def uniquer(seq, idfun=None):

if idfun is None:

def idfun(x):

return x

seen = {}

result = []

for item in seq:

marker = idfun(item)

if marker in seen:

continue

seen[marker] = 1

result.append(item)

return result

def \_md5\_for\_file(filepath):

md5hash = hashlib.md5()

blocksize = 0x10000

f = open(filepath, "rb")

while True:

data = f.read(blocksize)

if not data:

break

md5hash.update(data)

return md5hash.hexdigest()

def unzip(fname):

import zipfile

zf = zipfile.ZipFile(fname, 'r')

for name in zf.namelist():

dir = os.path.dirname(name)

try:

os.makedirs(dir)

except:

pass

open(name, 'w').write(zf.read(name))

def do\_diskcheck\_match(node, predicate, errorfmt):

result = predicate()

try:

if node.\_memo['stat'] is None:

del node.\_memo['stat']

except (AttributeError, KeyError):

pass

if result:

raise TypeError(errorfmt % node.abspath)

def asVertsLocation(verts2d, faces) :

t = clock()

uv = []

for f in faces :

uvface = []

for vi in f :

uvface.extend(verts2d[vi])

uv.append(uvface)

print('uvs convert in', clock() - t)

return uv

def read\_lwostring(raw\_name):

i = raw\_name.find(b'\0')

name\_len = i + 1

if name\_len % 2 == 1:

name\_len += 1

if i > 0:

name = raw\_name[0:i].decode("utf-8", "ignore")

else:

name = ""

return name, name\_len

def \_do\_create\_list\_action(act, kw):

acts = []

for a in act:

aa = \_do\_create\_action(a, kw)

if aa is not None: acts.append(aa)

if not acts:

return ListAction([])

elif len(acts) == 1:

return acts[0]

else:

return ListAction(acts)

def \_GetInfoRNA(bl\_rna, cls, parent\_id=""):

if bl\_rna is None:

return None

key = parent\_id, bl\_rna.identifier

try:

return cls.global\_lookup[key]

except KeyError:

instance = cls.global\_lookup[key] = cls(bl\_rna)

return instance

def \_BorderEdges(facelist):

ans = set()

for i in range(0, len(facelist)):

f = facelist[i]

for j in range(1, len(f)):

ans.add((f[j - 1], f[j]))

ans.add((f[-1], f[0]))

return ans

def \_IsDiag(i, v, hv, face, points):

n = len(face)

vm1 = face[(i - 1) % n]

v1 = face[(i + 1) % n]

k = \_AngleKind(vm1, v, v1, points)

if not \_InCone(hv, vm1, v, v1, k, points):

return False

for j in range(0, n):

vj = face[j]

vj1 = face[(j + 1) % n]

if SegsIntersect(v, hv, vj, vj1, points):

return False

return True

def removeDoubles(ob):

global theMergeLimit

if toggle & T\_Merge:

scn = bpy.context.scene

scn.objects.active = ob

bpy.ops.object.mode\_set(mode = 'EDIT')

bpy.ops.mesh.remove\_doubles(threshold = theMergeLimit)

bpy.ops.object.mode\_set(mode = 'OBJECT')

def parseDefineProperty(args, tokens):

prop = "%sProperty" % (args[1])

c = '('

for option in args[2:]:

prop += "%s %s" % (c, option)

c = ','

prop += ')'

setattr(bpy.types.Object, args[0], prop)

return

def cleanup\_and\_quit():

status = 'Killing torrents..'

for file, threadinfo in threads.items():

status = 'Killing torrent %s' % file

threadinfo['kill'].set()

threadinfo['thread'].join()

del threads[file]

displaykiller.set()

displaythread.join()

def \_GetInfoRNA(bl\_rna, cls, parent\_id=""):

if bl\_rna is None:

return None

key = parent\_id, bl\_rna.identifier

try:

return cls.global\_lookup[key]

except KeyError:

instance = cls.global\_lookup[key] = cls(bl\_rna)

return instance

def resolve\_cert\_reqs(candidate):

if candidate is None:

return CERT\_NONE

if isinstance(candidate, str):

res = getattr(ssl, candidate, None)

if res is None:

res = getattr(ssl, 'CERT\_' + candidate)

return res

return candidate