Colours

```
import Rhino.Geometry as rg
import Rhino.Display as rdis
import random as rd
import math as m

ptList = []
colList = []

rd.seed(s)

for i in range(50):
  for j in range(50):

    z = rd.gauss(me,dev)
    ptList.append(rg.Point3d(i,j,z))
    col = rdis.ColorHSL(z*0.2,1,0.5)
    colList.append(col.ToArgbColor())

a = ptList
```

//

Planarize

```
import Rhino.Geometry as rg
import Rhino.Display as rd

#Creamos un plano con tres puntos cualesquiera
plane = rg.Plane(pts[0],pts[1],pts[2])

#Proyectamos el punto 3 sobre el plano calculado
newPt3 = plane.ClosestPoint(pts[3])

#Calculamos el desplazamiento de cada punto
dev = pts[3].DistanceTo(newPt3)

#Creamos un color HSL(float) con dev y lo transformamos en RGB
devCol = rd.ColorHSL(dev*0.3,1,0.5)
devCol = devCol.ToArgbColor()

#Sustituimos el punto 3 por su nuevo valor
pts[3] = newPt3
lines = []

#Creamos una marca en cada panel fuera de tolerancia
if dev>T:

  l0 = rg.Line(pts[0],pts[2])
  l1 = rg.Line(pts[1],pts[3])
  lines.extend([l0,l1])

#Creamos la polilinea de cada panel y calculamos su area
pol = rg.PolylineCurve((pts[0],pts[1],pts[2],pts[3],pts[0]))

areaObj = rg.AreaMassProperties.Compute(pol)
area = areaObj.Area

w = lines
```

//

divSrf

```
import Rhino.Geometry as rg
from Grasshopper import DataTree as Tree
from Grasshopper.Kernel.Data import GH_Path as Path
ptList = []
polList = []

#Initializing an empty data tree
ptTree = Tree[object]()

for i in range(uDiv+1):
```

```
    ptListTemp = []

    for j in range(vDiv+1):

        ptTemp = srf.Evaluate(i/uDiv,j/vDiv,2)[1]
        ptListTemp.append(ptTemp)

    ptList.append(ptListTemp)

counter = 0
for i in range(uDiv):
    for j in range(vDiv):

        pt0 = ptList[i][j]
        pt1 = ptList[i+1][j]
        pt2 = ptList[i][j+1]
        pt3 = ptList[i+1][j+1]

        # Adding points to datatree
        # The commented lines produce the flipped tree.
        #ptTree.Add(pt0,Path(0))
        #ptTree.Add(pt1,Path(1))
        #ptTree.Add(pt2,Path(2))
        #ptTree.Add(pt3,Path(3))

        ptTree.AddRange([pt0,pt1,pt3,pt2],Path(counter))
        counter += 1

        polList.append(rg.Polyline([pt0,pt1,pt3,pt2]))

panel = polList
```

//

Koch_curve

```
import rhinoscriptsyntax as rs
import math as m

ptList = []

def koch(ptA, ptB, r0, r1, r2, rM):

    #points = []

    pt0 = ((ptB-ptA)*r0)+ptA
    pt1 = ((ptB-ptA)*r1)+ptA

    #Calculating vector perpendicular to ptB-ptA
    cross = rs.VectorCrossProduct((ptB-ptA),(0,0,1))
    cross = rs.VectorUnitize(cross)
    dist = rs.VectorLength((ptB-ptA))
    h = m.sqrt(((dist/3)**2.0)-((dist/6)**2.0))
    pt1 += cross*(h*rM)

    pt2 = ((ptB-ptA)*r2)+ptA

    #points.extend((pt0,pt1,pt2))
    return [ptA,pt0,pt1,pt2,ptB]

def recursive (ptA, ptB, gens, list):
    if gens>0:

        newPts = koch (ptA,ptB,rat0,rat1,rat2,ratM)
        curve = rs.AddPolyline(newPts)

        if gens == 1:
            list.append(curve)

        recursive(newPts[0],newPts[1],gens-1,list)
        recursive(newPts[1],newPts[2],gens-1,list)
        recursive(newPts[2],newPts[3],gens-1,list)
        recursive(newPts[3],newPts[4],gens-1,list)

        gens-=1
        return list

crv = recursive(PtA,PtB,G,ptList)
```

//

DivSrf + Panels

```
import Rhino.Geometry as rg
from Grasshopper import DataTree as Tree
from Grasshopper.Kernel.Data import GH_Path as Path


def planarize(pts,T):
    '''
    Creates a planar quad panel from a serie of 4 points projecting one of
    those over a plane defined by the other 3 points.

    Returns a list where [0] is a polyline
    and [1] is the panel area (Float).
    '''

    data = []

    #Creamos un plano con tres puntos cualesquiera
    plane = rg.Plane(pts[0],pts[1],pts[2])

    #Proyectamos el punto 3 sobre el plano calculado
    newPt3 = plane.ClosestPoint(pts[3])

    #Calculamos el desplazamiento de cada punto
    dev = pts[3].DistanceTo(newPt3)

    #Sustituimos el punto 3 por su nuevo valor
    pts[3] = newPt3
    lines = []

    #Creamos una marca en cada panel fuera de tolerancia
    if dev>T:

        l0 = rg.Line(pts[0],pts[2])
        l1 = rg.Line(pts[1],pts[3])

        lines.extend([l0,l1])

    #Creamos la polilinea de cada panel y calculamos su area
    pol = rg.PolylineCurve((pts[0],pts[1],pts[2],pts[3],pts[0]))

    areaObj = rg.AreaMassProperties.Compute(pol)
    area = areaObj.Area

    w = lines

    data.extend([pol,area])
    return data

panels = []
ptList = []
polList = []

#Initializing an empty data tree
ptTree = Tree[object]()

for i in range(uDiv+1):
    ptListTemp = []

    for j in range(vDiv+1):

        ptTemp = srf.Evaluate(i/uDiv,j/vDiv,2)[1]
        ptListTemp.append(ptTemp)

    ptList.append(ptListTemp)

# Contador necesario para crear una rama de datos por cada panel
counter = 0

for i in range(uDiv):
    for j in range(vDiv):

        pt0 = ptList[i][j]
        pt1 = ptList[i+1][j]
        pt2 = ptList[i][j+1]
```

```
        pt3 = ptList[i+1][j+1]

        # Adding points to datatree
        # The commented lines produce the flipped tree.
        #ptTree.Add(pt0,Path(0))
        #ptTree.Add(pt1,Path(1))
        #ptTree.Add(pt2,Path(2))
        #ptTree.Add(pt3,Path(3))

        ptTree.AddRange([pt0,pt1,pt3,pt2],Path(counter))
        counter += 1

        polList.append(rg.Polyline([pt0,pt1,pt3,pt2,pt0]))

# Loop para iterar en el arbol de datos creado en el loop anterior
for  i in range(ptTree.BranchCount):
    panels.append(planarize(ptTree.Branch(Path(i)),tol))


'''
## Salida de datos
'''
panel = polList
# Salida de datos de nuestra lista
fpanel = [p[0] for p in panels]
area = [p[1] for p in panels]
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Recursive_scaling

```
import rhinoscriptsyntax as rs
import Rhino.Geometry as rg
import math as m


def scaling(c):

    crvArea = rs.CurveArea(c)[0]
    crvCentroid = rs.CurveAreaCentroid(c)[0]
    #print crvCentroid

    # Comprobando casos
    if abs(target-crvArea)>tolerance:
        if target > crvArea:
            print "caso_0"
            print "Targe-Area= %f" %abs(target-crvArea)
            print "Tolerance= %f" %tolerance
            print "////////////////////////////////////////////"

            crvNew = rs.ScaleObject(c,crvCentroid,[1+step,1+step,0])
            c = scaling(crvNew)

        elif target < crvArea:
            print "caso_1"
            print "Targe-Area= %f" %(target-crvArea)
            print "Tolerance= %f" %tolerance
            print "////////////////////////////////////////////"

            crvNew = rs.ScaleObject(c,crvCentroid,[1-step,1-step,1-step])
            c = scaling(crvNew)

    #print "out"
    return c


a = scaling(crv)
print a
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

RW_polygons

```
import Rhino.Geometry as rg
import rhinoscriptsyntax as rs
import random as rd

def addPolygon(p,n):
    #inicializamos pts para diferenciarla de pts global
    pts = []
    for i in range(n):
        pts.append(rs.Polar(p,i*step,r))

    ptM = (pts[0]+pts[1])/2
    dist = rs.Distance(ptM,p)

    pts.append(pts[0]) # Appends the first point again to close the curve
    pol = rg.PolylineCurve(pts)
    return [pol,dist]

rd.seed(s)
pts = []
pol = []
step = 360.0/n

# Creates a loop to build points with polar coordinates
# at 2Pi/n increments.

angleList = [((step)*i)+step/2 for i in range(n)]
pts.append(pt)
pol.append(addPolygon(pt,n)[0])
ptNew = pt

for i in range(it):

    rdAngle = rd.choice(angleList)
    polygon = addPolygon(ptNew,n)
    pol.append(polygon[0])
    #print pol

    ptNew = rs.Polar(ptNew,rdAngle,polygon[1]*2)
    pts.append(ptNew)

print pol
pol = rg.Brep.CreatePlanarBreps(pol)
```