UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE
LABORATÓRIO DE ENGENHARIA E EXPLORAÇÃO DE PETRÓLEO

PROJETO ENGENHARIA SOFTWARE SIMULADOR DE DIFUSÃO TÉRMICA 3D

TRABALHO DE CONCLUSÃO DE CURSO

Versão 1:
Nicholas de Almeida Pinto
Prof. André Duarte Bueno
Prof. Guilherme Rodrigues Lima

MACAÉ - RJ
Novembro - 2021

Sumário

Capítulo 1

Introdução

No presente projeto de engenharia, desenvolveu-se o software Simulador de difusão térmica em objetos 3D, com paradigma orientado ao objeto, com o objetivo de implementar conceitos aprendidos nas disciplinas de fenômeno dos transportes, modelagem numérica e programação.

Dessa forma, a principal finalidade do simulador é fornecer o cálculo da temperatura ao longo do tempo, em um objeto genérico, com propriedades térmofísicas também inseridas pelo usuário. Tornando-se uma ferramenta poderosa para o ensino de transferência de calor, calculo numérico, programação orientada ao objeto, programação de multi-threads, e para o desenvolvimento de projetos de engenharia.

1.1 Escopo do problema

Troca de calor é um tema importantíssimo na indústria do Petróleo, sendo estudado e aplicado em absolutamente todas as etapas. Geólogos estudam a maturidade do óleo, engenheiros de reservatório estudam mecanismos térmicos para produzir mais óleo, engenheiros de planta de plataforma buscam formas de minimizar a perda energética devido às trocas de calor dos fluidos produzidos, engenheiros de refinaria buscam melhores controles de temperatura nas destilarias, e engenheiros de logística, melhores materiais para transportar o óleo/gás até o consumidor.

É importantíssimo resolver os problemas de engenharia citados, para diversos casos e situações, com alta precisão. Portanto, o que se propõe é um simulador de difusão térmica, que consegue resolver todos os casos possíveis, para qualquer temperatura, superfícies ou volumes, e para qualquer material. Tornando-se uma ferramenta prática para alunos e engenheiros.

1.2 Objetivos

O objetivo deste projeto são:

- Objetivo geral:
  - desenvolver um software capaz de simular a transferência de calor em qualquer superfície ou objeto, para qualquer material.
  - facilitar o entendimento e ensinamento de transferência de calor, modelagem numérica, programação orientada ao objeto e paralelismo.
- Objetivo específico
  - Simulação da transferência de calor de qualquer superfície.
  - Programação em C++, utilizando o paradigma de orientação ao objeto, e pacotes externos, permitindo modificações e adições no código fonte disponibilizado.
  - Métodos numéricos: solução de equações diferenciais da conservação de energia por meio de diferenças finitas, e desenvolvimento de algoritmo para resolver qualquer problema de fronteira.
  - Modelagem física e matemática do problema.
  - Simulação com dados de materiais obtidos em laboratório
  - Programação com paralelismo
  - Gerar gráfico e interface de usuário com software externo.
  - Resolver métodos iterativos.

Capítulo 2

Especificação

Apresenta-se neste capítulo do projeto de engenharia a concepção, a especificação do sistema a ser modelado e desenvolvido.

2.1 Nome do sistema/produto

Nome	Simulador de difusão térmica 3D
Componentes principais	Distribuição da temperatura em um objeto, ao longo do tempo, utilizando método numérico implícito.
Missão	Calcular a temperatura em objetos.

2.2 Especificação

Deseja-se desenvolver um software com interface gráfica amigável ao usuário, onde seja possível desenhar o objeto 3D, por meio de perfis, com o usuário escolhendo a temperatura e o material. A simulação é governada pela Equação da Difusão Térmica, a qual é modelada por diferenças finitas, pelo método BTCS, com fronteiras seladas.

Na dinâmica de execução, o usuário deverá escolher o tamanho do objeto, a temperatura, em qual perfil está desenhando, o material e suas propriedades termofísicas, e onde quer gerar gráficos para estudar. O usuário terá a liberdade para utilizar um dentre três métodos para obter as propriedades dos materiais: propriedades constantes, correlação e interpolação.

Os principais termos e suas unidades são listadas abaixo:

- Dados relativos ao material:
  - c_p
  - k
  - ρ

- Dados relativos ao objeto
  - $\Delta x, \Delta y$ distância entre nodos, valor inicial: 1px=0.0026m $[m]$;
  - Δz distância entre perfis, valor inicial: 0.05m $[m]$;
  - T temperatura no nodo $[K]$;
- Variáveis usadas na simulação:
  - i posição do nodo em relação ao eixo x;
  - k posição do nodo em relação ao eixo y;
  - g qual grid/perfil está sendo analisado;
  - t tempo atual;
  - ν número da iteração.

Após os desenhos do usuário e colocado o simulador para rodar, o simulador irá calcular iterativamente a temperatura em cada ponto, e só parará se o erro entre iterações for menor que um valor aceitável. Posteriormente, o desenho será atualizado, para mostrar a nova distribuição de temperatura, e plotará os gráficos com os novos valores.

O software será programado em C++, com paradigma orientado ao objeto, utilizando a biblioteca *Qt* para criar a interface do usuário, e *qcustomplot* para gerar os gráficos.

Para calcular as propriedades termofísicas dos materiais, são utilizados três modelos: propriedades constantes, por correlação e por interpolação.

2.2.1 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais.

RF-01	O usuário tem a liberdade de desenhar qualquer objeto 3D, escolhendo também sua temperatura em cada ponto.

RF-02	O usuário deverá ter liberdade para escolher o material em cada ponto do objeto.

RF-03	O usuário poderá salvar e/ou carregar dados da simulação.

RF-04	O usuário poderá salvar os resultados da simulação em um arquivo pdf.

RF-05	O usuário pode adicionar materiais no simulador, e escolhar a forma de calcular suas propriedades termofísicas: constante, correlação ou interpolação.

RF-06	O usuário poderá escolher em qual ponto quer gerar gráficos para estudar a evolução da temperatura com o tempo.

RF-07	O usuário poderá comparar as propriedades termofísicas dos materiais..

2.2.2 Requisitos não funcionais

RNF-01	Os cálculos devem ser feitos utilizando-se o método numérico de diferenças finitas BTCS.

RNF-02	O programa deverá ser multi-plataforma, podendo ser executado em *Windows*, *GNU/Linux* ou *Mac*.

2.3 Casos de uso

Tabela 2.1: Exemplo de caso de uso

Nome do caso de uso:	Cálculo da temperatura
Resumo/descrição:	Cálculo da distribuição de temperatura em determinadas condições.
Etapas:	1. Escolha da temperatura e do material 2. Desenhar o objeto desenhado 3. Escolher um ponto de estudo 4. Rodar a simulação e analisar resultados 5. Salvar objeto e resultados em pdf
Cenários alternativos:	Um cenário alternativo envolve uma entrada de propriedades de um metal obtidas em laboratório, escolher se essas propriedades vão ser calculadas por correlação ou interpolação.

2.3.1 Diagrama de caso de uso geral

O diagrama de caso de uso geral da Figura 2.1 mostra o usuário desenhando um objeto com material padrão do simulador, escolhendo um ponto de estudo, rodando a simulação, analisando os resultados e salvando o objeto e resultados em pdf.

2.3.2 Diagrama de caso de uso específico

O caso de uso específico na Figura 2.2 mostra um cenário onde o usuário quer utilizar os valores da condutividade térmica obtidos em laboratório. Ele deve montar um arquivo .txt com esses valores (a forma de criar esse arquivo é descrito no Apêndice B), e carregar no simulador.

O usuário terá a liberdade de comparar seu material com outros padrões do simulador, e escolhe-lo para o desenho do objeto.

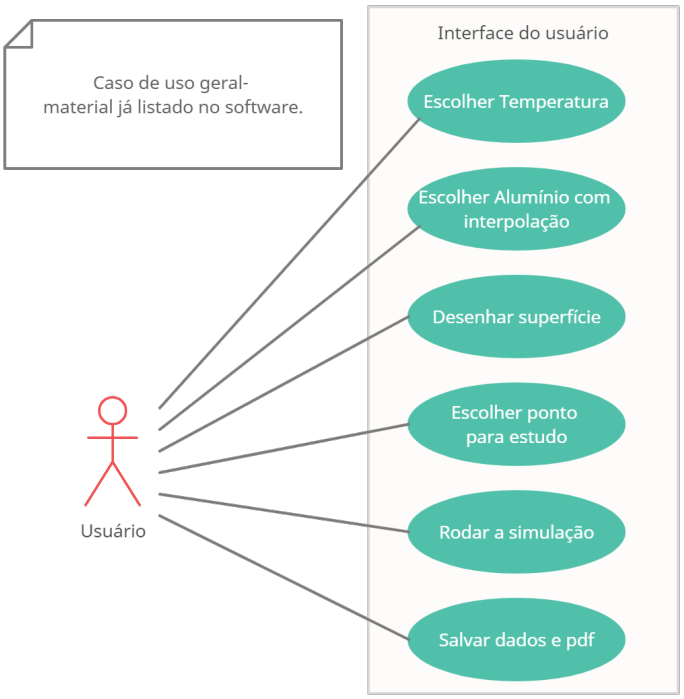

Figura 2.1: Diagrama de caso de uso – Caso de uso geral

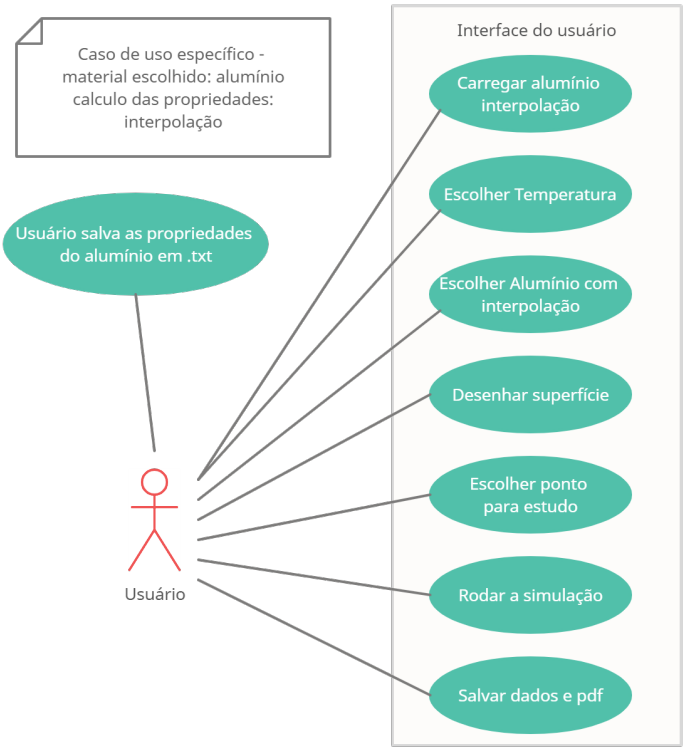

Figura 2.2: Diagrama de caso de uso específico

Capítulo 3

Elaboração

Depois da definição dos objetivos, da especificação do software e da montagem dos primeiros diagramas de caso de uso, a equipe de desenvolvimento do projeto de engenharia passa por um processo de elaboração que envolve o estudo de conceitos relacionados ao sistema a ser desenvolvido, a análise de domínio e a identificação de pacotes.

Na elaboração fazemos uma análise dos requisitos, ajustando os requisitos iniciais de forma a desenvolver um sistema útil, que atenda às necessidades do usuário e, na medida do possível, permita seu reuso e futura extensão.

3.1 Análise de domínio

Após estudo dos requisitos/especificações do sistema, algumas entrevistas, estudos na biblioteca e disciplinas do curso foi possível identificar nosso domínio de trabalho:

- Fenômeno dos transportes: área principal no qual o software foi desenvolvido. Utilizando equação do balanço de temperatura, propriedades termofísicas de materiais e condutividade térmica.
- Engenharia de petróleo: tópico principal para as simulações do software, especialmente a simulação de injeção térmica em reservatórios.
- Modelagem numérica computacional: desenvolvimento das equações diferenciais do balanço de temperatura, para que seja possível simular os mais diversos casos.
- Programação: utilização da linguagem C++ e paradigma orientado ao objeto, além de paralelismos para utilizar o máximo do poder de processamento e acelerar o software.
- Pacote de malhas: organiza o objeto desenhado em vetores.
- Pacote de simulação: resolve a equação da temperatura por métodos numéricos.

- Pacote de interpolação: utilizado para realizar interpolação com propriedades termofísicas dos materiais.
- Pacote de correlação: utilizado para realizar correlações com propriedades termofísicas dos materiais.
- Pacote de interface ao usuário: utilização da biblioteca Qt, para criar interface gráfica amigável.
- Pacote de gráficos: utilização da biblioteca qcustomplot, para montar os melhores gráficos para o problema.

3.2 Formulação

3.2.1 Formulação teórica

A equação da difusão de calor (Cap. 2 Incropera) pode ser estruturada a partir da Lei de Fourier. A equação geral da difusão de calor em meios tridimensionais cartesianos está na equação 3.1:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} = \frac{\rho c_p}{k}\frac{\partial T}{\partial t} \tag{3.1}$$

Onde ρ é a massa específica, c_p é a capacidade térmica, k é a condutividade térmica.

A modelagem pode ser feita por diferenças finitas atrasadas BTCS, onde cada derivada é representada abaixo:

$$\frac{\partial^2 T}{\partial x^2} = \frac{T^{n+1}_{i+1,j,k} - 2T^{n+1}_{i,j,k} + T^{n+1}_{i-1,j,k}}{\Delta x} \tag{3.2}$$

$$\frac{\partial^2 T}{\partial y^2} = \frac{T^{n+1}_{i,j+1,k} - 2T^{n+1}_{i,j,k} + T^{n+1}_{i,j-1,k}}{\Delta y} \tag{3.3}$$

$$\frac{\partial^2 T}{\partial z^2} = \frac{T^{n+1}_{i,j,k+1} - 2T^{n+1}_{i,j,k} + T^{n+1}_{i,j,k-1}}{\Delta z} \tag{3.4}$$

$$\frac{\partial T}{\partial t} = \frac{T^{n+1}_{i,j} - T^{n}_{i,j}}{\Delta t} \tag{3.5}$$

Substituindo as diferenças finitas na equação geral:

$$\begin{gathered}\frac{T^{n+1}_{i+1,j,k}-2T^{n+1}_{i,j,k}+T^{n+1}_{i-1,j,k}}{\Delta x} + \frac{T^{n+1}_{i,j+1,k}-2T^{n+1}_{i,j,k}+T^{n+1}_{i,j-1,k}}{\Delta y} + \\ \frac{T^{n+1}_{i,j,k+1}-2T^{n+1}_{i,j,k}+T^{n+1}_{i,j,k-1}}{\Delta z} = \frac{\rho c_p}{k}\frac{T^{n+1}_{i,j,k}-T^{n}_{i,j,k}}{\Delta t}\end{gathered} \tag{3.6}$$

Onde a malha é homogênea na superfície, mas não entre os perfis, ou seja, $\Delta x = \Delta y \neq \Delta z$. Substituindo:

$$\begin{aligned}\frac{T^{n+1}_{i+1,j,k}+T^{n+1}_{i,j+1,k}-4T^{n+1}_{i,j,k}+T^{n+1}_{i-1,j,k}+T^{n+1}_{i,j-1,k}}{\Delta x} + \frac{T^{n+1}_{i,j,k+1}-2T^{n+1}_{i,j,k}+T^{n+1}_{i,j,k-1}}{\Delta z} \\ = \frac{\rho c_p}{k}\frac{T^{n+1}_{i,j,k}-T^{n}_{i,j,k}}{\Delta t}\end{aligned} \tag{3.7}$$

Multiplicando pelo múltiplo comum:

$$\begin{aligned}\frac{\Delta z\left(T^{n+1}_{i+1,j,k}+T^{n+1}_{i,j+1,k}-4T^{n+1}_{i,j,k}+T^{n+1}_{i-1,j,k}+T^{n+1}_{i,j-1,k}\right)+\Delta x\left(T^{n+1}_{i,j,k+1}-2T^{n+1}_{i,j,k}+T^{n+1}_{i,j,k-1}\right)}{\Delta x\Delta z} \\ = \frac{\rho c_p}{k}\frac{T^{n+1}_{i,j,k}-T^{n}_{i,j,k}}{\Delta t}\end{aligned} \tag{3.8}$$

$$\begin{aligned}\Delta z\left(T^{n+1}_{i+1,j,k}+T^{n+1}_{i,j+1,k}-4T^{n+1}_{i,j,k}+T^{n+1}_{i-1,j,k}+T^{n+1}_{i,j-1,k}\right)+\Delta x\left(T^{n+1}_{i,j,k+1}-2T^{n+1}_{i,j,k}+T^{n+1}_{i,j,k-1}\right) \\ = \frac{\rho c_p \Delta x\Delta z}{k\Delta t}\left(T^{n+1}_{i,j,k}-T^{n}_{i,j,k}\right)\end{aligned} \tag{3.9}$$

$$\begin{aligned}\Delta z T^{n+1}_{i+1,j,k}+\Delta z T^{n+1}_{i,j+1,k}-4\Delta z T^{n+1}_{i,j,k}+\Delta z T^{n+1}_{i-1,j,k}+\Delta z T^{n+1}_{i,j-1,k}+ \\ \Delta x T^{n+1}_{i,j,k+1}-2\Delta x T^{n+1}_{i,j,k}+\Delta x T^{n+1}_{i,j,k-1} = \frac{\rho c_p \Delta x\Delta z}{k\Delta t}T^{n+1}_{i,j,k}-\frac{\rho c_p \Delta x\Delta z}{k\Delta t}T^{n}_{i,j,k}\end{aligned} \tag{3.10}$$

Encontrando a seguinte equação:

$$\begin{aligned}\Delta z T^{n+1}_{i+1,j,k}+\Delta z T^{n+1}_{i,j+1,k}+\Delta x T^{n+1}_{i,j,k+1} \\ +\Delta z T^{n+1}_{i-1,j,k}+\Delta z T^{n+1}_{i,j-1,k}+\Delta x T^{n+1}_{i,j,k-1} \\ -\left(4\Delta z+2\Delta x+\frac{\rho c_p \Delta x\Delta z}{k\Delta t}\right)T^{n+1}_{i,j,k} \\ = -\frac{\rho c_p \Delta x\Delta z}{k\Delta t}T^{n}_{i,j,k}\end{aligned} \tag{3.11}$$

A Equação 3.11 é a geral da difusão de calor discretizada por diferenças finitas. Para implementar no software, é necessário modelar as fronteiras e, como é buscado uma generalização da equação para um objeto com superfície qualquer, será necessário entender alguns pontos.

Começamos entendendo a equação 3.11: na primeira linha, são concentrados pontos de temperaturas localizadas posteriormente ao estudado. Na segunda, são pontos anteriores ao estudado. Já o termo em parênteses na terceira linha, é o coeficiente para o termo estudado. Por fim, a última linha a direita da igualdade, é a temperatura no ponto estudado, mas no tempo anterior.

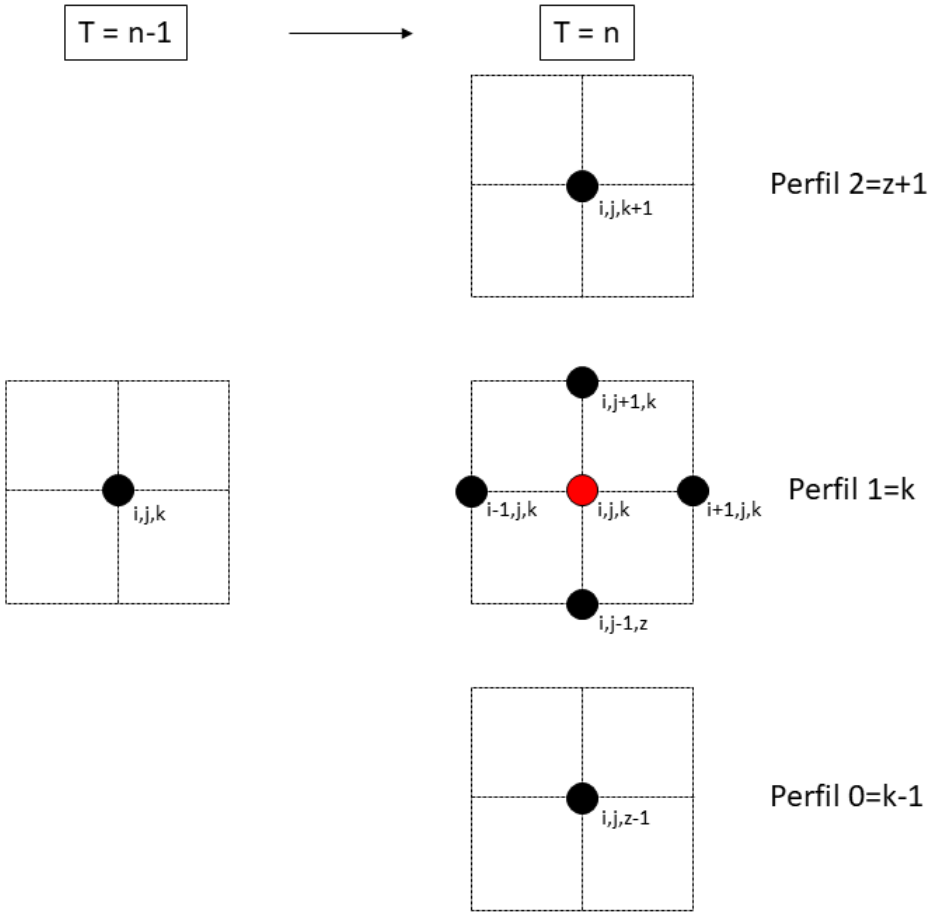

Figura 3.1: Malha utilizada para calcular um ponto de temperatura.

A seguir, serão realizadas duas etapas para finalizar a discretização. Primeiro a modelagem das fronteiras e, em seguida, a generalização da superfície de fronteira.

Primeira Parte

É discretizada a condição de contorno de Neumann, onde não há trocas com o meio externo, considerando que não há trocas com o ponto anterior no eixo x:

$$\frac{\partial T}{\partial x} = \frac{T_{i,j,k}^{n+1} - T_{i-1,j,k}^{n+1}}{\Delta x} = 0 \tag{3.12}$$

logo,

$$T_{i-1,j,k}^{n+1} = T_{i,j,k}^{n+1} \tag{3.13}$$

Todas as seis fronteiras possuem esse comportamento, então:

$$\begin{aligned} T_{i-1,j,k}^{n+1} &= T_{i,j,k}^{n+1} \\ T_{i+1,j,k}^{n+1} &= T_{i,j,k}^{n+1} \\ T_{i,j-1,k}^{n+1} &= T_{i,j,k}^{n+1} \\ T_{i,j+1,k}^{n+1} &= T_{i,j,k}^{n+1} \\ T_{i,j,k-1}^{n+1} &= T_{i,j,k}^{n+1} \\ T_{i,j,k+1}^{n+1} &= T_{i,j,k}^{n+1} \end{aligned} \tag{3.14}$$

Segunda Parte

Voltamos agora para a equação 3.11, faremos um caso onde há fronteira do lado esquerdo no eixo x (caso da primeira linha da equação 3.14):

$$\begin{aligned} &\Delta z T_{i+1,j,k}^{n+1} + \Delta z T_{i,j+1,k}^{n+1} + \Delta x T_{i,j,k+1}^{n+1} \\ &+\underline{\Delta z T_{i,j,k}^{n+1}} + \Delta z T_{i,j-1,k}^{n+1} + \Delta x T_{i,j,k-1}^{n+1} \\ &- \left(4\Delta z + 2\Delta x + \tfrac{\rho c_p \Delta x \Delta z}{k\Delta t}\right) T_{i,j,k}^{n+1} \\ &= -\tfrac{\rho c_p \Delta x \Delta z}{k\Delta t} T_{i,j,k}^{n} \end{aligned}$$

Arrumando a equação:

$$\begin{aligned} &\Delta z T_{i+1,j,k}^{n+1} + \Delta z T_{i,j+1,k}^{n+1} + \Delta x T_{i,j,k+1}^{n+1} \\ &+\Delta z T_{i,j-1,k}^{n+1} + \Delta x T_{i,j,k-1}^{n+1} \\ &- \left(3\Delta z + 2\Delta x + \tfrac{\rho c_p \Delta x \Delta z}{k\Delta t}\right) T_{i,j,k}^{n+1} \\ &= -\tfrac{\rho c_p \Delta x \Delta z}{k\Delta t} T_{i,j,k}^{n} \end{aligned} \tag{3.15}$$

Podemos perceber que o termo *i-1,j,k* sumiu da equação, e dimuiu o número 4 dentro do parênteses para 3, indicando que o número 4 é diretamente relacionado ao número de fronteiras da superfície xy, e o número 2, do eixo z.

Isso quer dizer que, caso exista uma fronteira na dimensão x ou y, esse termo deve ser anulado (condição de fronteira), e retirado 1 do total das 4 fronteiras e, caso exista uma fronteira no sentido de z, deve ser retirado a quantidade de fronteiras do total de 2. Portanto, é definito duas novas variáveis para o problema, nx e nz, onde $nx \in [0; 4]$e $nz \in [0; 2]$

Pode-se ir além, e provar o caso onde há fronteiras em todos os sentidos:

$$\begin{aligned} &\Delta z T_{i1,j,k}^{n+1} + \Delta z T_{i,j1,k}^{n+1} + \Delta x T_{i,j,k1}^{n+1} \\ &+\Delta z T_{i,j,k}^{n+1} + \Delta z T_{i,j1,k}^{n+1} + \Delta x T_{i,j,k1}^{n+1} \\ &- \left(4\Delta z + 2\Delta x + \tfrac{\rho c_p \Delta x \Delta z}{k\Delta t}\right) T_{i,j,k}^{n+1} \\ &= -\tfrac{\rho c_p \Delta x \Delta z}{k\Delta t} T_{i,j,k}^{n} \end{aligned} \tag{3.16}$$

resultando em

$$T_{i,j,k}^{n+1} = \frac{\frac{\rho c_p \Delta x \Delta z}{k\Delta t}}{\frac{\rho c_p \Delta x \Delta z}{k\Delta t}} T_{i,j,k}^{n} \tag{3.17}$$

$$T_{i,j,k}^{n+1} = T_{i,j,k}^{n} \tag{3.18}$$

Ou seja, um ponto isolado no espaço não tem variação de temperatura.

Portanto, para ser possível implementar a equação discretizada 3.11 em C++, será utilizado:

$$T_{i,j,k}^{n+1} = \left(nx\Delta z + nz\Delta x + \frac{\rho c_p \Delta x \Delta z}{k\Delta t}\right)^{-1} \left[\begin{array}{c} \frac{\rho c_p \Delta x \Delta z}{k\Delta t} T_{i,j,k}^{n} + \\ \Delta z T_{i+1,j,k}^{n+1} + \Delta z T_{i,j+1,k}^{n+1} + \Delta x T_{i,j,k+1}^{n+1} \ + \\ \Delta z T_{i-1,j,k}^{n+1} + \Delta z T_{i,j-1,k}^{n+1} + \Delta x T_{i,j,k-1}^{n+1}] \end{array}\right] \tag{3.19}$$

3.2.2 Paralelismos/multi-thread

Os chips de processadores atuais, são constituídos por vários processadores menores, o que permite que um mesmo processador consiga realizar tarefas distintas. A idéia é separar tarefas distintas, para que um processador não fique travado em uma única tarefa.

Uma analogia para melhorar a explicação é a dos estudantes. Uma sala cheia de estudantes, recebe uma tarefa de resolver uma lista de exercícios. Se todos os exercícios forem resolvidas por um único aluno, levará um tempo muito grande para terminarem a tarefa (caso sem paralelismo). Se os alunos dividirem as tarefas entre si, ela será resolvida muito mais rapidamente.

Similarmente ao cenário acima, foram implementados três casos de paralelismo, por questão de didática.

1. Sem paralelismo: uma única thread do processador resolve todos os cálculos.

2. Paralelismo por grid: cada thread resolve uma camada do objeto. Possuí certa otimização em relação ao anterior, mas, se só existir objeto em uma camada, outras threads ficam osciosas.

3. Paralelismo total: todas as threads do processador resolvem os cálculos de todo o objeto 3D, intercalando a posição com base no número da thread.

A figura ilustra melhor esses casos

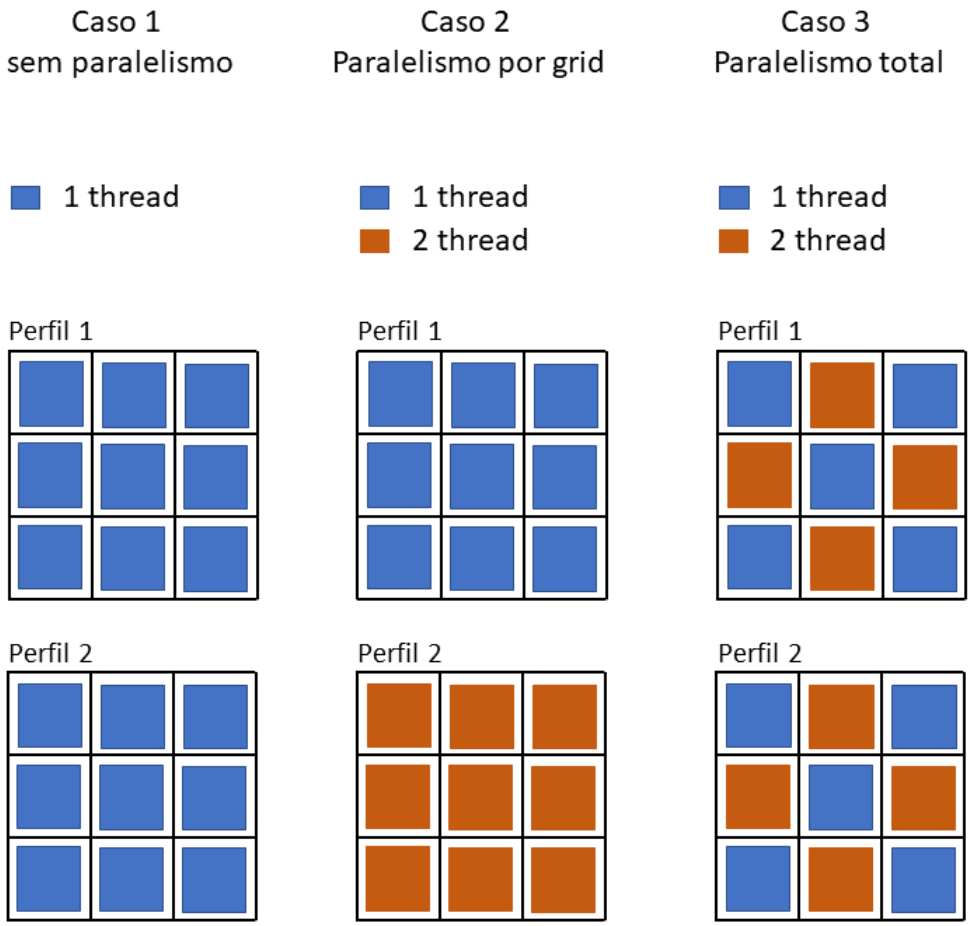

Figura 3.2: Figura ilustrando os três casos de paralelismo implementados para duas camadas com 9 células cada, e um processador com duas threads.

O algoritmo utilizado para o caso 3 é:
for(int i = NUM_ THREAD; i < size; i+=MAX_ THREADS)

3.2.3 Renderização 3D

Após o usuário desenhar algum objeto no software, pode ser de interesse dele observar como seria em renderização 3D. Portanto, é implementado algoritmos para essa renderização.

Inicialmente, é interessante observar a complexidade da renderização: um objeto 3D deve ser apresentado em uma tela 2D, com a ilusão de ótica que é um objeto com profundidade. Por exemplo, um cubo com arestas de tamanho 1 cm é mostrado nos quatro casos da figura abaixo:

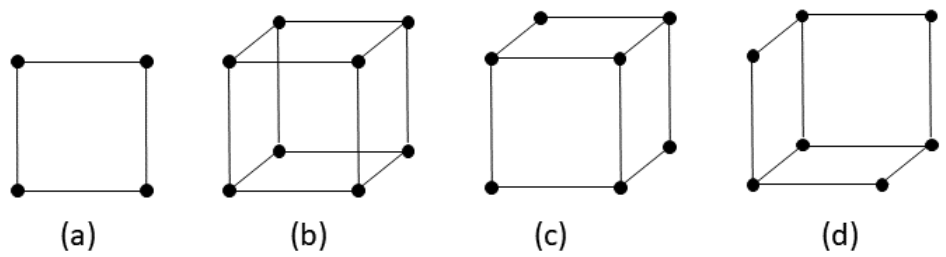

Figura 3.3: (a) Observador alinhado com uma das faces do cubo. (b) observador não está alinhado e não foram removidas arestas ocultas. O cérebro consegue interpretar que é um objeto 3D, mas fica confuso entre os casos (c) e (d).

Todos cantos do cubo da figura 3.2.3 estão na mesma posição, o que mudou foi o ângulo do observador com o objeto.

Portanto, tendo em mãos os pontos das arestas, é multiplicado esses vetores com a matriz de rotação do autor [Herter and Lott,] mostrada em (3.20), a qual permite rotacionar qualquer ponto a partir dos três angulos do observador.

$$R(\alpha,\beta,\gamma) = \begin{bmatrix} cos(\gamma)cos(\beta) & cos(\gamma)sin(\beta)sin(\alpha) - sin(\gamma)cos(\alpha) & cos(\gamma)sin(\beta)sin(\alpha) + sin(\gamma)cos(\alpha) \\ sin(\gamma)cos(\beta) & sin(\gamma)sin(\beta)sin(\alpha) + cos(\gamma)cos(\alpha) & sin(\gamma)sin(\beta)cos(\alpha) - cos(\gamma)sin(\alpha) \\ -sin(\beta) & cos(\beta) * sin(\alpha) & cos(\beta) * cos(\alpha) \end{bmatrix} \tag{3.20}$$

Ou seja, inicialmente, um cubo de aresta 3 cm, com uma margem de 1 cm, pode ser mostrado na tela (monitor) com os pontos do caso (a) da figura 3.2.3, onde o observador está alinhado com o objeto.

Conforme desejado, o objeto pode mudar seu ângulo com o observador, como no caso (b), onde os ângulos x e y passaram a ter o valor de 0.1 radianos. Não foi só os pontos de trás do cubo que aparecem (e mudaram seus valores), mas todos os pontos foram modificados.

Além disso, a aresta possui valor ligeiramente menor que 3, pois não é mais "de frente" que o observador está olhando, mas ligeiramente de lado. Mesmo que o objeto cubo tenha aresta de 3 centimetros.

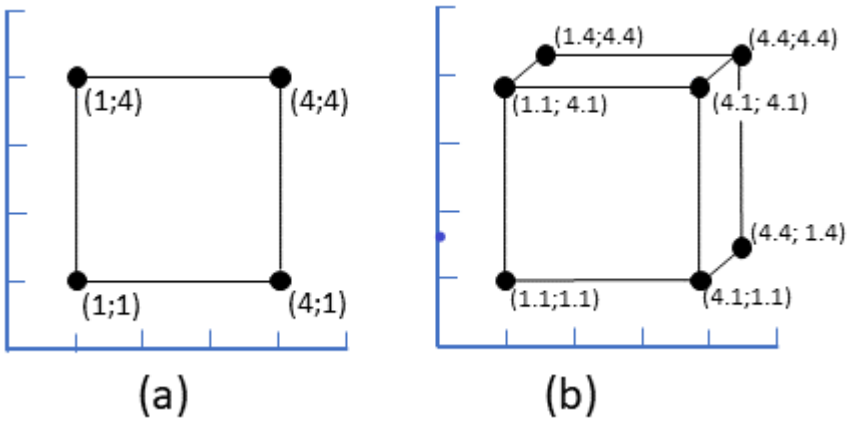

Figura 3.4: (a) o cubo está com ângulos nulos. (b) ângulo x e y estão com valor de 0.1 radianos.

Nos desenhos do simulador, cada pixel da figura, é uma célula com propriedades que serão calculadas, possuindo material, temperatura e volume. Como o usuário pode desenhar por pixel, a renderização 3D deve partir do princípio que cada pixel é um **potencial** objeto que deve ser renderizado.

Inicialmente, essa conclusão pode ficar vaga, pois todas as células do simulador devem ser renderizadas, mas, quando a simulação fica grande, é numeroso a quantidade de objetos renderizando ao mesmo tempo, tornando muito lenta a apresentação. Então algumas considerações são feitas no algoritmo para otimizar a renderização.

Primeiro, é desejável desenhar triângulos, e não pontos ou retas, por 2 motivos: geometria simples, possui normal e a biblioteca do Qt consegue desenhar e preencher a área com qualquer cor escolhida.

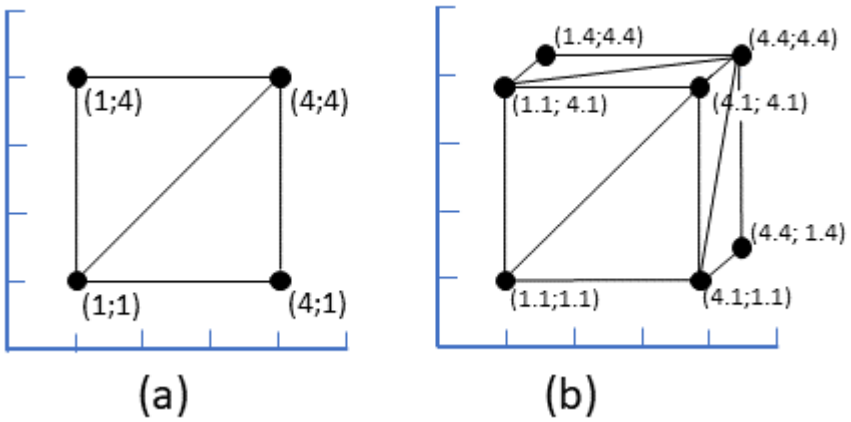

Figura 3.5: Mesmo desenho da figura anterior, mas agora renderizando a partir de triângulos.

O segundo motivo apresentado, é o mais importante dos três. Um triângulo possui três pontos, podendo ser reduzido para dois vetores (subtraindo o ponto de origem dos

outros dois pontos) e permite-se calcular a normal dessa superfície. Com isso, é obtido dos vetores $\boldsymbol{a} = \{a_1, a_2, a_3\}$ e o vetor $\boldsymbol{b} = \{b_1, b_2, b_3\}$ permitindo a realização do produto vetorial:

$$\boldsymbol{a} \times \boldsymbol{b} = \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} \tag{3.21}$$

Ou simplesmente:

$$\boldsymbol{a} \times \boldsymbol{b} = (a_2 b_3 - a_3 b_2)\boldsymbol{i} - (a_1 b_3 - a_3 b_1)\boldsymbol{k} + (a_1 b_2 - a_2 b_1)\boldsymbol{j} \tag{3.22}$$

Utilizando a Regra da Mão Direita[1], é possível entender a utilidade da equação 3.22: o caso (a) da figura 3.2.3, mostra uma normal saindo do papel, em direção ao olho do leitor, logo, é um triângulo que deve ser renderizado. O caso (b) possui uma normal no sentido contrário, e não faz sentido desenhar esse triângulo, pois está na parte de trás do objeto.

1

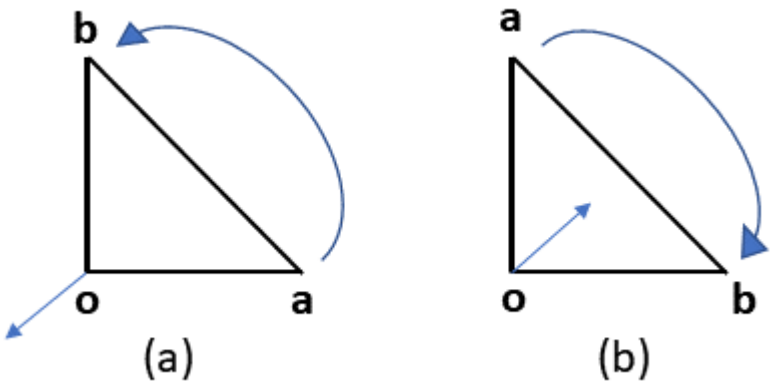

Figura 3.6: (a) mostra um caso onde a normal é na direção do leitor e (b) mostra um caso onde a normal é para dentro da folha.

Essa simples operação condicional do valor positivo/negativo de **j** da normal, reduz a renderização de objetos ocultos, e otimiza o software em duas vezes.

Uma outra condição implementada é a de avaliar se o objeto possui fronteira com outro objeto. Com isso, não é necessário renderizar 4 triângulos dessas duas superfícies em contato. Como estão em contato, não deve ser renderizada sob hipótese alguma.

Por fim, antes de renderizar os numerosos triângulos, eles são colocadas em ordem crescente com o valor de **j** da normal. Isso serve para ser desenhado primeiro o que está

[1]Para utilizar a Regra da Mão Direita, posicione o dedo polegar sobre o ponto **o**, e estique o indicador para o ponto **a**, agora, feche o indicador no sentido do ponto **b** (seta curvada mostra o sentido que a ponta do indicador deve realizar). No caso (a) da figura, o dedo polegar fica no sentido para fora do papel, e o caso (b), para dentro.

atrás, e depois desenhar o que está na frente, sobrescrevendo áreas que deveriam estar ocultas, evitando a criação de figuras confusas como no caso (b) da figura 3.2.3. É uma técnica lenta, mas de fácil implementação.

3.3 Identificação de pacotes – assuntos

- Pacote de malhas: organiza o objeto desenhado em vetores, facilita o acesso do simulador às propriedades de cada célula.
- Pacote de simulação: nela está presente o coração do simulador: o solver da equação da temperatura, discretizada por métodos numéricos, e resolvida por método iterativo.
- Pacote de interpolação: utilizado para realizar interpolação com propriedades termofísicas dos materiais, é acessado pelo simulador, e retorna as propriedades do material.
- Pacote de correlação: mesma função da linha acima, mas para método de correlação.
- Pacote de interface ao usuário: utilização da biblioteca Qt, para criar interface gráfica amigável. Fornece um ambiente onde o usuário pode enviar comandos para o simulador de maneira fácil, e apresenta os resultados.
- Pacote de gráficos: utilização da biblioteca qcustomplot, para montar os melhores gráficos para o problema. É solicitado ao pacote de malhas os resultados da temperatura. Está presente junto com o pacote de interface

3.4 Diagrama de pacotes – assuntos

Abaixo é apresentado o diagrama de pacotes (Figura 3.7).

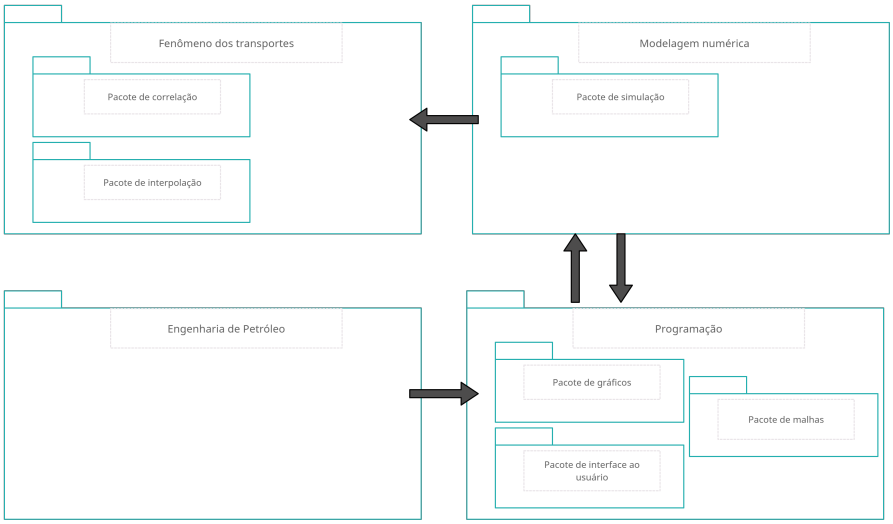

Figura 3.7: Diagrama de Pacotes

Capítulo 4

Projeto

Neste capítulo é apresentado questões relacionadas ao desenvolvimento do projeto, como ambiente de desenvolvimento e bibliotecas gráficas, comentados juntamente com a evolução de versões. Também é apresentado os diagramas de componentes e de implantação.

4.1 Projeto do sistema

O software desenvolvido foi implementado com a linguagem C++, sob o paradigma de orientação ao objeto.

Inicialmente, foi utilizado a biblioteca *SFML* para a criação de janelas para o usuário, e utilizado o ambiente de desenvolvimento *Visual Studio*, tudo isso no sistema operacional *Windows 10.*

Inicialmente, foi desenvolvido um software simples, com uma mistura de janela-terminal. O usuário podia desenhar e simular, mas não tinha muita liberdade para escolher e adicionar materiais.

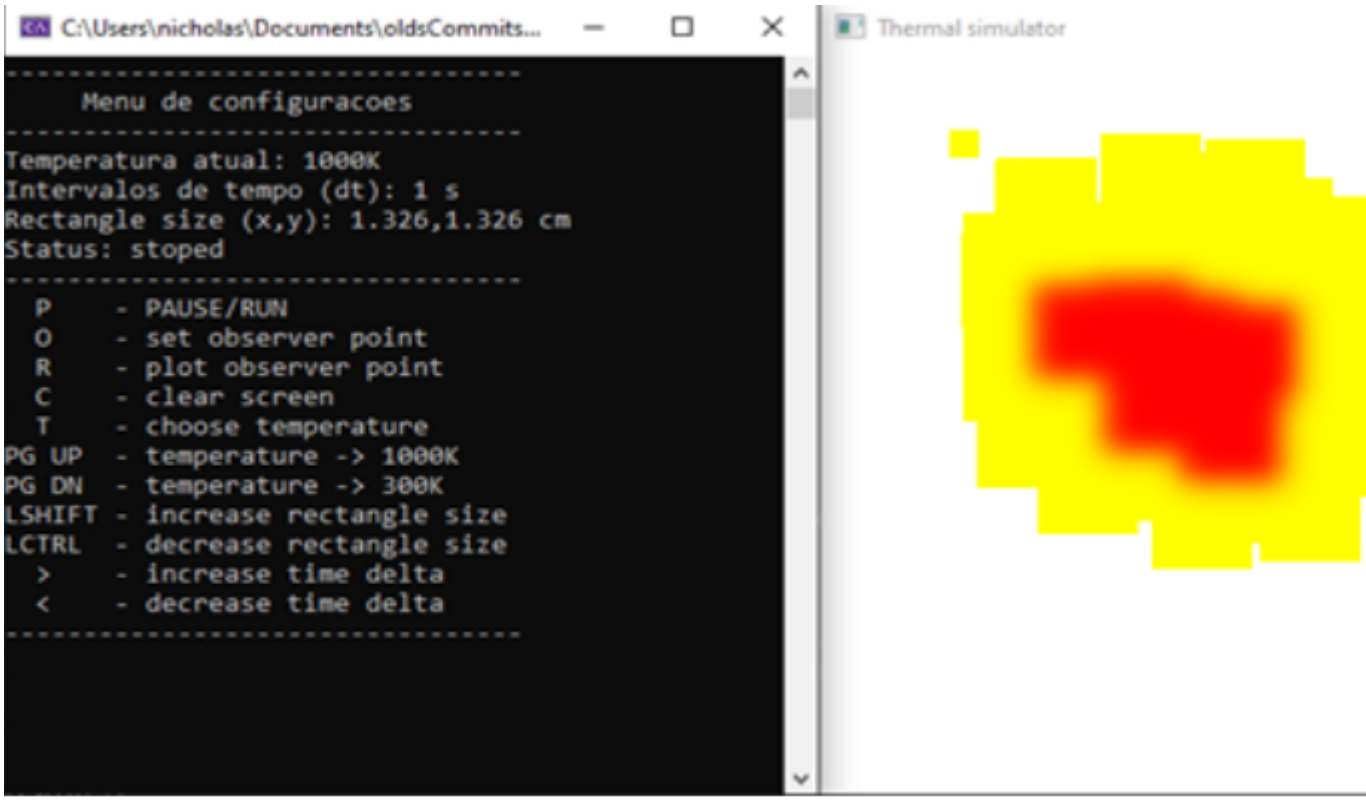

Figura 4.1: Versão 0.1, simples e utilizando a biblioteca *SFML*

Conforme a evolução pedia, foi criada uma segunda janela, a qual replica o desenho com as cores do material escolhido.

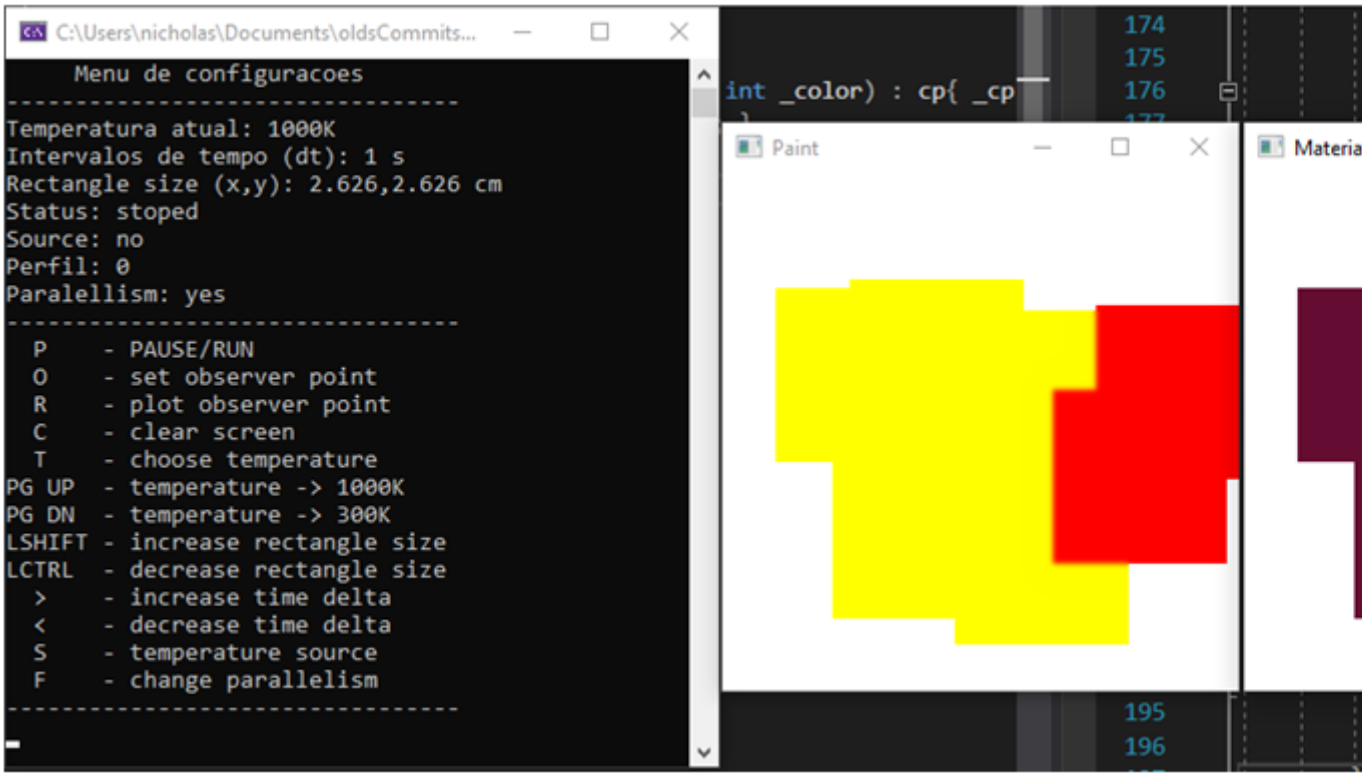

Figura 4.2: Versão 0.2, simples, mas preparando terreno para uma segunda janela dos materiais.

Por fim, foi montada a versão final utilizando essa biblioteca. Foi uma versão importantíssima para o aprendizado, pois o usuário não desenhava diretamente no software, mas era enviado uma lista de propriedades do desenho para o grid e, quando o desenho era atualizado, a biblioteca utilizava os valores do grid. Isso permitiu juntar as duas janelas.

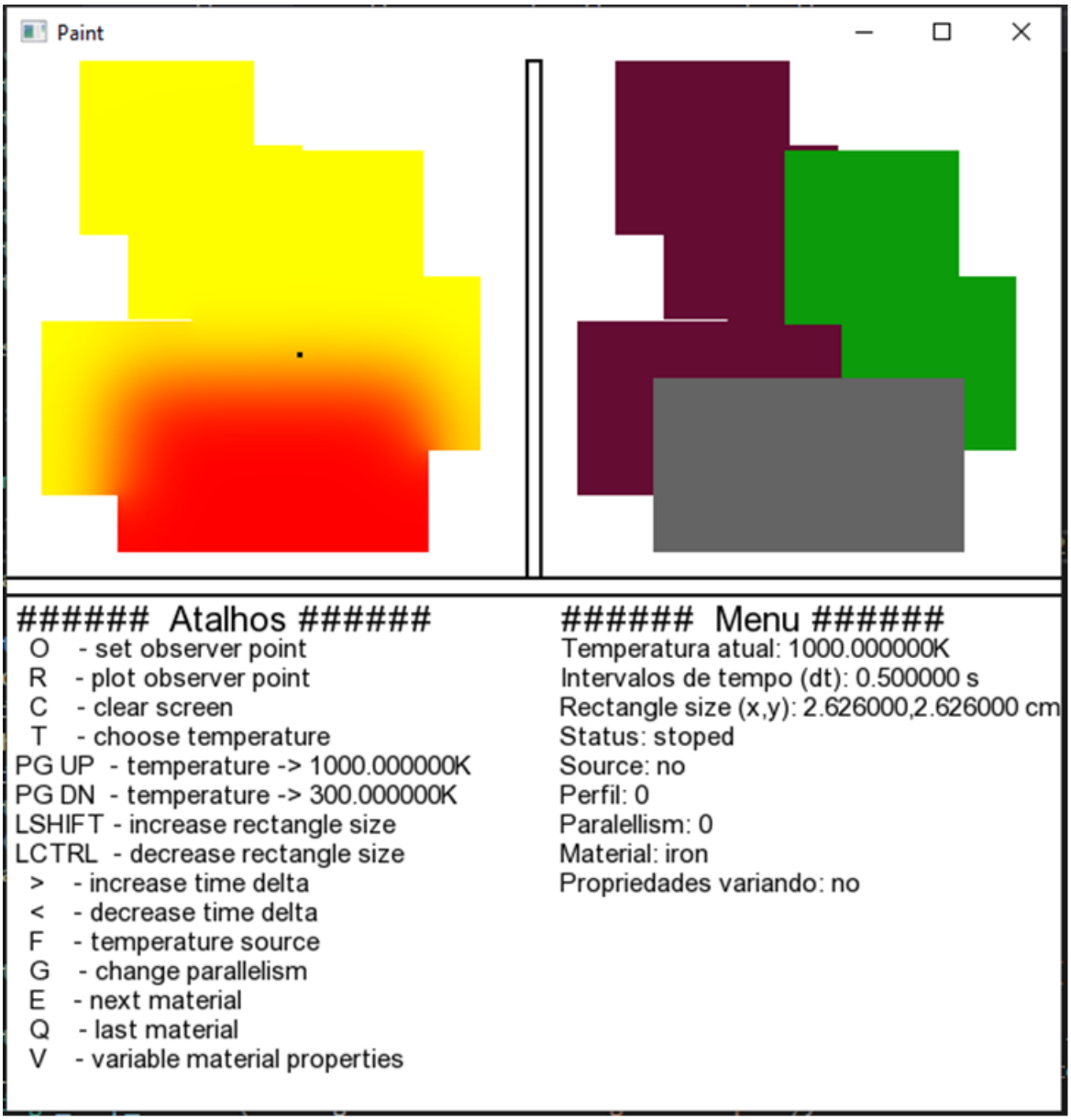

Figura 4.3: Versão 0.3, completa e complexa, mas muito lento.

Durante o desenvolvimento das versões anteriores, foi citado uma segunda biblioteca gráfica chamada *Qt*, mais rápida e completa que a anterior. Então surgiu essa necessidade de mudança.

Como o software foi programado com orientação ao objeto, foi rápido a migração, modificando, quase que somente, a classe da janela. Permitindo criar o software na versão 1.0.

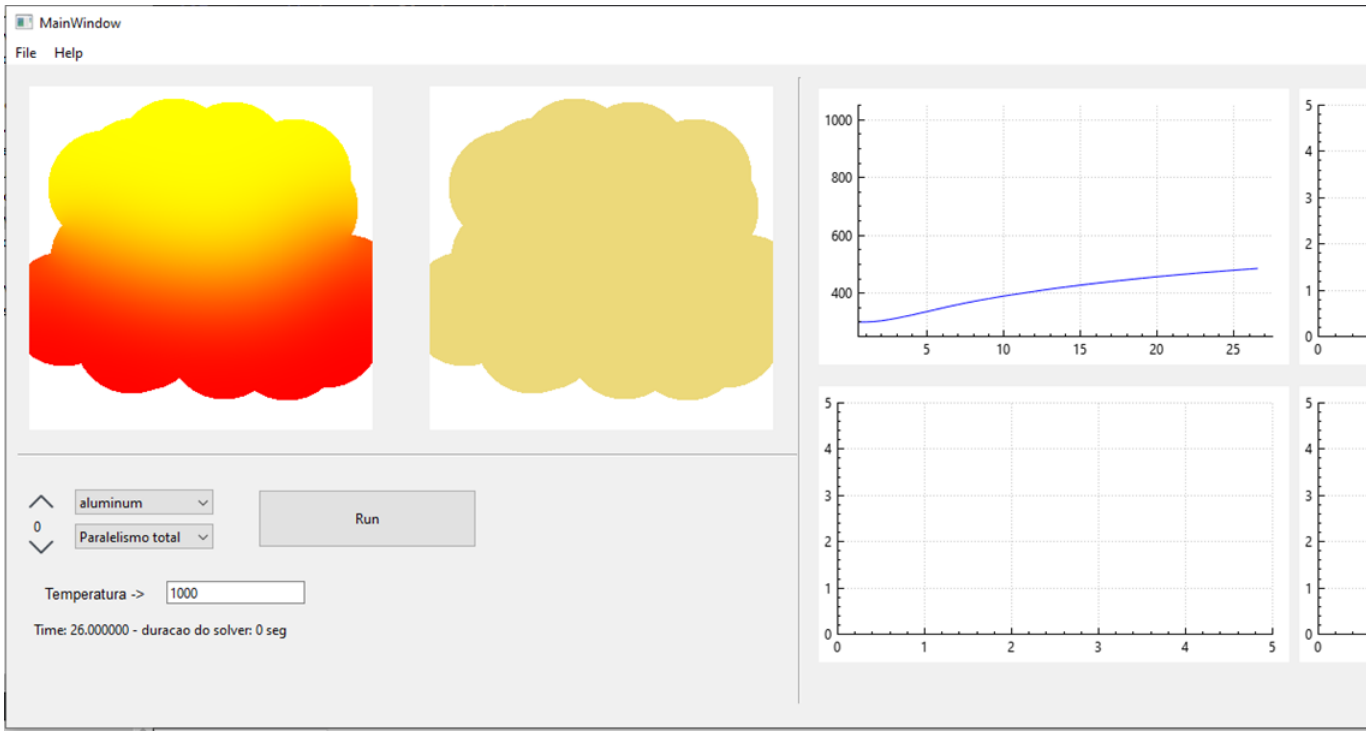

Figura 4.4: Versão 1.0, inicial e incompleta, mas utilizando a biblioteca *Qt*.

Para utilizar as ferramentas fornecidas por essa bibliteca, foi migrado do editor de texto *Visual Studio* para o *Qt Creator*. Abaixo é apresentado o ambiente de trabalho.

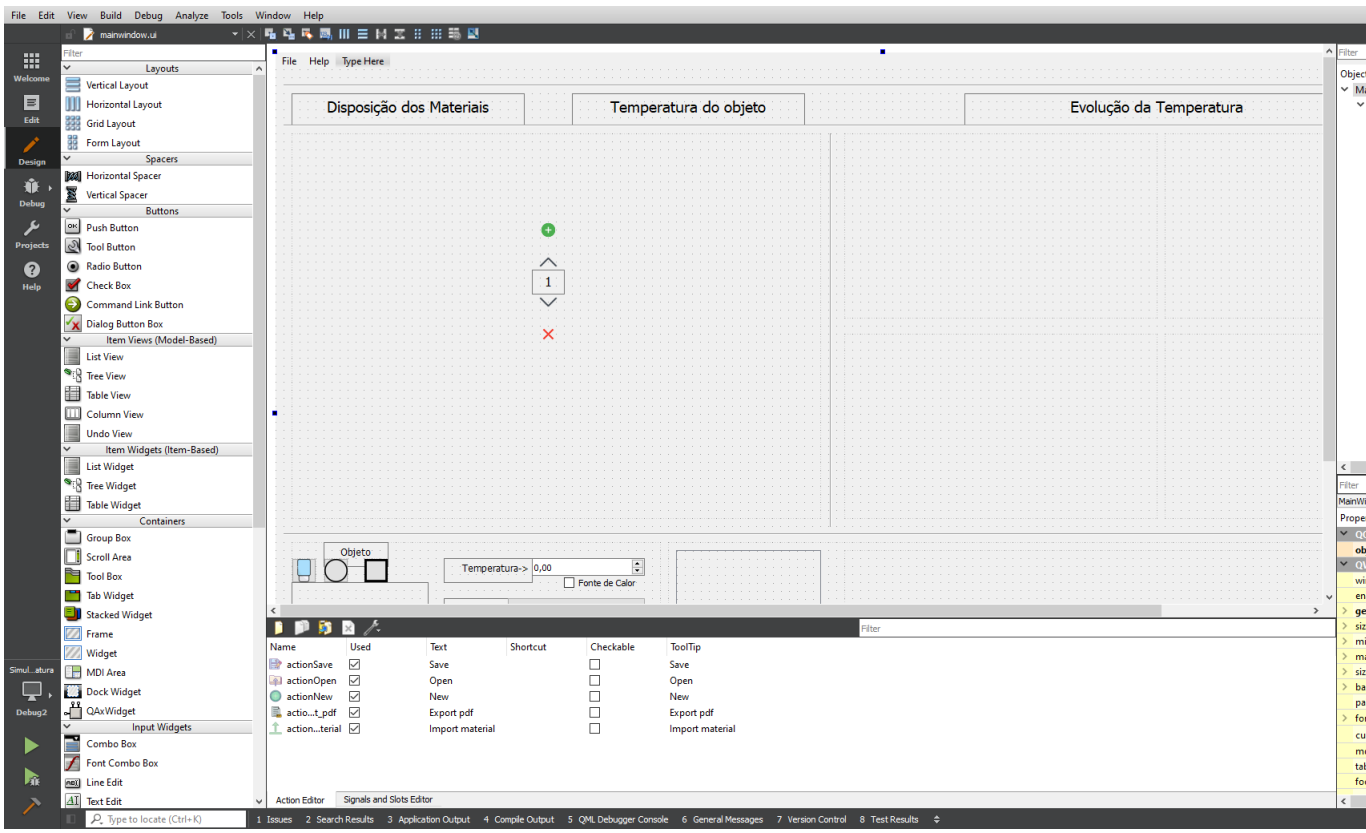

Figura 4.5: *Qt Creator*.

A curva de evolução do software dentro do *Qt Creator* foi exponencial, permitindo a criação da versão final apresentada na figura 4.6, com duas áreas que apresentam os cortes desenhados, 4 gráficos com valores da temperatura ao longo do tempo ou espaço. Na região do canto inferior esquerdo, mostram opções para a simulação ou criação do objeto. Na direita, é mostrado as propriedades termofísicas de vários materiais ao longo da temperatura.

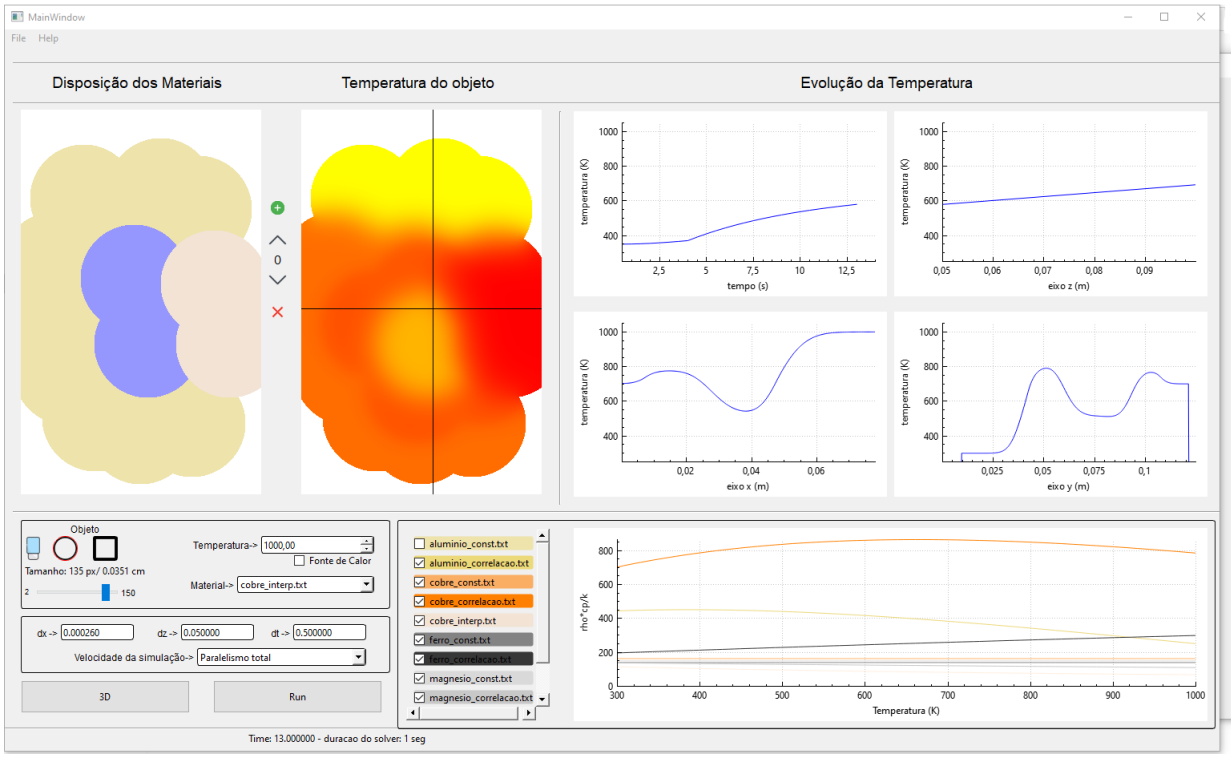

Figura 4.6: Versão 1.2, final. Na direita é apresentado a visualização 3D do objeto desenhado.

4.2 Diagrama de implantação

O diagrama de implantação é um diagrama de alto nível que inclui relações entre o sistema e o hardware e que se preocupa com os aspectos da arquitetura computacional escolhida. Seu enfoque é o hardware, a configuração dos nós em tempo de execução.

O diagrama de implantação deve incluir os elementos necessários para que o sistema seja colocado em funcionamento: computador, periféricos, processadores, dispositivos, nós, relacionamentos de dependência, associação, componentes, subsistemas, restrições e notas.

Veja na Figura 4.7 um exemplo de diagrama de implantação de um cluster. Observe a presença de um servidor conectado a um switch. Os nós do cluster (ou clientes) também estão conectados ao switch. Os resultados das simulações são armazenados em um servidor de arquivos (*storage*).

Pode-se utilizar uma anotação de localização para identificar onde determinado componente está residente, por exemplo {localização: sala 3}.

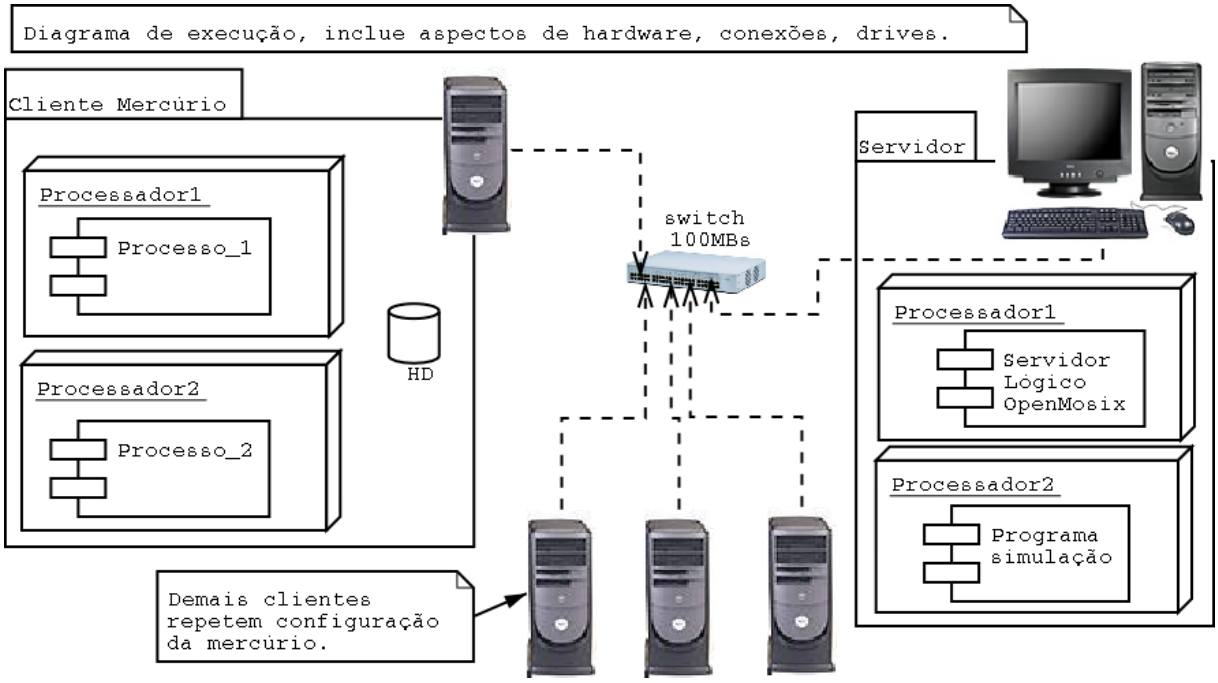

Figura 4.7: Diagrama de implantação

Nota:

Não perca de vista a visão do todo; do projeto de engenharia como um todo. Cada capítulo, cada seção, cada parágrafo deve se encaixar. Este é um diferencial fundamental do engenheiro em relação ao técnico, a capacidade de desenvolver projetos, de ver o todo e suas diferentes partes, de modelar processos/sistemas/produtos de engenharia.

Capítulo 5

Implementação

Neste capítulo do projeto de engenharia apresentamos os códigos fonte que foram desenvolvidos.

Nota: os códigos devem ser documentados usando padrão **javadoc**. Posteriormente usar o programa **doxygen** para gerar a documentação no formato html.

- Veja informações gerais aqui http://www.doxygen.org/.
- Veja exemplo aqui http://www.stack.nl/~dimitri/doxygen/manual/docblocks.html.

5.1 Código fonte

Apresenta-se a seguir um conjunto de classes (arquivos .h e .cpp) além do programa `main`.

Apresenta-se na listagem 5.1 o arquivo com código da função main.

Listing 5.1: Arquivo de implementação da função main.

```
1 #include "mainwindow.h"
2 #include <QApplication>
3
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
7     MainWindow w;
8     w.show();
9     return a.exec();
10 }
```

Apresenta-se na listagem 5.2 o arquivo de cabeçalho da classe mainwindow.

Listing 5.2: Arquivo de implementação da função mainwindow.

```
1 #ifndef MAINWINDOW_H
```

```
2 #define MAINWINDOW_H
3
4 #include <string>
5 #include <iostream>
6
7 #include <QDir>                    ///< Biblioteca que permite
      acessar diretorios.
8 #include <QDirIterator>
9 #include <QImage>                  ///< desenhar pixels
10 #include <QColor>                  ///< escolher a cor dos pixels
11 #include <QPainter>                ///< desenhar pixels
12 #include <QPrinter>                ///< Biblioteca  que habilita a
      geracao de pdf.
13 #include <QPainter>                ///< Biblioteca que auxilia a
      geracao do pdf.
14 #include <QPdfWriter>
15 #include <QMainWindow>
16 #include <QMouseEvent>             ///< pegar acoes/posicao do mouse
17 #include <QFileDialog>
18
19 #include "C3D.h"
20 #include "ui_mainwindow.h"
21 #include "CSimuladorTemperatura.h"
22
23
24 QT_BEGIN_NAMESPACE
25 namespace Ui { class MainWindow; }
26 QT_END_NAMESPACE
27
28 class MainWindow : public QMainWindow {
29     Q_OBJECT
30
31 public:
32     MainWindow(QWidget *parent = nullptr);
33     ~MainWindow();
34
35 private:
36     Ui::MainWindow *ui;
37     QPoint m_mousePos;
38     QPixmap pixmap;
39     QImage *mImage;
40     QWidget* checkboxes;
```

```
41      QVBoxLayout* layout;
42      std::vector<QCheckBox*> myCheckbox;
43      CSimuladorTemperatura *simulador;
44      std::string drawFormat = "circulo";
45
46      int timerId;
47      int parallelType = 2;
48      int size_x = 300, size_y = 480;
49      int currentGrid = 0;
50      int space_between_draws = 50;
51      int left_margin = 20, up_margin = 140;
52      bool runningSimulator = false;
53      bool eraserActivated = false;
54      QPoint studyPoint = QPoint(0,0);
55      int studyGrid;
56      std::vector<bool> selectedMateriails;
57      QVector<double> time, temperature;
58
59  protected:
60      void start_buttons();
61      void mousePressEvent(QMouseEvent *event) override;
62      void printPosition();
63      void printDrawSize();
64      void paintEvent(QPaintEvent *e) override;
65      QImage paint(int grid);
66
67      QColor calcRGB(double temperatura);
68      void runSimulator();
69      void timerEvent(QTimerEvent *e) override;
70
71  private slots:
72      void on_pushButton_clicked();
73      void on_gridDownButton_clicked();
74      void on_gridUpButton_clicked();
75
76      void createWidgetProps();
77
78      void makePlot1();
79      void makePlot2();
80      void makePlot3();
81      void makePlot4();
82      void makePlotMatProps();
```

```
83      bool checkChangeMaterialsState();
84      void on_actionSave_triggered();
85      void on_actionOpen_triggered();
86      void on_actionNew_triggered();
87      void on_actionExport_pdf_triggered();
88      QString save_pdf(QString file_name);
89      void on_buttonCircle_clicked();
90      void on_buttonSquare_clicked();
91      void on_actionImport_material_triggered();
92      void on_gridAddGrid_clicked();
93      void on_gridDelGrid_clicked();
94      void on_buttonEraser_clicked();
95      void on_button3D_clicked();
96  };
97  #endif
```

Apresenta-se na listagem 5.3 implementação da classe mainwindow.

Listing 5.3: Arquivo de implementação da função mainwindow.

```
1  #include "mainwindow.h"
2
3  MainWindow::MainWindow(QWidget *parent)
4      : QMainWindow(parent), ui(new Ui::MainWindow)
5  {
6      up_margin = 100;
7      simulador = new CSimuladorTemperatura();
8      simulador->resetSize(size_x, size_y);
9      ui->setupUi(this);
10     mImage = new QImage(size_x*2+space_between_draws, size_y,QImage
           ::Format_ARGB32_Premultiplied);
11     timerId = startTimer(20);
12
13     ui->plot1->addGraph();
14     ui->plot2->addGraph();
15     ui->plot3->addGraph();
16     ui->plot4->addGraph();
17     ui->plot_MatProps->addGraph();
18     ui->plot1->xAxis->setLabel("tempo␣(s)");
19     ui->plot1->yAxis->setLabel("temperatura␣(K)");
20     ui->plot2->xAxis->setLabel("eixo␣z␣(m)");
21     ui->plot2->yAxis->setLabel("temperatura␣(K)");
22     ui->plot3->xAxis->setLabel("eixo␣x␣(m)");
23     ui->plot3->yAxis->setLabel("temperatura␣(K)");
```

```
24      ui->plot4->xAxis->setLabel("eixo␣y␣(m)");
25      ui->plot4->yAxis->setLabel("temperatura␣(K)");
26      ui->plot_MatProps->xAxis->setLabel("Temperatura␣(K)");
27      ui->plot_MatProps->yAxis->setLabel("rho*cp/k");
28
29      for(unsigned int i = 0; i < simulador->getMateriais().size();i
            ++)
30          ui->plot_MatProps->addGraph();
31      start_buttons();
32  }
33
34  MainWindow::~MainWindow() {
35      delete mImage;
36      delete simulador;
37      delete ui;
38  }
39
40  void MainWindow::mousePressEvent(QMouseEvent *event) {
41      if (event->buttons() == Qt::LeftButton){
42          std::string actualMaterial = ui->material_comboBox->
                currentText().toStdString();
43          double temperature = ui->spinBox_Temperature->value();
44          bool isSource = ui->checkBox_source->checkState();
45          int size = ui->horizontalSliderDrawSize->value();
46          simulador->setActualTemperature(temperature); ///
                importante para atualizar Tmin/Tmax
47
48          if (drawFormat =="circulo")
49              simulador->grid[currentGrid]->draw_cir(event->pos().x()
                    -left_margin-size_x-space_between_draws, event->pos
                    ().y()-up_margin, size/2, temperature, isSource,
                    simulador->getMaterial(actualMaterial),
                    eraserActivated);
50          else
51              simulador->grid[currentGrid]->draw_rec(event->pos().x()
                    -left_margin-size_x-space_between_draws, event->pos
                    ().y()-up_margin, size, temperature, isSource,
                    simulador->getMaterial(actualMaterial),
                    eraserActivated);
52      }
53      else if (event->buttons() == Qt::RightButton){
54          int x = event->pos().x()-left_margin-size_x-
```

```
            space_between_draws;
55          int y = event->pos().y()-up_margin;
56          if (x >= 0 && x < size_x && y >= 0 && y < size_y){
57              studyPoint = QPoint(x, y);
58              studyGrid = currentGrid;
59              time.clear();
60              temperature.clear();
61          }
62      }
63      update();
64  }
65
66  void MainWindow::printPosition(){
67      int x = QWidget::mapFromParent(QCursor::pos()).x() -
           left_margin-size_x-space_between_draws;
68      int y = QWidget::mapFromParent(QCursor::pos()).y() - up_margin;
69      QWidget::mapFromParent(QCursor::pos()).x();
70      std::string txt;
71      if ((x>0) && (x<size_x) && (y>0) && (y<size_y))
72          if (!simulador->grid[currentGrid]->operator()(x, y)->active
               )
73              txt = "(" + std::to_string(x) + ", " + std::to_string(y
                   ) + ")";
74          else
75              txt = "(" + std::to_string(x) + ", " + std::to_string(y
                   ) + ") - T: " +
76                    std::to_string(simulador->grid[currentGrid]->
                         operator()(x, y)->temp) + "K - "+ simulador
                         ->grid[currentGrid]->operator()(x, y)->
                         material->getName();
77      else
78          txt = "";
79
80      ui->textMousePosition->setText(QString::fromStdString(txt));
81  }
82
83  void MainWindow::printDrawSize(){
84      int size = ui->horizontalSliderDrawSize->value();
85      ui->textDrawSize->setText("Tamanho: "+QString::number(size) + "
           px/ "+QString::number(size*simulador->getDelta_x()) + " cm"
           );
86  }
```

```
87
88 void MainWindow::start_buttons(){
89     /// adicionar borda em widget
90     ui->widget_props->setStyleSheet("border-width: 1;"
91                                     "border-radius: 3;"
92                                     "border-style: solid;"
93                                     "border-color: rgb(10,10,10)");
94
95     ui->widget_simulator_deltas->setStyleSheet( "border-width: 1;"
96                                                 "border-radius: 3;"
97                                                 "border-style:
                                                    solid;"
98                                                 "border-color: rgb
                                                    (10,10,10)");
99
100    ui->widget_drawStyles->setStyleSheet(       "border-width: 1;"
101                                                "border-radius: 3;"
102                                                "border-style:
                                                   solid;"
103                                                "border-color: rgb
                                                   (10,10,10)");
104
105    ui->widget_buttonCircle->setStyleSheet(     "border-width: 1;"
106                                                "border-radius: 15;
                                                   "
107                                                "border-style:
                                                   solid;"
108                                                "border-color: rgb
                                                   (255,0,0)");
109
110    /// remover borda das caixas de texto
111    ui->textBrowser_3->setFrameStyle(QFrame::NoFrame);
112    ui->textBrowser_4->setFrameStyle(QFrame::NoFrame);
113    ui->textBrowser_5->setFrameStyle(QFrame::NoFrame);
114    ui->textBrowser_6->setFrameStyle(QFrame::NoFrame);
115    ui->textBrowser_7->setFrameStyle(QFrame::NoFrame);
116    ui->textBrowser_8->setFrameStyle(QFrame::NoFrame);
117    ui->textBrowser_9->setFrameStyle(QFrame::NoFrame);
118    ui->textBrowser_10->setFrameStyle(QFrame::NoFrame);
119    ui->textBrowser_11->setFrameStyle(QFrame::NoFrame);
120    ui->textBrowser_12->setFrameStyle(QFrame::NoFrame);
121    ui->textBrowser_13->setFrameStyle(QFrame::NoFrame);
```

```
122     ui->textBrowser_14->setFrameStyle(QFrame::NoFrame);
123     ui->textBrowser_16->setFrameStyle(QFrame::NoFrame);
124     ui->textMousePosition->setFrameStyle(QFrame::NoFrame);
125     ui->textDrawSize->setFrameStyle(QFrame::NoFrame);
126 
127     /// spinBox temperatura
128     ui->spinBox_Temperature->setSingleStep(50);
129     ui->spinBox_Temperature->setMaximum(2000);
130     ui->spinBox_Temperature->setValue(300);
131 
132     /// texto do grid
133     ui->textGrid->setFrameStyle(QFrame::NoFrame);
134     ui->textGrid->setText(QString::fromStdString(std::to_string(
            currentGrid)));
135     QFont f = ui->textGrid->font();
136     f.setPixelSize(16);
137     ui->textGrid->setFont(f);
138     ui->textGrid->setAlignment(Qt::AlignCenter);
139 
140     /// lista de materiais
141     std::vector<std::string> materiais = simulador->getMateriais();
142     for (unsigned int i = 0; i < materiais.size(); i++)
143         ui->material_comboBox->addItem(QString::fromStdString(
                materiais[i]));
144 
145     ui->horizontalSliderDrawSize->setMinimum(2);
146     ui->horizontalSliderDrawSize->setMaximum(150);
147     ui->horizontalSliderDrawSize->setValue(50);
148 
149     /// lista de paralelismo
150     ui->parallel_comboBox->addItem("Paralelismo␣total");
151     ui->parallel_comboBox->addItem("Sem␣paralelismo");
152     ui->parallel_comboBox->addItem("Paralelismo␣por␣grid");
153 
154     ui->input_dt->setText(QString::fromStdString(std::to_string(
            simulador->getDelta_t())));
155     ui->input_dx->setText(QString::fromStdString(std::to_string(
            simulador->getDelta_x())));
156     ui->input_dz->setText(QString::fromStdString(std::to_string(
            simulador->getDelta_z())));
157 
158     createWidgetProps();
```

```
159 }
160
161 void MainWindow::createWidgetProps(){
162     /// scroll com os materiais para o grafico
163     std::vector<std::string> materiais = simulador->getMateriais();
164     checkboxes = new QWidget(ui->scrollArea);
165     layout = new QVBoxLayout(checkboxes);
166     myCheckbox.resize(materiais.size());
167     selectedMateriails.resize(materiais.size(), false);
168     QString qss;
169     for(unsigned int i = 0; i < materiais.size(); i++){
170         myCheckbox[i] = new QCheckBox(QString::fromStdString(
              materiais[i]), checkboxes);
171         qss = QString("background-color: %1").arg(simulador->
              getColor(materiais[i]).name(QColor::HexArgb));
172         myCheckbox[i]->setStyleSheet(qss);
173         layout->addWidget(myCheckbox[i]);
174     }
175     ui->scrollArea->setWidget(checkboxes);
176     makePlotMatProps();
177 }
178
179 void MainWindow::paintEvent(QPaintEvent *e) {
180     QPainter painter(this);
181     *mImage = paint(currentGrid);
182     painter.drawImage(left_margin,up_margin, *mImage);
183     e->accept();
184 }
185
186 QImage MainWindow::paint(int grid) {
187     QImage img = QImage(size_x*2+space_between_draws, size_y,QImage
          ::Format_ARGB32_Premultiplied);
188
189     /// desenho da temperatura
190     for (int i = 0; i < size_x; i++){
191         for (int k = 0; k < size_y; k++){
192             if (!simulador->grid[grid]->operator()(i, k)->active)
193                 img.setPixelColor(i+size_x+space_between_draws,k,
                      QColor::fromRgb(255,255,255));
194             else
195                 img.setPixelColor(i+size_x+space_between_draws,k,
                      calcRGB(simulador->grid[grid]->operator()(i, k)
```

```
                    ->temp));
196         }
197     }
198
199     if ((studyPoint.x() > 0 && studyPoint.x() < size_x) && (
            studyPoint.y() > 0 || studyPoint.y() < size_y) && grid ==
            studyGrid){
200         for(int i = 0; i < size_x; i++)
201             img.setPixelColor(i+size_x+space_between_draws,
                    studyPoint.y(), QColor::fromRgb(0,0,0));
202         for(int i = 0; i < size_y; i++)
203             img.setPixelColor(studyPoint.x()+size_x+
                    space_between_draws, i, QColor::fromRgb(0,0,0));
204     }
205
206     /// desenho dos materiais
207     for (int i = 0; i < size_x; i++){
208         for (int k = 0; k < size_y; k++){
209             if (!simulador->grid[grid]->operator()(i, k)->active)
210                 img.setPixelColor(i,k, QColor::fromRgb(255,255,255)
                        );
211             else
212                 img.setPixelColor(i,k, simulador->grid[grid]->
                        operator()(i, k)->material->getColor());
213         }
214     }
215     return img;
216 }
217
218 QColor MainWindow::calcRGB(double temperatura){
219     double maxTemp = simulador->getTmax();
220     double minTemp = simulador->getTmin();
221     return QColor::fromRgb(255, (maxTemp - temperatura)*255/(
            maxTemp - minTemp), 0, 255);
222 }
223
224 void MainWindow::runSimulator(){
225     simulador->setDelta_t(std::stod(ui->input_dt->text().
            toStdString()));
226     simulador->setDelta_x(std::stod(ui->input_dx->text().
            toStdString()));
227     simulador->setDelta_z(std::stod(ui->input_dz->text().
```

```
        toStdString()));
228
229     time_t start_time = std::time(0);
230     std::string type = ui->parallel_comboBox->currentText().
        toStdString();
231     if(type == "Sem␣paralelismo")
232         simulador->run_sem_paralelismo();
233     if(type=="Paralelismo␣por␣grid")
234         simulador->run_paralelismo_por_grid();
235     if(type=="Paralelismo␣total")
236         simulador->run_paralelismo_total();
237     time.append((time.size()+1)*simulador->getDelta_t());
238
239     std::string result = "Time:␣" + std::to_string(time[time.size()
        -1]) + "␣-␣duracao␣do␣solver:␣" + std::to_string(std::time
        (0) - start_time) + "␣seg";
240     ui->textBrowser_3->setText(QString::fromStdString(result));
241
242     update();
243     makePlot1();
244     makePlot2();
245     makePlot3();
246     makePlot4();
247 }
248
249 void MainWindow::timerEvent(QTimerEvent *e){
250     Q_UNUSED(e);
251     if (runningSimulator)
252         runSimulator();
253     makePlotMatProps();
254     printPosition();
255     printDrawSize();
256 }
257
258 void MainWindow::on_pushButton_clicked()
259 {
260     runningSimulator = runningSimulator?false:true;
261 }
262
263 void MainWindow::on_gridDownButton_clicked()
264 {
265     currentGrid--;
```

```
266     if (currentGrid < 0)
267         currentGrid = 0;
268     /// texto do grid
269     ui->textGrid->setText(QString::fromStdString(std::to_string(
            currentGrid)));
270     ui->textGrid->setAlignment(Qt::AlignCenter);
271     update();
272 }
273
274 void MainWindow::on_gridUpButton_clicked()
275 {
276     currentGrid++;
277     if (currentGrid > simulador->getNGRIDS()-1)
278         currentGrid = simulador->getNGRIDS()-1;
279     /// texto do grid
280     ui->textGrid->setText(QString::fromStdString(std::to_string(
            currentGrid)));
281     ui->textGrid->setAlignment(Qt::AlignCenter);
282     update();
283 }
284
285 void MainWindow::makePlot1(){
286     temperature.append(simulador->grid[studyGrid]->operator()(
            studyPoint.x(), studyPoint.y())->temp);
287
288     ui->plot1->graph(0)->setData(time,temperature);
289     ui->plot1->xAxis->setRange(time[0], time[time.size()-1]+1);
290     ui->plot1->yAxis->setRange(simulador->getTmin()-50, simulador->
            getTmax()+50);
291     ui->plot1->replot();
292     ui->plot1->update();
293 }
294
295 void MainWindow::makePlot2(){
296     QVector<double> temperature_z(simulador->getNGRIDS());
297     QVector<double> labor_z(simulador->getNGRIDS());
298     for (int i = 0; i < simulador->getNGRIDS(); i++){
299         labor_z[i] = simulador->getDelta_z()*(i+1);
300         temperature_z[i] = simulador->grid[i]->operator()(
                studyPoint.x(), studyPoint.y())->temp;
301     }
302
```

```
303     ui->plot2->graph(0)->setData(labor_z,temperature_z);
304     ui->plot2->xAxis->setRange(labor_z[0], labor_z[labor_z.size()
            -1]);
305     ui->plot2->yAxis->setRange(simulador->getTmin()-50, simulador->
            getTmax()+50);
306     ui->plot2->replot();
307     ui->plot2->update();
308 }
309
310 void MainWindow::makePlot3(){
311     QVector<double> temperature_x(size_x);
312     QVector<double> labor_x(size_x);
313     for (int i = 0; i < size_x; i++){
314         labor_x[i] = simulador->getDelta_x()*(i+1);
315         temperature_x[i] = simulador->grid[studyGrid]->operator()(i
                , studyPoint.y())->temp;
316     }
317
318     ui->plot3->graph(0)->setData(labor_x,temperature_x);
319     ui->plot3->xAxis->setRange(labor_x[0], labor_x[size_x-1]);
320     ui->plot3->yAxis->setRange(simulador->getTmin()-50, simulador->
            getTmax()+50);
321     ui->plot3->replot();
322     ui->plot3->update();
323 }
324
325 void MainWindow::makePlot4(){
326     QVector<double> temperature_y(size_y);
327     QVector<double> labor_y(size_y);
328     for (int i = 0; i < size_y; i++){
329         labor_y[i] = simulador->getDelta_x()*(i+1);
330         temperature_y[i] = simulador->grid[studyGrid]->operator()(
                studyPoint.x(), i)->temp;
331     }
332
333     ui->plot4->graph(0)->setData(labor_y,temperature_y);
334     ui->plot4->xAxis->setRange(labor_y[0], labor_y[size_y-1]);
335     ui->plot4->yAxis->setRange(simulador->getTmin()-50, simulador->
            getTmax()+50);
336     ui->plot4->replot();
337     ui->plot4->update();
338 }
```

```
339
340 void MainWindow::makePlotMatProps(){
341     bool changeState = checkChangeMaterialsState();
342     if (!changeState)
343         return;
344     int nPoints = 100;
345     QVector<double> props(nPoints);
346     QVector<double> temperature_x(nPoints);
347     std::vector<std::string> materiais = simulador->getMateriais();
348     double max_props = 600;
349
350     double dT = (simulador->getTmax() - simulador->getTmin())/(
        nPoints-1);
351     for (unsigned int mat = 0; mat < materiais.size(); mat++){
352         if (selectedMateriails[mat]){
353             for (int i = 0; i < nPoints; i++){
354                 temperature_x[i] = dT*i + simulador->getTmin();
355                 props[i] = simulador->getProps(temperature_x[i],
                    materiais[mat]);
356             }
357         ui->plot_MatProps->graph(mat)->setPen(QPen(simulador->
            getColor(materiais[mat])));
358         ui->plot_MatProps->graph(mat)->setData(temperature_x,props)
            ;
359         for (int i = 0; i < nPoints; i++)
360             max_props = max_props < props[i]? props[i] : max_props;
                /// aqui ajusto o ylabel
361         }else{
362             ui->plot_MatProps->graph(mat)->data()->clear();
363         }
364     }
365     ui->plot_MatProps->xAxis->setRange(temperature_x[0],
        temperature_x[nPoints-1]);
366     ui->plot_MatProps->yAxis->setRange(0, max_props);
367
368     ui->plot_MatProps->replot();
369     ui->plot_MatProps->update();
370 }
371
372 bool MainWindow::checkChangeMaterialsState(){
373     bool change = false;
374     bool temp = false;
```

```
375     for (unsigned int i = 0; i<selectedMateriails.size(); i++){
376         temp = myCheckbox[i]->checkState();
377         if (!(selectedMateriails[i] == temp)){
378             change = true;
379             selectedMateriails[i] = temp;
380         }
381     }
382     return change;
383 }
384
385 void MainWindow::on_actionSave_triggered()
386 {
387     QDir dir; QString path = dir.absolutePath();
388     QString file_name = QFileDialog::getSaveFileName(this, "Save a
        file", path+"//save", tr("Dados (*.dat)"));
389     std::string txt = simulador->saveGrid(file_name.toStdString());
390     ui->textBrowser_3->setText(QString::fromStdString(txt));
391 }
392
393
394 void MainWindow::on_actionOpen_triggered()
395 {
396     QDir dir; QString path = dir.absolutePath();
397     QString file_name = QFileDialog::getOpenFileName(this, "Open a
        file", path+"//save", tr("Dados (*.dat)"));
398     std::string txt = simulador->openGrid(file_name.toStdString());
399     ui->textBrowser_3->setText(QString::fromStdString(txt));
400 }
401
402 void MainWindow::on_actionNew_triggered()
403 {
404     simulador->resetGrid();
405     update();
406 }
407
408
409 void MainWindow::on_actionExport_pdf_triggered()
410 {
411     QString file_name = QFileDialog::getSaveFileName(this, "Save
        report as", "C://Users", tr("Dados (*.pdf)"));
412     QString txt = save_pdf(file_name);
413     ui->textBrowser_3->setText(txt);
```

```
414 }
415
416 void MainWindow::on_actionImport_material_triggered() {
417     QString file_name = QFileDialog::getOpenFileName(this, "Open a
        file", "C://Users//nicholas//Desktop//ProjetoEngenharia//
        Projeto-TCC-SimuladorDifusaoTermica//SimuladorTemperatura//
        materiais", tr("Dados (*.txt)"));
418     std::string name = simulador->openMaterial(file_name.
        toStdString());
419     ui->textBrowser_3->setText(QString::fromStdString("Material "+
        name+" carregado!"));
420     ui->material_comboBox->addItem(QString::fromStdString(name));
421
422     createWidgetProps();
423 }
424
425 void MainWindow::on_buttonCircle_clicked()
426 {
427
428     ui->widget_buttonCircle->setStyleSheet(      "border-width: 1;"
429                                                  "border-radius: 15;
                                                     "
430                                                  "border-style:
                                                     solid;"
431                                                  "border-color: rgb
                                                     (255,0,0)");
432     ui->widget_buttonSquare->setStyleSheet(      "border-width: 0;"
433                                                  "border-radius: 0;"
434                                                  "border-style:
                                                     solid;"
435                                                  "border-color: rgb
                                                     (255,0,0)");
436     drawFormat = "circulo";
437 }
438
439
440 void MainWindow::on_buttonSquare_clicked()
441 {
442     ui->widget_buttonSquare->setStyleSheet(      "border-width: 1;"
443                                                  "border-radius: 0;"
444                                                  "border-style:
                                                     solid;"
```

```
445                                                 "border-color: rgb
                                                       (255,0,0)");
446     ui->widget_buttonCircle->setStyleSheet(     "border-width: 0;"
447                                                 "border-radius: 15;
                                                       "
448                                                 "border-style:
                                                       solid;"
449                                                 "border-color: rgb
                                                       (255,0,0)");
450     drawFormat = "quadrado";
451
452 }
453
454
455 void MainWindow::on_buttonEraser_clicked()
456 {
457     if (eraserActivated){
458         ui->widget_eraser->setStyleSheet("border-width: 0;"
459                                          "border-radius: 0;"
460                                          "border-style: solid;"
461                                          "border-color: rgb
                                                (255,0,0)");
462     eraserActivated = false;
463     }
464     else{
465         ui->widget_eraser->setStyleSheet("border-width: 1;"
466                                          "border-radius: 5;"
467                                          "border-style: solid;"
468                                          "border-color: rgb
                                                (255,170,100)");
469         eraserActivated = true;
470     }
471 }
472
473 QString MainWindow::save_pdf(QString file_name){
474
475     QPdfWriter writer(file_name);
476     writer.setPageSize(QPageSize::A4);
477     writer.setPageMargins(QMargins(30, 30, 30, 30));
478
479     QPrinter pdf;
480     pdf.setOutputFormat(QPrinter::PdfFormat);
```

```
481     pdf.setOutputFileName(file_name);
482 
483     QPainter painterPDF(this);
484     if (!painterPDF.begin(&pdf))        //Se nao conseguir abrir o
        arquivo PDF ele nao executa o resto.
485         return "Erro ao abrir PDF";
486 
487 
488     painterPDF.setFont(QFont("Arial", 8));
489     painterPDF.drawText(40,140, "==> PROPRIEDADES DO GRID <==");
490     painterPDF.drawText(40,160, "Delta x: " + QString::number(
        simulador->getDelta_x())+" m");
491     painterPDF.drawText(40,180, "Delta z: " + QString::number(
        simulador->getDelta_z())+" m");
492     painterPDF.drawText(40,200, "Delta t: " + QString::number(
        simulador->getDelta_t())+" s");
493 
494     painterPDF.drawText(40,240, "Largura total horizontal: " +
        QString::number(simulador->getDelta_x()*size_x)+" m");
495     painterPDF.drawText(40,260, "Largura total vertical: " +
        QString::number(simulador->getDelta_x()*size_y)+" m");
496     painterPDF.drawText(40,280, "Largura total entre perfis (eixo z
        ): " + QString::number(simulador->getDelta_z()*simulador->
        getNGRIDS())+" m");
497 
498 
499 
500     painterPDF.drawText(400,140, "==> PROPRIEDADES DA SIMULACAO <==
        ");
501     painterPDF.drawText(400,160, "Temperatura maxima: " + QString::
        number(simulador->getTmax())+" K");
502     painterPDF.drawText(400,180, "Temperatura minima: " + QString::
        number(simulador->getTmin())+" K");
503     painterPDF.drawText(400,200, "Tempo maximo: " + QString::number
        (time[time.size()-1])+" s");
504 
505     painterPDF.drawText(400,240, "Tipo de paralelismo: " + ui->
        parallel_comboBox->currentText());
506     painterPDF.drawText(400,260, "Coordenada do ponto de estudo (x,
        y,z): " + QString::number(studyPoint.x()*simulador->
        getDelta_x())+","+QString::number(studyPoint.y()*simulador->
        getDelta_x())+","+QString::number(studyGrid*simulador->
```

```
        getDelta_z()));
507
508     /// print dos 4 desenhos
509     painterPDF.setPen(Qt::blue);
510     painterPDF.setRenderHint(QPainter::LosslessImageRendering);
511     int startDraw_x  = 40;
512     int startDraw_y  = 300;
513     int space_draw_x = 40;
514     int space_draw_y = 30;
515     int d = 5;
516     painterPDF.setFont(QFont("Arial", 8));
517
518     painterPDF.drawPixmap(startDraw_x, startDraw_y, (size_x*2+
            space_between_draws)/2, size_y/2, ui->plot1->toPixmap());
519     QRect retangulo5(startDraw_x-d, startDraw_y-d, (size_x*2+
            space_between_draws)/2+2*d, size_y/2+2*d);
520     painterPDF.drawRoundedRect(retangulo5, 2.0, 2.0);
521
522     painterPDF.drawPixmap((size_x*2+space_between_draws)/2+
            startDraw_x+space_draw_x, startDraw_y, (size_x*2+
            space_between_draws)/2, size_y/2, ui->plot2->toPixmap());
523     QRect retangulo6((size_x*2+space_between_draws)/2+startDraw_x+
            space_draw_x-d, startDraw_y-d, (size_x*2+space_between_draws
            )/2+2*d, size_y/2+2*d);
524     painterPDF.drawRoundedRect(retangulo6, 2.0, 2.0);
525
526     painterPDF.drawPixmap(startDraw_x, size_y/2+startDraw_y+
            space_draw_y, (size_x*2+space_between_draws)/2, size_y/2, ui
            ->plot3->toPixmap());
527     QRect retangulo7(startDraw_x-d, size_y/2+startDraw_y+
            space_draw_y-d, (size_x*2+space_between_draws)/2+2*d, size_y
            /2+2*d);
528     painterPDF.drawRoundedRect(retangulo7, 2.0, 2.0);
529
530     painterPDF.drawPixmap((size_x*2+space_between_draws)/2+
            startDraw_x+space_draw_x, size_y/2+startDraw_y+space_draw_y,
             (size_x*2+space_between_draws)/2, size_y/2, ui->plot4->
            toPixmap());
531     QRect retangulo8((size_x*2+space_between_draws)/2+startDraw_x+
            space_draw_x-d, size_y/2+startDraw_y+space_draw_y-d, (size_x
            *2+space_between_draws)/2+2*d, size_y/2+2*d);
532     painterPDF.drawRoundedRect(retangulo8, 2.0, 2.0);
```

```
533
534     painterPDF.drawPixmap(startDraw_x, size_y+startDraw_y+
            space_draw_y*2, (size_x*2+space_between_draws*2), size_y/2,
            ui->widget_props->grab());
535
536
537     startDraw_y = 100;
538     space_draw_y = 50;
539
540     for (int i = 0; i<simulador->getNGRIDS(); i++){
541         if (i%6 == 0){
542             startDraw_y = 100;
543             writer.newPage();
544             pdf.newPage();
545         }
546         if (i%2 == 0){
547             painterPDF.drawText(startDraw_x+size_x/2, startDraw_y-d
                    -8, "Grid␣"+QString::number(i));
548             painterPDF.drawPixmap(startDraw_x, startDraw_y, (size_x
                    *2+space_between_draws)/2, size_y/2, QPixmap::
                    fromImage(paint(i)));
549             QRect retangulo1(startDraw_x-d, startDraw_y-d, (size_x
                    *2+space_between_draws)/2+2*d, size_y/2+2*d);
550             painterPDF.drawRoundedRect(retangulo1, 2.0, 2.0);
551         }
552         else {
553             painterPDF.drawText(startDraw_x+space_draw_x+size_x+
                    size_x/2+4*d, startDraw_y-d-8, "Grid␣"+QString::
                    number(i));
554             painterPDF.drawPixmap((size_x*2+space_between_draws)/2+
                    startDraw_x+space_draw_x, startDraw_y, (size_x*2+
                    space_between_draws)/2, size_y/2, QPixmap::fromImage
                    (paint(i)));
555             QRect retangulo2((size_x*2+space_between_draws)/2+
                    startDraw_x+space_draw_x-d, startDraw_y-d, (size_x
                    *2+space_between_draws)/2+2*d, size_y/2+2*d);
556             painterPDF.drawRoundedRect(retangulo2, 2.0, 2.0);
557             startDraw_y+=size_y/2+space_draw_y;
558         }
559     }
560     return "PDF␣salvo!";
561 }
```

```
562
563
564 void MainWindow::on_gridAddGrid_clicked()
565 {
566     simulador->addGrid();
567     currentGrid = simulador->getNGRIDS()-1;
568
569     /// texto do grid
570     ui->textGrid->setText(QString::fromStdString(std::to_string(
        currentGrid)));
571     ui->textGrid->setAlignment(Qt::AlignCenter);
572     update();
573
574 }
575
576 void MainWindow::on_gridDelGrid_clicked()
577 {
578     if (simulador->getNGRIDS() > 1){
579         simulador->delGrid(currentGrid);
580         currentGrid = currentGrid==0? 0:currentGrid-1;
581     }
582
583     /// texto do grid
584     ui->textGrid->setText(QString::fromStdString(std::to_string(
        currentGrid)));
585     ui->textGrid->setAlignment(Qt::AlignCenter);
586     update();
587 }
588
589 void MainWindow::on_button3D_clicked(){
590     C3D *newWindow = new C3D(simulador);
591     //C3D *newWindow = new C3D();
592     newWindow->show();
593 }
```

Apresenta-se na listagem **??** o arquivo de cabeçalho da classe C3D.

Listing 5.4: Arquivo de implementação da classe C3D.

```
1 #ifndef C3D_H
2 #define C3D_H
3
4 #include <QMainWindow>
5 #include <QPainter>
```

```
6 #include <QPaintEvent>
7 #include <QVector>
8 #include <math.h>
9 //#include <QPoint>
10 #include <QMouseEvent>
11 //#include <QPolygon>
12 #include <omp.h>
13 #include <algorithm>
14
15 #include "CSimuladorTemperatura.h"
16
17 QT_BEGIN_NAMESPACE
18 namespace Ui { class C3D; }
19 QT_END_NAMESPACE
20
21 class C3D : public QMainWindow
22 {
23     Q_OBJECT
24
25 public:
26     C3D( QWidget *parent = nullptr);
27     C3D( CSimuladorTemperatura *simulador, QWidget *parent =
          nullptr);
28     ~C3D();
29 protected:
30     void paintEvent(QPaintEvent *event) override;
31
32     void timerEvent(QTimerEvent *e) override;
33     QVector3D rotate(QVector3D a);
34     QColor getRGB(int z);
35     void keyPressEvent(QKeyEvent *event) override;
36     void mousePressEvent(QMouseEvent *e) override;
37     void mouseReleaseEvent(QMouseEvent *e) override;
38     void mouseMoveEvent(QMouseEvent *e) override;
39
40     void minimizeAngles();
41     void createPoints();
42     void createTriangles();
43
44     QVector<bool> edges(int i, int j, int g);
45     QVector<QVector3D> createCube(QVector3D point);
46     QVector3D produtoVetorial(QVector3D origem, QVector3D a,
```

```
        QVector3D b);
47
48 private:
49     QPoint mousePos;
50     int timerId;
51     QImage *mImage;
52     int size_x, size_y;
53     double angle_x = 0.0;
54     double angle_y = 0.0;
55     double angle_z = 0.0;
56     double distance = 1.0;
57     int margin_x = 250;
58     bool mousePress = false;
59     int margin_y = 250;
60     int size;
61     int MAX_THREADS = omp_get_max_threads()-5;
62     const float PI = 3.141592;
63     double dx = 1, dy = 1, dz = 2;
64     QVector<QVector<QVector3D>> cube;
65     QVector<QVector<bool>> activeEdges;
66     QVector<QColor> colors;
67     QVector<QVector3D> drawCube;
68     QVector<QVector3D> triangles;
69     CSimuladorTemperatura *simulador;
70
71 };
72 #endif // MAINWINDOW_H
```

Apresenta-se na listagem 5.5 implementação da classe C3D.

Listing 5.5: Arquivo de implementação da função `main()`.

```
1 #include "C3D.h"
2 C3D::C3D(QWidget *parent)
3     : QMainWindow(parent)
4 {
5     //ui->setupUi(this);
6     this->setFixedSize(800,800);
7     this->adjustSize();
8     size_x = 500;
9     mImage = new QImage(size_x, size_y,QImage::
          Format_ARGB32_Premultiplied);
10    timerId = startTimer(0);
11
```

```
12      QVector3D point(0,0,0);
13      cube.push_back(createCube(point));
14
15      createTriangles();
16      drawCube.resize(8);
17      update();
18  }
19
20  C3D::C3D(CSimuladorTemperatura *_simulador, QWidget *parent)
21      : QMainWindow(parent)
22  {
23      simulador = _simulador;
24      this->setFixedSize(800,800);
25      this->adjustSize();
26      size_x = 500;
27      mImage = new QImage(size_x, size_y,QImage::
            Format_ARGB32_Premultiplied);
28      timerId = startTimer(0);
29      margin_x = 400;//simulador->getWidth();
30      margin_y = 400;//simulador->getHeight();
31      std::cout<<"criando cubos"<<std::endl;
32      dx = 1;//simulador->getDelta_x();
33      dy = dx;
34      dz = 1*simulador->getDelta_z()/simulador->getDelta_x();
35      for(int g = 0; g<simulador->getNGRIDS(); g++){
36          for(int i = 0; i < simulador->grid[g]->getWidth(); i++){
37              for(int j = 0; j < simulador->grid[g]->getHeight(); j
                    ++){
38                  if (simulador->grid[g]->operator()(i,j)->active){
39                      cube.push_back(createCube(QVector3D(i,j,(g+1)*
                            dz)));
40                      activeEdges.push_back(edges(i,j,g));
41                      colors.push_back(simulador->grid[g]->operator()
                            (i,j)->material->getColor());
42                  }
43              }
44          }
45      }
46
47      std::cout<<"cubos criados"<<std::endl;
48      createTriangles();
49      drawCube.resize(8);
```

```
50      update();
51  }
52
53
54  C3D::~C3D()
55  {
56      //delete ui;
57  }
58
59  QVector<bool> C3D::edges(int i, int j, int g){
60      QVector<bool> actives(12, true);
61      int max_i = simulador->getWidth()-1;
62      int max_j = simulador->getHeight()-1;
63      int max_g = simulador->grid.size()-1;
64
65
66      if (g > 0){
67          if (simulador->grid[g-1]->operator()(i,j)->active){
68              actives[0] = false;
69              actives[1] = false;
70          }
71      }
72      if (i < max_i){
73          if (simulador->grid[g]->operator()(i+1,j)->active){
74              actives[2] = false;
75              actives[3] = false;
76          }
77      }
78      if (i > 0){
79          if (simulador->grid[g]->operator()(i-1,j)->active){
80              actives[4] = false;
81              actives[5] = false;
82          }
83      }
84      if (j < max_j){
85          if (simulador->grid[g]->operator()(i,j+1)->active){
86              actives[6] = false;
87              actives[7] = false;
88          }
89      }
90      if (g < max_g){
91          if (simulador->grid[g+1]->operator()(i,j)->active){
```

```
92              actives[8] = false;
93              actives[9] = false;
94          }
95      }
96      if (j > 0){
97          if (simulador->grid[g]->operator()(i,j-1)->active){
98              actives[10] = false;
99              actives[11] = false;
100         }
101     }
102     return actives;
103 }
104
105 void C3D::createTriangles(){
106     triangles.resize(12);
107     triangles[0]  = QVector3D( 0,1,2);
108     triangles[1]  = QVector3D( 4,2,1);
109
110     triangles[2]  = QVector3D( 1,5,4);
111     triangles[3]  = QVector3D( 7,4,5);
112
113     triangles[4]  = QVector3D( 6,3,2);
114     triangles[5]  = QVector3D( 0,2,3);
115
116     triangles[6]  = QVector3D( 4,7,2);
117     triangles[7]  = QVector3D( 6,2,7);
118
119     triangles[8]  = QVector3D( 6,7,3);
120     triangles[9]  = QVector3D( 5,3,7);
121
122     triangles[10] = QVector3D( 1,0,5);
123     triangles[11] = QVector3D( 3,5,0);
124 }
125
126 QVector<QVector3D> C3D::createCube(QVector3D point){
127     double x = point.x(), y = point.y(), z = point.z();
128
129     QVector<QVector3D> cube(8);
130     cube[0] = QVector3D( x-dx/2.0, y-dy/2.0, z-dz/2.0);
131     cube[1] = QVector3D( x+dx/2.0, y-dy/2.0, z-dz/2.0);
132     cube[2] = QVector3D( x-dx/2.0, y+dy/2.0, z-dz/2.0);
133     cube[3] = QVector3D( x-dx/2.0, y-dy/2.0, z+dz/2.0);
```

```
134     cube[4] = QVector3D( x+dx/2.0, y+dy/2.0, z-dz/2.0);
135     cube[5] = QVector3D( x+dx/2.0, y-dy/2.0, z+dz/2.0);
136     cube[6] = QVector3D( x-dx/2.0, y+dy/2.0, z+dz/2.0);
137     cube[7] = QVector3D( x+dx/2.0, y+dy/2.0, z+dz/2.0);
138     return cube;
139 }
140
141 void C3D::keyPressEvent(QKeyEvent *event){
142     if (event->key() == Qt::Key_Up){
143         margin_y+=30.0f;
144     }
145     else if (event->key() == Qt::Key_Down){
146         margin_y-=30.0f;
147     }
148     else if (event->key() == Qt::Key_Left){
149         margin_x+=30.0f;
150     }
151     else if (event->key() == Qt::Key_Right){
152         margin_x-=30.0f;
153     }
154     else if (event->key() == Qt::Key_PageUp){
155         distance*=1.1;
156     }
157     else if (event->key() == Qt::Key_PageDown){
158         distance*=0.9;
159     }
160     else if (event->key() == Qt::Key_W){
161         angle_x-=0.1;
162     }
163     else if (event->key() == Qt::Key_S){
164         angle_x+=0.1;
165     }
166     else if (event->key() == Qt::Key_D){
167         angle_y-=0.1;
168     }
169     else if (event->key() == Qt::Key_A){
170         angle_y+=0.1;
171     }
172     update();
173 }
174
175 void C3D::mousePressEvent(QMouseEvent *e){
```

```
176     mousePos = e->pos();
177     mousePress = true;
178     update();
179 }
180 void C3D::mouseReleaseEvent(QMouseEvent *e){
181     angle_y -= (e->pos().x() - mousePos.x());
182     angle_x -= (e->pos().y() - mousePos.y());
183     mousePress = false;
184     update();
185 }
186
187 void C3D::mouseMoveEvent(QMouseEvent *e){
188     if (mousePress){
189         angle_y -= (e->pos().x() - mousePos.x())/60.0;
190         angle_x+= (e->pos().y() - mousePos.y())/60.0;
191         mousePos = e->pos();
192     }
193     update();
194 }
195
196 void C3D::minimizeAngles(){
197     if(angle_x > 2.0f*PI)
198         angle_x = 0.0f;
199     if(angle_x < 0.0f)
200         angle_x = 2.0f*PI;
201
202     if(angle_y > 2.0f*PI)
203         angle_y = 0.0f;
204     if(angle_y < 0.0f)
205         angle_y = 2.0f*PI;
206
207     if(angle_z > 2.0f*PI)
208         angle_z = 0.0f;
209     if(angle_z < 0.0f)
210         angle_z = 2.0f*PI;
211 }
212
213 void C3D::paintEvent(QPaintEvent *e) {
214
215     //QPolygon triangle;
216
217     QPainter painter(this);
```

```
218     minimizeAngles();
219     QVector<QPolygon> triangulosDesenho;
220     QVector<QColor> coresDesenho;
221     QVector<std::pair<int, double>> pos_norm;
222 
223     double prodVet;
224     int a, b, c;
225     int count = 0;
226     for(int cb = 0; cb < cube.size(); cb++){
227         for(int i = 0; i < 8; i++)
228             drawCube[i] = rotate(cube[cb][i]);
229 
230         for(int r = 0; r < 12; r++){
231             if(activeEdges[cb][r]){
232                 a = triangles[r].x();
233                 b = triangles[r].y();
234                 c = triangles[r].z();
235                 prodVet = produtoVetorial(drawCube[a], drawCube[b],
                        drawCube[c]).z();
236                 if(prodVet > 0){
237                     pos_norm.push_back(std::pair(count, prodVet));
238                     count++;
239                     if(r == 0 || r == 1 || r == 8 || r == 9) ///
                            fronteiras de g
240                         coresDesenho.push_back(QColor(colors[cb].
                            red(), colors[cb].green(), colors[cb].
                            blue(), 255));
241                     else
242                         coresDesenho.push_back(QColor(QColor(colors
                            [cb].red()*0.6, colors[cb].green()*0.6,
                            colors[cb].blue()*0.6, 255)));
243                     QPolygon pol;
244                     pol << QPoint(drawCube[a].x(),drawCube[a].y())
245                         << QPoint(drawCube[b].x(),drawCube[b].y())
246                         << QPoint(drawCube[c].x(),drawCube[c].y());
247                     triangulosDesenho.push_back(pol);
248                 }
249             }
250         }
251     }
252 
253     /// organizo conforme a profundidade
```

```
254     std::sort(pos_norm.begin(), pos_norm.end(), [](auto &left, auto
            &right) {
255         return left.second > right.second;
256     });
257 
258     /// desenho na tela
259     int pos;
260     painter.setPen(QColor(0,0,0,0));
261     for(int i = 0; i<triangulosDesenho.size(); i++){
262         pos = pos_norm[i].first;
263         painter.setBrush(coresDesenho[pos]);
264         painter.drawPolygon(triangulosDesenho[pos]);
265     }
266 
267     painter.drawImage(0,0, *mImage);
268     e->accept();
269 }
270 
271 QColor C3D::getRGB(int z){
272     return QColor::fromRgb(150+z, 150+z, 150+z);
273 }
274 
275 void C3D::timerEvent(QTimerEvent *e){
276     //angle_x-=0.05;
277     //angle_y+=0.05;
278     update();
279     Q_UNUSED(e);
280 }
281 
282 QVector3D C3D::rotate(QVector3D a){
283     double A[3] = {a.x(), a.y(), a.z()};
284     double rotation[3][3];
285     double result[3] = {0,0,0};
286 
287     /// rotation in x
288     rotation[0][0] = cos(angle_z)*cos(angle_y);
289     rotation[0][1] = cos(angle_z)*sin(angle_y)*sin(angle_x)-sin(
            angle_z)*cos(angle_x);
290     rotation[0][2] = cos(angle_z)*sin(angle_y)*cos(angle_x)+sin(
            angle_z)*sin(angle_x);
291 
292     rotation[1][0] = sin(angle_z)*cos(angle_y);
```

```
293     rotation[1][1] = sin(angle_z)*sin(angle_y)*sin(angle_x)+cos(
            angle_z)*cos(angle_x);
294     rotation[1][2] = sin(angle_z)*sin(angle_y)*cos(angle_x)-cos(
            angle_z)*sin(angle_x);
295
296     rotation[2][0] = -sin(angle_y);
297     rotation[2][1] = cos(angle_y)*sin(angle_x);
298     rotation[2][2] = cos(angle_y)*cos(angle_x);
299
300     for(int i = 0;i<3; i++)
301         for(int j = 0;j<3; j++)
302                 result[i]+=A[j]*rotation[i][j];
303
304     return QVector3D((result[0]+margin_x-200)*distance,(result[1]+
            margin_y-200)*distance,result[2]*distance);
305 }
306
307 QVector3D C3D::produtoVetorial(QVector3D origem, QVector3D a,
        QVector3D b){
308     QVector3D ax = a - origem;
309     QVector3D bx = b - origem;
310     return QVector3D(ax.y()*bx.z()-ax.z()*bx.y(), -ax.x()*bx.z()+ax
            .z()*bx.x(), ax.x()*bx.y()-ax.y()*bx.x());
311 }
```

Apresenta-se na listagem 5.6 o arquivo de cabeçalho da classe CSimuladorTemperatura.

Listing 5.6: Arquivo de implementação da classe CSimuladorTemperatura.

```
1 #ifndef CSIMULADORTEMPERATURA_H
2 #define CSIMULADORTEMPERATURA_H
3
4 #include <map>
5 #include <QDir>
6 #include <omp.h>
7 #include <QPoint>
8 #include <fstream>
9 #include <iomanip>
10 #include "CMaterial.h"
11 #include <QDirIterator>
12
13 #include "CGrid.h"
14 #include "CMaterialCorrelacao.h"
```

```
15 #include "CMaterialInterpolacao.h"
16
17 class CSimuladorTemperatura {
18 private:
19     //int parallel = 0;
20     QDir dir;
21     int MAX_THREADS = omp_get_max_threads()-1;
22     int width, height;
23     bool materialPropertiesStatus = true;
24     int NGRIDS = 1;
25     const double MIN_ERRO = 1.0e-1;
26     const int MAX_ITERATION = 39;
27     double delta_x = 2.6e-4, delta_t = 5.0e-1, delta_z = 0.05;
28
29     double Tmax = 1000, Tmin = 300;
30
31     double actualTemperature = 300;
32     double actual_time = 0.0;
33     std::map<std::string, CMaterial*> materiais;
34     std::vector<std::string> name_materiais;
35
36 public:
37     std::vector<CGrid*> grid;
38 public:
39     /// --------- FUNCOES DE CRIACAO ---------
40     CSimuladorTemperatura() { createListOfMaterials();  }
41
42     void resetSize(int width, int height);
43     void resetGrid();
44
45     void createListOfMaterials();
46     CMaterial* chooseMaterialType(std::string name);
47
48     void addGrid();
49     void delGrid(int _grid);
50
51     /// --------- FUNCOES DO SOLVER ---------
52     void run_sem_paralelismo();
53     void run_paralelismo_por_grid();
54     void run_paralelismo_total();
55     void solverByGrid(int g);
56     void solverByThread(int thread_num);
```

```
57      double calculatePointIteration(int  x, int y, int g);
58
59      std::string saveGrid(std::string nameFile);
60      std::string openGrid(std::string nameFile);
61      std::string openMaterial(std::string nameFile);
62
63      /// --------- FUNCOES SET ---------
64      void setActualTemperature(double newTemperature);
65      void changeMaterialPropertiesStatus();
66      void setDelta_t(double _delta_t) { delta_t = _delta_t; }
67      void setDelta_x(double _delta_x) { delta_x = _delta_x; }
68      void setDelta_z(double _delta_z) { delta_z = _delta_z; }
69
70      /// --------- FUNCOES GET ---------
71      int getWidth(){return width;}
72      int getHeight(){return height;}
73      double getProps(double temperature, std::string material);
74      QColor getColor(std::string material);
75      int getNGRIDS() { return NGRIDS; }
76      bool getMaterialStatus() { return materialPropertiesStatus; }
77      double maxTemp();
78      double minTemp();
79      double get_ActualTemperature() { return actualTemperature; }
80
81      double getTmax() { return Tmax; }
82      double getTmin() { return Tmin; }
83
84      double getDelta_t() { return delta_t; }
85      double getDelta_x() { return delta_x; }
86      double getDelta_z() { return delta_z; }
87      double getTime() { return actual_time; }
88
89      CMaterial* getMaterial(std::string mat) { return materiais[mat
            ]; }
90
91      std::vector<std::string> getMateriais() { return name_materiais
            ; }
92  };
93  #endif
```

Apresenta-se na listagem 5.7 implementação da classe CSimuladorTemperatura.

Listing 5.7: Arquivo de implementação da função `main()`.

```
#include "CSimuladorTemperatura.h"

void CSimuladorTemperatura::resetSize(int width, int height) {
    grid.resize(NGRIDS);
    this->width = width;
    this->height = height;
    for (int i = 0; i < NGRIDS; i++)
        grid[i] = new CGrid(width, height, 0.0);
}

void CSimuladorTemperatura::resetGrid() {
    for (int i = 0; i < NGRIDS; i++)
        grid[i]->resetGrid(0.0);
}

void CSimuladorTemperatura::createListOfMaterials() {
    ///*
    std::string matName;
    QDirIterator it(dir.absolutePath()+"//materiais", {"*.txt"},
       QDir::Files, QDirIterator::Subdirectories);
    while (it.hasNext()) {
        it.next();
        matName = it.fileName().toStdString();
        materiais[matName] = chooseMaterialType(matName);
    }
    for(auto const& imap: materiais)
        name_materiais.push_back(imap.first);
}

CMaterial* CSimuladorTemperatura::chooseMaterialType(std::string
    name){
    std::ifstream file(dir.absolutePath().toStdString()+"/materiais
       //"+name);

    std::string type;
    std::getline(file, type);
    file.close();
    if (type == "correlacao")
        return new CMaterialCorrelacao(name);
    else
        return new CMaterialInterpolacao(name);
}
```

```
40
41 void CSimuladorTemperatura::addGrid(){
42     NGRIDS++;
43     grid.push_back(new CGrid(width, height, 0.0));
44 }
45
46 void CSimuladorTemperatura::delGrid(int _grid){
47     NGRIDS--;
48     grid.erase(grid.begin()+_grid);
49 }
50
51 std::string CSimuladorTemperatura::openMaterial(std::string
       nameFile){
52     std::ifstream file(nameFile);
53
54     std::string type;
55     std::string name;
56     std::getline(file, type);
57     std::getline(file, name);
58     std::cout<<name<<std::endl;
59
60     file.close();
61     if (type == "correlacao")
62         materiais[name] = new CMaterialCorrelacao(nameFile);
63     else
64         materiais[name] = new CMaterialInterpolacao(nameFile);
65     name_materiais.push_back(name);
66     return name;
67 }
68
69
70 void CSimuladorTemperatura::run_sem_paralelismo() {
71     for (int g = 0; g < NGRIDS; g++){
72         grid[g]->startIteration();
73         solverByGrid(g);
74     }
75 }
76
77 void CSimuladorTemperatura::run_paralelismo_por_grid() {
78     omp_set_num_threads(NGRIDS);
79     #pragma omp parallel
80     {
```

```
81          grid[omp_get_thread_num()]->startIteration();
82          solverByGrid(omp_get_thread_num());
83      }
84  }
85
86  void CSimuladorTemperatura::run_paralelismo_total() {
87      for (int g=0;g<NGRIDS;g++)
88          grid[g]->startIteration();
89
90      omp_set_num_threads(MAX_THREADS);
91      #pragma omp parallel
92      {
93      solverByThread(omp_get_thread_num());
94      }
95      for (int g = 0; g < NGRIDS; g++)
96          grid[g]->updateSolver();
97  }
98
99  void CSimuladorTemperatura::solverByGrid(int g) {
100     double erro = 1;
101     int iter = 0;
102     while (erro > MIN_ERRO && iter <= MAX_ITERATION) {
103         grid[g]->updateIteration(); /// atualizo temp_nu para
              calcular o erro da iteracao
104         for (int i = 0; i < grid[g]->getWidth(); i++)
105             for (int k = 0; k < grid[g]->getHeight(); k++)
106                 calculatePointIteration(i, k, g);
107         erro = grid[g]->maxErroIteration();
108         iter++;
109     }
110     grid[g]->updateSolver();
111 }
112
113 void CSimuladorTemperatura::solverByThread(int thread_num) {
114     double erro = 1, _erro;
115     int iter = 0;
116     int x, y;
117     while (erro > MIN_ERRO && iter <= MAX_ITERATION) {
118         for (int g = 0; g < NGRIDS; g++) {
119             for (int i = thread_num; i < grid[g]->getSize(); i+=
                  MAX_THREADS) {
120                 x = i % grid[g]->getWidth();
```

```
121                 y = i / grid[g]->getWidth();
122
123                 (*grid[g])(x, y)->temp_nu = (*grid[g])(x, y)->
                        temp_nup1;
124                 _erro = calculatePointIteration(x, y, g);
125                 erro = erro < _erro ? _erro : erro;
126             }
127         }
128         iter++;
129     }
130 }
131
132 double CSimuladorTemperatura::calculatePointIteration(int x, int y,
        int g) {
133     if (!(*grid[g])(x,y)->active)
134         return 0.0;
135     if ((*grid[g])(x, y)->source)
136         return 0.0;
137     float n_x = 0;
138     float n_z = 0;
139     double inf = .0, sup = .0, esq = .0, dir = .0, cima = .0, baixo
            =.0;
140     double thermalConstant;
141
142     if (y - 1 > 0) {
143         if ((*grid[g])(x, y - 1)->active) {
144             n_x++;
145             inf = (*grid[g])(x, y - 1)->temp_nup1*delta_z;
146         }
147     }
148
149     if (y + 1 < grid[g]->getHeight()) {
150         if ((*grid[g])(x, y + 1)->active) {
151             n_x++;
152             sup = (*grid[g])(x, y + 1)->temp_nup1 * delta_z;
153         }
154     }
155
156     if (x - 1 > 0) {
157         if ((*grid[g])(x - 1, y)->active) {
158             n_x++;
159             esq = (*grid[g])(x - 1, y)->temp_nup1 * delta_z;
```

```
160         }
161     }
162 
163     if (x + 1 < grid[g]->getWidth()) {
164         if ((*grid[g])(x + 1, y)->active) {
165             n_x++;
166             dir = (*grid[g])(x + 1, y)->temp_nup1 * delta_z;
167         }
168     }
169 
170     if ( g < NGRIDS-1) {
171         if (grid[g + 1]->operator()(x, y)->active) {
172             n_z++;
173             cima = (*grid[g + 1])(x, y)->temp_nup1*delta_x;
174         }
175     }
176 
177     if (g > 0) {
178         if (grid[g - 1]->operator()(x, y)->active) {
179             n_z++;
180             baixo = (*grid[g - 1])(x, y)->temp_nup1 * delta_x;
181         }
182     }
183 
184     thermalConstant = (*grid[g])(x, y)->material->getThermalConst
        ((*grid[g])(x, y)->temp_nup1);
185 
186     (*grid[g])(x, y)->temp_nup1 = (thermalConstant * (*grid[g])(x,
        y)->temp*delta_x*delta_z/delta_t + inf + sup + esq + dir +
        cima + baixo) / (n_x*delta_z + n_z*delta_x + thermalConstant
         *delta_x*delta_z/delta_t);
187     return (*grid[g])(x, y)->temp_nup1 - (*grid[g])(x, y)->temp_nu;
188 }
189 
190 std::string CSimuladorTemperatura::saveGrid(std::string nameFile) {
191     std::ofstream file(nameFile);
192     int sizeGrid = grid[0]->getSize();
193     file << NGRIDS <<"\n";
194     for (int g = 0; g < NGRIDS; g++) {
195         for (int i = 0; i < sizeGrid; i++) {
196             if((*grid[g])[i]->active){
197                 file << i << "␣" << g << "␣";
```

```
198                 file << (*grid[g])[i]->temp << "␣";
199                 file << (*grid[g])[i]->active << "␣";
200                 file << (*grid[g])[i]->source << "␣";
201                 file << (*grid[g])[i]->material->getName() << "\n";
202             }
203         }
204     }
205     file.close();
206     return "Arquivo␣salvo!";
207 }
208
209 std::string CSimuladorTemperatura::openGrid(std::string nameFile) {
210
211     std::ifstream file(nameFile);
212
213     std::string _name;
214     int i, g;
215     double _temperature;
216     int _active, _source;
217     std::string _strGrids;
218     std::getline(file, _strGrids);
219
220     NGRIDS = std::stoi(_strGrids);
221     grid.resize(NGRIDS);
222     for(int gg = 0; gg<NGRIDS; gg++)
223         grid[gg] = new CGrid(width, height, 0.0);
224     while(file >> i >> g >> _temperature >> _active >> _source >>
        _name){
225             grid[g]->draw(i, _temperature, _active, _source, _name)
                ;
226     }
227
228     file.close();
229     return "Arquivo␣carregado!";
230 }
231
232 void CSimuladorTemperatura::setActualTemperature(double
    newTemperature) {
233     if (newTemperature > Tmax)
234         Tmax = newTemperature;
235     if (newTemperature < Tmin)
236         Tmin = newTemperature;
```

```
237     actualTemperature = newTemperature;
238 }
239
240 void CSimuladorTemperatura::changeMaterialPropertiesStatus() {
241     materialPropertiesStatus = materialPropertiesStatus ? false :
            true;
242 }
243
244 double CSimuladorTemperatura::getProps(double temperature, std::
        string material){
245     return materiais[material]->getThermalConst(temperature);
246 }
247
248 QColor CSimuladorTemperatura::getColor(std::string material){
249     return materiais[material]->getColor();
250 }
251
252 double CSimuladorTemperatura::maxTemp() {
253     double maxErro = 0;
254     double tempErro = 0;
255     for (int i = 0; i < NGRIDS; i++) {
256         tempErro = grid[i]->maxTemp();
257         maxErro = maxErro < tempErro ? tempErro : maxErro;
258     }
259     return maxErro;
260 }
261
262 double CSimuladorTemperatura::minTemp() {
263     double minErro = 0;
264     double tempErro = 0;
265     for (int i = 0; i < NGRIDS; i++) {
266         tempErro = grid[i]->minTemp();
267         minErro = minErro > tempErro ? tempErro : minErro;
268     }
269     return minErro;
270 }
```

Apresenta-se na listagem 5.8 o arquivo de cabeçalho da classe CGrid.

Listing 5.8: Arquivo de implementação da classe CGrid.

```
1 #ifndef CGRID_HPP
2 #define CGRID_HPP
3
```

```
4#include <vector>
5#include <string>
6#include "CCell.h"
7#include <iostream>
8#include "CMaterialCorrelacao.h"
9
10class CGrid {
11private:
12    int width, height;
13    std::vector<CCell> grid;
14public:
15    CGrid() {
16        width = 0;
17        height = 0;
18    }
19
20    CGrid(int _width, int _height) : width{_width}, height{_height
        }{
21        grid.resize(width * height);
22    }
23    CGrid(int _width, int _height, double temperature) {
24        resetSize(_width, _height, temperature);
25    }
26    void resetGrid(double temperature);
27
28    void printGrid();
29    void resetSize(int _width, int _height, double temperature);
30
31    void draw_rec(int x, int y, double size, double temperature,
        bool isSourceActive, CMaterial* _material, bool eraser);
32    void draw_cir(int x, int y, double size, double temperature,
        bool isSourceActive, CMaterial* _material, bool eraser);
33    void draw(int x, double temperature, bool active, bool isSource
        , std::string _material);
34
35    int getSize() { return width * height; }
36
37    void updateIteration();
38    void updateSolver();
39    void startIteration();
40
41    int getWidth() { return width;}
```

```
42      int getHeight() { return height; }
43      double maxErroIteration();
44
45      double getTemp(int position) { return grid[position].temp_nup1;
            }
46
47      void update_Temp_nup1(int x, int y, double temp) { grid[x + y *
            width].temp_nup1 = temp; }
48
49      double maxTemp();
50      double minTemp();
51
52      bool isActive(int x){ return grid[x].active;}
53
54      CCell* operator () (int x, int y) { return &grid[y * width + x
           ]; }
55      CCell* operator [] (int x) { return &grid[x]; }
56
57  };
58  #endif
```

Apresenta-se na listagem 5.9 implementação da classe CGrid.

Listing 5.9: Arquivo de implementação da função `main()`.

```
1  #include "CGrid.h"
2
3  void CGrid::printGrid() {
4      for (int i = 0; i < width; i++) {
5          for (int k = 0; k < height; k++)
6              std::cout << grid[k * width + i].temp << "␣␣";
7          std::cout << std::endl;
8      }
9  }
10
11 void CGrid::resetSize(int _width, int _height, double temperature)
       {
12      width = _width;
13      height = _height;
14      grid.resize(width * height);
15      for (int i = 0; i < width * height; i++)
16          grid[i].temp = temperature;
17 }
18
```

```
19 void CGrid::resetGrid(double temperature) {
20     for (int i = 0; i < width * height; i++) {
21         grid[i].active = false;
22         grid[i].active = false;
23         grid[i].source = false;
24         grid[i].temp = temperature;
25         grid[i].temp_nup1 = temperature;
26         grid[i].material = new CMaterial("ar");
27     }
28 }
29
30 void CGrid::draw_rec(int x, int y, double size, double _temperature
       , bool isSourceActive, CMaterial* _material, bool eraser) {
31     int start_x = (x - size / 2 >= 0) ? x - size / 2 : 0;
32     int start_y = (y - size / 2 >= 0) ? y - size / 2 : 0;
33     int max_x   = (x + size / 2 >= width)  ? width  : x - size/2 +
          size;
34     int max_y   = (y + size / 2 >= height) ? height : y - size/2 +
          size;
35     double temperatura = eraser?0:_temperature;
36
37     for (int i = start_x; i < max_x; i++){
38         for (int k = start_y; k < max_y; k++) {
39             grid[k * width + i].active = !eraser;
40             grid[k * width + i].temp = temperatura;
41             grid[k * width + i].source = isSourceActive;
42             grid[k * width + i].material = _material;
43         }
44     }
45 }
46
47 void CGrid::draw_cir(int x, int y, double radius, double
       _temperature, bool isSourceActive, CMaterial* _material, bool
       eraser) {
48     /// vou montar um quadrado, e analisar se o cada ponto dessa
          regiao faz parte do circulo
49     int start_x = (x - (int)radius >= 0) ? ((int)x - (int)radius) :
           0;
50     int start_y = (y - (int)radius >= 0) ? ((int)y - (int)radius) :
           0;
51     int max_x   = (x + (int)radius >= width)  ? width  : ((int)x +
          (int)radius);
```

```
52      int max_y   = (y + (int)radius >= height) ? height : ((int)y +
            (int)radius);
53      double temperatura = eraser?0:_temperature;
54
55      for (int i = start_x; i < max_x; i++) {
56          for (int k = start_y; k < max_y; k++) {
57              if (((i*1.0 - x) * (i*1.0 - x) + (k*1.0 - y) * (k*1.0 -
                    y)) < radius * radius) {
58                  grid[k * width + i].active = !eraser;
59                  grid[k * width + i].temp = temperatura;
60                  grid[k * width + i].source = isSourceActive;
61                  grid[k * width + i].material = _material;
62              }
63          }
64      }
65  }
66
67  void CGrid::draw(int x, double _temperature, bool active, bool
        isSource, std::string _material) {
68      grid[x].temp = _temperature;
69      grid[x].active = active;
70      grid[x].source = isSource;
71      if (active)
72          grid[x].material = new CMaterialCorrelacao(_material+".txt"
                );
73      else
74          grid[x].material = new CMaterial("ar");
75  }
76
77  void CGrid::updateIteration() {
78      for (int i = 0; i < width * height; i++)
79          grid[i].temp_nu = grid[i].temp_nup1;
80  }
81
82  void CGrid::updateSolver() {
83      for (int i = 0; i < width * height; i++)
84          grid[i].temp = grid[i].temp_nup1;
85  }
86
87  double CGrid::maxErroIteration() {
88      double erro = 0.0;
89      double erro_posicao = 0.0;
```

```
90      for (int i = 0; i < width * height; i++) {
91          erro_posicao = grid[i].temp_nup1 - grid[i].temp_nu;
92          erro = abs(erro_posicao) > erro ? erro_posicao : erro;
93      }
94      return erro;
95  }
96
97  void CGrid::startIteration() {
98      for (int i = 0; i < width * height; i++)
99          grid[i].temp_nup1 = grid[i].temp;
100 }
101
102 double CGrid::maxTemp() {
103     double maxTemp = 0;
104     for (int i = 0; i < width * height; i++)
105         maxTemp = maxTemp < grid[i].temp ? grid[i].temp : maxTemp;
106     return maxTemp;
107 }
108
109 double CGrid::minTemp() {
110     double minTemp = 1000000;
111     for (int i = 0; i < width * height; i++)
112         minTemp = minTemp > grid[i].temp ? grid[i].temp : minTemp;
113     return minTemp;
114 }
```

Apresenta-se na listagem 5.10 o arquivo de cabeçalho da classe CCell.

Listing 5.10: Arquivo de implementação da classe CCell.

```
1  #ifndef CCELL_HPP
2  #define CCELL_HPP
3
4  #include <iostream>
5  #include "CMaterial.h"
6
7  class CCell {
8  public:
9      bool active = false;
10     bool source = false;
11     double temp = 0;
12     double temp_nu = 0;
13     double temp_nup1 = 0;
14
```

```
15     CMaterial *material;
16     friend std::ostream& operator << (std::ostream& os, const CCell
          & cell) { return os << cell.temp; }
17 };
18 #endif
```

Apresenta-se na listagem 5.11 implementação da classe CCell.

Listing 5.11: Arquivo de implementação da função `main()`.

```
1 #include "CCell.h"
```

Apresenta-se na listagem 5.12 o arquivo de cabeçalho da classe CMaterial.

Listing 5.12: Arquivo de implementação da classe CMaterial.

```
1 #ifndef CMATERIAL_HPP
2 #define CMATERIAL_HPP
3 #include <iostream>
4 #include <string>
5 #include <QColor>
6
7 class CMaterial {
8 public:
9     CMaterial(){}
10     CMaterial(std::string _name) {name = _name;}
11     virtual double getThermalConst(double T) {return 0.0*T;}
12
13     virtual QColor getColor()       { return QColor(0,0,0); }
14     virtual std::string getName()   { return name; }
15
16 protected:
17     std::string name;
18     QColor color;
19 };
20 #endif
```

Apresenta-se na listagem 5.13 implementação da classe CMaterial.

Listing 5.13: Arquivo de implementação da função `main()`.

```
1 #include "CMaterial.h"
```

Apresenta-se na listagem 5.14 o arquivo de cabeçalho da classe CMaterialCorrelacao.

Listing 5.14: Arquivo de implementação da classe CMaterialCorrelacao.

```
1 #ifndef CMATERIALCORRELACAO_H
2 #define CMATERIALCORRELACAO_H
```

```
3
4#include <iostream>
5#include <fstream>
6#include <string>
7#include <QColor>
8#include <QDir>
9
10#include "CMaterial.h"
11
12class CMaterialCorrelacao:public CMaterial {
13public:
14    CMaterialCorrelacao(std::string fileName);
15    double getThermalConst(double T);
16
17    QColor getColor()       { return color; }
18    std::string getName()   { return name; }
19
20protected:
21    std::string name;
22    QColor color;
23
24    double C0_rho, C1_rho;
25    double C0_cp, C1_cp, C2_cp;
26    double C0_k, C1_k, C2_k;
27};
28#endif
```

Apresenta-se na listagem 5.15 implementação da classe CMaterialCorrelacao.

Listing 5.15: Arquivo de implementação da função `main()`.

```
1#include "CMaterialCorrelacao.h"
2
3CMaterialCorrelacao::CMaterialCorrelacao(std::string fileName){
4    std::string strTemporaria;
5    int r, g, b, alpha;
6
7    QDir dir; std::string path = dir.absolutePath().toStdString();
8    std::ifstream file(path+"/materiais//"+fileName);
9    if (file.is_open()){
10        std::getline(file, name);
11        std::getline(file, name);
12
13        file >> r; file >> g; file >> b; file >> alpha;
```

```
14          color = QColor(r, g, b, alpha);
15
16          file >> C0_rho; file >> C1_rho;
17          file >> C0_cp;  file >> C1_cp;  file >> C2_cp;
18          file >> C0_k;   file >> C1_k;   file >> C2_k;
19      }
20      else{
21          std::ifstream file(fileName);
22          if(file.is_open()){
23              std::getline(file, name);
24              std::getline(file, name);
25
26              file >> r; file >> g; file >> b; file >> alpha;
27              color = QColor(r, g, b, alpha);
28
29              file >> C0_rho; file >> C1_rho;
30              file >> C0_cp;  file >> C1_cp;  file >> C2_cp;
31              file >> C0_k;   file >> C1_k;   file >> C2_k;
32              std::cout<<"file␣open!"<<std::endl;
33          }
34          else
35              std::cout<<"can't␣open␣file!" << std::endl;
36      }
37  }
38
39  double CMaterialCorrelacao::getThermalConst(double T) {
40      double rho = C0_rho - C1_rho * (T-298);
41      double cp  = C0_cp  + C1_cp  * T - C2_cp * T * T;
42      double k   = C0_k   + C1_k   * T + C2_k  * T * T;
43      return rho * cp / k;
44  }
```

Apresenta-se na listagem 5.16 o arquivo de cabeçalho da classe CMaterialInterpolacao.

Listing 5.16: Arquivo de implementação da classe CMaterialInterpolacao.

```
1 #ifndef CMATERIALINTERPOLACAO_H
2 #define CMATERIALINTERPOLACAO_H
3
4 #include <QDir>
5 #include <string>
6 #include <vector>
7 #include "CMaterial.h"
8 #include "CSegmentoReta.h"
```

```
9
10 class CMaterialInterpolacao :public CMaterial {
11 public:
12     CMaterialInterpolacao();
13     CMaterialInterpolacao(std::string _name);
14
15     double getThermalConst(double T);
16
17     QColor getColor()       { return color; }
18     std::string getName()   { return name; }
19
20     double getK(double T);
21 protected:
22     std::string name;
23     QColor color;
24
25 private:
26     std::vector<CSegmentoReta> retaInterpolacao;
27     double rho, cp;
28     double xmin, xmax, edx;
29
30 };
31
32 #endif // CMATERIALINTERPOLACAO_H
```

Apresenta-se na listagem 5.17 implementação da classe CMaterialInterpolacao.

Listing 5.17: Arquivo de implementação da função `main()`.

```
1 #include "CMaterialInterpolacao.h"
2
3 CMaterialInterpolacao::CMaterialInterpolacao(std::string fileName){
4     std::string strTemporaria;
5     int r, g, b, alpha;
6
7     QDir dir; std::string path = dir.absolutePath().toStdString();
8     std::ifstream file(path+"/materiais//"+fileName);
9     if (file.is_open()){
10         std::getline(file, strTemporaria);
11         std::getline(file, name);
12
13         file >> r; file >> g; file >> b; file >> alpha;
14         color = QColor(r, g, b, alpha);
15
```

```
16          file >> rho; file >> cp;
17
18          double x1, x2, y1, y2;
19          file >> x1 >> y1;
20          xmin = x1;
21          while(file >> x2 >> y2){
22              retaInterpolacao.push_back( CSegmentoReta(x1,y1,x2,y2)
                    );
23              x1 = x2;
24              y1 = y2;
25          }
26          xmax = x1;
27          edx = (xmax-xmin)/ (retaInterpolacao.size()-1);
28      }
29      else{
30          std::cout<<"can't␣open␣file!" << std::endl;
31      }
32  }
33
34  double CMaterialInterpolacao::getThermalConst(double T){
35      return rho*cp/getK(T);
36  }
37
38  double CMaterialInterpolacao::getK(double T){
39      if( T <= xmin )
40        return retaInterpolacao[0].Fx(T);
41      else if(T >= xmax)
42        return retaInterpolacao[retaInterpolacao.size()-1].Fx(T);
43      // chute inicial, et = Estimativa do Trecho de reta que atende
           valor de x.
44      int et = (T - xmin) / edx;
45      while(true){  // procura pelo trecho de reta que contempla x.
46        if( T < retaInterpolacao[et].Xmin() and et > 1 )
47          et--;
48        else if ( T > retaInterpolacao[et].Xmax() and et <
             retaInterpolacao.size()-1 )
49          et++;
50        else
51          break;
52      };
53      return retaInterpolacao[et].Fx( T );    // calculo de Fx(x).
54    }
```

Apresenta-se na listagem 5.18 o arquivo de cabeçalho da classe CSegmentoReta.

Listing 5.18: Arquivo de implementação da classe CSegmentoReta.

```
#ifndef CSegmentoReta_h
#define CSegmentoReta_h

#include <iomanip>
#include <vector>

#include "CReta.h"

/// Class CSegmentoReta, representa uma reta com intervalo xmin->
    xmax.
class CSegmentoReta : public CReta
{
private:
   double xmin = 0.0; ///< Inicio do segmento de reta.
   double xmax = 0.0; ///< Fim do segmento de reta.
   bool ok = false;     ///< Se verdadeiro, x usado esta dentro
      intervalo válido (xmin->xmax)

public:
  /// Construtor default.
  CSegmentoReta ( ) { }

  /// Construtor sobrecarregado, recebe pontos (x1,y1), (x2,y2).
  CSegmentoReta (double x1, double y1, double x2, double y2)
    :CReta(x1,y1,x2,y2),xmin{x1},xmax{x2} {}

  /// Construtor copia.
  CSegmentoReta (const CSegmentoReta& retaInterpolacao )  {
    xmin = retaInterpolacao.xmin;  xmax = retaInterpolacao.xmax;
       ok = retaInterpolacao.ok;
    x = retaInterpolacao.x;     y = retaInterpolacao.y;
    a = retaInterpolacao.a;     b = retaInterpolacao.b;
  }

  // Metodos Get/Set
  double Xmin( )           { return xmin;  }
  void Xmin(double _xmin ) { xmin = _xmin; }
  double Xmax( )           { return xmax;  }
  void Xmax(double _xmax ) { xmax = _xmax; }

```

```
38  /// Se retorno for verdadeiro, valor de y esta dentro intervalo
       xmin->xmax.
39  bool Ok()                 { return ok; }
40
41  /// Verifica se esta no intervalo de xmin->xmax.
42  bool TestarIntervalo (double _x)   { return ok = ( _x >= xmin and
        _x <= xmax)? 1:0;  }
43
44  /// Calcula valor de y = Fx(x);
45  virtual double Fx (double _x)  {
46    TestarIntervalo(_x);
47    return CReta::Fx(_x);
48  }
49
50  /// Calcula valor de y = Fx(x);
51  double operator()(double _x) { return Fx(_x); }
52
53  /// Sobrecarga operador <<, permite uso cout << reta;
54  friend std::ostream& operator<<( std::ostream& os, const
       CSegmentoReta& retaInterpolacao )  {
55    os.precision(10);
56    os<< retaInterpolacao.xmin << "␣->␣" << retaInterpolacao.xmax
57      << "␣:␣y␣=␣"<< std::setw(15) << std::setprecision(10) <<
          retaInterpolacao.a << "␣+␣"
58      << std::setw(15) << std::setprecision(10) << retaInterpolacao
          .b << "*x␣";
59    return os;
60  }
61
62  /// Sobrecarga operador >>, permite uso cin >> reta;
63  friend std::istream& operator>>( std::istream& in, CSegmentoReta&
       retaInterpolacao  )  {
64   in >> retaInterpolacao.xmin >> retaInterpolacao.xmax
65      >> retaInterpolacao.a >> retaInterpolacao.b;
66   return in;
67  }
68
69  friend class CInterpolacaoLinear;
70};
71#endif //CSegmentoReta_h
```

Apresenta-se na listagem 5.19 implementação da classe CSegmentoReta.

Listing 5.19: Arquivo de implementação da função `main()`.

```
#include "CSegmentoReta.h"
```

Apresenta-se na listagem 5.20 o arquivo de cabeçalho da classe CReta.

Listing 5.20: Arquivo de implementação da classe CReta.

```
#ifndef CReta_H
#define CReta_H

#include <sstream>
#include <iomanip>
#include <fstream>
/// Class CReta,  representa uma reta y = a + b * x.
class CReta
{
protected:
   double x = 0.0; ///< Representa valor de x.
   double y = 0.0; ///< Representa valor de y.
   double b = 0.0; ///< Representa valor de b da  equacao y = a + b
      *x; normalmente e calculado.
   double a = 0.0; ///< Representa valor de a da  equacao y = a + b
      *x; normalmente e calculado.

public:
  /// Construtor default.
  CReta ( ){ }
  /// Construtor sobrecarregado, recebe a e b.
  CReta (double _a, double _b): b{_b},a{_a}{ }

  /// Construtor sobrecarregado, recebe dados pontos (x1,y1) e (x2,
     y2).
  CReta (double x1, double y1, double x2, double y2) : b{(y2-y1)/(
     x2-x1)}, a{y1-b*x1} { }

  /// Construtor de copia.
  CReta( const CReta& reta): x{reta.x}, y{reta.y},a{reta.a}, b{reta
     .b} { }

  // Metodos Get/Set
  double X( )        { return x; }
  void X(double _x ) { x = _x;   }
  double Y( )        { return y; }
  void Y(double _y ) { y = _y;   }
```

```
33  double A( )         { return a; }
34  void A(double _a ) { a = _a;   }
35  double B( )         { return b; }
36  void B(double _b ) { b = _b;   }
37
38  /// Calcula valor de y = Fx(x);
39  virtual double Fx (double _x)          { x = _x; return y = a + b
        * x; }
40
41  /// Calcula valor de y = Fx(x);
42  double operator()(double x)            { return Fx(x); }
43
44  /// Sobrecarga operador <<, permite uso cout << reta;
45  friend std::ostream& operator<<( std::ostream& os, CReta& reta )
        {
46    os << "y␣=␣" << std::setw(10) << reta.a << "␣+␣" << std::setw
         (10) << reta.b << "*x␣";
47    return os;  }
48
49  /// Sobrecarga operador >>, permite uso cin >> reta;
50  friend std::istream& operator>>( std::istream& in, CReta& reta )
        {
51    in >> reta.a >> reta.b ;
52    return in;  }
53
54  /// Retorna string com a equacao y = a + b*x;
55  std::string Equacao()   {    std::ostringstream os;     os << *
       this;
56    return os.str();  }
57};
58#endif //CReta_H
```

Apresenta-se na listagem 5.21 implementação da classe CReta.

Listing 5.21: Arquivo de implementação da função `main()`.

```
1#include "CReta.h"
```

Referências Bibliográficas

[Herter and Lott,] Herter, T. and Lott, K. Algorithms for decomposing 3-d orthogonal matrices into primitive rotations. 17(5):517–527. 14

Índice Remissivo