Documentação Escultor Digital

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 tal tal tal.
`#include <sculptor.h>`

Membros Públicos

- **Sculptor** (int _nx, int _ny, int _nz)
- **~Sculptor** ()
- void **setColor** (float r, float g, float b, float a)
- 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)
- 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 tal tal tal.

Teste de detalhes

Construtores e Destrutores

Sculptor::Sculptor (int _*nx*, int _*ny*, int _*nz*)

```
20                                              { // construtor
21   // Incializacao das matrizes de voxel
22   this->nx =  nx;
23   this->ny =  ny;
24   this->nz =  nz;
25   this->r = 0;
26   this->g = 0;
27   this->b = 0;
28   this->a = 0;
29
30   // Alocacao dinamica
31   v = new Voxel **[nx]; // Alocar matriz 3D
32   for (int i = 0; i < nx; i++) {
33     v[i] = new Voxel *[ny];
34     for (int j = 0; j < ny; j++) {
35       v[i][j] = new Voxel[nz];
36     }
37   }
38
39   for (int i = 0; i < nx; i++) {
40     for (int j = 0; j < ny; j++) {
41       for (int k = 0; k < nz; k++) {
42         v[i][j][k].show = false;
43         v[i][j][k].r = 0;
44         v[i][j][k].g = 0;
45         v[i][j][k].b = 0;
46       }
47     }
48   }
```

```
49 }
```

Sculptor::~Sculptor ()

```
51                     { // destrutor - liberar memoria
52   for (int i = 0; i < nx; i++) {
53     for (int j = 0; j < ny; j++) {
54       delete[] v[i][j];
55     }
56   }
57   for (int i = 0; i < nx; i++) {
58     delete[] v[i];
59   }
60   delete[] v;
61 }
```

Documentação das funções

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

```
92                                                                          {
93   for (int i = x0; i <= x1; i++) {
94     for (int j = y0; j <= y1; j++) {
95       for (int k = z0; k <= z1; k++) {
96         cutVoxel(i, j, k);
97       }
98     }
99   }
100 }
```

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

```
144
{
145   for (int i = 0; i < nx; i++) {
146     for (int j = 0; j < ny; j++) {
147       for (int k = 0; k < nz; k++) {
148         if (((pow((i - xcenter), 2) / (pow(rx, 2))) +
149              (pow((j - ycenter), 2) / (pow(ry, 2))) +
150              (pow((k - zcenter), 2) / (pow(rz, 2)))) <= 1) {
151           cutVoxel(i, j, k);
152         }
153       }
154     }
155   }
156 }
```

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

```
116                                                                         {
117   for (int i = 0; i < nx; i++) {
118     for (int j = 0; j < ny; j++) {
119       for (int k = 0; k < nz; k++) {
120         if (((pow((i - xcenter), 2) / (pow(radius, 2))) +
121              (pow((j - ycenter), 2) / (pow(radius, 2))) +
122              (pow((k - zcenter), 2) / (pow(radius, 2)))) <= 1) {
123           cutVoxel(i, j, k);
124         }
125       }
126     }
127   }
128 }
```

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

```
78                                              {
79   v[x][y][z].show = false;
80 }
```

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

```
82                                                                          {
83   for (int i = x0; i <= x1; i++) {
```

```
84      for (int j = y0; j <= y1; j++) {
85        for (int k = z0; k <= z1; k++) {
86          putVoxel(i, j, k);
87        }
88      }
89    }
90 }
```

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

```
130
{
131    for (int i = 0; i < nx; i++) {
132      for (int j = 0; j < ny; j++) {
133        for (int k = 0; k < nz; k++) {
134          if (((pow((i - xcenter), 2) / (pow(rx, 2))) +
135               (pow((j - ycenter), 2) / (pow(ry, 2))) +
136               (pow((k - zcenter), 2) / (pow(rz, 2)))) <= 1) {
137            putVoxel(i, j, k);
138          }
139        }
140      }
141    }
142 }
```

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

```
102                                                                                 {
103    for (int i = 0; i < nx; i++) {
104      for (int j = 0; j < ny; j++) {
105        for (int k = 0; k < nz; k++) {
106          if (((pow((i - xcenter), 2) / (pow(radius, 2))) +
107               (pow((j - ycenter), 2) / (pow(radius, 2))) +
108               (pow((k - zcenter), 2) / (pow(radius, 2)))) <= 1) {
109            putVoxel(i, j, k);
110          }
111        }
112      }
113    }
114 }
```

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

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

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

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

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

```
158                                             {
159    // abrir o fluxo de saido p o arquivo
160
161    // varrer a matriz v
162    int nvertices, nfaces;
163    int vOn = 0;
164
165    for (int i = 0; i < nx; i++) {  // laço triplamente alinhado
166      for (int j = 0; j < ny; j++) {
167        for (int k = 0; k < nz; k++) {
168          if (v[i][j][k].show == true) {
169            vOn++;
170          }
171        }
```

```
172     }
173   }
174   std::ofstream arquivo off;
175   arquivo off.open(filename);
176
177   if (!arquivo_off.is_open()) {
178     cout << "Erro ao abrir o arquivo " << endl;
179   }
180   arquivo off << "OFF" << std::endl;
181   arquivo off << (vOn * 8) << " "
182               << (vOn * 6) << " " << 0 << " "
183               << std::endl;
184
185 //explicar esse laço
186   for (int i = 0; i < nx; i++) {
187     for (int j = 0; j < ny; j++) {
188       for (int k = 0; k < nz; k++) {
189         if (v[i][j][k].show == true) {
190           arquivo off << i - 0.5 << " " << j + 0.5 << " " << k - 0.5 << std::endl
191                       << i - 0.5 << " " << j - 0.5 << " " << k - 0.5 << std::endl
192                       << i + 0.5 << " " << j + 0.5 << " " << k - 0.5 << std::endl
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         }
199       }
200     }
201   }
202
203   int contadorFace = 0;
204
205   // explicar esse laço
206   for (int i = 0; i < nx; i++) {
207     for (int j = 0; j < ny; j++) {
208       for (int k = 0; k < nz; k++) {
209         if (v[i][j][k].show == true) {
210           arquivo_off << "4"
211                       << " " << contadorFace + 0 << " "
212                       << contadorFace + 3 << " " << contadorFace + 2
213                       << " " << contadorFace + 1 << " " << std::fixed
214                       << std::setprecision(1) << v[i][j][k].r << " "
215                       << v[i][j][k].g << " " << v[i][j][k].b << " "
216                       << v[i][j][k].a << "\n"
217                       << "4"
218                       << " " << contadorFace + 4 << " "
219                       << contadorFace + 5 << " " << contadorFace + 6
220                       << " " << contadorFace + 7 << " " << v[i][j][k].r
221                       << " " << v[i][j][k].g << " " << v[i][j][k].b << " "
222                       << v[i][j][k].a << "\n"
223                       << "4"
224                       << " " << contadorFace + 0 << " "
225                       << contadorFace + 1 << " " << contadorFace + 5
226                       << " " << contadorFace + 4 << " " << v[i][j][k].r
227                       << " " << v[i][j][k].g << " " << v[i][j][k].b << " "
228                       << v[i][j][k].a << "\n"
229                       << "4"
230                       << " " << contadorFace + 0 << " "
231                       << contadorFace + 4 << " " << contadorFace + 7
232                       << " " << contadorFace + 3 << " " << v[i][j][k].r
233                       << " " << v[i][j][k].g << " " << v[i][j][k].b << " "
234                       << v[i][j][k].a << "\n"
235                       << "4"
236                       << " " << contadorFace + 7 << " "
237                       << contadorFace + 6 << " " << contadorFace + 2
238                       << " " << contadorFace + 3 << " " << v[i][j][k].r
239                       << " " << v[i][j][k].g << " " << v[i][j][k].b << " "
240                       << v[i][j][k].a << "\n"
241                       << "4"
242                       << " " << contadorFace + 1 << " "
243                       << contadorFace + 2 << " " << contadorFace + 6
244                       << " " << contadorFace + 5 << " " << v[i][j][k].r
245                       << " " << v[i][j][k].g << " " << v[i][j][k].b << " "
246                       << v[i][j][k].a << std::endl;
247           contadorFace = contadorFace + 8;
248         }
```

```
249       }
250     }
251   }
252
253   arquivo_off.close();
254 }
```

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

- C:/Users/evell/Downloads/EscultorDigitalAtualizado/**sculptor.h**
- C:/Users/evell/Downloads/EscultorDigitalAtualizado/**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/evell/Downloads/EscultorDigitalAtualizado/**sculptor.h**

Arquivos

Referência do Arquivo C:/Users/evell/Downloads/EscultorDigitalAtualizado/main.cpp

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

Funções

- int **main** ()

Funções

int main ()

A arquivo main define o desenho esculpido pelo programa

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

```
56    sculptor.putBox(12, 13, 14, 15, 20, 20); //caixa do olho
57
58    sculptor.writeOFF("write.off");
59
60    // std::cout << "desenho concluido" << std::endl;
61
62    return 0;
63 }
```

Referência do Arquivo C:/Users/evell/Downloads/EscultorDigitalAtualizado/sculptor.cpp

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

Referência do Arquivo C:/Users/evell/Downloads/EscultorDigitalAtualizado/sculptor.h

Componentes

- struct **Voxel**class **Sculptor**
  *A classe **Sculptor** implementa tal tal tal.*

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

Sumário

INDEX