Blender 3D: Blending Into Python/Cookbook

Image Functions

Edit Image in External Program

This runs in linux (probably any unix), and launches The Gimp. It could probably be modified to launch Photoshop in windows.
In Gnome, KDE and Mac OS X, you can use a command to open documents using the default or a specified application.

- KDE: kfmclient openURL <URL, relative or absolute path>
- Gnome: gnome-open <any URL or path Gnome understands>
- Mac OS X: open [-a <application name>] <path>

Win32 has some open command too, maybe somebody could add this in.

```
#!BPY
"""
Name: 'Edit Image in the Gimp'
Blender: 232
Group: 'UV'
Tooltip: 'Edit Image in the Gimp.'
"""
from Blender import *
import os

def main():
    image = Image.GetCurrent()
    if not image: # Image is None
        print 'ERROR: You must select an active Image.'
        return
    imageFileName = sys.expandpath( image.filename )


    #appstring = 'xnview "%f"'
    #appstring = 'gqview "%f"'
    appstring = 'gimp-remote "%f"'

    # --------------------------------

    appstring = appstring.replace('%f', imageFileName)
    os.system(appstring)

if __name__ == '__main__':
    main()
```

Find Images

This script recursively searches for images that have broken files references.
It works by giving the user a root path, then finds and re-links all images within that path.
Its very useful when migrating projects to different computers.

```
#!BPY
"""
Name: 'Find all image files'
Blender: 232
Group: 'UV'
Tooltip: 'Finds all image files from this blend an relinks'
"""

__author__ = "Campbell Barton AKA Ideasman"
__url__ = ["http://members.iinet.net.au/~cpbarton/ideasman/"  , "blender", "elysiun"]
```

```
__bpydoc__ = """\
Blah
"""

from Blender import *
import os

#============================================#
# Strips the slashes from the back of a string #
#============================================#
def stripPath(path):
    return path.split('/')[-1].split('\\')[-1]

# finds the file starting at the root.
def findImage(findRoot, imagePath):
    newImageFile = None

    imageFile = imagePath.split('/')[-1].split('\\')[-1]

    # ROOT, DIRS, FILES
    pathWalk = os.walk(findRoot)
    pathList = [True]

    matchList = [] # Store a list of (match, size), choose the biggest.
    while True:
        try:
            pathList  = pathWalk.next()
        except:
            break

        for file in pathList[2]:
            # FOUND A MATCH
            if file.lower() == imageFile.lower():
                name = pathList[0] + sys.sep + file
                try:
                    size = os.path.getsize(name)
                except:
                    size = 0

                if size:
                    print '   found:', name
                    matchList.append( (name, size) )

    if matchList == []:
        print 'no match for:', imageFile
        return None
    else:
        # Sort by file size
        matchList.sort(key=lambda x: x[1], reverse=True )

        print 'using:', matchList[0][0]
        # First item is the largest
        return matchList[0][0] # 0 - first, 0 - pathname


# Makes the pathe relative to the blend file path.
def makeRelative(path):
    blendBasePath = sys.expandpath('//')
    if path.startswith(blendBasePath):
        path = path.replace(blendBasePath, '//')
        path = path.replace('//\\', '//')
    return path

def find_images(findRoot):
    print findRoot

    # findRoot = Draw.PupStrInput ('find in: ', '', 100)

    if findRoot == '':
        Draw.PupMenu('No Directory Selected' )
        return

    # Account for //
    findRoot = sys.expandpath(findRoot)

    # Strip filename
    while findRoot[-1] != '/' and findRoot[-1] != '\\':
        findRoot = findRoot[:-1]


    if not findRoot.endswith(sys.sep):
        findRoot += sys.sep


    if findRoot != '/' and not sys.exists(findRoot[:-1]):
        Draw.PupMenu('Directory Dosent Exist' )
```

```
    Window.WaitCursor(1)
    # ============ DIR DONE\
    images = Image.Get()
    len_images = float(len(images))
    for idx, i in enumerate(images):

        progress = idx / len_images
        Window.DrawProgressBar(progress, 'searching for images' )

        # If files not there?
        if not sys.exists(sys.expandpath(i.filename )):
            newImageFile = findImage(findRoot, i.filename)
            if newImageFile != None:
                newImageFile = makeRelative(newImageFile)
                print 'newpath:', newImageFile
                i.filename = newImageFile
                i.reload()

    Window.RedrawAll()
    Window.DrawProgressBar(1.0, '')
    Window.WaitCursor(0)


if __name__ == '__main__':
    Window.FileSelector(find_images, 'SEARCH ROOT DIR', sys.expandpath('//'))
```

Remove Double Images

This script finds images that are referenced more than once, and looks through all meshes texface's, and assigns only one of the images.

If a face has no users the image is removed.

This is useful because when an image is loaded more than once, it's also loaded into system memory and graphics card memory more than once, wasting resources.

Support for image type textures still needs doing.

```
#!BPY
"""
Name: 'Remove Double Images'
Blender: 232
Group: 'UV'
Tooltip: 'Remove Double Images'
"""

from Blender import *

def main():
    # Sync both lists
    fNameList = []#
    bImageList = [] # Sync with the one abovr.

    bImageReplacePointer = dict() # The length of IMage.Get()

    imgIdx = 0

    # Sort by name lengths so image.001 will be replaced by image
    Images = Image.Get()
    Images.sort(key=lambda x: len(x.name), reverse=True )

    for bimg in Images:
        expendedFName = sys.expandpath(bimg.filename)
        bImageReplacePointer[expendedFName] = bimg

    print 'Remove Double Images, loading mesh data...' ,
    uniqueMeshNames = []
    # get all meshs
    doubles = 0
    for ob in Object.Get():
        if ob.getType() == 'Mesh' and ob.getData(1) not in uniqueMeshNames:
            m = ob.getData(mesh=1)
            uniqueMeshNames.append(ob.getData(1))

            # We Have a new mesh,
            imageReplaced = 0
            for f in m.faces:
                image = None
                try: image = f.image
                except: pass
```

```
                if image:
                    replaceImage = bImageReplacePointer[ sys.expandpath(f.image.filename) ]
                    if replaceImage.name != image.name:
                        f.image = replaceImage
                        imageReplaced = 1

            if imageReplaced:
                doubles += 1
                m.update()
                print '\tchanged', m.name
            else:
                print '\tunchanged', m.name

    print 'Done, %i doubles removed.' % doubles

if __name__ == '__main__':
    main()
```

Material Functions

Toon Material Batch ConversionScript

this script changes *all* materials in your currently open blend file to toon materials. after executing, check the blender console for script output.

as this script alters *all* of your material settings in your currently opened blend file, you should *not* run it on a project that has not been saved yet! changes made to materials while using this script cannot be undone!

note: changes are not permanently committed to your blend file unless you choose to save your file after executing this script.

```
import Blender

from Blender import Material, Scene
from Blender.Scene import Render

print "\nTOON MATERIAL CONVERSION SCRIPT V1.0 STARTED...  \n"

# Get list of active materials from Blender
materials = Blender.Material.Get()

# Get render information needed for edge setting
scn = Scene.GetCurrent()
context = scn.getRenderingContext()

print "PROGRESS: CONVERTING ALL MATERIALS TO TOON TYPE..."

# Change materials to Toon Diffuse/Specular
for m in materials:

   # Diffuse Shader (2 = Toon)
   m.setDiffuseShader(2)

   # Specular Shader (3 = Toon)
   m.setSpecShader(3)

   # THE FOLLOWING SETTINGS CAN
   # BE CHANGED TO DIFFERENT
   # VALUES WITHIN THE SPECIFIED
   # RANGE OF ACCEPTABLE NUMBERS:

   # Diffuse Size (0 to 3.14)
   m.setDiffuseSize(1.5)

   # Diffuse Smooth (0 to 1.0)
   m.setDiffuseSmooth(.5)

   # Reflect Amount (0 to 1.0)
   # - optionally here to help you
   # with any necessary batch changes
   # to all material reflection values
   # Remove "#" from line below to use:
   # m.setRef(.75)

   # Specular (0 to 2.0)
   m.setSpec(.3)
```

```
    # Specular Smooth (0 to 1.0)
    m.setSpecSmooth(.5)

    # Specular Size (0 to 3.14)
    m.setSpecSize(.4)

    # Enable toon edge: 0 = off, 1 = on
    context.enableToonShading(1)

    # Edge Intension (0 to 255)
    context.edgeIntensity(30)

print "PROGRESS: CONVERSION FINISHED! \nTWEAK MATERIALS AND LIGHTING AS NECESSARY."

Blender.Redraw()
```

Curve Functions

Length of a curve

This function gets the combined length of all edges. Most useful to get the length of a curve. Be careful, because it will get the length of every curve in a curve object!

Note that this function doesn't take the object's transformation into account when getting the length.

```
from Blender import Mesh, Object
def curve_length(ob): # Can realy be any object
    me= Mesh.New()
    me.getFromObject(cu_ob.name)
    totlength= 0.0
    for ed in me.edges:
        # Blender 2.42 can simply do
        # totlength+= ed.length
        totlength+= (ed.v1.co-ed.v2.co).length

    return totlength

# TEST THE FUNCTION
cu_ob= Object.Get('mycurve')
print curve_length(cu_ob)
```

Text Functions

Paste Text in Unix

Paste text in X11, requires uclip [[1]]

```
#!BPY
"""
Name: 'Text from Clipboard'
Blender: 234
Group: 'Add'
Tooltip: 'Text from Clipboard X11'
"""
from Blender import Text
import os

clip = os.popen('uclip -o')

clipTxt = clip.read()
text = Text.New(clipTxt[0:10])
text.write(clipTxt)
```

Save all Texts as files

Saves all text editor texts as files in the current working directory.WARNING: This will overwrite files with those names!

```
import Blender

texts=Blender.Text.Get()

for text in texts:
        out=file(text.name, 'w')
        for line in text.asLines():
                out.write(line+'\n')
```

Mesh Functions

Examples and functions for Blenders NMesh, GMesh and new Mesh module.

Mesh tool template

Use this to base your editmode mesh tool on.

```
#!BPY
""" Registration info for Blender menus:
Name: 'Template Mesh Editmode tool...'
Blender: 237
Group: 'Mesh'
Tooltip: 'Change this template text tooltip'
"""

__author__ = "Your Name"
__url__ = ("blender", "elysiun")
__version__ = "1.0"

__bpydoc__ = """\
Multilin Script Help
Document your script here.
"""

from Blender import *

def main():
    scn = Scene.GetCurrent()
    ob = scn.getActiveObject() # Gets the current active object (If Any)

    if ob == None or ob.getType() != 'Mesh': # Checks the active objects a mesh
        Draw.PupMenu('ERROR%t|Select a mesh object.' )
        return

    Window.WaitCursor(1) # So the user knowns the script is busy.

    is_editmode = Window.EditMode() # Store edit mode state
    if is_editmode: Window.EditMode(0) # Python must get a mesh in object mode.

    me = ob.getData()

    #================#
    # EDIT MESH HERE #
    #================#
    for v in me.verts:
        if v.sel: # Operating on selected verts is what the user expects.
            v.co.x = v.co.x * 2

    #================#
    # FINISH EDITING #
    #================#

    me.update() # Writes the mesh back into Blender.

    # Go back into editmode if we started in edit mode.
    if is_editmode: Window.EditMode(1)
    Window.WaitCursor(0)


if __name__ == '__main__': # Dont run the script if its imported by another script.
    main()
```

Desaturate Meshes VertCol

Uses the new Mesh module. In Blender 2.4 onlyDesaturates using the same weighting as Photoshop uses.

```
from Blender import Mesh, Object
for ob in Object.GetSelected():
        if ob.getType() == 'Mesh':
                me = ob.getData(mesh=1)
                if me.faceUV:
                        for f in me.faces:
                                for c in f.col:
                                        # Weighted colour conversion, as used by photoshop.
                                        c.r = c.g = c.b = int(((c.r*30) + (c.g*59) + (c.b*11)) / 100.0)
```

Point inside a Mesh

This function returns 1/0 depending on whether the provided point is inside a mesh. It relies on the mesh having a continuous skin, n holes in it. (Otherwise the problem doesn't make sense.)

It uses the method of seeing how many face intersections there are along a line segment between that point, and a point somewhere outside the meshs bounds.

An even number of intersections means it's outside, an odd for inside. Hence we **return len(intersections) % 2** where **intersections** generates a list of intersections.

This function uses a Z direction vector so we can save some CPU cycles by first doing an X/Y bounds test to see if the points could intersect, before doing a full ray intersection.

```
from Blender import *

def pointInsideMesh(ob, pt):
    Intersect = Mathutils.Intersect # 2 less dict lookups.
    Vector = Mathutils.Vector

    def ptInFaceXYBounds(f, pt):

        co= f.v[0].co
        xmax= xmin= co.x
        ymax= ymin= co.y

        co= f.v[1].co
        xmax= max(xmax, co.x)
        xmin= min(xmin, co.x)
        ymax= max(ymax, co.y)
        ymin= min(ymin, co.y)

        co= f.v[2].co
        xmax= max(xmax, co.x)
        xmin= min(xmin, co.x)
        ymax= max(ymax, co.y)
        ymin= min(ymin, co.y)

        if len(f.v)==4:
            co= f.v[3].co
            xmax= max(xmax, co.x)
            xmin= min(xmin, co.x)
            ymax= max(ymax, co.y)
            ymin= min(ymin, co.y)

        # Now we have the bounds, see if the point is in it.
        return xmin <= pt.x <= xmax and \
            ymin <= pt.y <= ymax

    def faceIntersect(f):
        isect = Intersect(f.v[0].co, f.v[1].co, f.v[2].co, ray, obSpacePt, 1) # Clipped.
        if not isect and len(f.v) == 4:
            isect = Intersect(f.v[0].co, f.v[2].co, f.v[3].co, ray, obSpacePt, 1) # Clipped.

        return bool(isect and isect.z > obSpacePt.z) # This is so the ray only counts if its above the
point.


    obImvMat = Mathutils.Matrix(ob.matrixWorld)
    obImvMat.invert()
```

```
    pt.resize4D()
    obSpacePt = pt* obImvMat
    pt.resize3D()
    obSpacePt.resize3D()
    ray = Vector(0,0,-1)
    me= ob.getData(mesh=1)

    # Here we find the number on intersecting faces, return true if an odd number (inside), false
(outside) if its true.
    return len([None for f in me.faces if ptInFaceXYBounds(f, obSpacePt) if faceIntersect(f)]) % 2

# Example, see if the cursor is inside the mesh.
if __name__ == '__main__':
    scn= Scene.GetCurrent()
    ob= scn.getActiveObject()
    pt= Mathutils.Vector(Window.GetCursorPos())
    print 'Testing if cursor is inside the mesh' ,
    inside= pointInsideMesh(ob, pt)
    print inside
```

Scanfill for importers

This function takes a mesh and a list of vert indicies representing an ngon. It returns a list of tri indicies that make up the scanfilled face. This is much more useful for importers.

it also handles cases where scanfill does not work by returning a triangle fan.

It may be faster then using the mesh.fill() function on your original mesh because cycling editmode can be slow on a lot of data. Another advantage with this function over simply using fill() is you can be sure the faces will be flipped the right way, according to the order of the indicies.

```
from Blender import *
def ngon(from_mesh, indicies):
    if len(indicies) < 4:
        return [indicies]
    is_editmode= Window.EditMode()
    if is_editmode:
        Window.EditMode(0)
    temp_mesh = Mesh.New()
    temp_mesh.verts.extend( [from_mesh.verts[i].co for i in indicies] )
    temp_mesh.edges.extend( [(temp_mesh.verts[i], temp_mesh.verts[i-1]) for i in
xrange(len(temp_mesh.verts))] )

    oldmode = Mesh.Mode()
    Mesh.Mode(Mesh.SelectModes['VERTEX'])
    for v in temp_mesh.verts:
        v.sel= 1

    # Must link to scene
    scn= Scene.GetCurrent()
    temp_ob= Object.New('Mesh')
    temp_ob.link(temp_mesh)
    scn.link(temp_ob)
    temp_mesh.fill()
    scn.unlink(temp_ob)
    Mesh.Mode(oldmode)

    new_indicies= [ [v.index for v in f.v]  for f in temp_mesh.faces ]

    if not new_indicies: # JUST DO A FAN, Cant Scanfill
        print 'Warning Cannot scanfill!- Fallback on a triangle fan.'
        new_indicies = [ [indicies[0], indicies[i-1], indicies[i]] for i in xrange(2, len(indicies)) ]
    else:
        # Use real scanfill.
        # See if its flipped the wrong way.
        flip= None
        for fi in new_indicies:
            if flip != None:
                break

            for i, vi in enumerate(fi):
                if vi==0 and fi[i-1]==1:
                    flip= 0
                    break
                elif vi==1 and fi[i-1]==0:
                    flip= 1
                    break
        if flip:
```

```
            for fi in new_indicies:
                fi.reverse()

    if is_editmode:
        Window.EditMode(1)
    return new_indicies


# === ===== EG
scn= Scene.GetCurrent()
me = scn.getActiveObject().getData(mesh=1)
ind= [v.index for v in me.verts if v.sel] # Get indicies

indicies = ngon(me, ind) # fill the ngon.

# Extand the faces to show what the scanfill looked like.
print len(indicies)
me.faces.extend([[me.verts[ii] for ii in i] for i in indicies])
```

Triangulate NMesh

This is a function to be used by other scripts, its useful if you want to make your life simpler by only dealing with triangles.

The shortest edge method is used for dividing the quad into 2 tri's.

```
def triangulateNMesh(nm):
    '''
    Converts the meshes faces to tris, modifies the mesh in place.
    '''

    #============================================================================#
    # Returns a new face that has the same properties as the origional face      #
    # but with no verts                                                          #
    #============================================================================#
    def copyFace(face):
        newFace = NMesh.Face()
        # Copy some generic properties
        newFace.mode = face.mode
        if face.image != None:
            newFace.image = face.image
        newFace.flag = face.flag
        newFace.mat = face.mat
        newFace.smooth = face.smooth
        return newFace

    # 2 List comprehensions are a lot faster then 1 for loop.
    tris = [f for f in nm.faces if len(f) == 3]
    quads = [f for f in nm.faces if len(f) == 4]


    if quads: # Mesh may have no quads.
        has_uv = quads[0].uv
        has_vcol = quads[0].col
        for quadFace in quads:

            # Triangulate along the shortest edge
            if (quadFace.v[0].co - quadFace.v[2].co).length < (quadFace.v[1].co -
quadFace.v[3].co).length:
                # Method 1
                triA = 0,1,2
                triB = 0,2,3
            else:
                # Method 2
                triA = 0,1,3
                triB = 1,2,3

            for tri1, tri2, tri3 in (triA, triB):
                newFace = copyFace(quadFace)
                newFace.v = [quadFace.v[tri1], quadFace.v[tri2], quadFace.v[tri3]]
                if has_uv: newFace.uv = [quadFace.uv[tri1], quadFace.uv[tri2], quadFace.uv[tri3]]
                if has_vcol: newFace.col = [quadFace.col[tri1], quadFace.col[tri2], quadFace.col[tri3]]

                nm.addEdge(quadFace.v[tri1], quadFace.v[tri3]) # Add an edge where the 2 tris are
devided.
                tris.append(newFace)

        nm.faces = tris
```

Fix vertex winding in "bowtie" quads

Sometimes you may encounter quad faces which, although correctly coplanar, aren't quite "full". This comes from the vertices being in the wrong order, and causes the face to overlap itself, often leaving undesired holes and black areas where the normals point the wrong way. To see what this all means, just make a plane and switch places for the vertices at any edge. The connecting edges will then cross. Since the normals of the face don't make sense in this situation, the script can't guarantee that the normal points outwards after finishing.

```
#!BPY
"""
Name: 'Quadsorter'
Blender: 233
Group: 'Mesh'
Tip: 'Fix winding order for quad faces for all selected meshes'
Author: Yann Vernier (LoneTech)
"""

from Blender.Mathutils import Vector, CrossVecs, DotVecs

def sortface(f):
  if len(f) != 4:
    return f
  v=[Vector(list(p)) for p in f]
  v2m0=v[2]-v[0]
  # The normal of the plane
  n=CrossVecs(v[1]-v[0], v2m0)
  #k=DotVecs(v[0],n)
  #if DotVecs(v[3],n) != k:
  #  raise ValueError("Not Coplanar")
  # Well, the above test would be a good hint to make triangles.
  # Get a vector pointing along the plane perpendicular to v[0]-v[2]
  n2=CrossVecs(n, v2m0)
  # Get the respective distances along that line
  k=[DotVecs(p,n2) for p in v[1:]]
  # Check if the vertices are on the proper side
  if cmp(k[1],k[0]) == cmp(k[1],k[2]):
    #print "Bad",v
    f.v=[f[0],f[2],f[3],f[1]]

from Blender.Object import GetSelected
for obj in GetSelected():
  if obj.getType() == 'Mesh':
    mesh=obj.data
    for face in mesh.faces:
      sortface(face)
    mesh.update()
```

Remove Verts Without removing entire faces

Script that uses the mesh template above. removes verts but surrounding quads will be converted to tri's.

Note This only works with NMesh.

```
        #================#
        # EDIT MESH HERE #
        #================#
        for f in me.faces:
            face_verts = f.v[:] # make a copy of the list.
            for v in face_verts:
                if v.sel:
                    f.v.remove(v)

        # Remove all with less then 3 verts,
        # When removing objects from a list its best to loop backwards
        fIdx = len(me.faces)
        while fIdx:
            fIdx -=1
            f = me.faces[fIdx]
            if len(f.v) < 3:
                del me.faces[fIdx]

        # Remove all selected verts
        # Loop backwards.
        vIdx = len(me.verts)
        while vIdx:
            vIdx -=1
            v = me.verts[vIdx]
```

```
            if v.sel:
                del me.verts[vIdx]
        #================#
        # FINISH EDITING #
        #================#
```

GMesh AutoSmooth Mesh

This function uses GMesh to autosmooth manifold meshes, it requires the GMesh module.

Be careful because it will smooth your mesh in place, so make a copy of your original object if you don't want it modified.

```
from Blender import *
import GMesh

smooth = Draw.PupIntInput('smooth:', 20,1,89)

for ob in Object.GetSelected():
        mesh = ob.getData()
        gmesh = GMesh.NMesh2GMesh(mesh)

        try:
                gmesh.autoSmooth(smooth)
        except:
                print 'Error non manifold mesh'
                continue # go onto the next item

        mesh = GMesh.GMesh2NMesh(gmesh)

        # Make the faces smooth
        for f in mesh.faces:
                f.smooth = 1

        ob.link(mesh) # Link the new mesh with the original object
```

ScanFill

To simulate typing "Shift F" in blender to create scanfill selected edge loop (edit mode only)

This will only work if the 3D view is open.

```
import Blender

winid = Blender.Window.GetScreenInfo(Blender.Window.Types.VIEW3D)[0]['id']
Blender.Window.SetKeyQualifiers(Blender.Window.Qual.SHIFT)
Blender.Window.QAdd(winid, Blender.Draw.FKEY,1)
Blender.Window.QHandle(winid)
Blender.Window.SetKeyQualifiers(0)
```

Expanded Scanfill function

Self contained scanfill function, based on the code above*Note, this function needs a 3D view to be available*

```
import Blender

# Take a list of points and return a scanfilled NMesh
def scanFillPoints(pointList):
        Blender.Window.EditMode(0)

        nme = Blender.NMesh.New()
        # 2.37 compatability, not needed in 2.4
        if not nme.edges:
                nme.addEdgesData()

        for p in pointList:
```

```
                v = Blender.NMesh.Vert( p[0], p[1], p[2] )
                nme.verts.append(v)
                v.sel = 1

                if len(nme.verts) >= 2:
                        nme.addEdge(nme.verts[-2], nme.verts[-1])

        nme.addEdge(nme.verts[0], nme.verts[-1])


        scn = Blender.Scene.GetCurrent()

        actOb = scn.getActiveObject()
        if actOb:
                actSel = actOb.sel
        else:
                actSel = 0


        ob = Blender.Object.New('Mesh')
        ob.link(nme)
        scn.link(ob)
        scn.layers = range(1,20)
        ob.sel = 1
        Blender.Window.EditMode(1)

        winid = Blender.Window.GetScreenInfo(Blender.Window.Types.VIEW3D)[0]['id']
        Blender.Window.SetKeyQualifiers(Blender.Window.Qual.SHIFT)
        Blender.Window.QAdd(winid, Blender.Draw.FKEY,1)
        Blender.Window.QHandle(winid)
        Blender.Window.SetKeyQualifiers(0)

        Blender.Window.EditMode(0)
        # scn.unlink(ob)

        # Select the old active object.
        if actOb:
                actOb.sel = actSel

        # Reture the scanfilled faces.
        return ob.getData()
```

Example function usage.

```
scanFillPoints([[-1,-1,0], [1,-1,1], [1,1,0], [0,0,0.2], [0,1,-.1], [0.1,1,-0.3] ])
```

Copy NMesh Face

Returns a new face that has the same properties as the original face but with no verts

```
def faceCopy(face):
  newFace = NMesh.Face()
  # Copy some generic properties
  newFace.mode = face.mode
  if face.image != None:
    newFace.image = face.image
  newFace.flag = face.flag
  newFace.mat = face.mat
  newFace.smooth = face.smooth
  return newFace
```

Returns the Faces centre as a Vector

Note, Blenders Mesh API now has face.cent access

Take 1 NMFace and return a vector as its centre, will use an existing vector object "cent" if provide

```


def faceCent(f, cent=None):
    x = y = z = 0
    for v in f.v:
```

```
        x+=v.co[0]
        y+=v.co[1]
        z+=v.co[2]
    if not cent:
        return Mathutils.Vector([x/len(f.v), y/len(f.v), z/len(f.v)])

    # Modify the provided vec
    cent.x = x/len(f.v)
    cent.y = y/len(f.v)
    cent.z = z/len(f.v)
```

Flip Faces Up

I used this script to flip all faces in many terrain meshes to point up. It uses Mesh as opposed to NMesh.

```
from Blender import *

#==================#
# Apply Tpransform #
#==================# Used for skin
def apply_transform(vec, matrix):
        x, y, z = vec
        xloc, yloc, zloc = matrix[3][0], matrix[3][1], matrix[3][2]
        vec.x = x*matrix[0][0] + y*matrix[1][0] + z*matrix[2][0] + xloc
        vec.y = x*matrix[0][1] + y*matrix[1][1] + z*matrix[2][1] + yloc
        vec.z = x*matrix[0][2] + y*matrix[1][2] + z*matrix[2][2] + zloc

def apply_transform3x3(vec, matrix):
        x, y, z = vec
        vec.x = x*matrix[0][0] + y*matrix[1][0] + z*matrix[2][0]
        vec.y = x*matrix[0][1] + y*matrix[1][1] + z*matrix[2][1]
        vec.z = x*matrix[0][2] + y*matrix[1][2] + z*matrix[2][2]


# Point to z up.
noVec = Mathutils.Vector(0,0,-10000)
cent = Mathutils.Vector(0,0,0)

for ob in Object.GetSelected():
        if ob.getType() != 'Mesh':
                continue
        mat = ob.matrixWorld


        me = ob.getData(mesh=1)
        # We know were mesh

        # Select none
        for f in me.faces: f.sel = 0

        # Flip based on facing.
        for f in me.faces:
                no = f.no
                apply_transform3x3(no, mat)

                # Get the faces centre
                cent.x, cent.y, cent.z = 0,0,0
                for v in f.v:
                        cent += v.co
                cent = cent * (1.0 / len(f.v))
                apply_transform(cent, mat)

                # Move the vec over the centre of the face.
                noVec.x = cent.x
                noVec.y = cent.y

                # Are we not facing up?, if not then select and flip later.
                if ((cent+no)-noVec).length <= (cent-noVec).length:
                        f.sel = 1
        me.flipNormals()
```

Scale UVs

Scales all uv coords for selected mesh objects.

```
from Blender import *
def main():
```

```
        # Scale the UV down.
        # This examples scales down by 1 pixel on a 512x512 image.
        shrink = 1-(1/512.0)

        for ob in Object.GetSelected():
                if ob.getType() == 'Mesh':
                        me = ob.getData(mesh=1)
                        if me.faceUV:
                                for f in me.faces:
                                        f.uv =\
                                        tuple([ Mathutils.Vector(\
                                        ((uv[0]-0.5)*shrink)+0.5,\
                                        ((uv[1]-0.5)*shrink)+0.5,\
                                         ) for uv in f.uv])

if __name__ == '__main__':
        main()
```

Find a material in all Meshes in all Scenes

Sometimes You have a lot of mesh object and materials you don't want any of them to use. This script can help you find those objects.

```
#!BPY
"""
Name: 'Find Mesh with Material'
Blender: 234
Group: 'Object'
Tooltip: 'Find Mesh with Material'
"""

from Blender import *

def main():
    matToFind = Draw.PupStrInput('matName:', '', 21)
    if matToFind == None:
        return

    Window.WaitCursor(1)
    for scn in Scene.Get():
        for ob in scn.getChildren():
            if ob.getType() == 'Mesh':
                for mat in ob.getData(mesh=1).materials:
                    matname = None
                    try:
                        matname = mat.name
                    except:
                        # Material must be None
                        continue

                    if matname == matToFind:
                        # Unselect all in the scene
                        for ob_ in scn.getChildren():
                            ob_.sel = 0

                        # Select the found object
                        ob.sel = 1

                        scn.makeCurrent()
                        Draw.PupMenu('Material "%s" found in object "%s".' % (matToFind, ob.name))
                        Window.WaitCursor(0)
                        return

    Window.WaitCursor(0)
    Draw.PupMenu('Material "%s" Not found.' % matToFind)

if __name__ == '__main__':
    main()
```

Faces share an edge

The the 2 faces share an edge, its best to make sure your not comparing the same faces, and remove the first 'if'.

```
# Do the 2 faces share an edge?
# return true or false.
def faceShareEdge(face1, face2):
    # Are we using the same verts. could be more comprehensive, since vert order may differ but still be
the same.
    if face1.v == face2.v:
        return False
    firstMatch = None
    for v1 in face1:
        if v1 in face2:
            if firstMatch is None:
                firstMatch = True
            else:
                return True
    return False
```

Get Edge Angles

Returns a list of angles, the combine angle difference of all faces that use the edges. the angles returned are in sync with mesh.edges. Edges with 0 or 1 faces will have Zero angle.

This function uses BlenderMesh not BlenderNMesh mesh data.

```
def getEdgeAngles(me):
    Ang= Blender.Mathutils.AngleBetweenVecs
    Vector= Blender.Mathutils.Vector

    edges = dict( [ (ed.key,  (i, [])) for i, ed in enumerate(me.edges) ] )

    for f in me.faces:
        #print f.index
        for key in f.edge_keys:
            edges[key][1].append(f.no)

    edgeAngles=[0.0] * len(me.edges)
    for eIdx, angles in edges.itervalues():
        angles_len= len(angles)

        if angles_len < 2:
            pass
        if angles_len==2:
            edgeAngles[eIdx] = Ang(angles[0], angles[1])
        else:
            totAngDiff=0
            for j in reversed(xrange(angles_len)):
                for k in reversed(xrange(j)):
                    totAngDiff+= (Ang(angles[j], angles[k])/180) # /180 isnt needed, just to keeop the
vert small.
            edgeAngles[eIdx] = totAngDiff
    return edgeAngles
```

Mesh Ray Intersect

Intersect a ray with a mesh, Assume the mesh has no loc/size/rot.

```
import Blender
from Blender import Window, Mathutils, Object
Vector= Mathutils.Vector
Intersect= Mathutils.Intersect
Matrix= Mathutils.Matrix

def meshRayIntersect(me, Origin, Direction):
    def faceIntersect(f):
        isect = Intersect(f.v[0].co, f.v[1].co, f.v[2].co, Direction, Origin, 1) # Clipped.
        if isect:
            return isect
        elif len(f.v) == 4:
            isect = Intersect(f.v[0].co, f.v[2].co, f.v[3].co, Direction, Origin, 1) # Clipped.
        return isect


    ''' Ray is a tuple of vectors (Origin, Direction) '''
    isect= best_isect= None
    dist_from_orig= 1<<30
```

```
    for f in me.faces:
        isect= faceIntersect(f)
        if isect:
            l= (isect-Origin).length
            if l < dist_from_orig:
                dist_from_orig= l
                best_isect= isect
    return best_isect, dist_from_orig
```

Copy Vertex UV to Face UV

Copies the Vertex UV coordinates (Sticky) to faceUV coordinates (TexFace).

```
#!BPY

#sticky2uv.py

""" Registration info for Blender menus:
Name: 'Vertex UV to face UV'
Blender: 241
Group: 'Mesh'
Tooltip: 'Copy vertex UV to face UV'
"""
__author__ = "Brandano"
__url__ = ("blender", "elysiun")
__version__ = "1.0"
__bpydoc__ = """\
Copies the Vertex UV coordinates (Sticky) to face UV coordinates (TexFace).
Warning: the original face UV's will be overwritten.
"""
import Blender
from Blender import Mesh

if (Blender.Object.GetSelected() != None):
    for me in [ob.getData(mesh=1) for ob in Blender.Object.GetSelected() if ob.getType() == "Mesh"]:
        if me.vertexUV:
            me.faceUV = 1
            for f in me.faces: f.uv = [v.uvco for v in f.verts]
        me.update()
```

Math Functions

Here is the place to add math examples, they can be blender specific or generic python math functions.

Changing Rotation Axis Order

If you ever have trouble converting between different rotation systems its possible that the order of rotations is the problem.

```
import Blender
RotationMatrix= Blender.Mathutils.RotationMatrix

MATRIX_IDENTITY_3x3 = Blender.Mathutils.Matrix([1.0,0.0,0.0],[0.0,1.0,0.0],[0.0,0.0,1.0])
def eulerRotateOrder(x,y,z):
    x,y,z = x%360,y%360,z%360 # Clamp all values between 0 and 360, values outside this raise an error.
    xmat = RotationMatrix(x,3,'x')
    ymat = RotationMatrix(y,3,'y')
    zmat = RotationMatrix(z,3,'z')
    # Standard BVH multiplication order, apply the rotation in the order Z,X,Y
    # Change the order here
    return (ymat*(xmat * (zmat * MATRIX_IDENTITY_3x3 ))).toEuler()
```

Get Angle between 3 points

Get the angle between line AB and BC where b is the elbow

```
import Blender
AngleBetweenVecs = Blender.Mathutils.AngleBetweenVecs

def getAng3pt3d(avec, bvec, cvec):
    try:
        ang = AngleBetweenVecs(avec - bvec,  cvec - bvec)
        if ang != ang:
            raise  "ERROR angle between Vecs"
        else:
            return ang
    except:
        print '\tAngleBetweenVecs failed, zero length?'
        return 0
```

Point inside a tri (2D)

Returns True if pt is inside the triangle.

only gives a correct answer when pt lies on the triangles plane.

```
from Blender import Mathutils
SMALL_NUM = 0.000001
def pointInTri2D(pt, tri1, tri2, tri3):
    a = Mathutils.TriangleArea(tri1, tri2, tri3)
    othera = Mathutils.TriangleArea(pt, tri1, tri2) + SMALL_NUM
    if othera > a: return False
    othera += Mathutils.TriangleArea(pt, tri2, tri3)
    if othera > a: return False
    othera += Mathutils.TriangleArea(pt, tri3, tri1)
    if othera > a: return False
    return True
```

Apply Matrix

Applys a 4x4 transformation as returned by object.getMatrix(), to a vector (3d point in space)

This is useful for finding out the position of a vertex in worldspace.

Blender 2.43 supports this simply by doing "newvwec = vec*matrix" but its good to know how to do it manually

```
#==================#
# Apply Tpransform #
#==================#
def apply_transform(vec, matrix):
    x, y, z = vec
    xloc, yloc, zloc = matrix[3][0], matrix[3][1], matrix[3][2]
    return  x*matrix[0][0] + y*matrix[1][0] + z*matrix[2][0] + xloc,\
            x*matrix[0][1] + y*matrix[1][1] + z*matrix[2][1] + yloc,\
            x*matrix[0][2] + y*matrix[1][2] + z*matrix[2][2] + zloc

def apply_transform3x3(vec, matrix):
    x, y, z = vec
    return x*matrix[0][0] + y*matrix[1][0] + z*matrix[2][0],\
            x*matrix[0][1] + y*matrix[1][1] + z*matrix[2][1],\
            x*matrix[0][2] + y*matrix[1][2] + z*matrix[2][2]
```

2D Line Intersection

Intersect 2 lines, if so where.

If there is no intersection the retured X value will be None and the y will be an error code.

The first line is *(x1,y1, x2,y2)*, the second *(_x1,_y1, _x2,_y2)*

```
SMALL_NUM = 0.000001
def lineIntersection2D(x1,y1, x2,y2, _x1,_y1, _x2,_y2):

    # Bounding box intersection first.
```

```
    if min(x1, x2) > max(_x1, _x2) or \
    max(x1, x2) < min(_x1, _x2) or \
    min(y1, y2) > max(_y1, _y2) or \
    max(y1, y2) < min(_y1, _y2):
        return None, 100 # Basic Bounds intersection TEST returns false.

    # are either of the segments points? Check Seg1
    if abs(x1 - x2) + abs(y1 - y2) <= SMALL_NUM:
        return None, 101

    # are either of the segments points? Check Seg2
    if abs(_x1 - _x2) + abs(_y1 - _y2) <= SMALL_NUM:
        return None, 102

    # Make sure the HOZ/Vert Line Comes first.
    if abs(_x1 - _x2) < SMALL_NUM or abs(_y1 - _y2) < SMALL_NUM:
        x1, x2, y1, y2, _x1, _x2, _y1, _y2 = _x1, _x2, _y1, _y2, x1, x2, y1, y2

    if abs(x2-x1) < SMALL_NUM: # VERTICLE LINE
        if abs(_x2-_x1) < SMALL_NUM: # VERTICLE LINE SEG2
            return None, 111 # 2 verticle lines dont intersect.

        elif abs(_y2-_y1) < SMALL_NUM:
            return x1, _y1 # X of vert, Y of hoz. no calculation.

        yi = ((_y1 / abs(_x1 - _x2)) * abs(_x2 - x1)) + ((_y2 / abs(_x1 - _x2)) * abs(_x1 - x1))

        if yi > max(y1, y2): # New point above seg1's vert line
            return None, 112
        elif yi < min(y1, y2): # New point below seg1's vert line
            return None, 113

        return x1, yi # Intersecting.


    if abs(y2-y1) < SMALL_NUM: # HOZ LINE
        if abs(_y2-_y1) < SMALL_NUM: # HOZ LINE SEG2
            return None, 121 # 2 hoz lines dont intersect.

        # Can skip vert line check for seg 2 since its covered above.
        xi = ((_x1 / abs(_y1 - _y2)) * abs(_y2 - y1)) + ((_x2 / abs(_y1 - _y2)) * abs(_y1 - y1))
        if xi > max(x1, x2): # New point right of seg1's hoz line
            return None, 112
        elif xi < min(x1, x2): # New point left of seg1's hoz line
            return None, 113

        return xi, y1 # Intersecting.

    # Accounted for hoz/vert lines. Go on with both anglular.
    b1 = (y2-y1)/(x2-x1)
    b2 = (_y2-_y1)/(_x2-_x1)
    a1 = y1-b1*x1
    a2 = _y1-b2*_x1

    if b1 - b2 == 0.0:
        return None, None

    xi = - (a1-a2)/(b1-b2)
    yi = a1+b1*xi
    if (x1-xi)*(xi-x2) >= 0 and (_x1-xi)*(xi-_x2) >= 0 and (y1-yi)*(yi-y2) >= 0 and (_y1-yi)*(yi-_y2)>=0:
        return xi, yi
    else:
        return None, None
```

Round to power of 2

This function takes any number and rounds it to the nearest power of 2 value (2,4,8,16,32,64,128,256,512,1024,2048,4096...) This is useful for rounding textures to sized that load into graphics card memory

It returns 3 values. rounded down, round closest, rounded up.

```
def roundPow2(roundVal):
    base2val = 1
    while roundVal >= base2val:
        base2val*=2

    # dont round up if there the same, just give the same vars
    if roundVal == base2val/2:
        return base2val/2, base2val/2, base2val/2 # Round down and round up.
```

```
    smallRound = base2val/2
    largeRound = base2val

    # closest to the base 2 value
    diffLower = abs(roundVal - smallRound)
    diffHigher = abs(roundVal - largeRound)
    if diffLower < diffHigher:
        mediumRound = smallRound
    else:
        mediumRound = largeRound

    smallRound = base2val/2
    largeRound = base2val

    return smallRound, mediumRound, largeRound # round down, round mid and round up.
```

Closest Point (snapping)

Returns the vector that is closest to point. Good for snapping.

```
def getSnapVec(point, snap_points):
    '''
    Returns the closest vec to snap_points
    '''
    close_dist= 1<<30
    close_vec= None

    x= point[0]
    y= point[1]
    z= point[2]
    for v in snap_points:
        # quick length cmp before a full length comparison.
        if abs(x-v[0]) < close_dist and\
        abs(y-v[1]) < close_dist and\
        abs(z-v[2]) < close_dist:
            l= (v-point).length
            if l<close_dist:
                close_dist= l
                close_vec= v
    return close_vec
```

Blender Object Scripts

Linked Duplicate Object

There is no function in B:Python to make a linked duplicate of an object (**Alt+D**) so here is a function that does it for you.

Note Since this was written Blender.Object.Duplicate() has been added as well as object.copy()

```
from Blender import *

# Like pressing Alt+D
def linkedCopy(ob, scn=None): # Just like Alt+D
    if not scn:
        scn = Scene.GetCurrent()
    type = ob.getType()
    newOb = Object.New(type)
    if type != 'Empty':
      newOb.shareFrom(ob)
    scn.link(newOb)
    newOb.setMatrix(ob.getMatrix())
    # Copy other attributes.
    newOb.setDrawMode(ob.getDrawMode())
    newOb.setDrawType(ob.getDrawType())
    newOb.Layer = ob.Layer
        # Update the view
        ob.select(0)
        newOb.select(1)
    return newOb

# You can call the function like this
try:
```

```
    ob2duplicate = Object.GetSelected()[0]
    linkedCopy(ob2duplicate)
        Redraw()
except:
    print "Nothing Selected"
```

Select Double Objects

Finds double object- objects that have the same dataname, type and loc/size/rot

```
#!BPY
"""
Name: 'Select only double objects.'
Blender: 232
Group: 'Object'
Tooltip: 'Select double objects from the existing selection.'
"""

from Blender import *

def main():
    # Collect the extra object data once only, so we dont need to request it again.
    obinfo = [{'object':ob, 'dataname':ob.getData(1), 'type':ob.getType(),
'matrix':tuple(ob.matrixWorld)} for ob in Object.GetSelected() ]
    print '\n\n\nStarting to select doubles for  %i objects.' % len(obinfo)
    doubleObs = [] # store doubles in this list
    doubles = 0

    # Comparison loop, compare items in the list only once.
    obIdx1 = len(obinfo)
    while obIdx1:
        obIdx1 -=1
        ob1 = obinfo[obIdx1]

        # Deselect as we go, any doubles will be selected again.          ob1['object'].sel = 0
        ob1['object'].sel = 0

        obIdx2 = obIdx1

        while obIdx2:
            obIdx2 -=1
            ob2 = obinfo[obIdx2]
            # Comparison loop done.


            # Now we have both objects we can compare ob2 against ob1.
            if \
            ob1['dataname'] == ob2['dataname'] and\
            ob1['type']     == ob2['type'] and\
            ob1['matrix']   == ob2['matrix']:
                # We have a double, print output and add to the double list.
                doubles +=1
                print '\t%i doubles found: "%s", "%s"' % (doubles, ob1['object'].name,
ob2['object'].name)
                doubleObs.append(ob2)

    for ob in doubleObs:
        ob['object'].sel = 1

if __name__ == '__main__':
    t = sys.time()
    main()
    print 'Done in %.4f seconds.' % (sys.time()-t)
```

NLA strips Examples

Example code for manipulating NLA strips and action strips

```
# Nov 16 2006
#
# Mike Stramba
# mstramba@sympatico.ca
# BlenderArtists  Mike_S
#

import Blender
from Blender import *
```

```
from Blender.Armature  import NLA


ctr = 1
numStrips = 1

vflags ={32:'LOCK_ACTION',1:'SELECT',2:'STRIDE_PATH',8:'HOLD',16:'ACTIVE'}

def tranflag(flag):
    if flag:
        print
        for v in vflags:
            t = flag & v
            if t:
                print '\t\t',v,vflags[t]

def showStrip(strip):
    print ctr,'/',numStrips
    print strip.action.name
    print '\tstripStart',strip.stripStart
    print '\tstripEnd',strip.stripEnd
    print '\tactionStart',strip.actionStart
    print '\tactionEnd',strip.actionEnd
    print '\tblendin',strip.blendIn
    print '\tblendout',strip.blendOut


    print '\tflag',strip.flag,
    tranflag(strip.flag)

    print '\tmode',strip.mode
    print '\tbrepeat',strip.repeat
    print '\tstrideAxis',strip.strideAxis
    print '\tstrideBone',strip.strideBone
    print '\tstrideLength',strip.strideLength


armOb=Object.Get('Armature')

actions=Armature.NLA.GetActions()

#
#  Actions named 'rot', 'move', 'Run' assumed to exist, or substitute
#  your own action names
#

rotAct = actions['rot']
movAct = actions['move']
runAct = actions['Run']

#
# get all NLA strips for this object
#

Char1NLAstrips  = armOb.actionStrips

#
# set the current frame to where you want NLA strips to initially appear
# in the NLA editor

frame = 1
Blender.Set('curframe',frame)



#
# remove all NLA strips
#


Char1NLAstrips[:] = []



#
#  some different ways of adding action strips to the NLA editor
#

Blender.Object.Get('Armature').actionStrips.append(Blender.Armature.NLA.GetActions()['tester'])

Char1NLAstrips.append(Blender.Armature.NLA.GetActions()['UpDown'])

armOb.actionStrips.append(rotAct)

Char1NLAstrips.append(movAct)

Char1NLAstrips.append(actions['Run'])
```

```
#
#  get a strip
#

strip0 = Char1NLAstrips[0]

print '\nstrip0.action.name ="' +strip0.action.name+'"'

#
# show it's properties
#

showStrip(strip0)

#
# change it's stripStart, stripEND (add 50 frames)
#  (effectively moving the strip

strip0.stripEnd   += 50
strip0.stripStart  += 50

#
# show the changes
#

showStrip(strip0)

#
# select the strip in the NLA editor
#

strip0.flag += NLA.Flags['SELECT']
Blender.Window.RedrawAll()

showStrip(strip0)

#
# move all strips by FrameOffset
#

def moveallStrips(FrameOffset):
    for strip in Char1NLAstrips:
        strip.stripEnd   += FrameOffset
        strip.stripStart  += FrameOffset


moveallStrips(30)



#
# show all strips Properties
#

print
print '============  ALL STRIPS ================'

numStrips = len(Char1NLAstrips)
print numStrips,' NLA strips for ',armOb
for strip in Char1NLAstrips:
    showStrip(strip)
```

Blender Windowing and User Interface Scripts

Mouse Location 3D Space

```
import Blender
from Blender import Mathutils, Window, Scene, Draw, Mesh
from Blender.Mathutils  import Matrix, Vector, Intersect


# DESCRIPTION:
# screen_x, screen_y the origin point of the pick ray
# it is either the mouse location
# localMatrix is used if you want to have the returned values in an objects localspace.
#    this is usefull when dealing with an objects data such as verts.
# or if useMid is true, the midpoint of the current 3dview
# returns
# Origin - the origin point of the pick ray
```

```
# Direction - the direction vector of the pick ray
# in global coordinates
epsilon = 1e-3 # just a small value to account for floating point errors

def getPickRay(screen_x, screen_y, localMatrix=None, useMid = False):

    # Constant function variables
    p = getPickRay.p
    d = getPickRay.d

    for win3d in Window.GetScreenInfo(Window.Types.VIEW3D): # we search all 3dwins for the one containing
the point (screen_x, screen_y) (could be the mousecoords for example)
        win_min_x, win_min_y, win_max_x, win_max_y = win3d['vertices']
        # calculate a few geometric extents for this window

        win_mid_x  = (win_max_x + win_min_x + 1.0) * 0.5
        win_mid_y  = (win_max_y + win_min_y + 1.0) * 0.5
        win_size_x = (win_max_x - win_min_x + 1.0) * 0.5
        win_size_y = (win_max_y - win_min_y + 1.0) * 0.5

        #useMid is for projecting the coordinates when we subdivide the screen into bins
        if useMid: # == True
            screen_x = win_mid_x
            screen_y = win_mid_y

        # if the given screencoords (screen_x, screen_y) are within the 3dwin we fount the right one...
        if (win_max_x > screen_x > win_min_x) and (  win_max_y > screen_y > win_min_y):
            # first we handle all pending events for this window (otherwise the matrices might come out
wrong)
            Window.QHandle(win3d['id'])

            # now we get a few matrices for our window...
            # sorry - i cannot explain here what they all do
            # - if you're not familiar with all those matrices take a look at an introduction to
OpenGL...
            pm  = Window.GetPerspMatrix()   # the prespective matrix
            pmi  = Matrix(pm); pmi.invert() # the inverted perspective matrix

            if (1.0 - epsilon < pmi[3][3] < 1.0 + epsilon):
                # pmi[3][3] is 1.0 if the 3dwin is in ortho-projection mode (toggled with numpad 5)
                hms = getPickRay.hms
                ortho_d = getPickRay.ortho_d

                # ortho mode: is a bit strange - actually there's no definite location of the camera ...
                # but the camera could be displaced anywhere along the viewing direction.

                ortho_d.x, ortho_d.y, ortho_d.z = Window.GetViewVector()
                ortho_d.w = 0

                # all rays are parallel in ortho mode - so the direction vector is simply the viewing
direction
                #hms.x, hms.y, hms.z, hms.w = (screen_x-win_mid_x) /win_size_x, (screen_y-win_mid_y) /
win_size_y, 0.0, 1.0
                hms[:] = (screen_x-win_mid_x) /win_size_x, (screen_y-win_mid_y) / win_size_y, 0.0, 1.0

                # these are the homogenious screencoords of the point (screen_x, screen_y) ranging from
-1 to +1
                p=(hms*pmi) + (1000*ortho_d)
                p.resize3D()
                d[:] = ortho_d[:3]


            # Finally we shift the position infinitely far away in
            # the viewing direction to make sure the camera if outside the scene
            # (this is actually a hack because this function
            # is used in sculpt_mesh to initialize backface culling...)
            else:
                # PERSPECTIVE MODE: here everything is well defined - all rays converge at the camera's
location
                vmi  = Matrix(Window.GetViewMatrix()); vmi.invert() # the inverse viewing matrix
                fp = getPickRay.fp

                dx = pm[3][3] * (((screen_x-win_min_x)/win_size_x)-1.0) - pm[3][0]
                dy = pm[3][3] * (((screen_y-win_min_y)/win_size_y)-1.0) - pm[3][1]

                fp[:] = \
                pmi[0][0]*dx+pmi[1][0]*dy,\
                pmi[0][1]*dx+pmi[1][1]*dy,\
                pmi[0][2]*dx+pmi[1][2]*dy

                # fp is a global 3dpoint obtained from "unprojecting" the screenspace-point (screen_x,
screen_y)
                #- figuring out how to calculate this took me quite some time.
                # The calculation of dxy and fp are simplified versions of my original code
                #- so it's almost impossible to explain what's going on geometrically... sorry
                p[:] = vmi[3][:3]
```

```
                # the camera's location in global 3dcoords can be read directly from the inverted
viewmatrix
                d[:] = p.x-fp.x, p.y-fp.y, p.z-fp.z


            # the direction vector is simply the difference vector from the virtual camera's position
            #to the unprojected (screenspace) point fp

            # Do we want to return a direction in object's localspace?
            if localMatrix:
                localInvMatrix = Matrix(localMatrix)
                localInvMatrix.invert()
                p = p*localInvMatrix
                d = d*localInvMatrix  # normalize_v3
                p.x += localInvMatrix[3][0]
                p.y += localInvMatrix[3][1]
                p.z += localInvMatrix[3][2]

            #else: # Worldspace, do nothing

            d.normalize()
            return True, p, d # Origin, Direction

    # Mouse is not in any view, return None.
    return False, None, None

# Constant function variables
getPickRay.d = Vector(0,0,0) # Perspective, 3d
getPickRay.p = Vector(0,0,0)
getPickRay.fp = Vector(0,0,0)

getPickRay.hms = Vector(0,0,0,0) # ortho only 4d
getPickRay.ortho_d = Vector(0,0,0,0) # ortho only 4d




# TEST FUNCTION
# MOVES & VERTS ON THE ACTIVE MESH.
def main():
    ob = Scene.GetCurrent().getActiveObject()
    me = ob.getData(mesh=1)

    # Loop until the mouse is in the view.
    mouseInView = False
    while not mouseInView:
        screen_x, screen_y = Window.GetMouseCoords()
        mouseInView, Origin, Direction = getPickRay(screen_x, screen_y)

    if Window.GetMouseButtons() == 1 and mouseInView:
        i = 0
        time = Blender.sys.time()
        while Window.GetMouseButtons() == 1:
            i+=1
            screen_x, screen_y = Window.GetMouseCoords()
            mouseInView, Origin, Direction = getPickRay(screen_x, screen_y, ob.matrix)
            if mouseInView:

                me.verts[0].co.x = Origin.x
                me.verts[0].co.y = Origin.y
                me.verts[0].co.z = Origin.z

                me.verts[1].co.x = Origin.x - (Direction.x*1000)
                me.verts[1].co.y = Origin.y - (Direction.y*1000)
                me.verts[1].co.z = Origin.z - (Direction.z*1000)
                Window.Redraw(Window.Types.VIEW3D)
        print '100 draws in %.6f' % (((Blender.sys.time()-time) / float(i))*100)

if __name__ == '__main__':
    main()
```

Auto Buttons

Auto Buttons is a really easy way to add a stack of buttons into any script.

Add the AutoButtons text to the bottom of any script and any function ending with**bgui** will have a button that calls it.

```
# All functions to be displayed as buttons must use this suffix
GUI_SUFFIX= '_bgui'
BUTTON_LIST = [] # A list if dicts
EVENT = 1000
EVENTNUM = 1000
```

```
for func in dir():
    if func.endswith(GUI_SUFFIX):
        newButton = {}
        newButton['name'] = func[:-5].replace('_', ' ')[2:]
        newButton['func'] = func + '()'
        newButton['event'] = EVENT
        BUTTON_LIST.append( newButton )
        EVENT+=1

def draw_gui():
    # find the width of the widest button
    button_height = 16; button_width = 100; ROW = 0
    for button in BUTTON_LIST:
        Draw.PushButton(button['name'], button['event'], 0, button_height*ROW, button_width,
button_height, ''); ROW+=1
def handle_event(evt, val):
    if evt in (Draw.ESCKEY, Draw.QKEY) and not val:
        Draw.Exit()
def handle_button_event(evt):
    if evt >= EVENTNUM and evt < EVENTNUM + len(BUTTON_LIST):
        exec(BUTTON_LIST[evt - EVENTNUM]['func'])
    else:
        print 'invalid', evt
Draw.Register(draw_gui, handle_event, handle_button_event)
```

An example of a function that could use this

```
def Print_Object_Selection_bgui():
    Blender.Draw.PupMenu('|'.join(ob.name for ob in Blender.Object.GetSelected()))
```

Popup Menu Wrapper

This script takes a string that you would normally give to **Draw.PupMenu()** splitting up the menus by the groupsize.

```
def PupMenuLess(menu, groupSize=30):
    '''
    Works like Draw.PupMenu but will add a more/less buttons if the number of
    items is greater then the groupSize.
    '''
    more = ['   more...']
    less = ['   less...']

    menuList= menu.split('|')

    # No Less Needed, just call.
    if len(menuList) < groupSize:
        return Draw.PupMenu(menu)

    title = menuList[0].split('%t')[0]

    # Split the list into groups
    menuGroups = [[]]
    for li in menuList[1:]:
        if len(menuGroups[-1]) < groupSize:
            menuGroups[-1].append(li)
        else:
            menuGroups.append([li])

    # Stores the current menu group we are looking at
    groupIdx = 0
    while True:
        # Give us a title with the menu number
        numTitle = [ ' '.join([title, str(groupIdx + 1), 'of', str(len(menuGroups)), '%t'])]
        if groupIdx == 0:
            menuString = '|'.join(numTitle + menuGroups[groupIdx] + more)
        elif groupIdx == len(menuGroups)-1:
            menuString = '|'.join(numTitle + less + menuGroups[groupIdx])
        else: # In the middle somewhere so Show a more and less
            menuString = '|'.join(numTitle + less + menuGroups[groupIdx] + more)
        result = Draw.PupMenu(menuString)
        # User Exit
        if result == -1:
            return -1

        if groupIdx == 0: # First menu
            if result-1 < groupSize:
```

```
                return result
            else: # must be more
                groupIdx +=1
        elif groupIdx == len(menuGroups): # Last Menu
            if result == 1: # Must be less
                groupIdx -= 1
            else: # Must be a choice
                return result + (groupIdx*groupSize)

        else:
            if result == 1: # Must be less
                groupIdx -= 1
            elif result-2 == groupSize:
                groupIdx +=1
            else:
                return result - 1 + (groupIdx*groupSize)
```

General Python

Here add general python code, useful when Python scripting.

Benchmark Script

Script that times different functions.

```
def loopFor():
    '''For loop test'''
    for i in xrange(1000000):
        a=i

def loopWhile():
    '''While loop test'''
    i=0
    while i<1000000:
        a=i
        i+=1

def time_func(bench_func, iter=4):
    ''' Run the function 10 times '''
    print '',bench_func.__doc__
    t= Blender.sys.time()
    for i in xrange(iter):
        bench_func()
    tme= (Blender.sys.time()-t) / 10
    print '\tBenchmark %.4f average sec' % tme
    return tme

def main():
    print '\nRunning tests'
    time_func(loopFor)
    time_func(loopWhile)

if __name__ == '__main__':
    main()
```

Iterate Multiple Lists

Sometimes you want to loop over more than 1 list at once. if the lists your dealing with are large then creating a new list for the purpose can be slow and use too much memory. This class takes multiple lists and treats them like 1 big list. without having to make a new list.

```
type_list= type([])
type_tuple= type(())
class listIter:
    def __init__(self, lists):
        if type(lists) != type_list:
            self.lists= list(lists)
        else:
            self.lists= lists
        self.idx= self.lidx= 0
```

```
    def next(self):
        if self.lidx==len(self.lists):
            raise StopIteration
        idx=self.idx
        lidx=self.lidx
        self.idx+=1
        if self.idx==len(self.lists[self.lidx]):
            self.idx= 0
            self.lidx+=1

        return self.lists[lidx][idx]

    def __iter__(self):
        return self
    def __getitem__(self, index):
        i=0
        for l in self.lists:
            if i+len(l)>index:
                return l[index-i]
            i+=len(l)
        raise IndexError
    def __setitem__(self, index, value):
        i=0
        for l in self.lists:
            if i+len(l)>index:
                l[index-i]= value
                return
            i+=len(l)
        raise IndexError

    def __len__(self):
        length=0
        for l in self.lists:
            length+=len(l)
        return length

    def index(self, value):
        i=0
        for l in self.lists:
            for li in l:
                if li == value:
                    return i
                i+=1
        raise ValueError

    def remove(self, value):
        for l in self.lists:
            if value in li:
                l.remove(i)
                return
        raise ValueError

    def count(self, value):
        return sum(l.count(value) for l in self.lists)

    def extend(self, value):
        for i in value: # See its an iterator
            break
        self.lists.append(value)

    def pop(self, index):
        i=0
        for l in self.lists:
            if i+len(l)>index:
                return l.pop(index-i)
            i+=len(l)
        raise IndexError
    def __str__(self):
        return '['+ ''.join(str(l)[1:-1] for l in self.lists) +']'

    def sort(self):
        '''Cant to a full sort, just do a par'''
        self.lists.sort()
        for l in self.lists:
            l.sort()

    def append(self, value):
        self.lists[-1].append(value)

    def reverse(self):
        for l in self.lists:
            l.reverse()
        self.lists.reverse()
```

Some examples

```
for i in listIter( (range(10), range(22), range(5)) ):
    print i
```

Another example that takes verts from 3 meshes and adds them to 1 mesh using this iterator and list comprehension.

```
from Blender import Mesh
newme= Mesh.New()

# Using the iterator
newme.verts.extend( [v.co for v in listIter((me1.verts, me2.verts, me3.verts))] )

# Without the iterator
newme.verts.extend( [v.co for v in me1.verts ] )
newme.verts.extend( [v.co for v in me2.verts ] )
newme.verts.extend( [v.co for v in me3.verts ] )
```

Binary Conversion (Without Struct)

Thanks to SJH 07/29/2004 20:26:03

```
nybblechr_to_01_dqs ={'-':'-','0':'0000', '1':'0001', '2':'0010', '3':'0011',
                      '4':'0100', '5':'0101', '6':'0110', '7':'0111',
                      '8':'1000', '9':'1001', 'A':'1010', 'B':'1011',
                      'C':'1100', 'D':'1101', 'E':'1110', 'F':'1111'}

# Int to binary
def i2b(j, wd=0):
    return ''.join(nybblechr_to_01_dqs [x] for x in '%02X' % j))[-wd:].zfill(wd)

# Char to binary
def c2b(c, wd=0):
    return i2b(ord(c))

# String to binary
def s2b(s, wd=0):
    return ''.join(nybblechr_to_01_dqs [x] for x in ''.join('%02X' % ord(c) for c in s))[-wd:].zfill(wd)

# Binary to char
def b2c(b):
    chr(int(b,2))
```

Randomize List

Returns a randomized list.**Note** if you can import random, use random.shuffle(ls) instead.

```
def randList(ls):
    lsCopy = ls[:]
    randList = []
    lenList = len(lsCopy)
    while lenList != len(randList):
        randIndex = int( Noise.random() * len(lsCopy)  )
        randList.append( lsCopy.pop( randIndex ) )
    return randList
```

Remove Doubles in List

Removes doubles in a list, modifying the original list. (will use an objects cmp() function)

```
def RemDoubles(List):
    lIdx = 0
    while lIdx < len(List):
        if List.count(List[lIdx]) > 1:
            List.pop(lIdx)
            continue
        lIdx+=1
```

Remove Doubles in List (Hash)

Return a new list with no doubles.

```
def RemDoublesHash(myList):
    return list(set(myList))
```

Get flag properties of a sum

A lot of properties in Blender are flags and stored in a sum of exponentials of 2. To find out that a specific flag is set and it is included in the sum try this function:

```
def powList(self, x):
    tmpx = x
    exp = 0
    expList = []
    while tmpx != 0:
        tmp = 2**exp
        if tmp > tmpx:
            elem = 2**(exp-1)
            expList.append(elem)
            tmpx -= elem
            exp = 0
        else:
            exp += 1;
    return expList
```

Call the function in that way:

```
lmp = Lamp.Get(thisObj.data.getName())
lmpMode = lmp.getMode()
lmpFlags = self.powList(lmpMode)
if 16 in lmpFlags:
    ...
```

Fraction Data Type

Allows creation of fraction data and all operations on them within the real number system.

```
class fraction:
    # Types without importing type - Does not retuire a python install.
    type_float = type(0.1)
    type_int = type(1)

    def __init__(self, num, den=1):
        if den == 0:
            raise ValueError, 'Division by zero'
        g = self.gcd(num, den)
        self.num = num / g
        self.den = den / g

    def __str__(self):
        return "%d/%d" % (self.num, self.den)

    def __mul__(self, other):
        if type(other) is fraction.type_int:
            other = fraction(other)
        elif type(other) is fraction.type_float:
            return self.eval() * other
        if not isinstance(other, fraction):
            raise ValueError, 'Unsupported operand type for multiply operation '   + str(type(other))
        return fraction(self.num * other.num, self.den * other.den)

    __rmul__ = __mul__

    def __add__(self, other):
        if type(other) is fraction.type_int:
            other = fraction(other)
        elif type(other) is fraction.type_float:
            return self.eval() + other
        if not isinstance(other, fraction):
```

```
            raise ValueError, 'Unsupported operand type for addition operation '   + str(type(other))
        num = self.num * other.den + self.den * other.num
        den = self.den * other.den
        return fraction(num, den)

    def __cmp__(self, other):
        if type(other) is fraction.type_int or type(other) is fraction.type_float:
            return self.eval() - other
        if not isinstance(other, fraction):
            raise ValueError, 'Comparative operation no supported for operand '   * type(other)
        return (self.num * other.den - other.num * self.den)

    def __neg__(self):
        return fraction(self.num * -1, self.den)

    def __invert__(self):
        return fraction(self.den, self.num)

    def __sub__(self, other):
        return self + -other

    def __rsub__(self, other):
        return other + -self

    def __div__(self, other):
        return fraction(self.num, self.den) * fraction(other.den, other.num)

    def __rdiv__(self, other):
        return fraction(self.den, self.num) * fraction(other.num, other.den)

    def __pow__(self, other):
        if type(other) is fraction.type_int:
            return fraction(self.num ** other, self.den ** other)
        elif type(other) is fraction.type_float:
            a = self.eval()
            if a > 0:
                return a ** other
            else:
                raise ValueError, 'Negative number raised to fractional power'
        if not isinstance(other, fraction):
            raise ValueError, 'Unsupported operand type for exponential operation '   + str(type(other))
        return self.eval() ** other.eval()

    def gcd(self, m, n):
        if m % n:
            return self.gcd(n, m % n)
        return n


    def eval(self):
        return float(self.num) / self.den

##### Usage: #####

a = fraction(3, 4)
b = fraction(5, 6)
print a * b
print a - 3
print a ** b

#invalid - raising a negative number to a fractional power
print (-a)**b
```

Get a name with sane chars

This function can be used when you are making a filename from an object/mesh/scene...name. Blender supports many characters in a name that a filesystem may not.

saneFilechars replaces these characters with "_"

```
def saneFilechars(name):
    for ch in ' /\\~!@#$%^&*()+=[];\':",./<>?\t\r\n':
        name = name.replace(ch, '_')
    return name
```

Colour Scripts

A place for scripts that deal with colours.

RGB to HSV

Convert Red/Green/Blue to Hue/Saturation/Value

r,g,b values are from 0.0 to 1.0

h = [0,360], s = [0,1], v = [0,1]

if s == 0, then h = -1 (undefined)

The Hue/Saturation/Value model was created by A. R. Smith in 1978. It is based on such intuitive color characteristics as tint, shade and tone (or family, purety and intensity). The coordinate system is cylindrical, and the colors are defined inside a hexcone. The hue value H runs from 0 to 360º. The saturation S is the degree of strength or purity and is from 0 to 1. Purity is how much white is added to the color, so S=1 makes the purest color (no white). Brightness V also ranges from 0 to 1, where 0 is the black.

```
def RGBtoHSV(R,G,B):
    # min, max, delta;
    min_rgb = min( R, G, B )
    max_rgb = max( R, G, B )
    V = max_rgb

    delta = max_rgb - min_rgb
    if not delta:
        H = 0
        S = 0
        V = R # RGB are all the same.
        return H,S,V

    elif max_rgb: # != 0
        S = delta / max_rgb
    else:
        R = G = B = 0 # s = 0, v is undefined
        S = 0
        H = 0 # -1
        return H,S,V

    if R == max_rgb:
        H = ( G - B ) / delta # between yellow & magenta
    elif G == max_rgb:
        H = 2 + ( B - R ) / delta # between cyan & yellow
    else:
        H = 4 + ( R - G ) / delta # between magenta & cyan

    H *= 60 # convert to deg
    if H < 0:
        H += 360

    return H,S,V
```

HSV to RGB

Convert Hue/Saturation/Value to Red/Green/Blue

```
def HSVtoRGB(H,S,V):
    if not S: # S == 0
        # achromatic (grey)
        # R = G = B = V
        return V,V,V # RGB == VVV

    H /= 60;            # sector 0 to 5
    i = int( H ) # round down to int. in C its floor()
    f = H - i           # factorial part of H
    p = V * ( 1 - S )
    q = V * ( 1 - S * f )
    t = V * ( 1 - S * ( 1 - f ) )

    if i == 0:
```

```
        R,G,B = V,t,p
    elif i == 1:
        R,G,B = q,V,p
    elif i == 2:
        R,G,B = p,V,t
    elif i == 3:
        R,G,B = p,q,V
    elif i == 4:
        R,G,B = t,p,V
    else: # 5
        R,G,B = V,p,q
    return R,G,B
```

Interactive Tools

Freehand Polyline Draw Tool

```
from Blender import *

def main():
        # New Curve and add to Scene.
        scn = Scene.GetCurrent()
        cu = Curve.New()
        # cu.setResolu(1)
        x=y=z=w=t = 1
        cu.appendNurb([x,y,z,w,t])
        cu[0].type = 0 # Poly line
        ob = Object.New('Curve')
        ob.link(cu)
        scn.link(ob)
        ob.sel = 1 # Make active and selected

        # Initialize progress bar for writing
        Window.DrawProgressBar(0.0, '')

        ticker = 0.0 # Used to cycle the progress bar

        # Pause before drawing
        while not Window.GetMouseButtons() & Window.MButs['L']:
                sys.sleep(10)

                Window.DrawProgressBar(ticker, 'Left Mouse to Draw')
                ticker += 0.01
                if ticker > 0.98: ticker = 0

        oldx=oldy = -100000
        # Mouse Clicked, lets draw
        while Window.GetMouseButtons() & Window.MButs['L']:
                x,y = Window.GetMouseCoords()
                print abs(x-oldx)+abs(y-oldy)
                if (oldx == x and oldy == y) or abs(x-oldx)+abs(y-oldy) < 10: # Mouse must have moved 10
before adding the next point
                        pass
                else:
                        z = 0 # 2D Drawing for now
                        w = 100 #Weight is 1
                        cu.appendPoint(0, [x*0.001,y*0.001,z,w]) # Can add tilt here.
                        cu.update()
                        Window.Redraw(Window.Types.VIEW3D)

                        Window.DrawProgressBar(ticker, 'Drawing...')
                        ticker += 0.01
                        if ticker > 0.98: ticker = 0

                        oldx,oldy = x,y # Store the old mouse location to compare with new.

        # Clear the progress bar
        Window.DrawProgressBar(1.0, '')

main()
```

Render Functions

Render to a dir

```
# recursive dir creation.
def _mkdir(newdir):
    import os, sys
    """works the way a good mkdir should :)
            - already exists, silently complete
            - regular file in the way, raise an exception
            - parent directory(ies) does not exist, make them as well
    """
    if os.path.isdir(newdir):
        pass
    elif sys.exists(newdir):
        raise OSError("a file with the same name as the desired "   \
                                "dir, '%s', already exists."  % newdir)
    else:
            head, tail = os.path.split(newdir)
            if head and not os.path.isdir(head):
                _mkdir(head)
            #print "_mkdir %s" % repr(newdir)
            if tail:
                os.mkdir(newdir)

from Blender import *
from Blender.Scene import Render
if sys.sep == '\\':
    path="c:\\tmp\\renderfarm\\render"
else:
    path="/tmp/renderfarm/render"

# Should probably create the paths if not existing.
_mkdir(path)

scene= Scene.GetCurrent()
context = scene.getRenderingContext()
context.setRenderPath(path)
context.setImageType(Scene.Render.PNG)
context.enableExtensions(1)

context.renderAnim()
```

Render from all cameras

This script renders an animation from all cameras in a scene, it makes a new name from the camera and leaves the scene as it was originally.

Be sure to use useful camera names.

```
from Blender import Object, Scene

sce= Scene.GetCurrent()
context = sce.getRenderingContext()
output_path_orig = context.getRenderPath()

cams= [ob for ob in sce.getChildren() if ob.getType()=='Camera']

# backup the active cam.
orig_cam= sce.getCurrentCamera()

# loop over all the cameras in this scene, set active and render.
for i, c in enumerate(cams):
    print '\tRendering %i of %i cameras.' % (i, len(cams))
    context.setRenderPath('%s_%s_' % (output_path_orig, c.name)) # use a unique name
    sce.setCurrentCamera(c) # set this camera to be active.
    context.renderAnim()
print 'Done per camera render'

if orig_cam:
    sce.setCurrentCamera(orig_cam) # restore the original cam
```

```
# restore the original path.
context.setRenderPath(output_path_orig)
```

Scriptlinks

Monitor Image

This Script needs to be used as a redraw scriptlink. It checks the date of the current image and attempts to reload the image, and if successful it redraws the image.

```
import Blender
try:
    Blender.my_image_time
except:
    Blender.my_image_time=0
import os
img= Blender.Image.GetCurrent() # currently displayed picture.
if img: # Image isnt None
    path= Blender.sys.expandpath(img.filename)
    if Blender.sys.exists(path):
        t= os.path.getctime(path)
        if t != Blender.my_image_time:
            img.reload()
            Blender.Window.Redraw(Blender.Window.Types.IMAGE)
        Blender.my_image_time= t # global, persists between running the scripts.
```

Dynamic Text

This script needs to be used as a FrameChanged script linked to a Text object. Change the Years.append line to choice the years of the timeline.

```
import Blender as B

Years = []

# year = [year, initial frame, duration of frames]
Years.append([1800, 1, 10])
Years.append([1850, 100, 50])
Years.append([1994, 170, 100])
Years.append([2008, 300, 50])
Years.append([2050, 400, 50])

def when (frame,  years):

    iniY = 0

    for y in range(len(years)):
        if frame > years[y][1]:
            iniY = y

    iniYear       = years[iniY][0]
    iniFrame      = years[iniY][1]
    iniFrameDelay = years[iniY][2]

    finYear  = years[iniY+1][0]
    finFrame = years[(iniY+1)][1]

    frameRange = finFrame - (iniFrame + iniFrameDelay)
    yearRange = finYear - iniYear

    normFrame = float(frame - iniFrame - iniFrameDelay)
    normFrame = normFrame/frameRange

    if normFrame > 0:
        newYear = str(int(iniYear + (yearRange * normFrame)))
    else:
        newYear = iniYear

    return str(newYear)

if B.bylink:
```

```
    actualFrame = B.Get("curframe")

    year = B.link
    dynYear = year.getData()

    oldYear=dynYear.getText()
    newYear=when (actualFrame,Years)

    if newYear != oldYear:
        dynYear.setText(newYear)
        year.makeDisplayList()
        B.Window.RedrawAll()
```

External Utils

Compress All Blend files (Unix Only)

Sine blender 2.41, support for GZip compression has been integrated into blender. So you can gzip all blend files and they will still open as expected.

This utility searches your hard-disk for blend files and gzips them if they are not already compressed.

Note: python3 required.

```
#!/usr/bin/python3

root_dir = '/mango/pro'

import os
import os.path


def blend_path_list(path):
    for dirpath, dirnames, filenames in os.walk(path):
        for filename in filenames:
            if filename.endswith(".blend"):
                yield os.path.join(dirpath, filename)


def isblend_nogz(path):
    try:
        f = open(path, 'rb')
        is_blend = (f.read(7) == b'BLENDER')
        f.close()
        return is_blend
    except:
        return False


def main():
    print('Searching "%s"...' % root_dir)
    files = list(blend_path_list(root_dir))
    files.sort()
    print('done.')
    #print files
    tot_files = len(files)
    tot_compressed = tot_blends = tot_alredy_compressed = 0
    tot_blend_size = 0
    tot_blend_size_saved = 0
    for f in files:
        if len(f) >= 6:  # .blend is 6 chars
            f_lower = f.lower()
            if (f_lower.endswith(".blend") or
                f_lower[:-1].endswith(".blend") or
                f_lower[:-2].endswith(".blend")):  # .blend10 +

                print(f, "...", end="")
                tot_blends += 1
                # allows for dirs with .blend, will just be false.
                if isblend_nogz(f):
                    print("compressing ...", end="")
                    tot_compressed += 1
                    orig_size = os.path.getsize(f)
                    tot_blend_size += orig_size
                    os.system('gzip --best "%s"' % f)
                    os.system('mv "%s.gz" "%s"' % (f, f)) # rename the gz file to the original.
```

```
                    new_size = os.path.getsize(f)
                    tot_blend_size_saved += orig_size - new_size
                    print('saved %.2f%%' % (100 - (100 * (float(new_size) / orig_size))))
                else:
                    print('alredy compressed.' )
                    tot_alredy_compressed += 1

    print('\nTotal files:', tot_files)
    print('Total Blend:', tot_blends)
    print('Total Blend Compressed:', tot_compressed)
    print('Total Alredy Compressed:', tot_alredy_compressed)
    print('\nTotal Size in Blends: %sMB' % (((tot_blend_size) / 1024) / 1024))
    print('Total Saved in Blends: %sMB' % (((tot_blend_size_saved) / 1024) / 1024))

if __name__ == '__main__':
        main()
```
