT_OW = BoolProperty(attr="Overwrite", default=False, description="Extract textures
again instead of re-using priorly extracted textures")


class OBJECT_PT_scenemassive(bpy.types.Panel):
    bl_label = "Convert BI Materials to Cycles"
    bl_space_type = "PROPERTIES"
    bl_region_type = "WINDOW"
    bl_context = "material"

    def draw(self, context):
        sc = context.scene
        layout = self.layout
        row = layout.row()
        box = row.box()
        box.operator("ml.refresh", text=&#039;Convert All to Cycles&#039;,
icon=&#039;MATERIAL&#039;)
        box.operator("ml.refresh_active", text=&#039;Convert Active to Cycles&#039;,
icon=&#039;MATERIAL&#039;)
        box.operator("ml.restore", text=&#039;To BI Nodes Off&#039;, icon=&#039;MATERIAL&#039;)
        row = layout.row()
        box = row.box()
        box.label(text = &#039;Blender Internal Texture (One Per Object)&#039;)
        box.prop(sc, "EXTRACT_ALPHA", text=&#039;Extract Alpha Textures (slow)&#039;)
        box.prop(sc, "EXTRACT_PTEX", text=&#039;Extract Procedural Textures (slow)&#039;)
        box.prop(sc, "EXTRACT_OW", text=&#039;Re-extract Textures&#039;)

&#039;&#039;&#039;
        # Locking of nodes objects
        try:
            cmat = bpy.context.selected_objects[0].active_material
            TreeNodes = cmat.node_tree # throws exception for non-nodes mats
            locked = False
            for n in TreeNodes.nodes: # throws exception if no node mat
                if n.type == &#039;ShaderNodeOutputMaterial&#039;:
```

```
                if n.label == &amp;#039;Locked&amp;#039;:
                    locked = True
                    break

        row = layout.row()
        row.label(text="Selected: " + cmat.name, icon=("LOCKED" if locked else "UNLOCKED"))
        row.operator("ml.lock", text=("Unlock" if locked else "Lock"))
    except:
        pass
&amp;#039;&amp;#039;&amp;#039;
def register():
    bpy.utils.register_module(__name__)
    pass

def unregister():
    bpy.utils.unregister_module(__name__)
    pass

if __name__ == "__main__":
    register()
```