Stockage et génération de topographie artificielle de fond océanique

Projet "Sonar de l'infini"

Samy Avrillon - 24817

Lycée Lafayette

6 février 2020

Sommaire

Sommaire

Sommaire

Sommaire

Introduction

Le but : un format inéxistant

Introduction

Le but : un format inéxistant

- Un champ des hauteurs

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0.04	0.098	0.14	0.14	0.1	0.031	-0.022	0.015	0.087	0.15	0.16	0.1	-0.03	-0.14	-0.19
1	-0.2	-0.16	-0.11	-0.067	-0.061	-0.094	-0.16	-0.2	-0.12	-0.0018	0.11	0.17	0.15	0.025	-0.099	-0.17
2	-0.4	-0.37	-0.33	-0.29	-0.28	-0.3	-0.35	-0.37	-0.25	-0.073	0.096	0.2	0.21	0.1	-0.024	-0.1
3	-0.49	-0.48	-0.46	-0.43	-0.42	-0.43	-0.45	-0.45	-0.3	-0.089	0.11	0.23	0.25	0.15	0.039	-0.031
4	-0.45	-0.46	-0.45	-0.44	-0.43	-0.42	-0.42	-0.4	-0.24	-0.039	0.14	0.25	0.25	0.15	0.061	0.019
5	-0.29	-0.3	-0.32	-0.32	-0.31	-0.3	-0.28	-0.24	-0.1	0.064	0.19	0.25	0.21	0.1	0.033	0.029
6	-0.072	-0.099	-0.12	-0.14	-0.13	-0.11	-0.079	-0.038	0.068	0.18	0.23	0.22	0.14	0.014	-0.036	-0.0099
7	0.088	0.059	0.034	0.026	0.036	0.062	0.09	0.12	0.19	0.25	0.25	0.19	0.074	-0.053	-0.099	-0.072
8	0.19	0.16	0.15	0.16	0.19	0.22	0.25	0.25	0.28	0.29	0.26	0.19	0.084	-0.024	-0.071	-0.068
9	0.23	0.19	0.19	0.22	0.28	0.32	0.34	0.31	0.29	0.27	0.23	0.18	0.12	0.049	0.0073	-0.016
10	0.19	0.14	0.15	0.2	0.28	0.34	0.34	0.27	0.21	0.17	0.15	0.15	0.14	0.12	0.091	0.058
11	0.11	0.036	0.052	0.12	0.21	0.26	0.25	0.15	0.052	0.0041	0.012	0.062	0.12	0.14	0.14	0.12
12	0.022	-0.064	-0.051	0.018	0.1	0.15	0.12	-0.0062	-0.12	-0.17	-0.15	-0.059	0.048	0.1	0.13	0.15
13	0.0051	-0.089	-0.086	-0.029	0.041	0.079	0.04	-0.098	-0.23	-0.3	-0.28	-0.2	-0.087	-0.027	0.032	0.1
14	0.056	-0.041	-0.048	-0.0013	0.059	0.091	0.052	-0.087	-0.24	-0.35	-0.39	-0.35	-0.28	-0.24	-0.15	-0.037
15	0.11	0.016	0.0042	0.046	0.1	0.14	0.11	-0.025	-0.2	-0.35	-0.45	-0.47	-0.45	-0.42	-0.33	-0.18

FIGURE – Exemple de champ de hauteur

Introduction

Le but : un format inéxistant

- Un champ des hauteurs
- Une discrétisation 3D

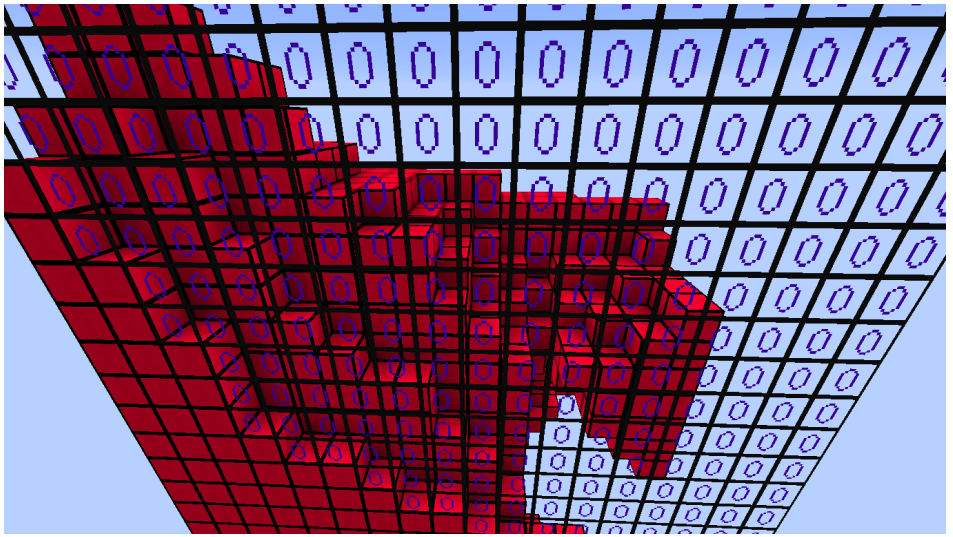

FIGURE – Exemple de discrétisation 3d de l'espace

Introduction

Le but : un format inéxistant

- Un champ des hauteurs
- Une discrétisation 3D
- Quelques formats privés

Introduction

Débouchés et utilisations

- Jeu vidéo on monde ouvert

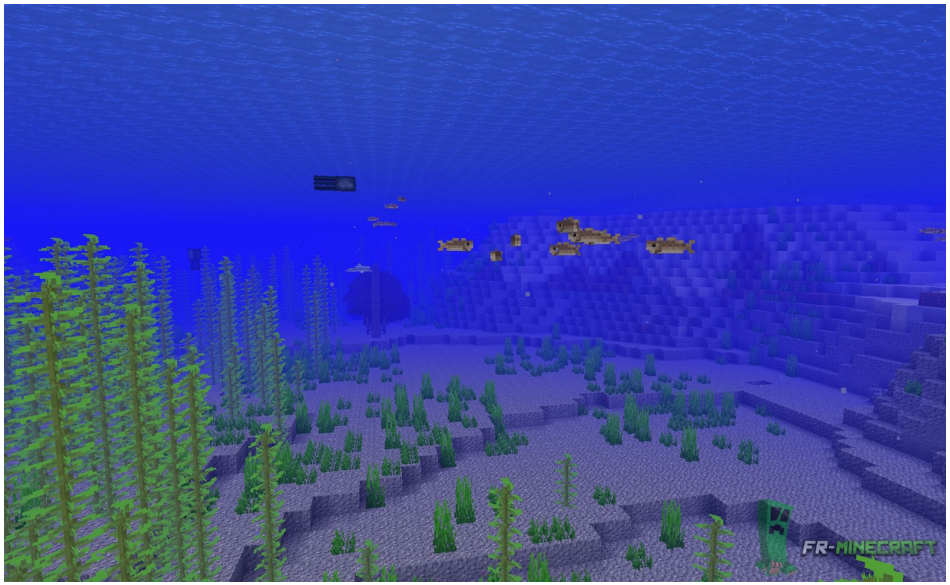

FIGURE – Capture du jeu vidéo Minecraft

Introduction

Débouchés et utilisations

- Jeu vidéo on monde ouvert
- Graphisme, cinéma

FIGURE – Extrait des films » Le monde de némo « et » Le Voyage extraordinaire de Samy «

Introduction

Débouchés et utilisations

- Jeu vidéo on monde ouvert
- Graphisme, cinéma
- Simulation physique ou de rover

Le format TMF

Contraintes

Le format TMF

Contraintes

- Liberté totale

Le format TMF

Contraintes

- Liberté totale
- Complexité spatiale

Le format TMF

Contraintes

- Liberté totale
- Complexité spatiale
- Référencabilité

Le format TMF

Modélisation

Le format TMF

Modélisation

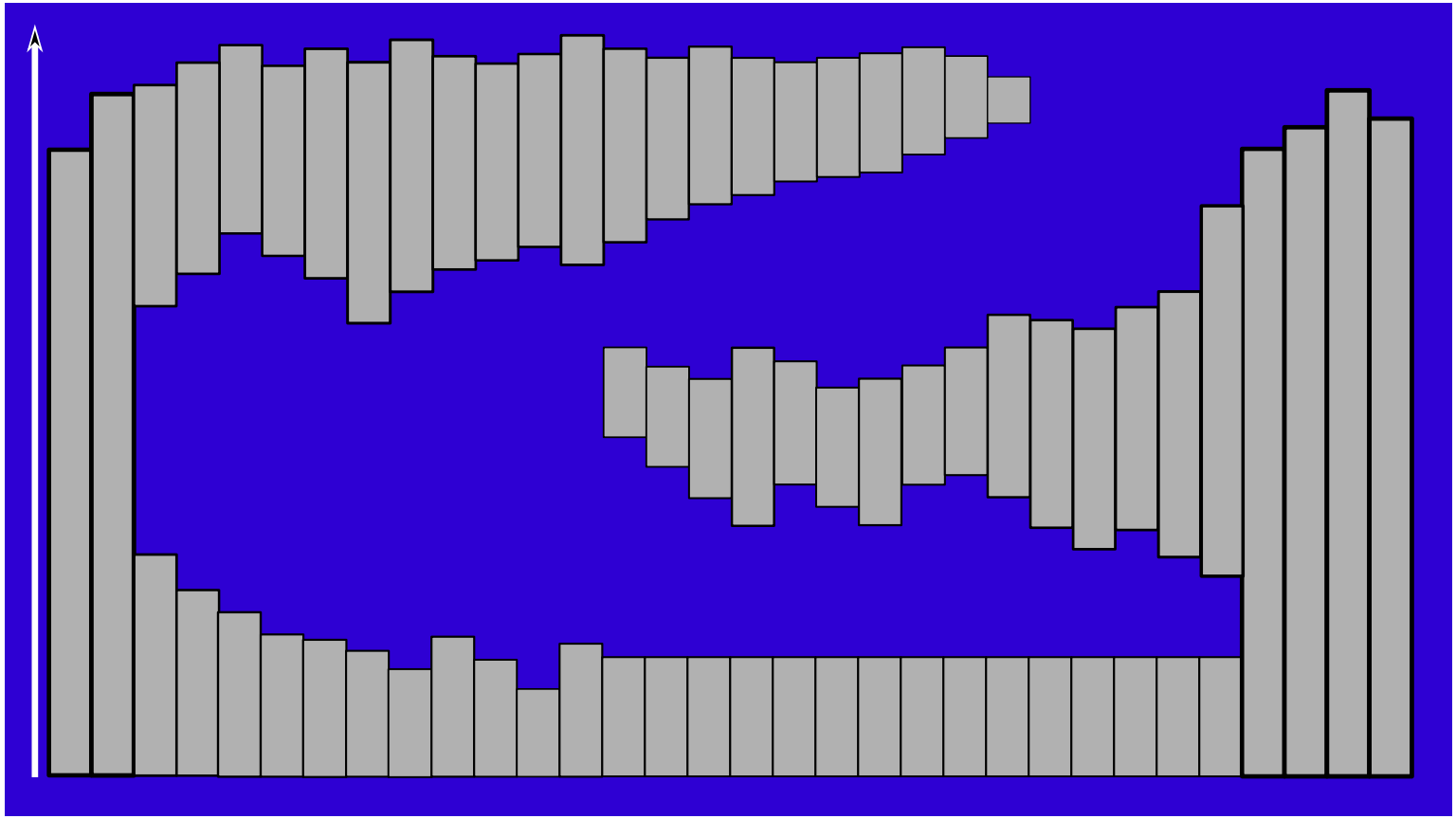

FIGURE – Représentation 2D du stockage des colonnes

Le format TMF

Réalité du stoquage

Ici format de fichier,

Module Objection

Minecraft

Module Objection

Minecraft

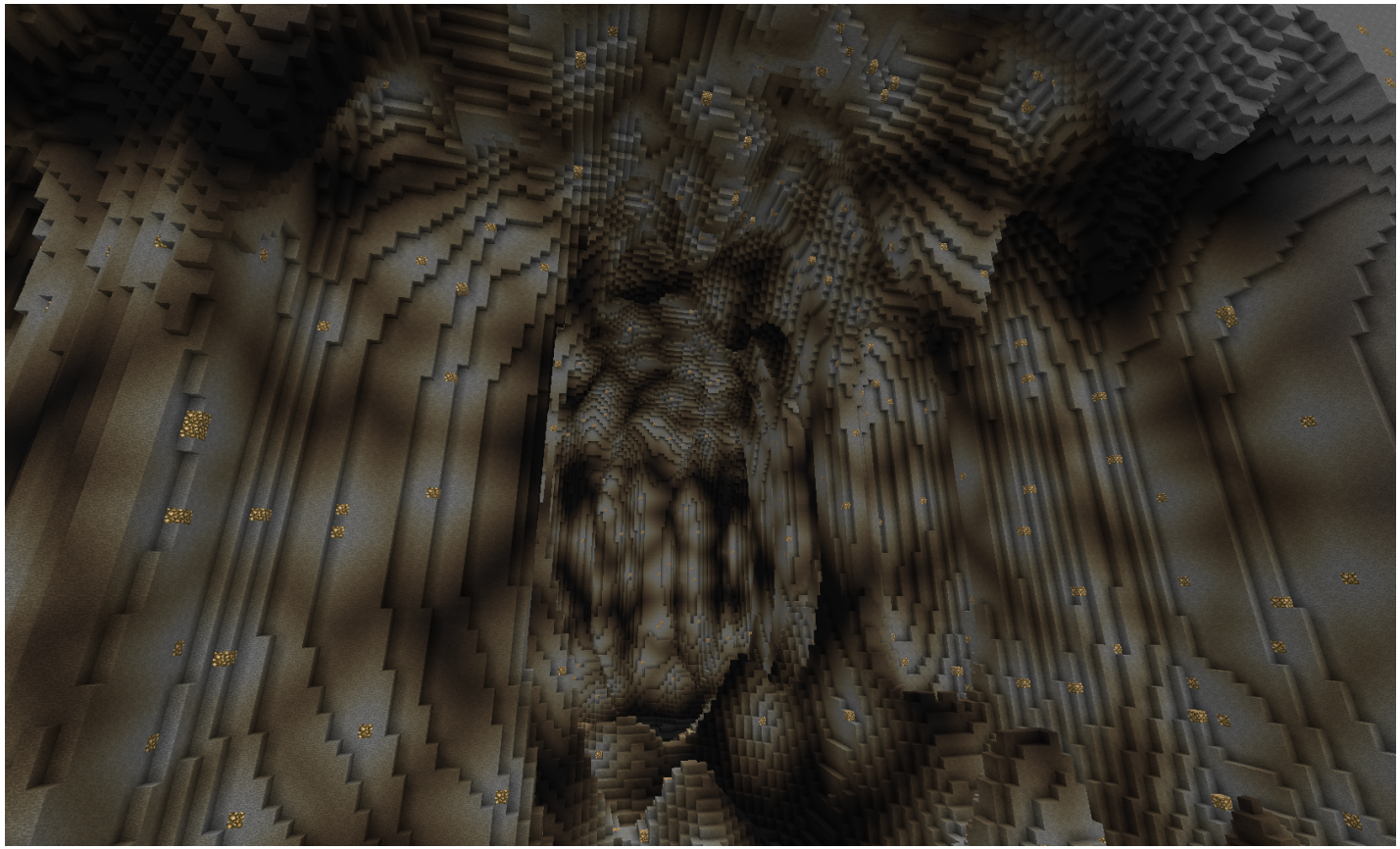

FIGURE – Usage de Minecraft comme moteur graphique

Module Objection

Rectangle

Module Objection

Rectangle

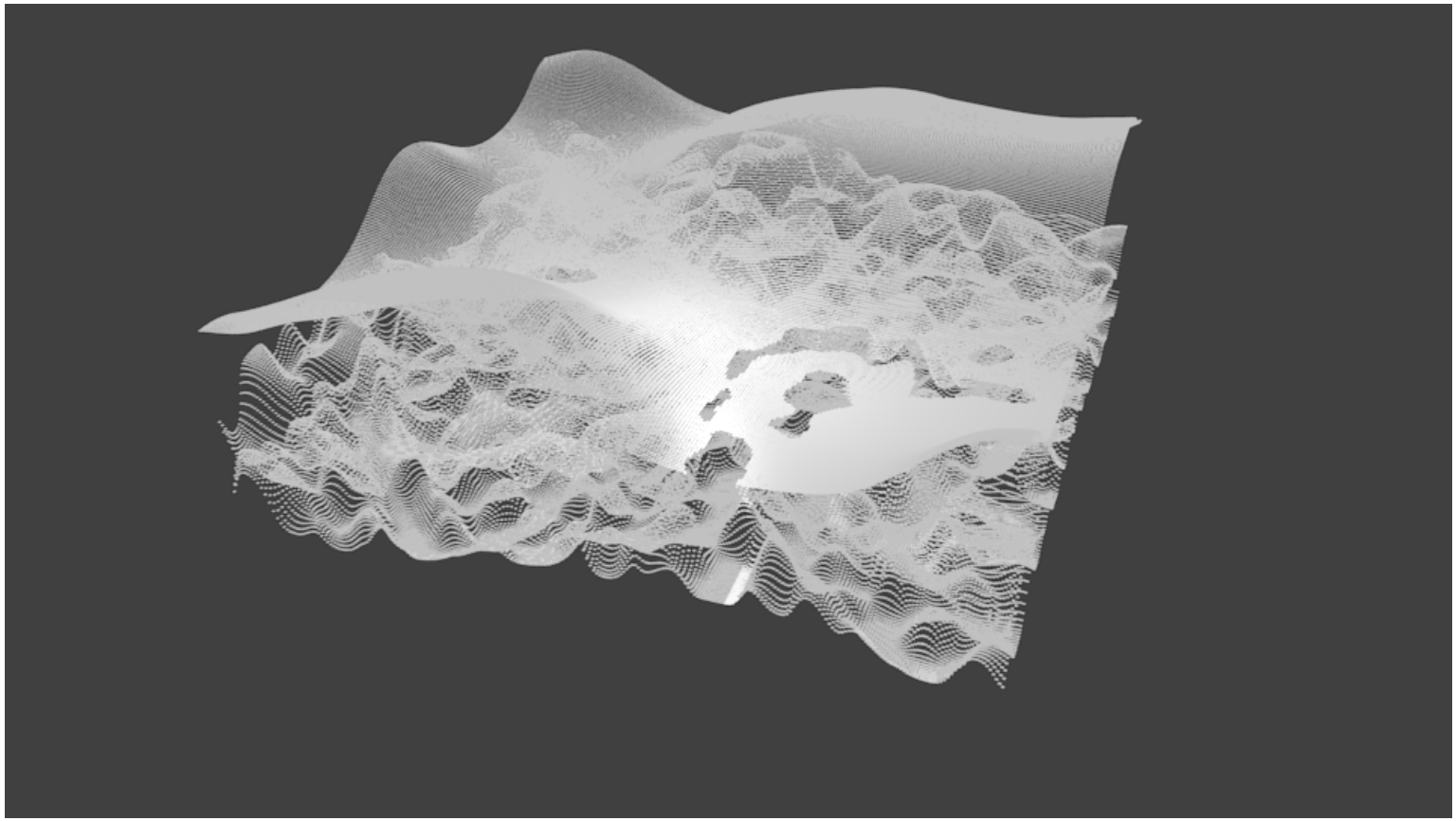

FIGURE – Usage de rectangles

Module Objection

Colonnes

Module Objection

Colonnes

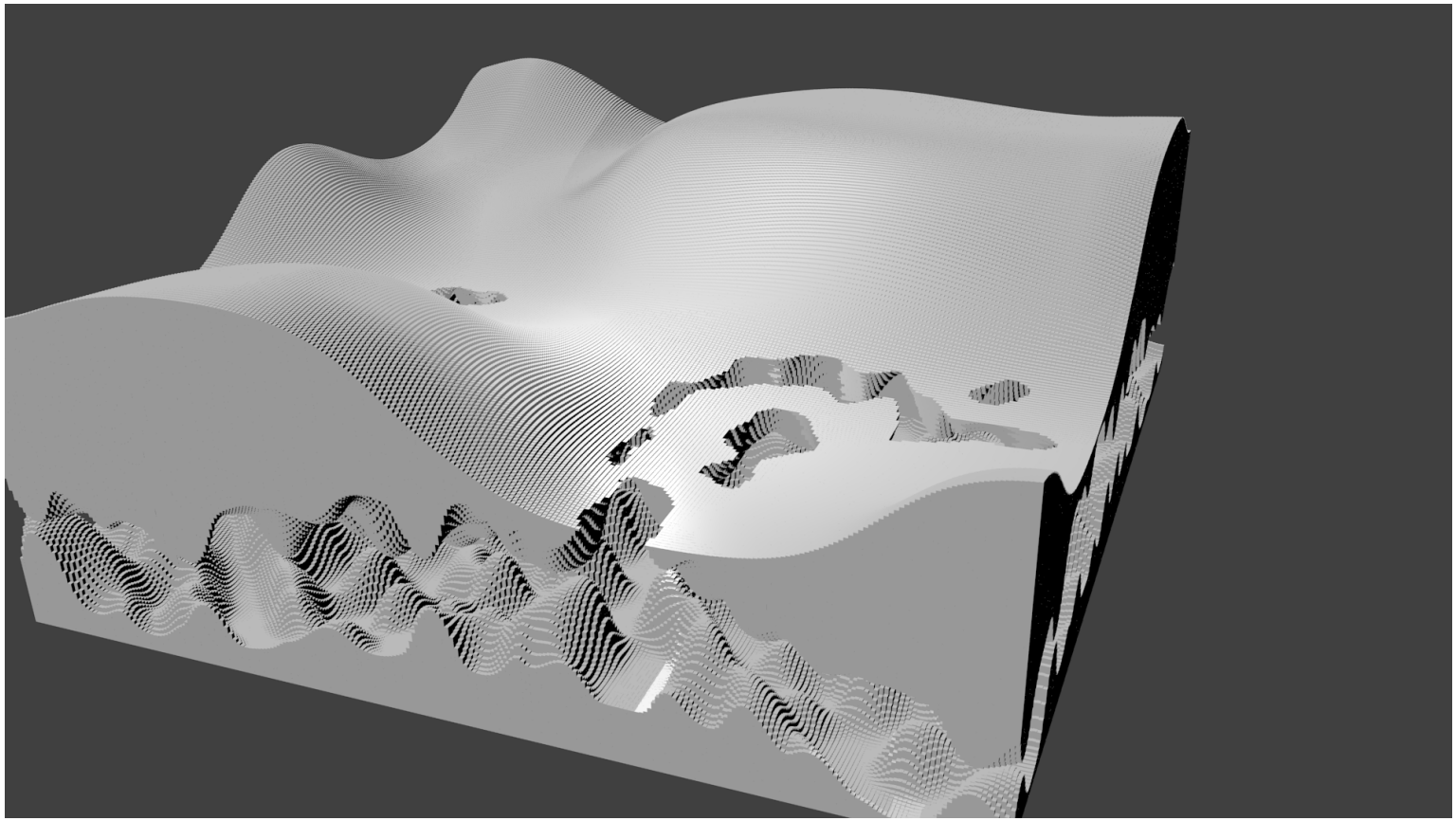

FIGURE – Usage de parallélépipèdes

Module Objection

Triangles

Module Objection

Triangles

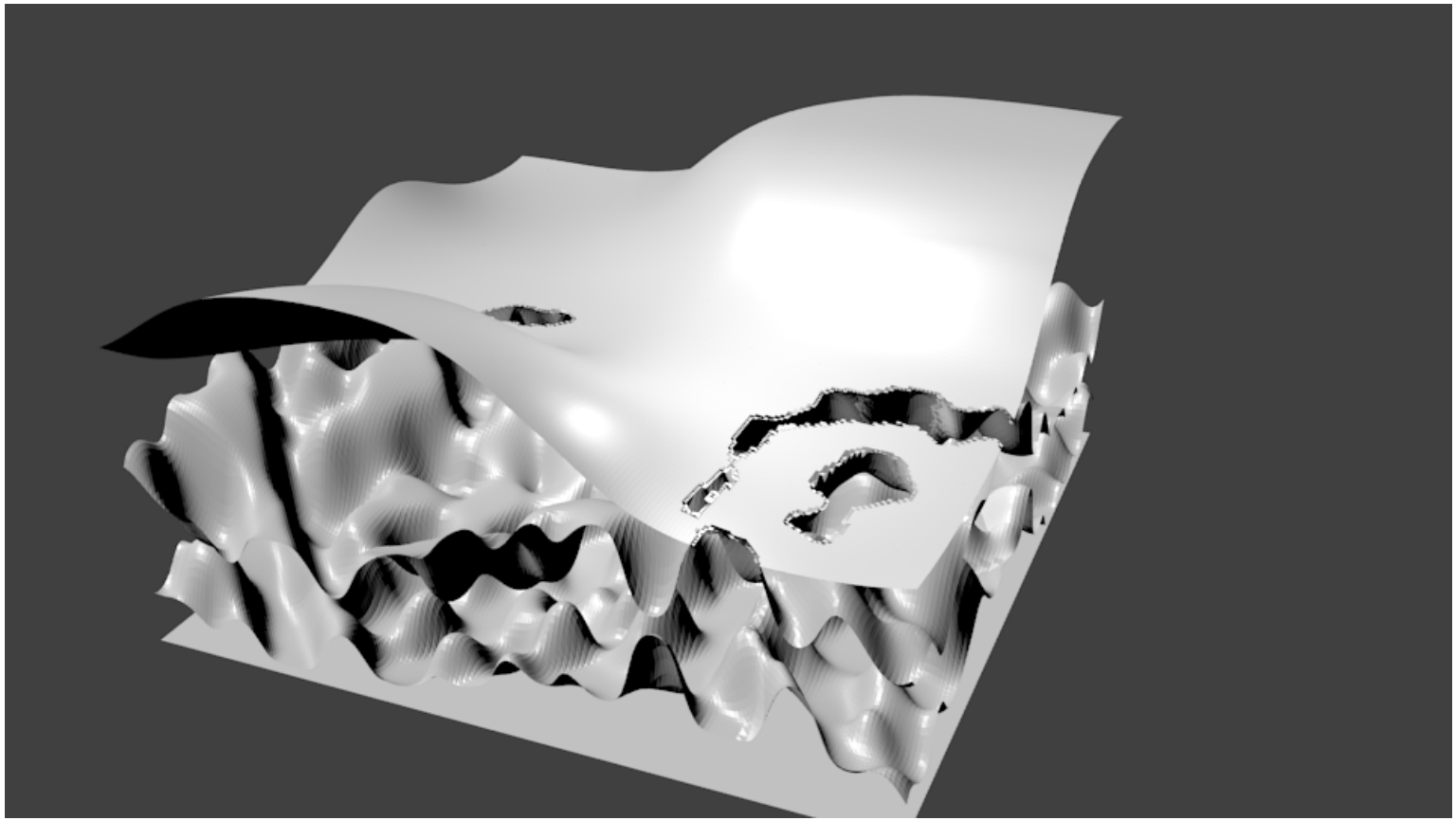

FIGURE – Usage de triangles

Génération

Contraintes

Génération

Contraintes

- Infinité

Génération

Contraintes

- Infinité
- Répétabilité

Génération

Contraintes

- Infinité
- Répétabilité
- Modulabilité

Génération

Noisette

Génération

Noisette

- `getChunk(self,x,y,n)`

Génération

Noisette

- `getChunk(self,x,y,n)`
- `__add__(self,other)`

Génération

Noisette

- `getChunk(self,x,y,n)`
- `__add__(self,other)`
- `__rmul__(self,other)`

Génération

Noisette

- `getChunk(self,x,y,n)`
- `__add__(self,other)`
- `__rmul__(self,other)`
- `__sub__(self,other)`

Génération

Noisette

Génération

Noisette

- Bruit sur le cercle trigonométrique

Génération

Noisette

- Bruit sur le cercle trigonométrique
- Bruit avec des interpolations linéaires

Génération

Noisette

- Bruit sur le cercle trigonométrique
- Bruit avec des interpolations linéaires
- Bruit avec des droites

Génération

Noisette

- Bruit sur le cercle trigonométrique
- Bruit avec des interpolations linéaires
- Bruit avec des droites
- Bruit de Perlin (2d)

Génération

Noisette

- Bruit sur le cercle trigonométrique
- Bruit avec des interpolations linéaires
- Bruit avec des droites
- Bruit de Perlin (2d)
- Bruit fractal

Génération

Cartman

Génération

Cartman

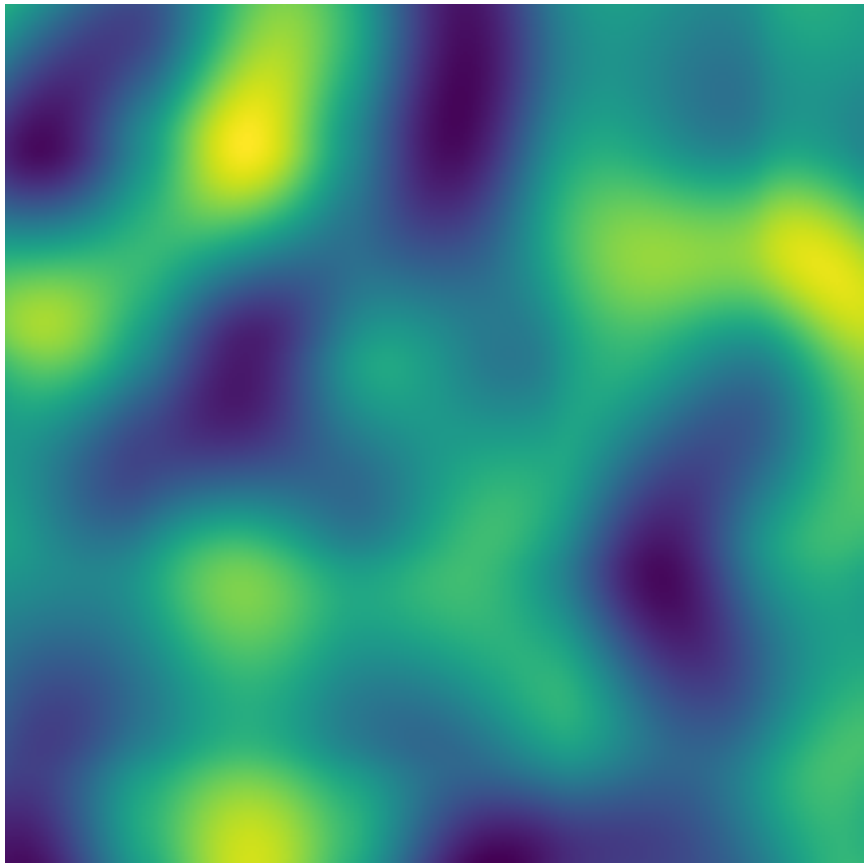

FIGURE – Heightmap créée par un bruit de perlin

Génération

Cartman

Génération

Cartman

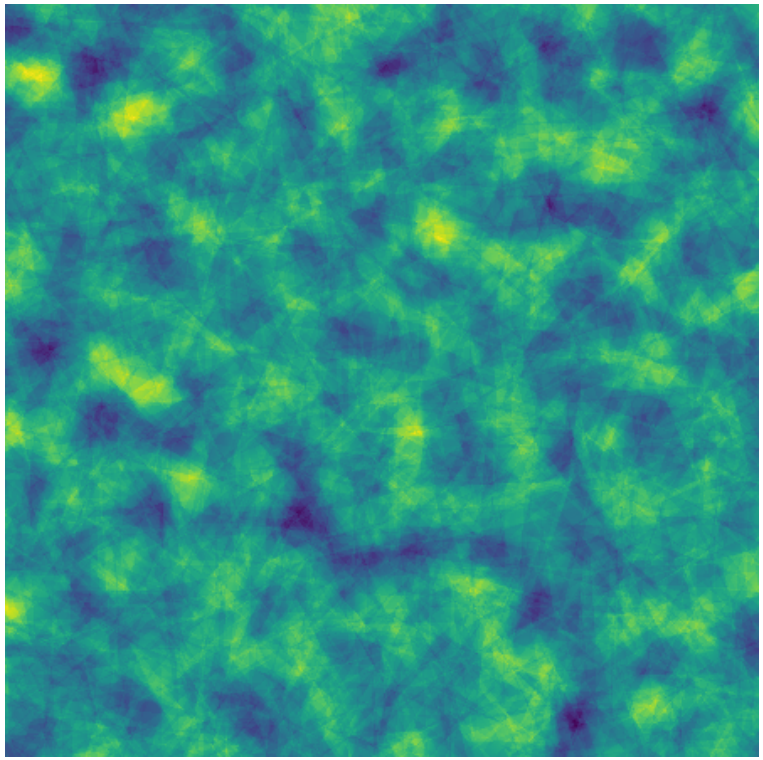

FIGURE – Bruit fractal avec peu de droites

Génération

Cartman

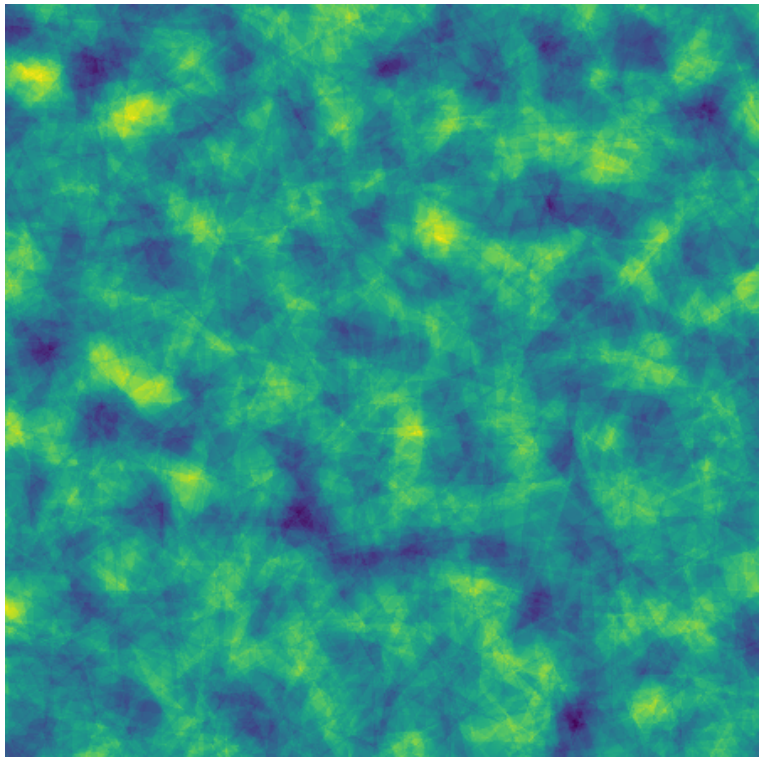

FIGURE – Bruit fractal avec peu de droites

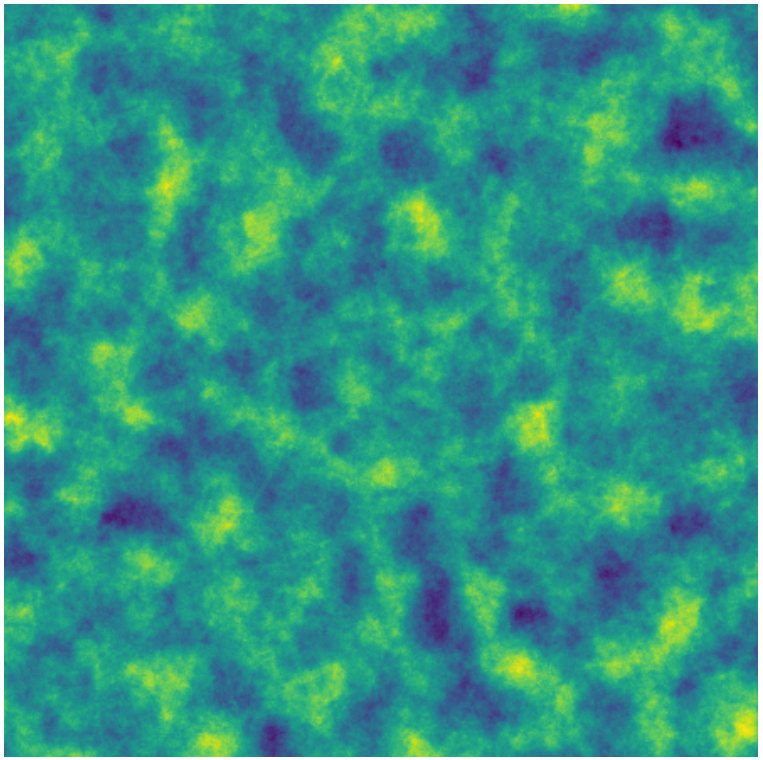

FIGURE – Bruit fractal avec plus de droites

Génération

Cartman

Bruit Caverne à présenter

Sommaire

TODO sommaire de l'appendice

Python

data.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Created on Wed Aug 28 17:39:43 2019

Module contanant les classes générales structurant les données,
↪  ainsi que les

@author: mysaa
"""
import numpy as np

def appendeur(l1,l2):
    """
        Effectue l1=l1+l2 de manière opti
    """
    for l in l2:
```

```
        l1.append(l)

class WorldChunk():

    def getSize(self):
        raise NotImplementedError("Vous avez fait un monde qui
        ↪  n'implemente pas cette méthode. Vous êtes bizzare
        ↪  vous savez ? Un monde sans taille !!!")


    def getColumn(self,x,y):
        raise NotImplementedError("Vous avez fait un monde qui
        ↪  n'implemente pas cette méthode. Vous êtes bizzare
        ↪  vous savez ?")


    def asList(self):
        return [ [self.getColumn(x,y) for y in
        ↪  range(self.size[1])]  for x in range(self.size[0])
        ↪  ]
```

Python

data.py

```
    def getIndexed(self,fullCoords=False,addZero=False):
        points = []
        pointIndexes = [0]
        for pos in np.ndindex(self.getSize()):
            col = self.getColumn(pos[0],pos[1])
            if(addZero): col = np.insert(col,0,0)
            if(fullCoords): col = [(pos[0],pos[1],c) for c in
            ↪  col]
            appendeur(points,col)
            pointIndexes.append(pointIndexes[-1]+len(col))
        return points,pointIndexes


class CollageWorldChunk(WorldChunk):

    def __init__(self,chunk,xp,yp,xy):
```

Python

data.py

```
        self.orgChunk,self.xp,self.yp,self.xy = chunk,xp,yp,xy
        self.orgSize = chunk.getSize()

    def getSize(self) : return
    ↪   (self.orgChunk.size[0]+1,self.orgChunk.size[1]+1)

    def getColumn(self,x,y):
        xout,yout = x>=self.orgSize[0],y>=self.orgSize[1]
        if(xout and yout):
            return
            ↪   self.xy.getColumn(x-self.orgSize[0],y-self.orgSize
        if(xout):
            return self.xp.getColumn(x-self.orgSize[0],y)
        if(yout):
            return self.yp.getColumn(x,y-self.orgSize[1])

        return self.orgChunk.getColumn(x,y)
```

Python

data.py

```
class ArrayedWorldChunk(WorldChunk):

    def fromList(liste):
        size = len(liste),len(liste[0])
        indexes=np.empty(size[0]*size[1]+1,dtype=np.uint32)
        index = 0
        data = []
        for y in range(size[1]):
            for x in range(size[0]):
                indexes[x+size[0]*y]=index
                data += liste[x][y]
                index += len(liste[x][y])
        indexes[size[0]*size[1]] = index
        data = np.array(data,dtype=np.float)
        return ArrayedWorldChunk(size,indexes,data)

    def __init__(self,size,indexes,data):
        self.size = size
```

Python

data.py

```
        self.indexes=indexes
        self.data=data


    def getColumn(self,x,y):
        i0,i1 =
        ↪  self.indexes[x+self.size[0]*y],self.indexes[x+self.siz
        return self.data[i0:i1]

    def getSize(self) : return self.size
```

```python
class Noise:

    def getChunk(self,x,y,n):
        """
            Cette fonction renvoie un array numpy de taille
↪   rx*ry correspondant au chunk x y avec le seed donné.
            Cette fonction doit être déterministe (si les
↪   attributs de l'objets ne sont pas changés bien sur)
        """
        raise NotImplementedError("Vous avez fait un bruit qui
        ↪   n'implemente pas cette méthode. Vous êtes bizzare
        ↪   vous savez ?")



    def __add__(self,other):
```

Python

data.py

```
        def addedChunk(self,x,y,n):
            return self.noise1.getChunk(x,y,n) +
            ↪  self.noise2.getChunk(x,y,n)
        noise = Noise()
        noise.noise1 = self
        noise.noise2 = other
        noise.getChunk = addedChunk
        return noise

    def __iadd__(self,other):

        return self.__add__(other)

    def __rmul__(self,other):

        if type(other) in ['float','int']:
            def mulChunk(self,x,y,n):
                return self.prop*self.noise1.getChunk(x,y,n)
```

```
            noise = Noise()
            noise.noise1 = self
            noise.prop = other
            noise.getChunk = mulChunk
        else:
            def mulChunk(self,x,y,n):
                return self.noise1.getChunk(x,y,n) *
                ↪  self.noise2.getChunk(x,y,n)
            noise = Noise()
            noise.noise1 = self
            noise.noise2 = other
            noise.getChunk = mulChunk
        return noise

    def __sub__(self,other):

        def subChunk(self,x,y,n):
            return self.noise1.getChunk(x,y,n) -
            ↪  self.noise2.getChunk(x,y,n)
```

Python

data.py

```
noise = Noise()
noise.noise1 = self
noise.noise2 = other
noise.getChunk = subChunk
return noise
```

Python

objection.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 22 19:46:36 2019

@author: mysaa
"""
import numpy as np
from data import CollageWorldChunk
from perlin import CavernedNoise2,TestNoise



def getTriangles(x0,y0,chunk,xp,yp,xy):


    nx,ny = chunk.size
    newChunk = CollageWorldChunk(chunk,xp,yp,xy)
```

Python

objection.py

```
#     pointIndexes = np.zeros((nx+1,ny+1),dtype=np.uint32)
#     pointLengthes = np.zeros((nx+1,ny+1),dtype=np.uint32)
#     points = []
#
#
#     pos = 0
#     for j in range(ny):
#         for i in range(nx):
#             carotte = [(x0+i/nx,y0+j/ny,z) for z in
↪   sorted([0.]+list(chunk.getColumn(i,j)))]
#             points += carotte
#             pointLengthes[i,j] = len(carotte)
#             pointIndexes[i,j] = pos
#             pos+=len(carotte)
#         carotte = [(x0+1,y0+j/ny,z) for z in
↪   sorted([0.]+list(xp.getColumn(0,j)))]
#         points += carotte
```

Python

objection.py

```
#         pointLengthes[nx,j]= len(carotte)
#         pointIndexes[nx,j] = pos
#         pos+=len(carotte)
#     for i in range(nx):
#         carotte = [(x0+i/nx,y0+1,z) for z in
↪   sorted([0.]+list(yp.getColumn(i,0)))]
#         points += carotte
#         pointLengthes[i,ny] = len(carotte)
#         pointIndexes[i,ny] = pos
#         pos+=len(carotte)
#     carotte = [(x0+1,y0+1,z) for z in
↪   sorted([0.]+list(xy.getColumn(0,0)))]
#     points += carotte
#     pointLengthes[nx,ny] = len(carotte)
#     pointIndexes[nx,ny] = pos
```

Python

objection.py

```
points,pointIndexes =
↪  newChunk.getIndexed(fullCoords=True,addZero=True)

points=[(p[0]/nx+x0,p[1]/ny+y0,p[2]) for p in points]
pointLengthes =
↪  np.reshape([pointIndexes[i+1]-pointIndexes[i] for i in
↪  range(len(pointIndexes)-1)],(nx+1,ny+1))
pointIndexes = np.reshape(pointIndexes[:-1],(nx+1,ny+1))
print(points,pointIndexes,pointLengthes)



triangles = []
```

Python

objection.py

```
for x in range(nx*2):
    for y in range(ny):
        # On récupère les coordonées entières du triangle
        ↪  indicé (x,y)
        if(x%2==0):
            col0=(x//2  ,y  )
            col1=(x//2+1,y  )
            col2=(x//2  ,y+1)
        else:
            col0=(x//2+1,y+1)
            col1=(x//2+1,y  )
            col2=(x//2  ,y+1)

        # On récupère la liste des points dans la colonne
        colonne0 =
        ↪  points[pointIndexes[col0[0],col0[1]]:pointIndexes
        colonne1 =
        ↪  points[pointIndexes[col1[0],col1[1]]:pointIndexes
```

Python

objection.py

```
colonne2 =
↪  points[pointIndexes[col2[0],col2[1]]:pointIndexes
#print("colonne:",colonne1)
# st contient des triplets (numéro de colonne,index
↪  interne dans la colonne,coordonée z)
st  = [(0,i,colonne0[i][2]) for i in
↪  range(len(colonne0))]
st += [(1,i,colonne1[i][2]) for i in
↪  range(len(colonne1))]
st += [(2,i,colonne2[i][2]) for i in
↪  range(len(colonne2))]


# On y trie par coordonée z
st = sorted(st,key=lambda c:c[2])

#Liste des coordonées des colonnes, pour pouvoir
↪  sélectionner les coordonées selon l'index de la
↪  colonne
```

Python

objection.py

```
cols = [col0,col1,col2]
# Là, tout est bon à peu près

i=0
try:
    while i<len(st):
        v1 = st[i] ; i+=1
        v2 = st[i] ; i+=1
        if v1[0]==v2[0]: continue #Cas où une
        ↪  colonne apparait puis disparaît
        v3 = st[i] ; i+=1
        v4=v3 # Pour pouvoir observer le changement
        ↪  de colonne dans la chaine aller/retour
        ↪  2/1
        while v1[0]+v2[0]+v3[0]!=3:
            # On est dans la chaine aller-retour
            ↪  2/1
```

```
            if v4[0]!=v3[0]:
                # Les deux colonnes ont disparu
                # On peut créer le rectangle
                ↪  v1,v2,v3,v4
                triangle1 =
                ↪  [pointIndexes[cols[v[0]]] +
                ↪  v[1] for v in (v1,v2,v3)]
                triangle2 =
                ↪  [pointIndexes[cols[v[0]]] +
                ↪  v[1] for v in (v1 if
                ↪  v1[0]==v3[0] else v2,v3,v4)]
                triangles.append(triangle1)
                triangles.append(triangle2)
                break
            v4=v3
            v3 = st[i] ; i+=1
        else:
            # On est dans le cas ou v1,v2,v3
            ↪  correspondent à trois colonnes
```

Python

objection.py

```
# différentes (le cas (1,1,1) ayant
↪  déjà été filtré par la première
↪  condition)

# On ajoute le triangle en dessous
triangle = [pointIndexes[cols[v[0]]] +
↪  v[1] for v in (v1,v2,v3)]
triangles.append(triangle)

# On "attends" jusqu'à ce que les trois
↪  colonnes soient à nouveau vides
vs = [None,None,None]
while None in vs:
    v = st[i] ; i+=1
    vs[v[0]] = v if vs[v[0]]==None else
    ↪  None

# Avec cette mthode, on déssine le
↪  triangle avec les derniers points
↪  étant apparus (les plus hauts)
```

```
                triangle = [pointIndexes[cols[v[0]]] +
                ↪  v[1] for v in vs]
                triangles.append(triangle)

            continue


        except ValueError:
            # Une fin de liste a été atteinte, la colonne
            ↪  n'a pas été refermée: lance un warn
            print("Attention ! Une colonne n'avait pas de
            ↪  toit. veuillez vérifier que vos colonnes
            ↪  aient un nombre impair de coordonées, merci
            ↪  !")

    return points,triangles
```

Python

objection.py

```
                ##################################
#                while i<len(st):
#                    #Plein
#                    v0 = st[i]
#                    i+=1
#                    v1 = st[i]
#                    i+=1
#                    if(v0[0] == v1[0]): # S'est la même colonne
↪   qui est apparu puis disparu
#
#                        print("Tribord")
#                        colz = cols[v0[0]]
#                        # Triangle sur les bords
#                        # Demis-points
#                        halfZ = (v0[2]+v1[2])/2
#                        #Les deux autres colonnes sont :
#                        cola = cols[(v0[0]+1)%3]
#                        colb = cols[(v0[0]+2)%3]
```

Python

objection.py

```
#
#
↪   zi1,zi2=pointIndexes[colz]+v0[1],pointIndexes[colz]+v1[1]
#                       print('OoOOoO',points[zi1],points[zi2])
#
##                       points.append(
↪   (x0+(cola[0])/nx,y0+(cola[1])/ny,halfZ) )
##                       points.append(
↪   (x0+(colb[0])/nx,y0+(colb[1])/ny,halfZ) )
##
##
↪   triangles.append([zi1,len(points)-1,len(points)-2])
##
↪   triangles.append([zi2,len(points)-1,len(points)-2])
#                       print("Tribord-fin")
#                       #
↪   print(points[-1],points[-2],points[zi1])
#
↪   #print(cola,colb,points[triangles[-1][0]-1],points[triangles[-
↪
```

Python

objection.py

```
#
#                 else: # Deux colonnes différentes ont apparus
↪  successivement
#                     # vs stoque les états des colonnes
#                     # vs[i] est l'état de la ième colonne, le
↪  point de st, dernier à apparaître si
#                     # cette colonne est présente, None sinon
#                     vs=[None,None,None]
#                     vs[v0[0]] = v0
#                     vs[v1[0]] = v1
#                     while None in vs and vs !=
↪  [None,None,None]:# Tant qu'il y a une abscente ou une
↪  présente
#                         v2 = st[i]
#                         i+=1
#                         vs[v2[0]] = v2 if vs[v2[0]]==None else
↪  None
#
```

Python

objection.py

```
#                       if not None in vs:
#                           # Une face complète a été créée
#                           # Triangle complet
#                           # Face dessous (apparition de la
↪   colonne)
#                           triangle = [pointIndexes[cols[i]] +
↪   vs[i][1] for i in range(3)]
#                           triangles.append(triangle)
#                           #print("#",[points[triangle[i]] for i
↪   in range(0,3)])
#
#
#                           # On inverse le sens de vs, et stoque
↪   les premiers points à apparaître
#                           vs = [None,None,None]
#                           while None in vs and i<len(st):
#                               v2 = st[i]
#                               i+=1
```

Python

objection.py

```
#                                   vs[v2[0]] = v2 if vs[v2[0]]==None
↪   else None
#
#                             #Face dessus
#                             if not None in vs:
#                                 triangle = [pointIndexes[cols[i]]
↪   + vs[i][1] for i in range(3)]
#                                 triangles.append(triangle)
#                                 #print("0",[points[triangle[i]]
↪   for i in range(3)])
#                         else:
#                             # Il faut placer un carré
#                 ##################################
#     print(points)
#     return points,triangles
```

Python

objection.py

```
def getRectangles(x0,y0,chunk):
    nx,ny = chunk.size

    triangles = []
    points = []

    for x in range(nx) :
        for y in range(ny):
            for z in [0]+chunk.getColumn(x,y):
                points.append([x/nx+x0,y/ny+y0,z])
                points.append([x/nx+x0+1/nx,y/ny+y0,z])
                points.append([x/nx+x0,y/ny+y0+1/ny,z])
                points.append([x/nx+x0+1/nx,y/ny+y0+1/ny,z])

                ↪  triangles.append([len(points)-4,len(points)-3,

                ↪  triangles.append([len(points)-3,len(points)-2,
```

Python

objection.py

```
    return points,triangles

def getRectCols(x0,y0,chunk):
    nx,ny = chunk.size

    triangles = []
    points = []
    e=0.3

    for x in range(nx) :
        for y in range(ny):
            for z in [0]+chunk.getColumn(x,y):
                points.append([x/nx+x0-e,y/ny+y0-e,z])
                points.append([x/nx+x0+e,y/ny+y0-e,z])
                points.append([x/nx+x0-e,y/ny+y0+e,z])
                points.append([x/nx+x0+e,y/ny+y0+e/ny,z])

                ↪  triangles.append([len(points)-4,len(points)-3,
```

Python

objection.py

```
                ↪  triangles.append([len(points)-3,len(points)-2,

    return points,triangles


def getFilled(x0,y0,chunk):
    nx,ny = chunk.size

    triangles = []
    points = []

    for x in range(nx) :
        for y in range(ny):
            boule = True
            lz = 0
            for z in sorted(chunk.getColumn(x,y)):
```

```
if boule:
    points.append([x/nx+x0      ,y/ny+y0      ,z
    ↪  ])
    points.append([x/nx+x0+1/nx,y/ny+y0      ,z
    ↪  ])
    points.append([x/nx+x0      ,y/ny+y0+1/ny,z
    ↪  ])
    points.append([x/nx+x0+1/nx,y/ny+y0+1/ny,z
    ↪  ])
    points.append([x/nx+x0      ,y/ny+y0
    ↪  ,lz])
    points.append([x/nx+x0+1/nx,y/ny+y0
    ↪  ,lz])
    points.append([x/nx+x0
    ↪  ,y/ny+y0+1/ny,lz])

    ↪  points.append([x/nx+x0+1/nx,y/ny+y0+1/ny,l
    l = len(points)
```

```
                triangles.append([l-4,l-2,l-1])
                triangles.append([l-4,l-3,l-1])
                triangles.append([l-4,l-2,l-6])
                triangles.append([l-4,l-8,l-6])
                triangles.append([l-4,l-3,l-7])
                triangles.append([l-4,l-8,l-7])
                triangles.append([l-5,l-1,l-2])
                triangles.append([l-5,l-6,l-2])
                triangles.append([l-5,l-6,l-8])
                triangles.append([l-5,l-7,l-8])
                triangles.append([l-5,l-7,l-3])
                triangles.append([l-5,l-1,l-3])

            boule = not boule
            lz = z

return points,triangles
```

Python

objection.py

```
def printObject(file,name,delta,points,triangles):
    file.write("o "+name+"\n\n")

    sf = lambda x : "%.6f" % float(x)
    si = lambda x : str(int(x+1)+delta)

    for p in points:
        file.write("v "+sf(p[0])+" "+sf(p[1])+"
        ↪  "+sf(np.array(p[2])/20.)+"\n")

    file.write("\n")

    for t in triangles:
        file.write("f "+" ".join([si(tp) for tp in t])+"\n")


def
↪  writeMap(filePath,noise,x0,y0,sx,sy,cx,cy,objType='triangle',l
```

Python

objection.py

```
log("Génération de la carte")
generated = {}
formatter='\rÉcriture du chunk {};{}
↪  '+" "*(sx//10+sy//10)
for i in range(x0,sx+x0+(1 if objType=='triangle' else 0)):
    for j in range(y0,sy+y0+(1 if objType=='triangle' else
    ↪  0)):
        log(formatter.format(i,j), end='\r')
        generated[(i,j)] = noise.getChunk(i,j,(cx,cy))
log("Génération des objets")
file = open(filePath,"w+")
file.write("g carte\n")
delta=0
for i in range(x0,sx+x0):
    for j in range(y0,sy+y0):
        log(formatter.format(i,j), end='\r')
        if objType=='triangle':
            points,triangles =
            ↪  getTriangles(i,j,generated[(i,j)],generated[(i
```

Python

objection.py

```
            elif objType=='rectangle':
                points,triangles =
                ↪  getRectangles(i,j,generated[(i,j)])
            elif objType=='filled':
                points,triangles =
                ↪  getFilled(i,j,generated[(i,j)])
            elif objType=='rectcols':
                points,triangles =
                ↪  getRectCols(i,j,generated[(i,j)])
            else:
                raise ValueError("Je en connais pas le type
                ↪  d'objet "+objType)

            ↪  printObject(file,"chunk_"+objType+"_"+str(i)+"-"+s
            file.write("\n\n")
            delta+=len(points)
log("\nTerminé !                "+" "*(sx//10+sy//10))
file.close()
```

Python

objection.py

```
noise = CavernedNoise2(93152)

size = 12
taille=16

#writeMap("gros.obj",noise,-size//2+1,-size//2+1,size,size,taille,
#writeMap("carte.obj",noise,2,1,1,1,4,4,'triangle')
#writeMap("carteTri.obj",noise,-8,-8,16,16,16,16,'triangle')
writeMap("carteRec.obj",noise,-8,-8,16,16,16,16,'rectangle')
writeMap("carteFil.obj",noise,-8,-8,16,16,16,16,'filled')
writeMap("carteRCo.obj",noise,-8,-8,16,16,16,16,'rectcols')
```

Python

objection.py

```
#xy0=-taille*size
#for x in range(2*taille):
#    for y in range(2*taille):
#
#
↪   writeMap("render2/carte"+str(x)+","+str(y)+".obj",noise,xy0-si
```

Python

perlin.py

```
# -*- coding: utf-8 -*-

"""
Created on Fri Mar  8 15:13:12 2019

Ce module contient de nombreuses implémentations de Noise,
↪  permetant en les assemblant de créer des mondes

@author: mysaa
"""

from data import ArrayedWorldChunk,Noise
import random as r
import numpy as np
import sys
import matplotlib.pyplot as pp
import matplotlib.image as img
from mpl_toolkits.mplot3d import Axes3D
```

Python

perlin.py

```python
from math import sqrt,floor,ceil,pi


##### Paramètres #####
G = 4.5
F = 5.2


class RandNoise(Noise):
    """
                Ce bruit renvoie une carte de vecteurs
↪   complexes (2d) du cercle trigonométrique (de module 1)
        """
    seed = None
    f = None

    def __init__(self,seed,f=lambda r : r.random()):
        self.seed = seed
```

Python

perlin.py

```
        self.f = f

    def getRandomly(self,xg,yg):
        """
            Cette fonction renvoie un nombre complexe aléatoire
↪   du cercle
            trigonométrique, uniformément distribué selon
↪   l'argument.
            Cette fonction est déterministe pour un même seed
↪   demandé.
        """
        s = ((self.seed & 0xFFFFFFFFFFFFFFFF) << 64) |
        ↪   ((int(xg) & 0xFFFFFFFF) << 32) | (int(yg) &
        ↪   0xFFFFFFFF)
        r.seed(s)
        return self.f(r)
```

Python

perlin.py

```
    def getChunk(self,x,y,n):
        """
                            x,y sont les coordonées du chunk à
↪   considérer (boucle au bout de 4294967296=2^32) java:int
                            n est un couple ou une liste d'aumoins
↪   deux éléments contenant la précision suivant x et y du
↪   chunk
                    """
        randomizer = lambda i,j : self.getRandomly(x+i,y+j)

        return
        ↪  np.fromfunction(np.vectorize(randomizer),(n[0],n[1]))



class RandLinNoise(RandNoise):

    y0 = 0
```

```
    y1 = 1

    def __init__(self,seed,y0,y1):
        super().__init__(seed,lambda r : (y1-y0)*r.random() +
        ↪  y0)

class RandTrigNoise(RandNoise):

    def __init__(self,seed):
        super().__init__(seed,lambda r :
        ↪  np.exp(1j*2*pi*r.random()))

class DroiteNoise(Noise):

    seed = None
    F,D = 0,0

    def __init__(self,seed,F,D):
```

Python

perlin.py

```
        self.seed = seed
        self.F,self.D = F,D

    def getChunk(self,x,y,n=None):

        return self.getLoadedDroites(x,y)

#         randomizer = lambda i,j :
↪   self.getRandomGradient(x+i,y+j)
#
#         return np.fromfunction(np.vectorize(randomizer),(n,))

    def getLoadedDroites(self,x,y):
        """
            Cette fonction renvoie la liste des droites devant
↪   être considérées dans la génération du chunk x,y. Cela
↪   permet d'effectuer la génération procédurale.
        """
```

Python

perlin.py

```
        def dst(x0,x1,y0,y1):
            """
                Cette fonction renvoie la ditance eucildienne
↪   2D entre les points (x0,y0) et (x1,y1)
            """
            return sqrt( (x1-x0)**2 + (y1-y0)**2 )

        F = self.F
        x0 = floor(x-F)
        x1 = floor(x+F+1)
        y0 = floor(y-F)
        y1 = floor(y+F+1)
#        print(x0,x1,y0,y1)
        drts = []
        for i in range(x0,x1+1):
            for j in range(y0,y1+1):
                for d in self.getDroitesOnChunk(i,j):
                    # Tester si la droite sera utile
```

```
                    dx = d[0]+i
                    dy = d[1]+j
                    if (x <= dx <= x+1 and y-F <= dy <= y+F+1)
                    ↪  or (y <= dy <= y+1 and x-F <= dx <=
                    ↪  x+F+1) or
                    ↪  (min(dst(x,dx,y,dy),dst(x+1,dx,y,dy),dst(x
                    ↪  <= F):
                        drts.append((dx,dy,d[2]))
#         print(len(drts))
        return drts


    def getDroitesOnChunk(self,xg,yg):
        s = ((self.seed & 0xFFFFFFFFFFFFFFFF) << 64) |
        ↪  ((int(xg) & 0xFFFFFFFF) << 32) | (int(yg) &
        ↪  0xFFFFFFFF)
        r.seed(s)
        L = []
```

```
        for i in range(self.D):
            lx = r.random()
            ly = r.random()
            theta = r.random()*2*pi
            L.append((lx,ly,theta))
        return L


class PerlinNoise(Noise):

    G = None
    randomizer = None
    interpol = None
    wrapper = None


    def __init__(self,G,randomizer,interpol=lambda a,b,w :
    ↪   (b-a)*w**2*6*(1/2-w/3)+a,wrapper = lambda x :
    ↪   np.tanh(x*3.8622)): # Par défaut, un bonale
```

Python

perlin.py

```
        self.G = G
        if type(randomizer) == int:
            randomizer = RandTrigNoise(randomizer)
        self.randomizer = randomizer
        self.interpol = interpol
        self.wrapper = wrapper

    def getChunkGradients(self,x,y):
        G=self.G # Python de merde !
        x0 = floor(x/G)
        x1 = ceil((x+1)/G)
        y0 = floor(y/G)
        y1 = ceil((y+1)/G)
        nx = x1-x0+1
        ny = y1-y0+1
#         grads = np.fromfunction(np.vectorize(lambda x,y :
↪   self.getPerlinGradient(x+x0,y+y0)),(nx,ny))
        grads = self.randomizer.getChunk(x0,y0,(nx,ny))
```

```
        return grads,x0,y0


    def getChunk(self,x,y,n):

        G = self.G
        chunk = np.zeros(n) # Initialise la sortie

        gradients,x0,y0 = self.getChunkGradients(x,y)

        def dotGridGradient(ix, iy, tx, ty):
            dx = tx - ix
            dy = ty - iy
            return
            ↪  (np.conj(gradients[ix-x0][iy-y0])*(dx+1j*dy)).real


        for i in range(n[0]):
```

Python

perlin.py

```
for j in range(n[1]):
    #C------------D#
    #|              |#
    #|              |#
    #|              |#
    #|    x M       |#
    #|              |#
    #A------------B#
    posx = x + i/n[0]
    posy = y + j/n[1]
    xx = posx / G
    yy = posy / G
    xx0 = floor(xx)
    yy0 = floor(yy)
    xx1 = xx0 + 1
    yy1 = yy0 + 1
```

```
                gA = dotGridGradient(xx0, yy0, xx, yy);
                gB = dotGridGradient(xx1, yy0, xx, yy);
                gC = dotGridGradient(xx0, yy1, xx, yy);
                gD = dotGridGradient(xx1, yy1, xx, yy);
                haut = self.interpol(gA, gB, xx - xx0);
                bas = self.interpol(gC, gD, xx - xx0);
                valeur =  self.interpol(haut, bas, yy - yy0);

                chunk[i,j] = valeur

        return self.wrapper(chunk)

class FractalNoise(Noise):

    F = None
    D = None
    epsilon = None
    interpol = None
```

Python

perlin.py

```
    droiteMaker = None

    def interpolizer(n,F):
        """
            Retourne une fonction polynomiale réelle sur [-F,F]
↪   et nulle autre part, s'annule en F et -F, vaut 1 en 0 et a
↪   comme dérivée 0 en -F,0 et F. n+1 est le degré de la racine
↪   0.
        """
        return lambda x : 0 if abs(x)>F else 1 +
        ↪   (2*n**2+6*n+4)/(F**(2*n+4)) *
        ↪   ((x**2)/(2*n+4)-(F**2)/(2*n+2))*abs(x)**(2*n+2)


    def __init__(self,F,D,epsilon,droiteMaker,n = 1):
        self.F = F
        self.D = D
        self.epsilon = epsilon
```

Python

perlin.py

```
        if type(droiteMaker) == int:
            droiteMaker = DroiteNoise(droiteMaker,F,D)
        self.droiteMaker = droiteMaker
        self.interpol = FractalNoise.interpolizer(n,F)

    def getChunk(self,x,y,n):
        drts = self.droiteMaker.getChunk(x,y)
        chunk = np.zeros(n)
        epsilon = self.epsilon
        interpol = self.interpol

        def dst(x0,x1,y0,y1):
            return sqrt( (x1-x0)**2 + (y1-y0)**2 )

        def kelkote(drt,x,y):
            dx = drt[0]
            dy = drt[1]
            #print(x,y,dx,dy)
```

Python

perlin.py

```
            return 1 if (np.exp(1j*(drt[2]+pi/2))    *
            ↪  ((x-dx)+(dy-y)*1j)).real >= 0 else -1

        # FractalNoise(0.7,511,0.01,42).getChunk(3,3,(16,16))
        #33.8 s ± 72.8 ms per loop (mean ± std. dev. of 7 runs,
        ↪  1 loop each)
        for d in drts:
            drteffect = lambda i,j : interpol(dst(d[0],i/n[0] +
            ↪  x,d[1],j/n[1] + y))*kelkote(d,i/n[0] + x,j/n[1]
            ↪  + y)*epsilon
            chunk += np.fromfunction(np.vectorize(drteffect),n)

        # FractalNoise(0.7,511,0.01,42).getChunk(3,3,(16,16))
        #28.9 s ± 344 ms per loop (mean ± std. dev. of 7 runs,
        ↪  1 loop each)
#           for i in range(n[0]):
#               for j in range(n[1]):
#                   posx = i/n[0] + x
```

Python

perlin.py

```
#                   posy = j/n[1] + y
#                   value = 0
##                   print(i,j)
#                   for d in drts:
#                       value +=
↪   interpol(dst(d[0],posx,d[1],posy))*kelkote(d,posx,posy)*epsilo
#                   chunk[i,j] = value
        return chunk


class TestNoise(Noise):

    nn = PerlinNoise

    def getChunk(self,x,y,n):

        indexes = np.array([i for i in range(n[0]*n[1]+1)])
        data = np.ones((n[0]*n[1]))
```

```
        return WorldChunk(n,indexes,data)



class CavernedNoise(Noise):

    perlinSurface = None
    perlinGrotte = None
    perlinFond = None

    def __init__(self):
        self.perlinSurface = PerlinNoise(.5,64)
        self.perlinGrotte  = PerlinNoise(7 ,77)
        self.perlinFond    = PerlinNoise(.3 ,23)


    def getChunk(self,x,y,n):
```

Python

perlin.py

```
chk = self.perlinSurface.getChunk(x,y,n)
fond =   self.perlinFond.getChunk(x,y,n)
grotte=self.perlinGrotte.getChunk(x,y,n)

out = []

for i in range(n[0]):
    lig = []
    for j in range(n[1]):
        if(grotte[i,j]>.2): # Pas de grotte
            lig.append([chk[i,j]])
        elif(grotte[i,j]>0):
            lig.append([fond[i,j]])
        else:

            ↪  lig.append([fond[i,j],chk[i,j]-0.1*abs(gro
    out.append(lig)
```

```
        return out


class CavernedNoise2(Noise):

    perlinCielH = None
    perlinGHaut = None
    perlinGH = None
    perlinGHp = None
    perlinGBas = None

    #-x^(4)+4x^(3)-6x^(2)+4x

    def __init__(self,seed):
        self.perlinCielH = PerlinNoise(7 ,seed)
        self.perlinGHaut = PerlinNoise(5 ,seed)
        self.perlinGH    = PerlinNoise(25 ,seed)
        self.perlinGHp   = PerlinNoise(1 ,seed)
```

Python

perlin.py

```
        self.perlinGBas  = PerlinNoise(5 ,seed)


    def getChunk(self,x,y,n):
        gtTransform = np.vectorize(lambda x : 0 if x<0 else
        ↪  sqrt(2*x-x**2)**1.5)
        transform=lambda M,a,b : M*b+a
        cielH =
        ↪  transform(self.perlinCielH.getChunk(x,y,n),28,28)
        gHaut =
        ↪  transform(self.perlinGHaut.getChunk(x,y,n),60,20)
        ghp   = transform(self.perlinGHp.getChunk(x,y,n)
        ↪  ,0.005,0.005)
        ghh   = transform(self.perlinGHp.getChunk(x,y,n)
        ↪  ,0.5,0.5)
        gBas =
        ↪  transform(self.perlinGBas.getChunk(x,y,n),10,10)
        gh = gtTransform(ghp+ghh)
```

Python

perlin.py

```
toit = 128

out = []



for i in range(n[0]):
    lig = []
    for j in range(n[1]):
        ch =  cielH[i,j] # la hauteur entre la surface
        ↪  et le ciel
        ght = gHaut[i,j] # haut limite de la grotte
        gbs = gBas[i,j] # bas limite de la grotte
        hauteur=gh[i,j] # pourcentage de hauteur de la
        ↪  grotte
        grh = (ght+gbs +hauteur*(ght-gbs))/2 # vrai
        ↪  plafond de la grotte
```

```
                grb = (ght+gbs -hauteur*(ght-gbs))/2 # vrai sol
                ↪  de la grotte
                if(ght<=40):print(ght)
                if hauteur==0:
                    # Pas de grotte
                    lig.append([toit-ch])
                elif(grh+ch>=toit):
                    # La grotte est ouverte sur la surface
                    lig.append([grb])
                else:
                    # Grotte souterraine et surface
                    lig.append([grb,grh,toit-ch])

            out.append(lig)

        return ArrayedWorldChunk.fromList(out)
```

Python

perlin.py

```
#for c in cmaps:
#    pp.figure()
#    print(c)
#    pp.imshow(I, cmap=c)



#sys.exit()
####### Fenêtre graphique #######
#from PyQt5.QtWidgets import
↪  QVBoxLayout,QHBoxLayout,QPushButton,QWidget,QApplication,QForm
#
#app = QApplication([])
#
#class ExplorerWidget(QWidget):
#
```

Python

perlin.py

```
#     def __init__():
#         print('wow')
#
##### Control Panel ####
#seedSelector = QTextEdit()
#ndroitesSelector = QDial()
#
#cPanel = QFormLayout()
#cPanel.addWidget(QLabel("Seed : "))
#cPanel.addWidget(seedSelector)
#cPanel.addWidget(QLabel("Nombre de droites :"))
#cPanel.addWidget(ndroitesSelector)
#
#globalL = QHBoxLayout()
#globalL.addStretch(1)
#globalL.addLayout(cPanel)
#
#window = QWidget()
```

Python

perlin.py

```
#window.setLayout(globalL)
#window.show()
#
#app.exec_()

#
#(x0,x1,y0,y1) = (0,4,0,4)
#n = 100
#
#
#x = np.linspace(x0,x1,n)
#y = np.linspace(y0,y1,n)
#x00 = int(x0)-1
#y00 = int(y0)-1
#x11 = int(x1)+1
#y11 = int(y1)+1
#gradient =
↪  np.exp(np.random.rand(x11-x00+1,y11-y00+1)*2*np.pi*1j)
```

Python

perlin.py

```
#X,Y = np.meshgrid(x,y)
##print(gradient)
#
#def lerp(a0, a1, w):
#    return a0 + (a1-a0)*(-2*w*w*w+3*w*w)
#
#def dotGridGradient(ix, iy, x, y):
#    dx = x - ix
#    dy = y - iy
#    return (np.conj(gradient[iy-y00][ix-x00])*(dx+1j*dy)).real
#
#def bruit(x,y):
#    (x0,y0) = (int(x),int(y))
#    (x1,y1) = (x0+1,y0+1)
#
#    sx = x - x0;
#    sy = y - y0;
#
```

Python

perlin.py

```
#     n0 = dotGridGradient(x0, y0, x, y);
#     n1 = dotGridGradient(x1, y0, x, y);
#     ix0 = lerp(n0, n1, sx);
#     n0 = dotGridGradient(x0, y1, x, y);
#     n1 = dotGridGradient(x1, y1, x, y);
#     ix1 = lerp(n0, n1, sx);
#     return lerp(ix0, ix1, sy);
#
#
#
#Z = np.zeros((n,n))
#for i in range(n):
#     for j in range(n):
#          Z[i,j] = bruit(x[i],y[j])
#
#pp.imshow(Z,cmap='autumn')
#
#fig = pp.figure()
```

Python

perlin.py

```
#ax = pp.axes(projection='3d')
#
#ax.view_init(80, 42)
#ax.plot_surface(X,Y,Z, rstride=1, cstride=1,
#                cmap='autumn', edgecolor='none')
```

Python

tmf.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
        Module contant les fonctions permettant de lire et
↪   écrire des fichiers dans le format TMF

        @author: mysaa
"""


class WorldSaver():

    def __init__():
        regSize=0
```