sculptor3d<sub>1.0</sub> Q Busca Página Principal Classes ▼ **Arquivos** ▼ Membros Públicos | Lista de todos os Membros Referência da Classe Sculptor #include <sculptor.h> Membros Públicos **Sculptor** (int \_nx, int \_ny, int \_nz) ~Sculptor () void **setColor** (float r, float g, float b, float a) putVoxel (int x, int y, int z) cutVoxel (int x, int y, int z) putBox (int x0, int x1, int y0, int y1, int z0, int z1) cutBox (int x0, int x1, int y0, int y1, int z0, int z1) putSphere (int xcenter, int ycenter, int zcenter, int radius) cutSphere (int xcenter, int ycenter, int zcenter, int radius) putEllipsoid (int xcenter, int ycenter, int zcenter, int rx, int ry, int rz) **cutEllipsoid** (int xcenter, int ycenter, int zcenter, int rx, int ry, int rz) writeOFF (char \*filename) Construtores e Destrutores Sculptor() Sculptor::Sculptor (int \_nx, int \_ny, int \_nz 11  $nx = _nx; ny = _ny; nz = _nz;$ 12 13 r = 1; g = 1; b = 1; 14 // Alocação dinâmica da matriz 3d v = new Voxel\*\*[nx]; 15 for (int i =0; i<nx; i++){</pre> 16 17 v[i] = new Voxel\*[ny]; 18 19 for (int j =0; j<ny; j++){</pre> v[i][j]= new Voxel[nz]; 20 21 22 23 } ~Sculptor() Sculptor::~Sculptor() for(int i = 0; i < nx; i++){</pre> 26 27 for (int j = 0; j < ny; j++){ delete[] v[i][j]; 28 29 30 for (int i=0; i <nx; i++){</pre> 31 32 delete[] v[i]; 33 34 35 delete[] v; 36 //Resetando coordenadas 37 nx = 0; ny = 0; nz = 0; 38 } Funções membros cutBox() void Sculptor::cutBox (int x0, int x1, int y0, int y1, int z0, int z1 72 73 74 //Desativa uma sequencia de voxel 75 for(int x = x0; x < x1; x++){ for (int y = y0; y<y1; y++){
 for (int z = z0; z<z1; z++){</pre> 76 77 v[x][y][z].isOn = false; 78 79 80 81 82 }

## 140 141 double newx, newy, newz; 142 int x, y, z; 143 144 for $(x = 0; x < nx; x++){$ for $(y = 0; y < ny; y++){$ 145 for $(z = 0; z < nz; z++){$ 146 newx = ((float)(x-xcenter)\*(float)(x-xcenter))/(rx \* rx); newy = ((float)(y-ycenter)\*(float)(y-ycenter))/(ry \* ry); newz = ((float)(z-zcenter)\*(float)(z-zcenter))/(rz \* rz); 147 148 149 150 151 if ((newx + newy + newz) < 1){ 152 cutVoxel(x,y,z); **153** 154 155 156 } 157 • cutSphere() void Sculptor::cutSphere (int xcenter, int ycenter, int zcenter, int radius

//Mesma lógica do putSphere, mas ao invés de ativar o voxel, ele desativa

if((i\*i+j\*j+k\*k) < radius\*radius){
 cutVoxel(i+xcenter,j+ycenter,k+zcenter);</pre>

cutEllipsoid()

138 139

101 102

103

104

105

106 107

108

109

110 111 112 //Desativa a esfera

for(i = -radius; i<radius; i++){</pre>

for(j = -radius; j<radius; j++){</pre>

for(k = -radius; k<radius; k++){</pre>

int i, j, k;

void Sculptor::cutEllipsoid (int xcenter,

//Desativa a elipsoid

int ycenter,

int zcenter,

int rx,

int ry,

int rz

113 115 } cutVoxel() void Sculptor::cutVoxel (int x, int y, int z 53 54 55 } { v[x][y][z].isOn = false; • putBox() void Sculptor::putBox (int x0, int x1, int y0, int y1, int z0, int z1 57 58 //Define uma sequencia de voxel 59 for(int  $x = x0; x<x1; x++){$ for (int y = y0; y<y1;  $\hat{y}$ ++){ 60 for (int z = z0; z<z1; z++){ 61 v[x][y][z].isOn=true; v[x][y][z].r = r; 62 63 v[x][y][z].g = g; v[x][y][z].b = b; 64 65 66 67 v[x][y][z].a = a;68 69 70 } • putEllipsoid() void Sculptor::putEllipsoid (int xcenter, int ycenter, int zcenter, int rx,

int ry, int rz 117 118 //Define uma elipsoide 119 //recebe as coordenadas das dimensões e coordenadas do centro 120 float newx, newy, newz; 121 int x, y, z; 122 123 for  $(x = 0; x < nx; x++){$ for  $(y = 0; y < ny; y++){}$ 124 125 for  $(z = 0; z < nz; z++){$ newx = ((float)(x-xcenter)\*(float)(x-xcenter))/(rx \* rx);
newy = ((float)(y-ycenter)\*(float)(y-ycenter))/(ry \* ry);
newz = ((float)(z-zcenter)\*(float)(z-zcenter))/(rz \* rz); 126 127 128 //calculo da condição de existencia da elipsoid 129 130 if ((newx + newy + newz) < 1)131 putVoxel(x,y,z); 132 133 134 135 136 } • putSphere() void Sculptor::putSphere (int xcenter, int ycenter, int zcenter, int radius //Recebe as coordenadas e o raio da esfera 85 86 87 int i, j, k; 88 89 for( i = -radius; i<=radius; i++){ //pecorre o diametro</pre> for(j = -radius; j<=radius; j++){</pre> 90 91 for(k = -radius; k<=radius; k++){ 92 93 if ((i\*i+j\*j+k\*k) < radius\*radius){ //condição da esfera</pre> 94 putVoxel(i+xcenter,j+ycenter,k+zcenter); //adiciona um voxel 95 96 97 98 99 } • putVoxel() void Sculptor::putVoxel (int x,

int y, int z 45 46 //insere um voxel v[x][y][z].isOn = true;47 48 //repassa cor e transparencia do voxel v[x][y][z].r = r; v[x][y][z].g = g; v[x][y][z].b = b; v[x][y][z].a = a;49 50 51 } setColor() void Sculptor::setColor (float r, float g, float b, float a 40 41 //O usuário define as propriedades do voxel Sculptor::r = r; Sculptor::g = g; Sculptor::b = b; Sculptor::a = a; 42 43 writeOFF() void Sculptor::writeOFF ( char \* filename ) 159 160 int qtd\_Voxel = 0; int ref; 161 int i, j, k, a, b, c; //auxiliares 162 float fix = 0.5; 163 std::ofstream endArq; 164 endArq.open(filename); 165 if (endArq.is\_open()){ 166 167 endArq<<"OFF\n"; //Define off na primeira linha 168 169 // Percorre todas as dimensoes verificando os voxel em exibição 170 for  $(i = 0; i < nx; i++){}$ for (j = 0; j <ny; j++){ for (k = 0; k <nz; k++){ 171 172 **173 if**(v[i][j][k].isOn == true){ qtd\_Voxel++; //Guarda a quantidade de voxel ativos 174 175 176 177 } 178 179 endArq<<qtd\_Voxel \* 8<<" "<<qtd\_Voxel \* 6 << " " << "0" << "\n";</pre> 180 // exibe a quantidade total de vertices, faces e arestas 181 182 //Pecorre as posições a, b, c e exibe as coordenadas do voxel 183 //No primeiro voxel o fix será o valor de referencia 184 for  $(a = 0; a < nx; a++){}$ 185 for  $(b = 0; b < ny; b++){}$ 186 for  $(c = 0; c < nz; c++){}$ if(v[a][b][c].isOn == true){
 endArq << a - fix << " " << b + fix << " " << c - fix << "\n" << flush;
 endArq << a - fix << " " << b - fix << " " << c - fix << "\n" << flush;</pre> 187 188 189 endArq << a + fix << " " << b - fix << " " << c - fix << "\n" << flush; endArq << a + fix << " " << b + fix << " " << c - fix << "\n" << flush; endArq << a - fix << " " << b + fix << " " << c - fix << "\n" << flush; endArq << a - fix << " " << b + fix << " " << c + fix << "\n" << flush; 190 191 192 endArq << a - fix << " " << b - fix << " " << c + fix << "\n" << flush; 193 endArq << a + fix << " " << b - fix << " " << c + fix << "\n" << flush; endArq << a + fix << " " << b + fix << " " << c + fix << "\n" << flush; 194 195 196 197 } 198 } 199 200 201 qtd\_Voxel = 0; //Reseta a quantidade de voxel 202 203 // percorre todas as dimensoes 204 // Verifica os voxel ativos e exibe as linhas com numeros de faces e as combinacoes que exibem a face 205 for (a= 0; a<nx; a++){ 206 for  $(b = 0; b < ny; b++){}$ 207 for (c= 0; c<nz; c++){ 208  $if(v[a][b][c].isOn == true){$ 209 ref = qtd\_Voxel \* 8; 210 endArq << fixed;</pre> 211 212 // Construir linha que monta as faces a partir do vertices: 213 //Valores exemplos do site quando ref = 0: 214 215 endArq << "4" << " " << 0+ref << " " << 3+ref << " " << 2+ref << " " << 1+ref << " "; //exibindo propriedades do voxel: 216 endArq << setprecision(2)<v[a][b][c].r << " " << setprecision(2)<<math>v[a][b][c].g << " " << setprecision(2)<<math>v[a][b][c].b << " " << setprecision(2) << | colored | colov[a][b][c].a << "\n"; 217 218 endArq << "4" << " " << 4+ref << " " << 5+ref << " " << 6+ref << " " << 7+ref << " "; endArq << setprecision(2)<<v[a][b][c].r << " " << setprecision(2)<<v[a][b][c].g <<" " << setprecision(2)<<v[a][b][c].b << " " << setprecision(2) << 219 v[a][b][c].a << "\n"; 220 221 endArq << "4" << " " << 0+ref << " " << 1+ref << " " << 5+ref << " " << 4+ref << " "; endArq << setprecision(2)<<v[a][b][c].r << " " << setprecision(2) <<v[a][b][c].b << " " << setprecision(2) << 222 v[a][b][c].a << "\n"; 223 endArq << "4" << " " << 0+ref << " " << 4+ref << " " << 7+ref << " " << 3+ref << " "; 224 endArq << setprecision(2)<<v[a][b][c].r << " " << setprecision(2) <<v[a][b][c].b << " " << setprecision(2) << 225 v[a][b][c].a << "\n"; 226 endArq << "4" << " " << 3+ref << " " << 7+ref << " " << 6+ref << " " << 2+ref << " "; endArq << setprecision(2)<<v[a][b][c].r << " " << setprecision(2)<<v[a][b][c].g <<" " << setprecision(2)<<v[a][b][c].b << " " << setprecision(2) << " " < setprecision(2) << " 227 228 v[a][b][c].a << "\n"; 229 endArg << "4" << " " << 1+ref << " " << 2+ref << " " << 6+ref << " " << 5+ref << " "; 230 endArq << setprecision(2)<<v[a][b][c].r << " " << setprecision(2)<<v[a][b][c].g << " " << setprecision(2)<<v[a][b][c].b << " " << setprecision(2) << 231 v[a][b][c].a << "\n"; 232 233 qtd\_Voxel++; //incrementa a qtd de voxels 234 235 236 237 238 239 cout << "Erro ao abrir arquivo de texto.";</pre> 240 241 242 243 endArq.close(); 244 } A documentação para essa classe foi gerada a partir dos seguintes arquivos: • C:/Users/wesle/Downloads/Sculptor3D\_PA\_Cpp-main/sculptor.h • C:/Users/wesle/Downloads/Sculptor3D\_PA\_Cpp-main/sculptor.cpp