EscultorDigital

AUTHOR
Versão 1.0

Sumário

Table of contents

Índice dos Componentes

Lista de Classes

Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições:

Índice dos Arquivos

Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

Classes

Referência da Classe Sculptor

A classe **Sculptor** implementa os métodos para a modelagem do desenho.
`#include <sculptor.h>`

Membros Públicos

- **Sculptor** (int _nx, int _ny, int _nz)
- **~Sculptor** ()
- void **setColor** (float r, float g, float b, float a)
  O método setColor define as cores ao desenho.

- void **putVoxel** (int x, int y, int z)
- void **cutVoxel** (int x, int y, int z)
- void **putBox** (int x0, int x1, int y0, int y1, int z0, int z1)
  O método putBox coloca uma caixa no espaço onde o desenho será esculpido.

- void **cutBox** (int x0, int x1, int y0, int y1, int z0, int z1)
- void **putSphere** (int xcenter, int ycenter, int zcenter, int radius)
- void **cutSphere** (int xcenter, int ycenter, int zcenter, int radius)
- void **putEllipsoid** (int xcenter, int ycenter, int zcenter, int rx, int ry, int rz)
- void **cutEllipsoid** (int xcenter, int ycenter, int zcenter, int rx, int ry, int rz)
- void **writeOFF** (const char *filename)

Descrição detalhada

A classe **Sculptor** implementa os métodos para a modelagem do desenho.

Construtores e Destrutores

Sculptor::Sculptor (int _nx, int _ny, int _nz)

```
18                                                  { // construtor
19   // Incializacao das matrizes de voxel
20   this->nx =  nx;
21   this->ny =  ny;
22   this->nz = _nz;
23   this->r = 0;
24   this->g = 0;
25   this->b = 0;
26   this->a = 0;
27
28   // Alocacao dinamica
29   v = new Voxel **[nx]; // Alocar matriz 3D
30   for (int i = 0; i < nx; i++) {
31     v[i] = new Voxel *[ny];
32     for (int j = 0; j < ny; j++) {
33       v[i][j] = new Voxel[nz];
34     }
35   }
36
37   for (int i = 0; i < nx; i++) {
38     for (int j = 0; j < ny; j++) {
39       for (int k = 0; k < nz; k++) {
40         v[i][j][k].show = false;
41         v[i][j][k].r = 0;
```

```
42          v[i][j][k].g = 0;
43          v[i][j][k].b = 0;
44        }
45      }
46    }
47 }
```

Sculptor::~Sculptor ()

```
49                      { // destrutor - liberar memoria
50    for (int i = 0; i < nx; i++) {
51      for (int j = 0; j < ny; j++) {
52        delete[] v[i][j];
53      }
54    }
55    for (int i = 0; i < nx; i++) {
56      delete[] v[i];
57    }
58    delete[] v;
59 }
```

Documentação das funções

void Sculptor::cutBox (int *x0*, int *x1*, int *y0*, int *y1*, int *z0*, int *z1*)

```
96                                                                    {
97    for (int i = x0; i <= x1; i++) {
98      for (int j = y0; j <= y1; j++) {
99        for (int k = z0; k <= z1; k++) {
100         cutVoxel(i, j, k);
101       }
102     }
103   }
104 }
```

void Sculptor::cutEllipsoid (int *xcenter*, int *ycenter*, int *zcenter*, int *rx*, int *ry*, int *rz*)

```
150                                              {
151   for (int i = 0; i < nx; i++) {
152     for (int j = 0; j < ny; j++) {
153       for (int k = 0; k < nz; k++) {
154         if (((pow((i - xcenter), 2) / (pow(rx, 2))) +
155              (pow((j - ycenter), 2) / (pow(ry, 2))) +
156              (pow((k - zcenter), 2) / (pow(rz, 2)))) <= 1) {
157           cutVoxel(i, j, k);
158         }
159       }
160     }
161   }
162 }
```

void Sculptor::cutSphere (int *xcenter*, int *ycenter*, int *zcenter*, int *radius*)

```
120                                                                   {
121   for (int i = 0; i < nx; i++) {
122     for (int j = 0; j < ny; j++) {
123       for (int k = 0; k < nz; k++) {
124         if (((pow((i - xcenter), 2) / (pow(radius, 2))) +
125              (pow((j - ycenter), 2) / (pow(radius, 2))) +
126              (pow((k - zcenter), 2) / (pow(radius, 2)))) <= 1) {
127           cutVoxel(i, j, k);
128         }
129       }
130     }
131   }
132 }
```

void Sculptor::cutVoxel (int *x*, int *y*, int *z*)

```
80 { v[x][y][z].show = false; }
```

void Sculptor::putBox (int *x0*, int *x1*, int *y0*, int *y1*, int *z0*, int *z1*)

O método putBox coloca uma caixa no espaço onde o desenho será esculpido.

```
86                                                                          {
87   for (int i = x0; i <= x1; i++) {
88     for (int j = y0; j <= y1; j++) {
89       for (int k = z0; k <= z1; k++) {
90         putVoxel(i, j, k);
91       }
92     }
93   }
94 }
```

void Sculptor::putEllipsoid (int *xcenter*, int *ycenter*, int *zcenter*, int *rx*, int *ry*, int *rz*)

```
135                                                 {
136   for (int i = 0; i < nx; i++) {
137     for (int j = 0; j < ny; j++) {
138       for (int k = 0; k < nz; k++) {
139         if (((pow((i - xcenter), 2) / (pow(rx, 2))) +
140              (pow((j - ycenter), 2) / (pow(ry, 2))) +
141              (pow((k - zcenter), 2) / (pow(rz, 2)))) <= 1) {
142           putVoxel(i, j, k);
143         }
144       }
145     }
146   }
147 }
```

void Sculptor::putSphere (int *xcenter*, int *ycenter*, int *zcenter*, int *radius*)

```
106                                                                         {
107   for (int i = 0; i < nx; i++) {
108     for (int j = 0; j < ny; j++) {
109       for (int k = 0; k < nz; k++) {
110         if (((pow((i - xcenter), 2) / (pow(radius, 2))) +
111              (pow((j - ycenter), 2) / (pow(radius, 2))) +
112              (pow((k - zcenter), 2) / (pow(radius, 2)))) <= 1) {
113           putVoxel(i, j, k);
114         }
115       }
116     }
117   }
118 }
```

void Sculptor::putVoxel (int *x*, int *y*, int *z*)

```
72                                              { // metodo putVoxel
73   this->v[x][y][z].show = true;
74   this->v[x][y][z].r = this->r;
75   this->v[x][y][z].g = this->g;
76   this->v[x][y][z].b = this->b;
77   this->v[x][y][z].a = this->a;
78 }
```

void Sculptor::setColor (float *r*, float *g*, float *b*, float *a*)

O método setColor define as cores ao desenho.

```
65                                                          { // metodo setColor
66   this->r = r;
67   this->g = g;
68   this->b = b;
69   this->a = a;
70 }
```

void Sculptor::writeOFF (const char * *filename*)

```
164                                             {
165   // abrir o fluxo de saido p o arquivo
166   int nvertices, nfaces;
167   int vOn = 0;
```

```
168
169   for (int i = 0; i < nx; i++) { // laço triplamente alinhado
170     for (int j = 0; j < ny; j++) {
171       for (int k = 0; k < nz; k++) {
172         if (v[i][j][k].show == true) {
173           vOn++;
174         }
175       }
176     }
177   }
178   std::ofstream arquivo_off;
179   arquivo_off.open(filename);
180
181   if (!arquivo off.is open()) {
182     cout << "Erro ao abrir o arquivo " << endl;
183   }
184   arquivo_off << "OFF" << std::endl;
185   arquivo_off << (vOn * 8) << " " << (vOn * 6) << " " << 0 << " " << std::endl;
186
187   // explicar esse laço
188   for (int i = 0; i < nx; i++) {
189     for (int j = 0; j < ny; j++) {
190       for (int k = 0; k < nz; k++) {
191         if (v[i][j][k].show == true) {
192           arquivo off
193               << i - 0.5 << " " << j + 0.5 << " " << k - 0.5 << std::endl
194               << i - 0.5 << " " << j - 0.5 << " " << k - 0.5 << std::endl
195               << i + 0.5 << " " << j + 0.5 << " " << k - 0.5 << std::endl
196               << i + 0.5 << " " << j + 0.5 << " " << k - 0.5 << std::endl
197               << i - 0.5 << " " << j + 0.5 << " " << k + 0.5 << std::endl
198               << i - 0.5 << " " << j - 0.5 << " " << k + 0.5 << std::endl
199               << i + 0.5 << " " << j - 0.5 << " " << k + 0.5 << std::endl
200               << i + 0.5 << " " << j + 0.5 << " " << k + 0.5 << std::endl;
201         }
202       }
203     }
204   }
205
206   int contadorFace = 0;
207
208   // explicar esse laço
209   for (int i = 0; i < nx; i++) {
210     for (int j = 0; j < ny; j++) {
211       for (int k = 0; k < nz; k++) {
212         if (v[i][j][k].show == true) {
213           arquivo_off << "4"
214                       << " " << contadorFace + 0 << " " << contadorFace + 3
215                       << " " << contadorFace + 2 << " " << contadorFace + 1
216                       << " " << std::fixed << std::setprecision(1)
217                       << v[i][j][k].r << " " << v[i][j][k].g << " "
218                       << v[i][j][k].b << " " << v[i][j][k].a << "\n"
219                       << "4"
220                       << " " << contadorFace + 4 << " " << contadorFace + 5
221                       << " " << contadorFace + 6 << " " << contadorFace + 7
222                       << " " << v[i][j][k].r << " " << v[i][j][k].g << " "
223                       << v[i][j][k].b << " " << v[i][j][k].a << "\n"
224                       << "4"
225                       << " " << contadorFace + 0 << " " << contadorFace + 1
226                       << " " << contadorFace + 5 << " " << contadorFace + 4
227                       << " " << v[i][j][k].r << " " << v[i][j][k].g << " "
228                       << v[i][j][k].b << " " << v[i][j][k].a << "\n"
229                       << "4"
230                       << " " << contadorFace + 0 << " " << contadorFace + 4
231                       << " " << contadorFace + 7 << " " << contadorFace + 3
232                       << " " << v[i][j][k].r << " " << v[i][j][k].g << " "
233                       << v[i][j][k].b << " " << v[i][j][k].a << "\n"
234                       << "4"
235                       << " " << contadorFace + 7 << " " << contadorFace + 6
236                       << " " << contadorFace + 2 << " " << contadorFace + 3
237                       << " " << v[i][j][k].r << " " << v[i][j][k].g << " "
238                       << v[i][j][k].b << " " << v[i][j][k].a << "\n"
239                       << "4"
240                       << " " << contadorFace + 1 << " " << contadorFace + 2
241                       << " " << contadorFace + 6 << " " << contadorFace + 5
242                       << " " << v[i][j][k].r << " " << v[i][j][k].g << " "
243                       << v[i][j][k].b << " " << v[i][j][k].a << std::endl;
244           contadorFace = contadorFace + 8;
```

```
245         }
246       }
247     }
248   }
249
250   arquivo_off.close();
251 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- C:/Users/Lucila/Downloads/EscultorDigital/**sculptor.h**
- C:/Users/Lucila/Downloads/EscultorDigital/**sculptor.cpp**

Referência da Estrutura Voxel

`#include <sculptor.h>`

Atributos Públicos

- float **r**
- float **g**
- float **b**
- float **a**
- bool **show**

Atributos

float Voxel::a

float Voxel::b

float Voxel::g

float Voxel::r

bool Voxel::show

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- C:/Users/Lucila/Downloads/EscultorDigital/**sculptor.h**

Arquivos

Referência do Arquivo C:/Users/Lucila/Downloads/EscultorDigital/main.cpp

```
#include "sculptor.h"
#include <iostream>
```

Funções

- int **main** ()

Funções

int main ()

Na main, o **Sculptor** implementa as classes e funções presentes no código deste projeto.

```
4           {
5   /*Sculptor write(15,30,12);
6   write.setColor(1, 1, 0, 1);
7   write.putEllipsoid(3, 3, 3, 3, 3, 3);
8   write.putBox(0, 3, 19, 4, 8, 10);*/
9
10   int nx = 21;
11   int ny = 22;
12   int nz = 15;
13
19   Sculptor sculptor(nx, ny, nz);
20   float alpha = 1.0;
21   sculptor.setColor(1, 1, 0, alpha); // cor amarela no desenho
22   sculptor.putBox(0, 20, 0, 21, 0,
23                   14); // coloca uma caixa com dimensões 21 nx 22 ny 15 nz
24   sculptor.cutBox(0, 0, 0, 7, 0, 14); // o ultimo par eh o nz
25   sculptor.cutBox(1, 1, 0, 5, 0, 14); // add um a mais do que esta
26   sculptor.cutBox(2, 2, 0, 3, 0, 14);
27   sculptor.cutBox(3, 3, 0, 2, 0, 14);
28   sculptor.cutBox(4, 4, 0, 1, 0, 14);
29   sculptor.cutBox(5, 5, 0, 1, 0, 14);
30   sculptor.cutBox(6, 7, 0, 0, 0, 14);
31   sculptor.cutBox(14, 15, 0, 0, 0, 14);
32   sculptor.cutBox(16, 20, 0, 1, 0, 14);
33   sculptor.cutBox(18, 20, 0, 2, 0, 14);
34   sculptor.cutBox(19, 20, 0, 3, 0, 14);
35   sculptor.cutBox(12, 20, 9, 9, 0, 14);
36   sculptor.cutBox(13, 20, 8, 8, 0, 14);
37   sculptor.cutBox(15, 20, 7, 7, 0, 14);
38   sculptor.cutBox(17, 20, 6, 6, 0, 14);
39   sculptor.cutBox(19, 20, 5, 5, 0, 14);
40   sculptor.cutBox(14, 20, 10, 10, 0, 14);
41   sculptor.cutBox(16, 20, 11, 11, 0, 14);
42   sculptor.cutBox(18, 20, 12, 12, 0, 14);
43   sculptor.cutBox(20, 20, 13, 13, 0, 14);
44   sculptor.cutBox(20, 20, 4, 4, 0, 14);
45   sculptor.cutBox(0, 0, 14, 21, 0, 14);
46   sculptor.cutBox(1, 1, 16, 21, 0, 14);
47   sculptor.cutBox(2, 2, 18, 21, 0, 14);
48   sculptor.cutBox(3, 3, 19, 21, 0, 14);
49   sculptor.cutBox(4, 5, 20, 21, 0, 14);
50   sculptor.cutBox(6, 7, 21, 21, 0, 14);
51   sculptor.cutBox(14, 15, 21, 21, 0, 14);
52   sculptor.cutBox(16, 17, 20, 21, 0, 14);
53   sculptor.cutBox(18, 18, 19, 21, 0, 14);
54   sculptor.cutBox(19, 19, 18, 21, 0, 14);
55   sculptor.cutBox(20, 20, 16, 21, 0, 14);
56
57   sculptor.setColor(0, 0, 0, alpha);       // cor do olho
```

```
58    sculptor.putBox(12, 13, 14, 15, 14, 14); // caixa do olho
59
60    sculptor.writeOFF("write.off");
61
62    // std::cout << "desenho concluido" << std::endl;
63
64    return 0;
65 }
```

Referência do Arquivo C:/Users/Lucila/Downloads/EscultorDigital/sculptor.cpp

```
#include "sculptor.h"
#include <cmath>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <stdlib.h>
#include <string>
```

Referência do Arquivo C:/Users/Lucila/Downloads/EscultorDigital/sculptor.h

Componentes

- struct **Voxel**class **Sculptor**
  *A classe **Sculptor** implementa os métodos para a modelagem do desenho.*

sculptor.h

Ir para a documentação desse arquivo.

```
1 #ifndef SCULPTOR_H
2 #define SCULPTOR_H
3
4 struct Voxel {
5   float r, g, b; // coloração
6   float a;       // transparencia
7   bool show;     // objeto incluido ou nao
8 };
9
14 class Sculptor {
15 private:
16   Voxel ***v;                          // 3D matrix
17   int nx, ny, nz;                      // dimencoes
18   float r, g, b, a;                    // cor e transparencia
19 public:                                // metodos:
20   Sculptor(int  nx, int  ny, int  nz); // construtor - alocacao dinamica
21   ~Sculptor();                         // destrutor
22   void setColor(float r, float g, float b, float a); // escolha da cor - ok
23   void putVoxel(int x, int y, int z);               // criar quadrado - ok
24   void cutVoxel(int x, int y, int z);               // deletar quadrado - ok
25   void putBox(int x0, int x1, int y0, int y1, int z0, int z1); // ok
26   void cutBox(int x0, int x1, int y0, int y1, int z0, int z1); // ok
27   void putSphere(int xcenter, int ycenter, int zcenter,
28                  int radius); // cria esfera
29   void cutSphere(int xcenter, int ycenter, int zcenter,
30                  int radius); // apaga esfera
31   void putEllipsoid(int xcenter, int ycenter, int zcenter, int rx, int ry,
32                     int rz); // cria elipse
33   void cutEllipsoid(int xcenter, int ycenter, int zcenter, int rx, int ry,
34                     int rz);           // apaga elipse
35   void writeOFF(const char *filename); // cria arquivo.off - ok
36 };
37
38 #endif
39
40 // Fazer ate a parte 4.1: Criação de um programa de testes
```

Sumário

INDEX