

Universidade do Minho
Departamento de Matemática

Mestrado Integrado em Engenharia Informática

Computação Gráfica



Graphical primitives

A82238 João Gomes
A81953 Pedro Barros
A80328 Pedro Lima
A80624 Sofia Teixeira

Braga
2020

1 Introdução

A primeira fase do trabalho proposto na unidade curricular de computação gráfica, inserida no Mestrado Integrado de Engenharia Informática, teve como principal objetivo a formulação e representação de várias primitivas gráficas.

Isto foi conseguido através da utilização de ferramentas sugeridas pelos docentes, nomeadamente o Microsoft Visual Studio, *CMake* assim como alguns toolkits, sendo o *GLUT* (*OpenGL Utility Toolkit*) o único a ser utilizado nesta fase inicial. Estas ferramentas permitiram, em suma, trabalhar em OpenGL de modo a criar modelos tridimensionais.

2 Arquitetura do Projeto

Após efetuarmos uma avaliação geral do enunciado proposto, deliberamos que a melhor abordagem seria dividir o projeto em dois pedaços principais:

1. **Generator** - Aqui estão definidas as respetivas formas geométricas. Este generator vai criar os vértices de modo a que seja possível representar as formas da maneira pretendida (dado um certo comprimento, altura...).
2. **Engine** - Esta é a aplicação que contém as funcionalidades requiridas para demonstrar os sólidos pretendidos. Permite a exibição tridimensional dos sólidos a ser especificados no ficheiro xml.

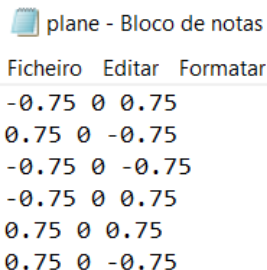
2.1 Generator

O generator ao ser executado vai gerar sempre um ficheiro *config.xml* e dependendo dos argumentos gera o ficheiro *.3d* pretendido.

Estes ficheiros podem posteriormente ser utilizados na config, que será parsed mais tarde pela nossa *engine*.

O generator é composto neste momento pelos seguintes componentes:

- **main.cpp** - A main do generator receberá os parâmetros do sólido que pretendemos gerar, e cria os pontos através do respetivo ficheiro dependendo do input recebido. No final deste processo, o writer escreverá ainda um ficheiro *.3d* onde estarão os pontos da forma geométrica criada.
- **writer.cpp** - O writer escreve os pontos gerados para um ficheiro *.3d*. Tal como é possível observar no exemplo da figura 1, são escritas em cada linha as coordenadas *x*, *y* e *z* de cada ponto.



```
plane - Bloco de notas
Ficheiro  Editar  Formatar
-0.75 0 0.75
0.75 0 -0.75
-0.75 0 -0.75
-0.75 0 0.75
0.75 0 0.75
0.75 0 -0.75
```

Figura 1: Ficheiro com pontos.

Os seguintes componentes formulam os pontos da respetiva primitiva gráfica que serão recebidos pelo **writer.cpp**:

- **plane.cpp**
- **box.cpp**
- **cone.cpp**
- **sphere.cpp**

2.2 *Engine*

Tal como foi mencionado no ponto anterior, a *Engine* tem como função principal gerar a representação gráfica dos sólidos a partir do nosso ficheiro *config.xml*, que por sua vez conterá os respetivos ficheiros *.3d*.

Assim como o *Generator*, o nosso *Engine* é constituído por uma série de ficheiros auxiliares que facilitam todo o processo de criação da forma geométrica:

- **main.cpp** - A *main* executará na respetiva ordem os ficheiros mencionados abaixo. Isto é, primeiro será feito um *Read* do ficheiro com as figuras geométricas que queremos criar, transformando uma lista de pontos num vetor com os mesmos. Posteriormente, estes pontos serão transformados em triângulos, que quando apresentados em simultâneo pela nossa *main* representarão as figuras geométricas propostas no enunciado.
- **renderer.cpp** - O *renderer* invoca o reader e após este realizar as suas funções e no fim do *renderer* processar esta informação, ficarão formulados os triângulos que formarão a primitiva geométrica.
- **reader.cpp** - O *reader* lê os pontos dos ficheiros *.3d* indicados no ficheiro *config.xml* e consequentemente transforma-os num vetor.

2.3 *Common*

A pasta *Common*, como o nome sugere, é comum a ambas as aplicações mencionadas previamente. Esta contém a classe *Point* que será a nossa estrutura de dados para este projeto. Como o nome da classe sugere, representa um ponto através das suas coordenadas (x,y,z).

- **point.cpp** - Neste ficheiro está definida a nossa estrutura de dados, um *Point*.

3 Primitivas Gráficas

Nesta secção do relatório faremos uma breve introdução de cada primitiva gráfica, dos respetivos parâmetros e processos de desenvolvimento. Neste projeto temos como primitivas gráficas o **plano**, a **caixa**, o **cone**, e a **esfera**. Estas denominam-se primitivas gráficas uma vez que são formas geográficas irredutíveis (para além do triângulo, a forma geográfica que permitirá a criação de todas as outras).

3.1 Triângulos

Como mencionado em cima, a forma geográfica fundamental deste projeto será o triângulo, uma vez que é este que permite todas as outras formas. Este é construído seguindo a regra da mão direita, para que seja possível a sua representação gráfica após ter sido processado pela máquina.

Esta regra é demonstrada na figura seguinte, onde está representada a construção de um triângulo para que este seja visível após ter sido criado.

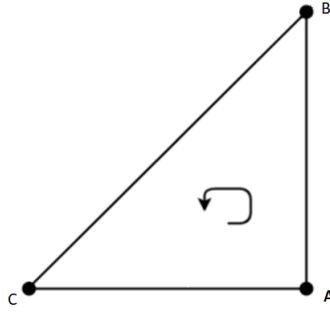


Figura 2: Construção do Triângulo.

3.2 Plano

3.2.1 Parâmetros

- Dimensão

O plano tem como input uma dimensão. Esta dimensão será o tamanho de cada aresta do plano. A solução conseguida pelo grupo para esta primitiva gráfica foi através da dimensão recebida pelo programa, descobrir as coordenadas para criar um plano centrado à origem.

Sendo d a dimensão recebida, querendo centrar o plano à origem, sabemos que os vértices serão em x e z , respetivamente ordenados por quadrante:

1. $(d/2, -d/2)$
2. $(-d/2, -d/2)$
3. $(-d/2, d/2)$
4. $(d/2, d/2)$

Sabendo as coordenadas, podemos facilmente construir dois triângulos de modo a criar um plano.

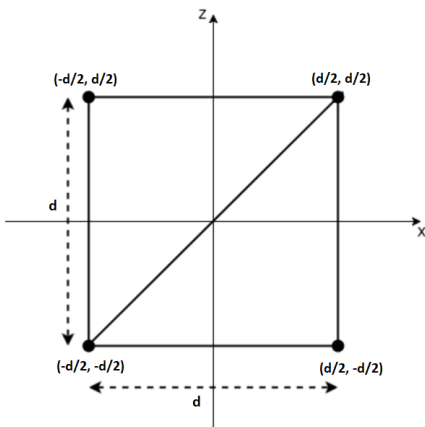


Figura 3: Construção do Plano.

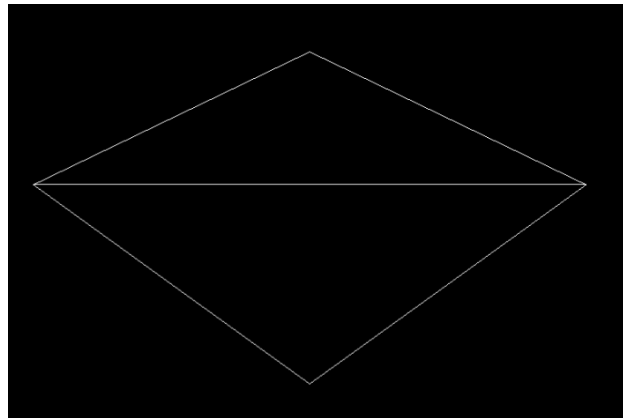


Figura 4: Plano.

3.3 Caixa

3.3.1 Parâmetros

- Altura
- Comprimento
- Profundidade
- Divisões

A caixa é construída a partir dos 4 parâmetros mencionados em cima. Após receber a altura, comprimento e largura, semelhante ao plano, estes valores serão divididos por 2 de modo a obtermos as coordenadas que corresponderão aos vértices da nossa caixa que estará centralizada nos eixos.

Para realizar uma caixa com n partições, as arestas serão então divididas por n , sendo então obtido o tamanho das arestas dos blocos que vão constituir a caixa.

O processo de construção da caixa efetua-se desenhando os triangulos necessários conforme se vai percorrendo as áreas das divisões. A ordem de construção trata-se então de $-z$ para z , $-x$ para x e $-y$ para y , tal como se pode observar na 5. Começa-se então pela camada inicial, pela linha da esquerda. A partir do fundo, começam a ser desenhados os triangulos da area do fundo avançando assim na direção de $-z$ para z . De seguida passa-se para a linha seguinte da direita e realiza-se o mesmo processo. Quando as linhas forem todas percorridas, reinicia-se o processo na camada de cima. (tem-se de escrever, antes ou depois, que 0 divisoes é a caixa normal tal como na figura 6, a opção de n divisões faz com que sejam feitas n divisoes em cada aresta no sólido, ou seja, uma divisao, vai partir cada aresta em dois blocos, partindo então cada face em 4 blocos, tal como na figura 7. e temos também o exemplo com 2 divisões, cortando assim cada aresta duas vezes, ficando a face dividida em 9)

explicar processo, os parametros, etc recebe 3 valores, altura, comprimento, largura, parte valores a meio para ficar centrado, fazes base, topo e faces

ordem começa-se pela camada de baixo começa-se na linha da esquerda e começa-se a construir os triangulos dessa linha na direção $-z$ para z depois passa-se uma linha para a direita e repete-se quando se terminar as linhas avança-se para a seguinte camada e começa-se de novo começa-se

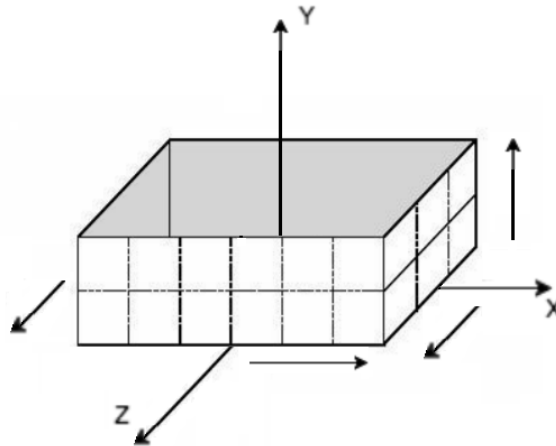


Figura 5: Construção da Caixa.

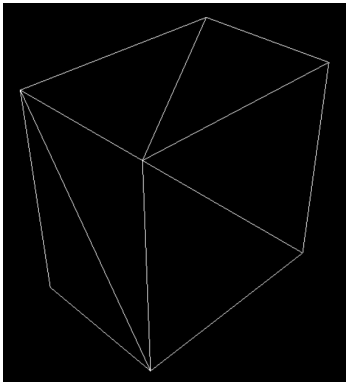


Figura 6: Caixa - sem divisões.

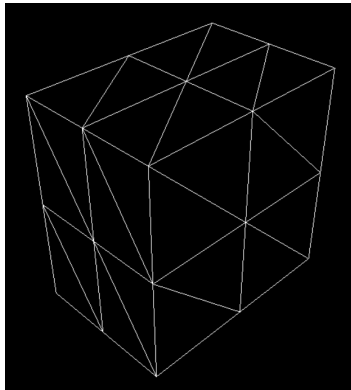


Figura 7: Caixa - 1 divisões.

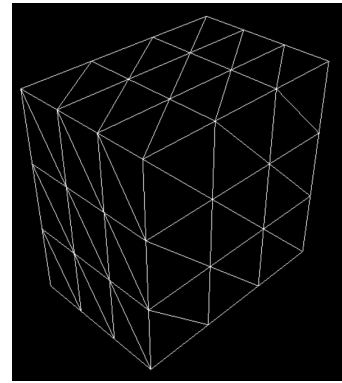


Figura 8: Caixa - 2 divisões.

3.4 Cone

Parâmetros

- altura
- raio
- *slices*
- *stacks*

Para construir o cone, a nossa aplicação recebe como input 4 parâmetros. A altura, raio, slices e stacks. A altura e o raio do cone são bastantes intuitivos de perceber a sua utilidade. Como as nossas primitivas são construídas a partir de triângulos, tivemos de arranjar uma forma de dividir a superfície do cone para conseguir desenhar todos os triângulos necessários. É aqui que entram os parâmetros slices e stacks. As slices representam cortes verticais perpendiculares à base do cone, enquanto que as stacks representam cortes horizontais paralelos à base do cone. Após todas as slices e stacks serem aplicadas, o nosso cone fica dividido em múltiplas secções, todas idênticas entre si. Cada uma destas secções servirá para desenhar dois triângulos que partilham entre si dois vértices.

Usando coordenadas esféricas

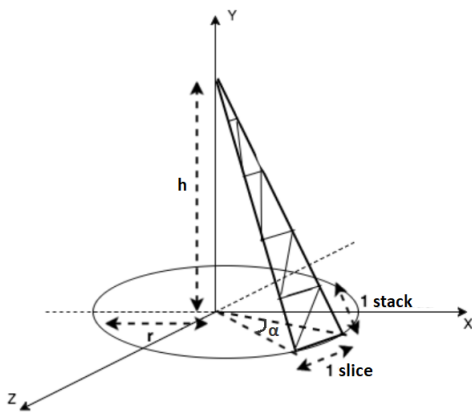


Figura 9: Construção do Cone.

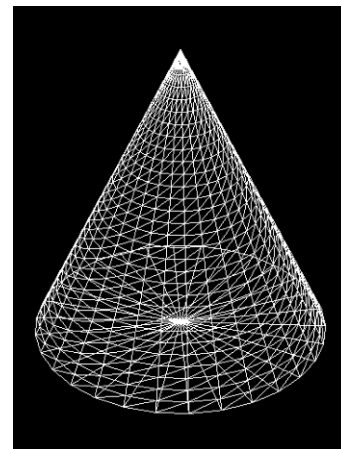


Figura 10: Cone.

3.5 Esfera

Parâmetros

- raio
- *slices*
- *stacks*

explicar processo, os parametros, etc coordenadas esfericas

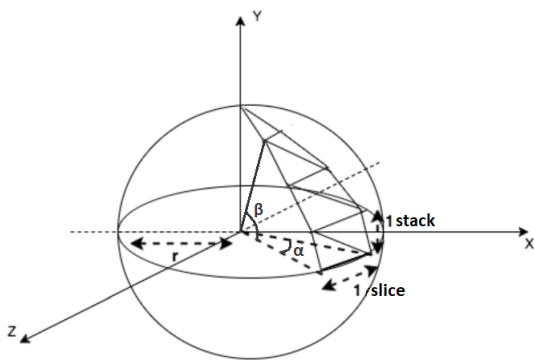


Figura 11: Construção da Esfera.

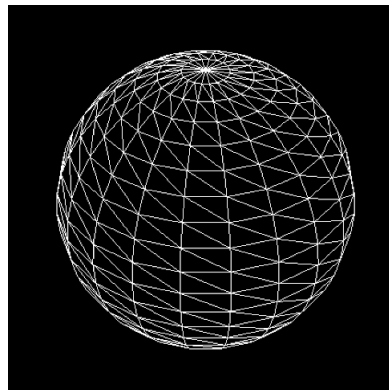


Figura 12: Esfera.

3.6 Conjuntos

bla

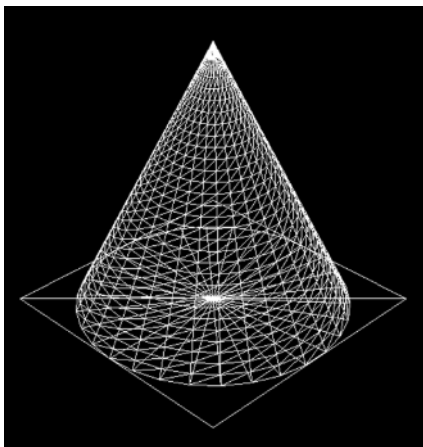


Figura 13: Conjunto: Cone e Plano.

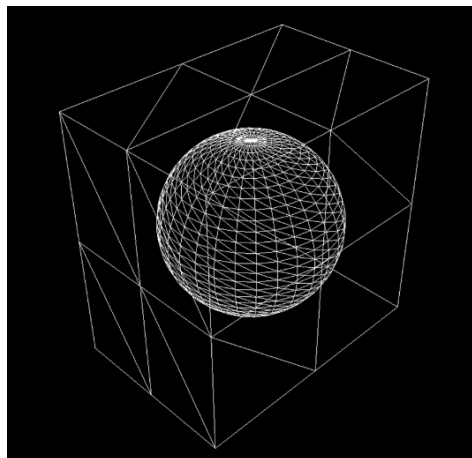


Figura 14: Conjunto: Esfera e Caixa.

4 Demonstração Gráfica da Aplicação

os outros têm a desenvolver o generator e mostrar os comandos e assim

5 Conclusão

6 just aqui para nao ter de ir buscar codigo a outro trabalho,
ignorar

6.1 Regular

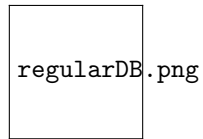


Figura 15: Base de dados numérica.

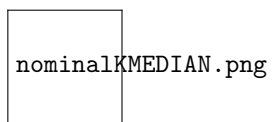


Figura 16: *K-Median Clustering*

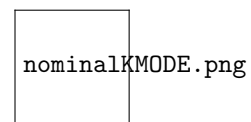


Figura 17: *K-Mode Clustering*

- idk
- idk

Bibliografia

- [1] Sheetal Kumrawat Archna Kumari Pramod S. Nair. “An Enhanced K-Medoid Clustering Algorithm”. Em: *International Journal on Recent and Innovation Trends in Computing and Communication* 4.6 ().
- [2] Stéphane Louis Clain. “Machine Learning - Theory and applications”. Em: (2019), pp. 43–58.
- [3] Zhexue Huang. “A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining”. Em: *Research Issues on Data Mining and Knowledge Discovery* (1997).

7 Anexos

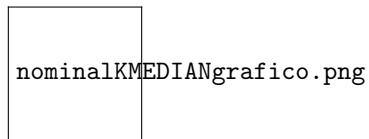


Figura 18: Gráfico de Clusters para *K-Median* na Base de Dados Nominal.

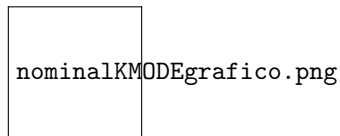


Figura 19: Gráfico de Clusters para *K-Mode* na Base de Dados Nominal.

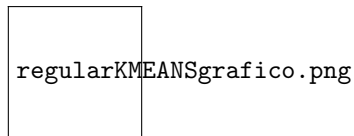


Figura 20: Gráfico de Clusters para *K-Mean* na Base de Dados Numérica Regular.

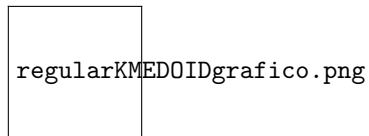


Figura 21: Gráfico de Clusters para *K-Medoid* na Base de Dados Numérica Regular.

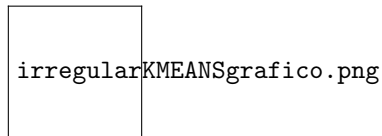


Figura 22: Gráfico de Clusters para *K-Mean* na Base de Dados Numérica Irregular.

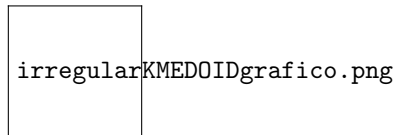


Figura 23: Gráfico de Clusters para *K-Medoid* na Base de Dados Numérica Irregular.

```

1 # imports
2 import csv
3 import scipy.spatial.distance as dist
4 import matplotlib.pyplot as plt
5 import pandas as pd
6 import numpy as np
7 import math
8
9
10
11 #===== my functions =====
12 def my_distance(x,y):
13     if x.size!=y.size:
14         return(-1)
15     nn=dist.euclidean(x,y)
16     return nn
17
18 def cost(dt, med, lab):
19     C = 0;
20     for i in set(lab):
21         cluster = dt[lab==i]
22         for n in range(0,len(cluster)):
23             C = C + my_distance(cluster[n,:], med[i])
24     return C
25
26 def cost_centros(med):
27     C = 0;
28     l = len(med)
29     M = [0,0];
30     for i in range(0,l):
31         M = M + med[i]
32     M = M / l
33     for n in range(0,l):
34         C = C + my_distance(M, med[n])
35     return C
36
37 def initialise_representative(data,K):
38     return data[0:K,:]
39
40 def assignement(data,rep,K,N):
41     clusters=np.zeros(N,dtype=np.int)
42     for n in range(0,N):
43         val_min=1E20
44         for k in range(0,K):
45             val=my_distance(data[n,:],rep[k,:])
46             if(val<val_min):
47                 val_min=val
48                 clusters[n]=k
49     return clusters
50
51 def centroid_mean(data,clusters,K,N):
52     rep=data[0:K,:]*0
53     nb=np.zeros(K,dtype=np.int)
54
55     for n in range(0,N):
56         k=clusters[n]
57         rep[k,:]=rep[k,:]+data[n,:]
58         nb[k] = nb[k] + 1
59
60     for k in range(0,K):
61         soma = rep[k,0]+rep[k,1]
62         if soma!=0:
63             rep[k,:]=rep[k,:] / nb[k]
64
65     return rep

```

```

66
67 def Error_representative(old_rep,rep,K):
68     value=0
69     for k in range(0,K):
70         value=value + my_distance(old_rep[k,:],rep[k,:])
71     return value
72
73 def within_clusters(data,rep,clusters,K,N):
74     Sw=np.zeros(K,dtype=np.float)
75     nb=np.zeros(K,dtype=np.int)
76     for n in range(0,N):
77         k=clusters[n]
78         Sw[k]=Sw[k]+my_distance(data[n,:],rep[k,:])
79         nb[k]=nb[k]+1
80     Sw=np.divide(Sw,nb)
81     return Sw
82
83 def between_cluster(rep,K):
84     Sb=np.ones(K,dtype=np.float)*1E20
85     for k in range(0,K):
86         for kk in range(0,K):
87             val=my_distance(rep[k,:],rep[kk,:])
88             if (k!=kk) and val<Sb[k]:
89                 Sb[k]=val
90     return Sb
91
92 def plotCentroid(rep, D):
93     fig = plt.figure(1, figsize=(8, 6))
94     plt.scatter( D[:, 0], D[:, 1], c=clusters,s=50,cmap='gnuplot')
95     plt.scatter( rep[:, 0], rep[:, 1], label='Centroids', c='grey', s=150, alpha=0.5)
96     plt.xlabel("Length", fontsize='medium', fontweight='bold', labelpad=15)
97     plt.ylabel("Width", fontsize='medium', fontweight='bold', labelpad=15)
98     plt.legend(loc='lower left')
99     plt.savefig("k-mean.png")
100    plt.show()
101
102 #===== MAIN =====
103
104 # read the data file
105 my_data = pd.read_csv('./Database/irregularDB.csv',sep=',');D=np.array(my_data.values
    [0:3000,0:3],dtype=np.float)
106
107 # set the sizes
108 N=D.shape[0]
109 d=D.shape[1]
110 custos = []
111
112 for K in range(1,N):
113     error=1;ncount=0;n_MAX=20
114
115     # initialise clustering structure
116     rep=initialise_representative(D,K)
117
118     # main loop
119     while ((error>1.E-8) and (ncount<n_MAX)):
120         clusters=assigment(D,rep,K,N)
121         old_rep=rep
122         rep=centroid_mean(D,clusters,K,N)
123         error=Error_representative(old_rep,rep,K)
124         ncount=ncount+1
125
126     c = cost(D, rep, clusters) + cost_centros(rep)
127     custos.append(c)
128
129 plt.xlabel("Number of clusters", fontsize='medium', fontweight='bold', labelpad=15)

```

```

130 plt.ylabel("Cost", fontsize='medium', fontweight='bold', labelpad=15)
131 plt.plot(custos)
132 plt.savefig("k-means-clusters.png")
133 plt.show()
134
135 K = custos.index(min(custos))
136 print('Numero ideal de clusters: ',K)
137 error=1;ncount=0;n_MAX=20
138
139 # initialise clustering structure
140 rep=initialise_representative(D,K)
141 # main loop
142 while ((error>1.E-8) and (ncount<n_MAX)):
143     clusters=assignment(D,rep,K,N)
144     old_rep=rep
145     rep=centroid_mean(D,clusters,K,N)
146     error=Error_representative(old_rep,rep,K)
147     ncount=ncount+1
148
149 plotCentroid(rep, D)
150
151 print('number of iteration:',ncount)
152
153 # validation
154 Sw=within_clusters(D,rep,clusters,K,N)
155 Sb=between_cluster(rep,K)
156 print("max within clusters error",Sw.max())
157 print("min between clusters error",Sb.min())
158 print("Clustering index:",Sw.max()/Sb.min())

```

Listing 1: *Script em Python para K-Mean*

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import scipy.spatial.distance as dist
5 from mpl_toolkits.mplot3d import Axes3D
6 from sklearn import datasets
7 from sklearn.decomposition import PCA
8
9 # Dataset
10 data = pd.read_csv("Database/irregularDB.csv")
11 datapoints=np.array(data.values[0:3000,0:3])
12
13 m, f = datapoints.shape
14
15 def my_distance(x,y):
16     if x.size!=y.size:
17         return(-1)
18     nn=dist.cityblock(x,y)
19     return nn
20
21 def init_medoids(X, k):
22     from numpy.random import choice
23     from numpy.random import seed
24
25     seed(1)
26     samples = choice(len(X), size=k, replace=False)
27     return X[samples, :]
28
29
30 def assigment(data,rep,K,N):
31     clusters=np.zeros(N,dtype=np.int)
32     for n in range(0,N):
33         val_min=1E20
34         for k in range(0,K):

```

```

35         val=my_distance(data[n,:],rep[k,:])
36         if(val<val_min):
37             val_min=val
38             clusters[n]=k
39     return clusters
40
41 def cost(dt, med, lab):
42     C = 0;
43     for i in set(lab):
44         cluster = dt[lab==i]
45         for n in range(0,len(cluster)):
46             C = C + my_distance(cluster[n,:], med[i])
47     return C
48
49 def cost_centros(med):
50     C = 0;
51     l = len(med)
52     M = [0,0];
53     for i in range(0,l):
54         M = M + med[i]
55     M = M / l
56     for n in range(0,l):
57         C = C + my_distance(M, med[n])
58     return C
59
60 def update_medoids(X, medoids):
61
62     labels = assignment(X, medoids, k, m)
63
64     for i in set(labels):
65         new_medoids = medoids.copy()
66
67         cluster_points = datapoints[labels == i]
68         avg_cost = cost(datapoints, new_medoids, labels)
69
70         for datap in cluster_points:
71             new_medoids[i] = datap
72             new_labels = assignment(X, new_medoids, k, m)
73
74             new_cost = cost(datapoints, new_medoids, new_labels)
75
76             if new_cost <= avg_cost :
77                 avg_cost = new_cost
78                 medoids[i] = datap
79
80     labels = assignment(X, medoids, k, m)
81
82     return medoids
83
84 def has_converged(old_medoids, medoids):
85     return set([tuple(x) for x in old_medoids]) == set([tuple(x) for x in medoids])
86
87 #Full algorithm
88 def kmedoids(X, max_steps=np.inf):
89     medoids = init_medoids(X, k)
90
91     converged = False
92     labels = np.zeros(len(X))
93
94     while (not converged):
95         old_medoids = medoids.copy()
96
97         labels = assignment(X, medoids, k, m)
98
99         medoids = update_medoids(X, medoids)

```

```

100         converged = has_converged(old_medoids, medoids)
101
102     return (medoids, labels)
103
104 custos = []
105
106 for k in range(1, m):
107     medoids_initial = init_medoids(datapoints, k)
108
109     labels = assignment(datapoints, medoids_initial, k, m)
110
111     results = kmedoids(datapoints)
112     final_medoids = results[0]
113     data['clusters'] = results[1]
114     labels = assignment(datapoints, final_medoids, k, m)
115
116     c = cost(datapoints, final_medoids, labels) + cost_centros(final_medoids)
117     custos.append(c)
118
119 plt.xlabel("Number of clusters", fontsize='medium', fontweight='bold', labelpad=15)
120 plt.ylabel("Cost", fontsize='medium', fontweight='bold', labelpad=15)
121 plt.plot(custos)
122 plt.savefig("k-medoid-clusters.png")
123 plt.show()
124
125
126 k = custos.index(min(custos))
127 print('Numero ideal de clusters: ', k)
128
129 medoids_initial = init_medoids(datapoints, k)
130
131 labels = assignment(datapoints, medoids_initial, k, m)
132
133 results = kmedoids(datapoints)
134 final_medoids = results[0]
135 data['clusters'] = results[1]
136 labels = assignment(datapoints, final_medoids, k, m)
137
138 c = cost(datapoints, final_medoids, labels) + cost_centros(final_medoids)
139 custos.append(c)
140
141 #Visualization
142 fig = plt.figure(1, figsize=(8, 6))
143 plt.scatter(datapoints[:, 0], datapoints[:, 1], c=labels, s=50, cmap='gnuplot')
144 plt.scatter(final_medoids[:, 0], final_medoids[:, 1], label='Medoids', c='grey', s
145             =200, alpha=0.5)
146 plt.xlabel("Length", fontsize='medium', fontweight='bold', labelpad=15)
147 plt.ylabel("Width", fontsize='medium', fontweight='bold', labelpad=15)
148 plt.legend(loc='lower left')
149 plt.savefig("k-medoid.png")
150 plt.show()

```

Listing 2: *Script em Python para K-Medoid*

```

1
2 from __future__ import division
3
4 import pandas as pd
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import scipy.spatial.distance as dist
8 from mpl_toolkits.mplot3d import Axes3D
9 from sklearn import datasets
10 from sklearn.decomposition import PCA
11

```



```

12 import matplotlib.pyplot as plt
13 from matplotlib import colors as mcolors
14
15 data = pd.read_csv("Database/nominalDB.csv")
16 datapoints=np.array(data.values[0:3000,0:3])
17
18 data_shapes = datapoints[:,0]
19 data_cores = datapoints[:,1]
20 lista_cores = [k.lower() for k in data_cores]
21 lista_shapes = [k.lower() for k in data_shapes]
22
23
24 #-----Organizar cores-----
25 colors = dict(mcolors.BASE_COLORS, **mcolors.CSS4_COLORS)
26 # Sort
27 by_hsv = sorted((tuple(mcolors.rgb_to_hsv(mcolors.to_rgba(color)[:3])), name)
28                 for name, color in colors.items()))
29 sorted_names = [name for hsv, name in by_hsv]
30 # Sort apenas as cores que temos no data_cores
31 cores = []
32 for key in sorted_names:
33     if(key in lista_cores):
34         cores.append(key)
35 #-----
36
37 # Passar de categorico a numerico
38 numerar_cores = [1,2,3,4,5,6,7]
39 d=dict(zip(cores, numerar_cores))
40 nrs_cores = [d[v] for v in lista_cores if v in d]
41
42 shapes = ['circle', 'triangle', 'square']
43 numerar_shapes = [1,2,3]
44 dd=dict(zip(shapes, numerar_shapes))
45 nrs_shapes = [dd[v] for v in lista_shapes if v in dd]
46
47 # Encontrar mediana
48 #nrs_cores.sort()
49 #nrs_shapes.sort()
50
51 N = datapoints.shape[0]
52 mediana = np.array([nrs_shapes[N//2], nrs_cores[N//2]])
53
54 for k in range(0,N):
55     datapoints[k,0] = nrs_shapes[k]
56     datapoints[k,1] = nrs_cores[k]

```

Listing 3: *Script em Python para K-Median*

```

1 import csv
2 import scipy.spatial.distance as dist
3 import matplotlib.pyplot as plt
4 import pandas as pd
5 import numpy as np
6 import math
7
8
9
10 #===== my functions =====
11
12
13 def cost(dt, med, lab):
14     C = 0;
15     for i in set(lab):
16         cluster = dt[lab==i]
17         for n in range(0,len(cluster)):
18             C = C + my_distance(cluster[n,:], med[i])

```

```

19     return C
20
21 def initialise_representative(data,K):
22     unique, counts = np.unique(data.astype("<U22"), return_counts=True, axis = 0)
23
24     # array com os indices da maior frequencia para a menor
25     frq_index = []
26     for c in range(0, len(counts)):
27         m = np.argmax(counts)
28         counts[m] = 0
29         frq_index.append(m)
30     rep = []
31     for k in range(0,K):
32         rep.append(unique[frq_index[k]])
33     rep = np.array(rep)
34     return rep
35
36 def my_distance(x, c):
37     return np.sum(np.array(x) != np.array(c), axis = 0)
38
39 def assignment(data,rep,K,N):
40     clusters=np.zeros(N,dtype=np.int)
41     for n in range(0,N):
42         val_min=1E20
43         for k in range(0,K):
44             val=my_distance(data[n,:],rep[k,:])
45             if(val<val_min):
46                 val_min=val
47                 clusters[n]=k
48     return clusters
49
50 def centroid_mode(data,clusters,K,N):
51     for i in set(clusters):
52         cluster_points = data[clusters==i]
53         cluster_shapes = cluster_points[:,0].copy()
54         cluster_cores = cluster_points[:,1].copy()
55
56         s, c = np.unique(cluster_shapes, return_counts=True)
57         c_shapes_mode = s[np.argmax(c)]
58         s, c = np.unique(cluster_cores, return_counts=True)
59         c_cores_mode = s[np.argmax(c)]
60
61         rep[i,:] = np.array([c_shapes_mode, c_cores_mode])
62
63     return rep
64
65 def plotCentroid(rep, D):
66
67     plt.scatter( D[:, 0], D[:, 1], c=clusters,s=50,cmap='tab20')
68     plt.scatter( rep[:, 0], rep[:, 1], c='grey', s=200, alpha=0.5)
69     plt.xlabel("Shapes", fontsize='medium', fontweight='bold', labelpad=15)
70     plt.ylabel("Colors", fontsize='medium', fontweight='bold', labelpad=15)
71     plt.savefig("k-mode.png")
72     plt.show()
73
74 #===== MAIN =====
75
76 # read the data file
77 my_data = pd.read_csv('./Database/nominalDB.csv',sep=',');D=np.array(my_data.values
78                               [0:3000,0:3],dtype='U10')
79
80 # set the sizes
81 N=D.shape[0] #number of elements/events
82 d=D.shape[1] #number of attribute
83 K=3 #number of clusters

```

```

83 custos = []
84
85 for K in range(1,7*3):
86
87     # initialise clustering structure
88     rep=initialise_representative(D,K)
89     old_rep = []
90
91     # main loop
92     while (not np.array_equal(rep, old_rep)):
93         clusters=assigment(D,rep,K,N)
94         old_rep = rep.copy()
95         rep=centroid_mode(D,clusters,K,N)
96
97     c = cost(D, rep, clusters) + K*K
98     custos.append(c)
99
100 plt.xlabel("Number of clusters", fontsize='medium', fontweight='bold', labelpad=15)
101 plt.ylabel("Cost", fontsize='medium', fontweight='bold', labelpad=15)
102 plt.plot(custos)
103 plt.savefig("k-mode-clusters.png")
104 plt.show()
105
106 K = custos.index(min(custos))
107 print('Numero ideal de clusters: ',K)
108
109
110 # initialise clustering structure
111 rep=initialise_representative(D,K)
112 old_rep = []
113
114 # main loop
115 while (not np.array_equal(rep, old_rep)):
116     clusters=assigment(D,rep,K,N)
117     old_rep = rep.copy()
118     rep=centroid_mode(D,clusters,K,N)
119
120 plotCentroid(rep, D)

```

Listing 4: *Script em Python para K-Mode*