



# **Relatório Técnico**

Version 1.0

Gustavo Lopes Rodrigues, Lucas Santiago de Oliveira

25 de novembro de 2021

# Conteúdo

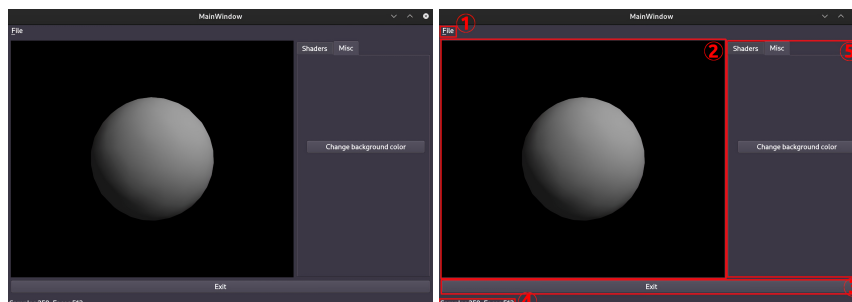
<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Qt</b>	<b>2</b>
2.1	File . . . . .	2
2.2	Visualizador de Objetos 3D . . . . .	4
2.3	Botão de saída . . . . .	5
2.4	Barra de status . . . . .	5
2.5	Abas de edição . . . . .	6
<b>3</b>	<b>Código</b>	<b>7</b>
3.1	glwidget.cpp . . . . .	7
3.1.1	Nova cor para o fundo do QGLWidget . . . . .	7
3.1.2	Gerenciamento de arquivos . . . . .	7
3.1.3	Escolhendo shader . . . . .	8
3.1.4	Entrada do usuário . . . . .	8
3.2	light.cpp . . . . .	8
3.3	material.cpp . . . . .	8
3.4	camera.cpp e trackball.cpp . . . . .	9
<b>4</b>	<b>Módelos matemáticos</b>	<b>10</b>
4.1	Operações geométricas . . . . .	10
4.2	Shaders . . . . .	10
4.2.1	Geração de tangentes . . . . .	10
4.2.2	Phong: . . . . .	10
4.2.3	Gouraud: . . . . .	10
4.2.4	Combinação linear: Specular, Ambient e Diffuse . . . . .	10

# 1 Introdução

Esse programa feito para a disciplina de Computação gráfica tem como objetivo integrar C++, Qt e OpenGL para fazer uma aplicação que seja capaz de abrir arquivos .off (Object File Format) e manipulá-los. O programa, em grande parte, consiste na implementação do artigo *"Interactive Graphics Applications with OpenGL Shading Language and Qt"* com apenas algumas melhoras a sua interface e a adição da leitura de malha mista como uma nova funcionalidade.

Utilizamos C++ como a linguagem de programação, o Qt como o framework para construir a interface visual no qual o usuário interage e o OpenGL é a API gráfica que renderiza os objetos, processa shaders e texturas. As seções a seguir serão separadas da seguinte forma: primeiro apresentaremos a comunicação entre a interface visual do Qt e o código para em seguida apresentarmos o código em si e sua estruturação.

## 2 Qt



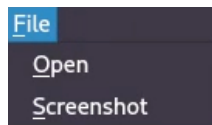
**Figura 1:** À esquerda temos a principal interface do programa e ao lado temos a mesma interface com marcações para sinalizar os principais componentes.

A área de interação do Qt é composta de Widgets, desde os botões, até o canvas do OpenGL, todos os elementos são Widgets que podem ser dimensionados na tela e mais importante, podem usar sinais para comunicar entre-si.

### 2.1 File

Primeiro temos o QMenuBar responsável em expandir e mostrar as duas opções presentes:

- **Open** - O primeiro QMenu possui um QAction, um sinal conectado com o visualizador dos objetos (QGLWidget). Quando acionado (triggered) realiza a ação de abrir o gerenciador de arquivos e abrir um arquivo .off.
- **Screenshot** - O outro QMenu possui um QAction que conecta ao QGLWidget. Quando acionado (triggered) realiza a ação de abrir o gerenciador de arquivos e permite o usuário salvar o frame atual do QGLWidget como uma imagem png.



**Figura 2:** O botão **File** expandido

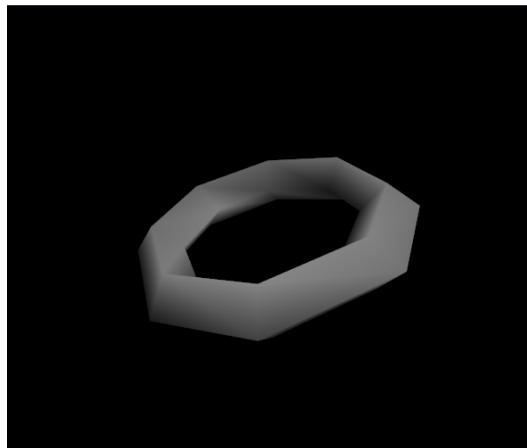
Além disso, o QMenuBar possui **Keybinds**, atalhos do teclado do usuário, que permitem que todas as interações com o QMenuBar sejam mais rápidas. Eis a seguir os keybinds implementados:

- **Alt + f** → clicar no botão *File*
  - **Alt + f + o** → abrir um arquivo
  - **Alt + f + s** → salvar uma captura de tela

## 2.2 Visualizador de Objetos 3D

O principal componente da aplicação, a janela onde os objetos 3D são exibidos é um `QGLWidget`, em que há grande parte da implementação da API do OpenGL.

Esta janela ilustra o resultado final do processamento dos objetos, com texturas e shaders, se necessário. Além de capturar as teclas do teclado e o movimento do mouse.



**Figura 3:** Objeto `octorus.off` sendo exibido na janela

As possíveis entradas de teclado na aplicação são:

### Teclado:

- **Número 1 do teclado** → aplicar o algoritmo de *Gouraud*
- **Número 2 do teclado** → aplicar o algoritmo de *Phong*
- **Número 3 do teclado** → aplicar uma textura
- **Número 4 do teclado** → aplicar uma textura com normal
- **Esc** → fechar a aplicação

### Mouse:

- **Scroll** - Realizar ampliação/distanciamento (zoom) no objeto atualmente selecionado.
- **Botão direito + movimentar o mouse** - Rotacionar o objeto.

## 2.3 Botão de saída

QPushButton é um widget que possui um sinal ligado a janela principal do programa, ao ser clicado fechará a aplicação.



**Figura 4:** Botão para sair do programa

## 2.4 Barra de status

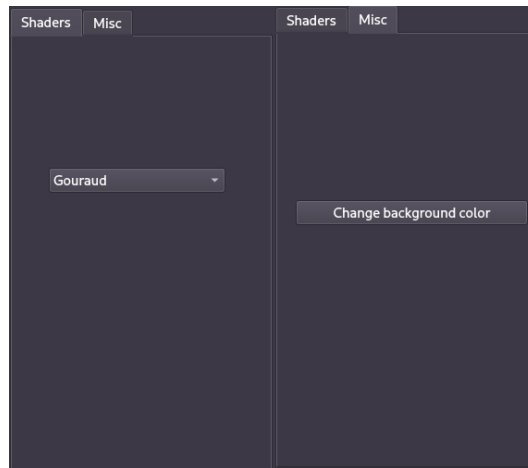
A barra de status fica no canto inferior da tela informando usuários de quantos *samples* e *faces* foram carregadas do objeto 3D. Esse possui um sinal ligado ao QGLWidget, que recebe uma QString com as informações mencionadas.



Samples 32, Faces 66

**Figura 5:** Barra de status com informações do **octtorus.off**

## 2.5 Abas de edição



**Figura 6:** As duas abas de edição

Nessas abas, o usuário pode escolher entre qual dos shader será aplicado no objeto 3D e qual a cor do fundo atrás do objeto. Na primeira, ao selecionarmos um algoritmo o objeto que tiver sido selecionado anteriormente, será redesenhado na tela com o novo algoritmo. No segundo, a cor escolhida será aplicada no fundo da imagem, não modificando o objeto que está sendo exibido (esses algoritmos não levam em conta a cor do ambiente).

- **Shaders** - Possui um QComboBox, que apresenta vários textos com nome dos possíveis shaders do programa. Após selecionar um, será acionado um sinal que será enviado para o QGLWidget pedindo para mudar para o shader especificado pelo usuário.
- **Misc** - Possui um QPushButton, que acionará a paleta de cor do Qt, permitindo que o usuário escolha uma cor para ser a nova cor do fundo (mais informações na seção do código).

## 3 Código

Agora que a comunicação na interface do Qt foi detalhada, podemos passar a explicar a comunicação dentro do código em si. Como dito anteriormente, o maior componente dentro do projeto que possui a grande porção do código é o `QGLWidget`. Ademais, temos a *main.cpp*, que inicia a aplicação e inclusive inicia a *mainwindow.cpp*, que é a janela principal, onde está o `QGLWidget`.

### 3.1 `glwidget.cpp`

Esse é o código referente a tela do `QGLWidget`, encapsulando todas as operações referentes ao OpenGL. Além de tratar operações recebidas do usuário na interface. O `glwidget` é o arquivo mais importante, pois além de ser responsável em comunicar com a API gráfica, todas as outras classes são orquestradas a partir dessa: a *camera.cpp* a *light.cpp*, o *material.cpp* e o *trackball.cpp*.

Neste momento não seria intuitivo falar sobre todas as funções, pois além de estarem comentadas, entraremos em mais detalhes sobre o funcionamento das principais funções quando falarmos sobre os modelos matemáticos. Então, agora iremos completar a discussão sobre a interação do Qt, percorrendo como o código faz o tratamento dos pedidos do usuário.

#### 3.1.1 Nova cor para o fundo do `QGLWidget`

Quando o usuário pressiona para escolher uma nova cor para o plano de fundo do visualizador, a função *chooseBackgroundColor()* é chamada. Ela é responsável em abrir o *QColorDialog*, um Widget já implementado dentro da framework, que possui a função *getColor()* permitindo chamar a paleta de cores para que o usuário faça uma escolha. Logo após isso, o programa apenas valida a cor e se for válida ela será definida como a nova cor de fundo.

#### 3.1.2 Gerenciamento de arquivos

Ao pedir para abrir um novo arquivo, a função *showFileOpenDialog()* é chamada para fazer essa tarefa. Esta função pode ser dividida em duas partes: a primeira é abrir o gerenciador de arquivos, usando o *QFileDialog*, para deixar o usuário escolher um objeto. Caso um objeto válido tenha sido escolhido, o programa então agora irá ler e processar o arquivo para gerar o objeto na tela.



Essa lógica é aplicada da mesma maneira quando é salvo uma imagem do frame atual do OpenGL usando a função *takeScreenshot()*. Nesta função, o *QFileDialog* também é chamado, porém, em vez de abrir um arquivo, ele salva no diretório escolhido pelo usuário.

### 3.1.3 Escolhendo shader

Existe duas maneiras que o usuário pode escolher os shaders dentro da interface. A primeira é usando os keybinds no teclado número, apertando qualquer um dos números de um a quatro, faz com que o OpenGL atualize os shaders, através da função *keyPressEvent ( QKeyEvent \*event )*. Entretanto, o usuário também pode atualizar os shaders usando a *QComboBox* mencionada na seção sobre a aba de edição. Neste caso, a função que será chamada é a *changeShader(const QString shaderName)*, que irá trocar o shader de acordo com o nome enviado para ela.

### 3.1.4 Entrada do usuário

A primeira função é o *keyPressEvent ( QKeyEvent \* event )*. Como o *GLWidget* foi programado para ter uma *focusPolicy* forte, então todos os eventos do teclado são capturados pela função mencionada, e então tratados para fazer os keybinds já mencionados.

Além de fazer leitura do teclado, o programa também faz leitura do mouse, ele reconhece entradas no botão esquerdo do mouse (quando o objeto quer ser rotacionado) e também o scroll do mouse quando o usuário quer fazer uma operação de ampliação. As funções responsáveis para interagir com as entradas do mouse são: *GLWidget :: mouseMoveEvent (QMouseEvent \*event)*, *void GLWidget :: mousePressEvent (QMouseEvent \*event)*, *void GLWidget :: mouseReleaseEvent (QMouseEvent \*event)*, *void GLWidget :: wheelEvent (QWheelEvent \*event)*.

## 3.2 light.cpp

Classe para gerenciar as luzes dentro do programa, o *glwidget.cpp* possui uma instância do objeto, que será usada durante o processo de "desenhar" a tela *paintGL()*, para calcular a iluminação.

## 3.3 material.cpp

Usado em junção com o *light.cpp*, o *Material* é uma classe para descrever as propriedades de um objeto quando for refletido pela luz. Assim como ela,

todos esses cálculos são feitos na hora de chamar a função *paintGL()*.

### **3.4 camera.cpp e trackball.cpp**

Por fim, para rotacionar e ampliar o objeto 3D, todos os cálculos são feitos em cima das classes Camera e Trackball. Essas classes também são chamadas quando o OpenGL altera o tamanho da janela da aplicação.

## 4 Modelos matemáticos

### 4.1 Operações geométricas

Operação de rotação em um ponto fixo:

$$R(0) = R_z(0_z) * R_y(0_y) * R_x(0_x)$$

### 4.2 Shaders

#### 4.2.1 Geração de tangentes

$$\begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix} = \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix}$$

#### 4.2.2 Phong:

$$I_p = k_a i_a + \sum_{m \in \text{lights}} (k_d (\hat{L}_m * \hat{N}) i_{m,d} + k_s (\hat{R}_m * \hat{V})^\alpha i_{m,s})$$

#### 4.2.3 Gouraud:

$$F_n = \frac{f_1 + f_2 + \dots + f_n}{|f_1 + f_2 + \dots + f_n|}$$

#### 4.2.4 Combinação linear: Specular, Ambient e Diffuse

$$I = I_a K_a + I_i [K_d (L * N) + K_s (R * V)^\eta]$$