|  |
| --- |
|  |
|  |

 **Menger Sponge Modifications**

After I was able to generate the Menger 3D sponge in Maya using the rib file I made alternations to the shape by playing with which number cube on the subdivision levels will be removed. Once I read the rib file into the Renderman archive path and created an alteration I played around with the lighting and shading of the shapes. I also used the sin functions to generate some interesting shapes. Also, when reading these modification scripts I continued to use the ri\_utils.py code and made no changes to it, but instead made adjusting to just the menger code.

 Menger Sponge Shape 1 Modifications

import ri\_utils

from math import sqrt

import math

class Menger3D:

 def \_\_init\_\_(self, bbox, depth, listOfHoles):

 self.deletedCubes = [] # list of deleted cubes

 self.retainedCubes = []

 listOfHoles.sort()

 listOfHoles.reverse()

 self.holeLUT = listOfHoles

 self.bbox = bbox # minx,miny,minz, maxx,maxy,maxz

 self.depth = depth

 self.divide(bbox, depth) # our recursive routine

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Given the minimum x,y,z and maximum x,y,z coordinates

 # of a bounding box this proc returns the bouding box

 # coordinates of a "row" of three cubes.

 def row(self, x0,y0,z0, w,h,d):

 x,y,z = x0,y0,z0

 X,Y,Z = x + w, y + h, z + d

 cubes = []

 for n in range(3):

 cube = [x,y,z, X,Y,Z]

 cubes.append(cube)

 z,Z = z + d, Z + d

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # A recursive proc that subdivides a bounding box into

 # 27 sub-cubes. Each time the proc is called the arg

 # "depth" is decremented. Recursion terminates when its

 # value becomes zero.

 def divide(self, bbox, depth):

 if depth == 0:

 self.retainedCubes.append(bbox)

 return []

 x0,y0,z0,x1,y1,z1 = bbox

 w = float(x1 - x0)/3

 h = float(y1 - y0)/3

 d = float(z1 - z0)/3

 x,y,z = x0,y0,z0

 cubes = []

 for layer in range(3):

 x = x0

 for rows in range(3):

 cubes.extend(self.row(x,y,z,w,h,d))

 x = x + w

 y = y + h

 cubes = self.delete(cubes)

 # Recursion\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 for cube in cubes:

 self.divide(cube, depth - 1)

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Uses the indices in the holeLUT to remove specific cubes

 # from the list of 27 cubes in the "cubes" arg.

 def delete(self,cubes):

 for n in range(len(self.holeLUT)):

 index = self.holeLUT[n]

 if index < len(cubes):

 deleted = cubes.pop(self.holeLUT[n])

 self.deletedCubes.append(deleted)

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Write a RenderMan archive rib file for the menger cubes

 # or for the cubes that were removed by the delete method.

 def writeAsCubes(self, rib\_path, cube\_type='retained'):

 f = open(rib\_path,'w')

 bboxStr = ' '.join(map(str, self.bbox))

 f.write('#bbox: %s\n' % bboxStr)

 if cube\_type == 'retained':

 cubes = self.retainedCubes

 else:

 cubes = self.deletedCubes

 counter = 1

 for cube in cubes:

 f.write(ri\_utils.Cube(cube, counter))

 counter += 1

 f.close()

 return len(cubes)

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Not used in this example implementation but could be

 # used to cull cubes on the basis of their distance from

 # a central location - to create a menger sphere.

 def distance(self, p1, p2):

 x = p1[0]-p2[0]

 y = p1[1]-p2[1]

 z = p1[2]-p2[2]

 return sqrt(x \* x + y \* y + z \* z)

#=======================================================

if \_\_name\_\_=="\_\_main\_\_":

 bounds = [-1,0,-1, 1,2,1]

 removals = [20,11,26,8,0,15,16]

 menger3d = Menger3D(bounds, 4, removals)

 menger3d.writeAsCubes('/home/njones26/mount/stuhome/tech312/python/Menger Sponge/menger3d.rib')

 menger3d.writeAsCubes('/home/njones26/mount/stuhome/tech312/python/Menger Sponge/menger3d\_holes.rib','')

Menger Sponge Shape 2 Modifications

import ri\_utils

from math import sqrt

import math

class Menger3D:

 def \_\_init\_\_(self, bbox, depth, listOfHoles):

 self.deletedCubes = [] # list of deleted cubes

 self.retainedCubes = []

 listOfHoles.sort()

 listOfHoles.reverse()

 self.holeLUT = listOfHoles

 self.bbox = bbox # minx,miny,minz, maxx,maxy,maxz

 self.depth = depth

 self.divide(bbox, depth) # our recursive routine

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Given the minimum x,y,z and maximum x,y,z coordinates

 # of a bounding box this proc returns the bouding box

 # coordinates of a "row" of three cubes.

 def row(self, x0,y0,z0, w,h,d):

 x,y,z = x0,y0,z0

 X,Y,Z = x + w, y + h, z + d

 cubes = []

 for n in range(3):

 cube = [x,y,z, X,Y,Z]

 cubes.append(cube)

 z,Z = z + d, Z + d

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # A recursive proc that subdivides a bounding box into

 # 27 sub-cubes. Each time the proc is called the arg

 # "depth" is decremented. Recursion terminates when its

 # value becomes zero.

 def divide(self, bbox, depth):

 if depth == 0:

 self.retainedCubes.append(bbox)

 return []

 x0,y0,z0,x1,y1,z1 = bbox

 w = float(x1 - x0)/3

 h = math.sin(float(y1 - y0)/3)

 d = float(z1 - z0)/3

 x,y,z = x0,y0,z0

 cubes = []

 for layer in range(3):

 x = x0

 for rows in range(3):

 cubes.extend(self.row(x,y,z,w,h,d))

 x = x + w

 y = y + h

 cubes = self.delete(cubes)

 # Recursion\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 for cube in cubes:

 self.divide(cube, depth - 1)

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Uses the indices in the holeLUT to remove specific cubes

 # from the list of 27 cubes in the "cubes" arg.

 def delete(self,cubes):

 for n in range(len(self.holeLUT)):

 index = self.holeLUT[n]

 if index < len(cubes):

 deleted = cubes.pop(self.holeLUT[n])

 self.deletedCubes.append(deleted)

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Write a RenderMan archive rib file for the menger cubes

 # or for the cubes that were removed by the delete method.

 def writeAsCubes(self, rib\_path, cube\_type='retained'):

 f = open(rib\_path,'w')

 bboxStr = ' '.join(map(str, self.bbox))

 f.write('#bbox: %s\n' % bboxStr)

 if cube\_type == 'retained':

 cubes = self.retainedCubes

 else:

 cubes = self.deletedCubes

 counter = 1

 for cube in cubes:

 f.write(ri\_utils.Cube(cube, counter))

 counter += 1

 f.close()

 return len(cubes)

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Not used in this example implementation but could be

 # used to cull cubes on the basis of their distance from

 # a central location - to create a menger sphere.

 def distance(self, p1, p2):

 x = p1[0]-p2[0]

 y = p1[1]-p2[1]

 z = p1[2]-p2[2]

 return sqrt(x \* x + y \* y + z \* z)

#=======================================================

if \_\_name\_\_=="\_\_main\_\_":

 bounds = [-1,0,-1, 1,2,1]

 removals = [26,14,3,2,10,15,16]

 menger3d = Menger3D(bounds, 4, removals)

 menger3d.writeAsCubes('/home/njones26/mount/stuhome/tech312/python/Menger Sponge/menger3d.rib')

 menger3d.writeAsCubes('/home/njones26/mount/stuhome/tech312/python/Menger Sponge/menger3d\_holes.rib','')

Menger Sponge Shape 3 Modifications

import ri\_utils

from math import sqrt

import math

class Menger3D:

 def \_\_init\_\_(self, bbox, depth, listOfHoles):

 self.deletedCubes = [] # list of deleted cubes

 self.retainedCubes = []

 listOfHoles.sort()

 listOfHoles.reverse()

 self.holeLUT = listOfHoles

 self.bbox = bbox # minx,miny,minz, maxx,maxy,maxz

 self.depth = depth

 self.divide(bbox, depth) # our recursive routine

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Given the minimum x,y,z and maximum x,y,z coordinates

 # of a bounding box this proc returns the bouding box

 # coordinates of a "row" of three cubes.

 def row(self, x0,y0,z0, w,h,d):

 x,y,z = x0,y0,z0

 X,Y,Z = x + w, y + h, z + d

 cubes = []

 for n in range(3):

 cube = [x,y,z, X,Y,Z]

 cubes.append(cube)

 z,Z = z + d, Z + d

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # A recursive proc that subdivides a bounding box into

 # 27 sub-cubes. Each time the proc is called the arg

 # "depth" is decremented. Recursion terminates when its

 # value becomes zero.

 def divide(self, bbox, depth):

 if depth == 0:

 self.retainedCubes.append(bbox)

 return []

 x0,y0,z0,x1,y1,z1 = bbox

 w = math.sin(float(x1 - x0)/3)

 h = math.sin(float(y1 - y0)/3)

 d = math.sin(float(z1 - z0)/3)

 x,y,z = x0,y0,z0

 cubes = []

 for layer in range(3):

 x = x0

 for rows in range(3):

 cubes.extend(self.row(x,y,z,w,h,d))

 x = x + w

 y = y + h

 cubes = self.delete(cubes)

 # Recursion\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 for cube in cubes:

 self.divide(cube, depth - 1)

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Uses the indices in the holeLUT to remove specific cubes

 # from the list of 27 cubes in the "cubes" arg.

 def delete(self,cubes):

 for n in range(len(self.holeLUT)):

 index = self.holeLUT[n]

 if index < len(cubes):

 deleted = cubes.pop(self.holeLUT[n])

 self.deletedCubes.append(deleted)

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Write a RenderMan archive rib file for the menger cubes

 # or for the cubes that were removed by the delete method.

 def writeAsCubes(self, rib\_path, cube\_type='retained'):

 f = open(rib\_path,'w')

 bboxStr = ' '.join(map(str, self.bbox))

 f.write('#bbox: %s\n' % bboxStr)

 if cube\_type == 'retained':

 cubes = self.retainedCubes

 else:

 cubes = self.deletedCubes

 counter = 1

 for cube in cubes:

 f.write(ri\_utils.Cube(cube, counter))

 counter += 1

 f.close()

 return len(cubes)

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Not used in this example implementation but could be

 # used to cull cubes on the basis of their distance from

 # a central location - to create a menger sphere.

 def distance(self, p1, p2):

 x = p1[0]-p2[0]

 y = p1[1]-p2[1]

 z = p1[2]-p2[2]

 return sqrt(x \* x + y \* y + z \* z)

#=======================================================

if \_\_name\_\_=="\_\_main\_\_":

 bounds = [-1,0,-1, 1,2,1]

 removals = [25,4,3,23,9,18,13]

 menger3d = Menger3D(bounds, 4, removals)

 menger3d.writeAsCubes('/home/njones26/mount/stuhome/tech312/python/Menger Sponge/menger3d.rib')

 menger3d.writeAsCubes('/home/njones26/mount/stuhome/tech312/python/Menger Sponge/menger3d\_holes.rib','')

Menger Sponge Shape 4 Modifications

# menger3d.py

# Implements a recursive subdivision of a cube into 27 sub

# cubes - three "layers" each with nine cubes. The indices

# in the "holeLUT" are used to delete specific cubes from

# each layer.

# Malcolm Kesson Jan 16 2013

# Converted to a class: Feb 9 2016

import ri\_utils

from math import sqrt

import math

import random

class Menger3D:

 def \_\_init\_\_(self, bbox, depth, listOfHoles):

 self.deletedCubes = [] # list of deleted cubes

 self.retainedCubes = []

 listOfHoles.sort()

 listOfHoles.reverse()

 self.holeLUT = listOfHoles

 self.bbox = bbox # minx,miny,minz, maxx,maxy,maxz

 self.depth = depth

 self.divide(bbox, depth) # our recursive routine

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Given the minimum x,y,z and maximum x,y,z coordinates

 # of a bounding box this proc returns the bouding box

 # coordinates of a "row" of three cubes.

 def row(self, x0,y0,z0, w,h,d):

 x,y,z = x0,y0,z0

 X,Y,Z = x + w, y + h, z + d

 cubes = []

 for n in range(3):

 cube = [x,y,z, X,Y,Z]

 cubes.append(cube)

 z,Z = z + d, Z + d

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # A recursive proc that subdivides a bounding box into

 # 27 sub-cubes. Each time the proc is called the arg

 # "depth" is decremented. Recursion terminates when its

 # value becomes zero.

 def divide(self, bbox, depth):

 if depth == 0:

 self.retainedCubes.append(bbox)

 return []

 x0,y0,z0,x1,y1,z1 = bbox

 w = float(x1 - x0)/3

 h = float(y1 - y0)/2

 d = float(z1 - z0)/3

 x,y,z = x0,y0,z0

 cubes = []

 for layer in range(3):

 x = x0

 for rows in range(3):

 cubes.extend(self.row(x,y,z,w,h,d))

 x = x + w

 y = y + h

 cubes = self.delete(cubes)

 # Recursion\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 for cube in cubes:

 self.divide(cube, depth - 1)

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Uses the indices in the holeLUT to remove specific cubes

 # from the list of 27 cubes in the "cubes" arg.

 def delete(self,cubes):

 for n in range(len(self.holeLUT)):

 index = self.holeLUT[n]

 if index < len(cubes):

 deleted = cubes.pop(self.holeLUT[n])

 self.deletedCubes.append(deleted)

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Write a RenderMan archive rib file for the menger cubes

 # or for the cubes that were removed by the delete method.

 def writeAsCubes(self, rib\_path, cube\_type='retained'):

 f = open(rib\_path,'w')

 bboxStr = ' '.join(map(str, self.bbox))

 f.write('#bbox: %s\n' % bboxStr)

 if cube\_type == 'retained':

 cubes = self.retainedCubes

 else:

 cubes = self.deletedCubes

 counter = 1

 for cube in cubes:

 f.write(ri\_utils.Cube(cube, counter))

 counter += 1

 f.close()

 return len(cubes)

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Not used in this example implementation but could be

 # used to cull cubes on the basis of their distance from

 # a central location - to create a menger sphere.

 def distance(self, p1, p2):

 x = p1[0]-p2[0]

 y = p1[1]-p2[1]

 z = p1[2]-p2[2]

 return sqrt(x \* x + y \* y + z \* z)

#=======================================================

if \_\_name\_\_=="\_\_main\_\_":

 bounds = [-1,0,-1, 1,2,1]

 removals = [random.randrange(1, 13) for \_ in range(0, 9)]

 menger3d = Menger3D(bounds, 4, removals)

 menger3d.writeAsCubes('/home/njones26/mount/stuhome/tech312/python/MengerSponge/menger3d\_6.rib')

 menger3d.writeAsCubes('/home/njones26/mount/stuhome/tech312/python/MengerSponge/menger3d\_holes(6).rib','')

Menger Sponge Shape 5 Modifications

# menger3d.py

# Implements a recursive subdivision of a cube into 27 sub

# cubes - three "layers" each with nine cubes. The indices

# in the "holeLUT" are used to delete specific cubes from

# each layer.

# Malcolm Kesson Jan 16 2013

# Converted to a class: Feb 9 2016

import ri\_utils

from math import sqrt

import math

import random

class Menger3D:

 def \_\_init\_\_(self, bbox, depth, listOfHoles):

 self.deletedCubes = [] # list of deleted cubes

 self.retainedCubes = []

 listOfHoles.sort()

 listOfHoles.reverse()

 self.holeLUT = listOfHoles

 self.bbox = bbox # minx,miny,minz, maxx,maxy,maxz

 self.depth = depth

 self.divide(bbox, depth) # our recursive routine

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Given the minimum x,y,z and maximum x,y,z coordinates

 # of a bounding box this proc returns the bouding box

 # coordinates of a "row" of three cubes.

 def row(self, x0,y0,z0, w,h,d):

 x,y,z = x0,y0,z0

 X,Y,Z = x + w, y + h, z + d

 cubes = []

 for n in range(3):

 cube = [x,y,z, X,Y,Z]

 cubes.append(cube)

 z,Z = z + d, Z + d

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # A recursive proc that subdivides a bounding box into

 # 27 sub-cubes. Each time the proc is called the arg

 # "depth" is decremented. Recursion terminates when its

 # value becomes zero.

 def divide(self, bbox, depth):

 if depth == 0:

 self.retainedCubes.append(bbox)

 return []

 x0,y0,z0,x1,y1,z1 = bbox

 w = float(x1 - x0)/3

 h = math.sin(float(y1 - y0)/3)

 d = float(z1 - z0)/3

 x,y,z = x0,y0,z0

 cubes = []

 for layer in range(3):

 x = x0

 for rows in range(3):

 cubes.extend(self.row(x,y,z,w,h,d))

 x = x + w

 y = y + h

 cubes = self.delete(cubes)

 # Recursion\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 for cube in cubes:

 self.divide(cube, depth - 1)

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Uses the indices in the holeLUT to remove specific cubes

 # from the list of 27 cubes in the "cubes" arg.

 def delete(self,cubes):

 for n in range(len(self.holeLUT)):

 index = self.holeLUT[n]

 if index < len(cubes):

 deleted = cubes.pop(self.holeLUT[n])

 self.deletedCubes.append(deleted)

 return cubes

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Write a RenderMan archive rib file for the menger cubes

 # or for the cubes that were removed by the delete method.

 def writeAsCubes(self, rib\_path, cube\_type='retained'):

 f = open(rib\_path,'w')

 bboxStr = ' '.join(map(str, self.bbox))

 f.write('#bbox: %s\n' % bboxStr)

 if cube\_type == 'retained':

 cubes = self.retainedCubes

 else:

 cubes = self.deletedCubes

 counter = 1

 for cube in cubes:

 f.write(ri\_utils.Cube(cube, counter))

 counter += 1

 f.close()

 return len(cubes)

 #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 # Not used in this example implementation but could be

 # used to cull cubes on the basis of their distance from

 # a central location - to create a menger sphere.

 def distance(self, p1, p2):

 x = p1[0]-p2[0]

 y = p1[1]-p2[1]

 z = p1[2]-p2[2]

 return sqrt(x \* x + y \* y + z \* z)

#=======================================================

if \_\_name\_\_=="\_\_main\_\_":

 bounds = [-2,0,-1, 2,3,1]

 removals = [random.randrange(1, 18) for \_ in range(0, 9)]

 menger3d = Menger3D(bounds, 4, removals)

 menger3d.writeAsCubes('/home/njones26/mount/stuhome/tech312/python/MengerSponge/menger3d.rib')

 menger3d.writeAsCubes('/home/njones26/mount/stuhome/tech312/python/MengerSponge/menger3d\_holes.rib','')