3 Simuladores

3.1. Simulador 1D

O primeiro modelo computacional a ser desenvolvido foi o simulador 1D (figura 3.1), que mostra o comportamento da corrente gravitacional em apenas uma direção. Esse programa foi criado usando a linguagem de programação orientada a objetos C++ no ambiente Microsoft Visual Studio 2005. Para a elaboração da interface e os elementos gráficos 2D, utilizou-se bibliotecas geradas no Tecgraf - PUC-Rio, as mais relevantes são:

• IUP - ferramenta portável para construção de interface ao usuário (http://www.tecgraf.puc-rio.br/iup).

• CD - biblioteca gráfica 2D (http://www.tecgraf.puc-rio.br/iup).

• XY - ferramenta para plotar em gráficos x-y.

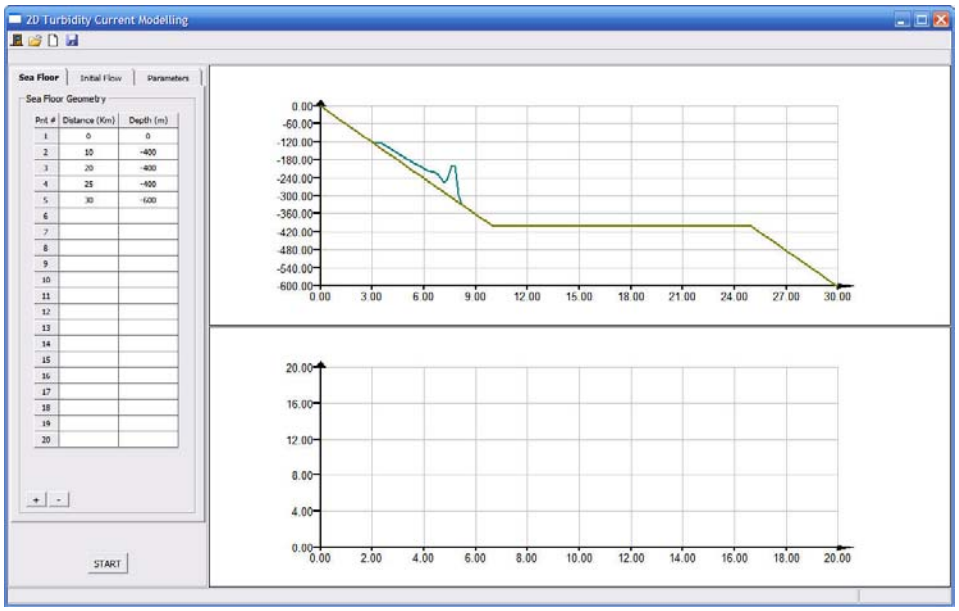

Figura 3.1 – Diálogo principal do simulador 1D

O principal objetivo em desenvolver tal simulador 1D foi aprender mais sobre equações diferencias parciais (EDP), especialmente a equação da conservação de fluxo, e como resolvê-las numericamente. Então, o método

usado para resolver essas equações foi o Método de Diferenças Finitas, que é a ferramenta matemática mais simples de resolver EDP`s.

A implementação desse método numérico para resolver a versão 1D mais simples das equações diferenciais parciais, citadas no capítulo 1, trouxe um conhecimento essencial para a implementação do algoritmo 2D. Mas se a criação do simulador 1D envolve solucionar EDP`s, o que faz ele ser mais simples que o simulador 2D? Primeiramente o simples fato de estar trabalhando com um problema 1D, e não 2D, já faz com que as equações fiquem mais fáceis de se resolver, se tem menos termos para calcular durante a resolução. Além disso, a equação da conservação de fluxo foi simplificada, adotando uma velocidade constante ao longo do tempo. Esta simplificação realmente reduziu a complexidade do problema, uma vez que não era necessário calcular novas velocidades para cada passo da simulação.

As figuras 3.2 e 3.3 ilustram como esse programa funciona. O usuário deve prover dados de entrada, sendo o primeiro deles a topografia do fundo do mar, que é fornecida preenchendo a tabela à esquerda, na figura 3.2, com as coordenadas do leito. Então, o programa interpola automaticamente tais pontos para gerar o leito do mar, representados pela linha amarela escura no primeiro gráfico da mesma figura. O segundo gráfico não é utilizado nesse ponto.

A figura 3.3 ilustra o segundo dado de entrada a ser fornecido ao programa, que é a geometria do fluxo inicial. Clicando no tab "*Initial Flow*" uma nova tabela aparece, a qual deve ser preenchida da mesma forma que a tabela do leito do mar. Ao mesmo tempo em que a tabela é preenchida, uma linha azul escura é desenhada automaticamente no primeiro gráfico da mesma figura, esta linha representa a corrente. O usuário também poderia alterar os parâmetros do fluxo, como densidade, mas nesse estágio da pesquisa procurou-se simplificar o máximo o programa, fixando tudo em código. Essa simplificação não ocorre no simulador 2D.

Após verificada a eficácia do simulador 1D, o que será explicado com detalhes na seção 4.2 do capítulo 4 de analise dos dados, duas melhorias foram feitas no programa 1D. A primeira foi a alteração da velocidade que estava sendo considerada constante no decorrer da simulação, para uma velocidade atualizada a cada passo da simulação. A outra melhoria consiste em uma nova

ferramenta, implementada para fazer uma análise pontual da corrente. Essa ferramenta também será descrita no capítulo 4 de análise na seção 4.1.

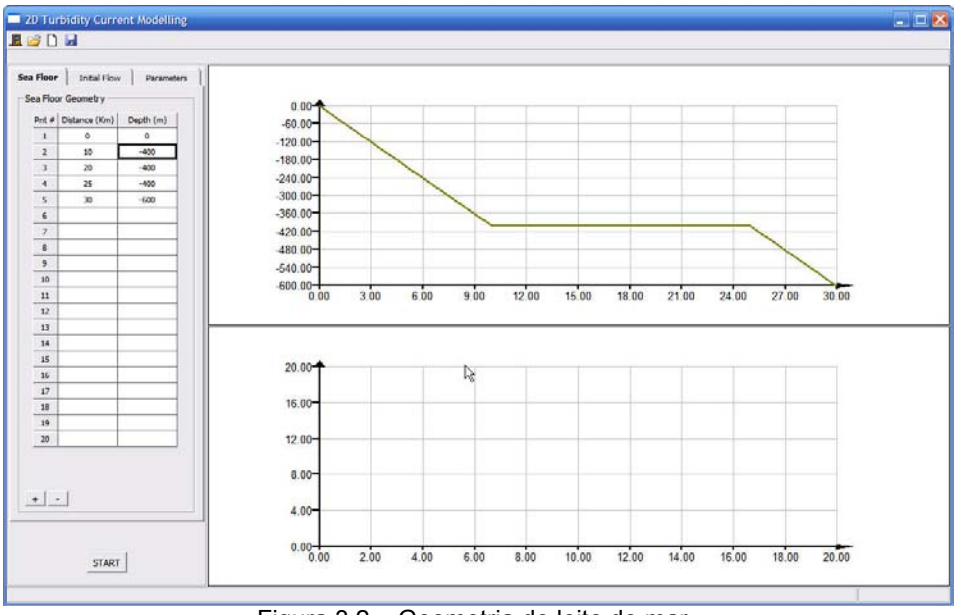

Figura 3.2 – Geometria do leito do mar

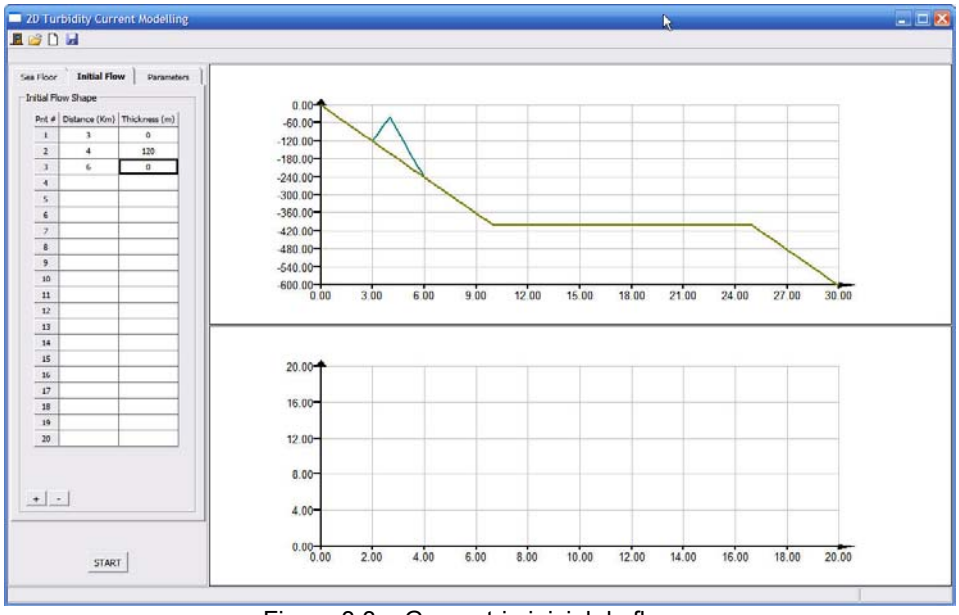

Figura 3.3 – Geometria inicial do fluxo

3.2. Simulador 2D

O simulador 2D (figura 3.4) foi gerado também com a função de resolver as equações diferenciais parciais, citadas na seção 1.3, que modelam o comportamento das correntes de turbidez, porém dessa vez sem simplificações. Desenvolveu-se o programa usando o mesmo ambiente Microsoft Visual 2005 e a mesma linguagem de programação C++. As bibliotecas do Tecgraf - PUC-Rio também continuaram a ser usadas, mas para parte gráfica 3D foi incluída a biblioteca OpenGL ("*Open Graphics Library*").

Os principais objetivos ao desenvolver-se o simulador 2D foram não somente simular correntes de turbidez, mas também reproduzir de forma mais precisa possível o tanque T usado na fase experimental. O motivo para simular ambos se deve ao foco da pesquisa, que é validar as equações do Dr. Waltham através da comparação dos resultados provenientes dos experimentos com os do simulador. Em outras palavras, procurou-se recriar os experimentos no simulador para facilitar a comparação dos resultados.

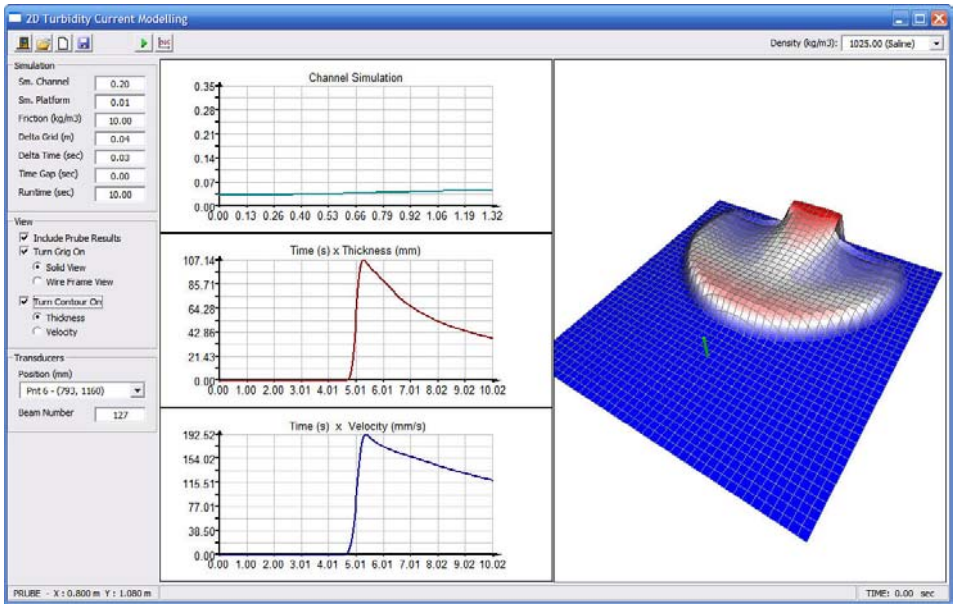

Figura 3.4 – Diálogo principal do simulador 2D

A interface do programa pode ser dividida em três partes principais: a barra de controle, o quadro de opções e a área de visualização. A barra de controle é mostrada na figura 3.5 e é responsável por gerenciar algumas funções do simulador, tais como: gerenciar arquivos (quartos primeiros botões à esquerda), começar a simulação (botão "*Play*") e comparar resultados (botão com ícone de

comparação de gráficos) e controlar a densidade da corrente ("*drop list*" a direita).

Figura 3.5 – Barra de controle

Enquanto isso, o quadro de opções pode ser subdividido em três grupos: simulação, visualização e transdutores. O grupo simulação (figura 3.6) permite o usuário configurar alguns atributos da simulação, como o tempo de duração, o fator de suavização, fator de atrito, dentre outros. Já no grupo visualização (figura 3.7) encontra-se algumas opções para melhorar a interpretação gráfica dos resultados do simulador. Uma opção interessante é o contour, que pode ser calculado tanto em função da espessura da corrente quanto da velocidade da mesma. O último grupo refere-se aos transdutores (figura 3.8), que permite o usuário escolher em qual dos 28 pontos da plataforma (figura 2.9) se deseja coletar os dados de velocidade. Esses pontos estão exatamente na mesma posição que nos experimentos, justamente para facilitar a comparação dos dados. O ponto escolhido é destacado com um pequeno bastão verde (figura 3.12).

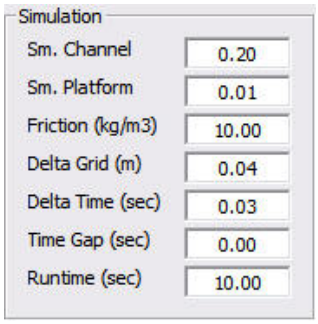

Figura 3.6 – Grupo simulação

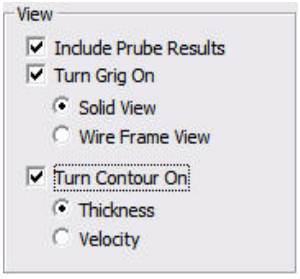

Figura 3.7 – Grupo Visualização

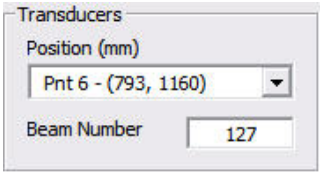

Figura 3.8 – Grupo transdutores

Indo para terceira e última parte da interface, área de visualização, encontra-se quarto canvases criados para ilustrar a simulação e mostrar alguns resultados pertinentes. O primeiro canvas (figura 3.9) ilustra a simulação da corrente enquanto confinada no canal do tanque. Como o fluxo dentro do canal pode ser modelado como um problema 1D, optou-se por uma ilustração simples num gráfico X-Y. Os próximos dois canvases mostram gráficos dos dados da corrente coletados durante a simulação, Tempo versus Espessura (figura 3.10) e Tempo versus Velocidade (figura 3.11). O último canvas (figura 3.12) ilustra a simulação numérica 2D da corrente de turbidez sobre a plataforma do tanque T.

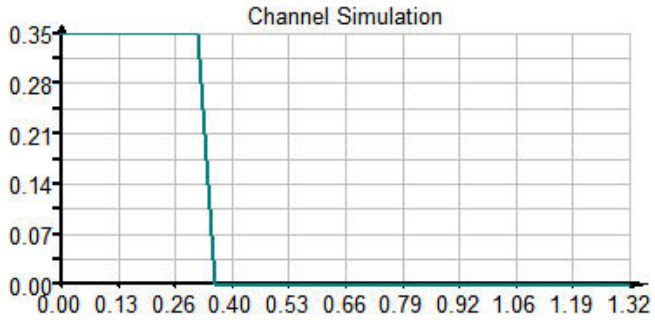

Figura 3.9 – Simulação da corrente dentro do canal

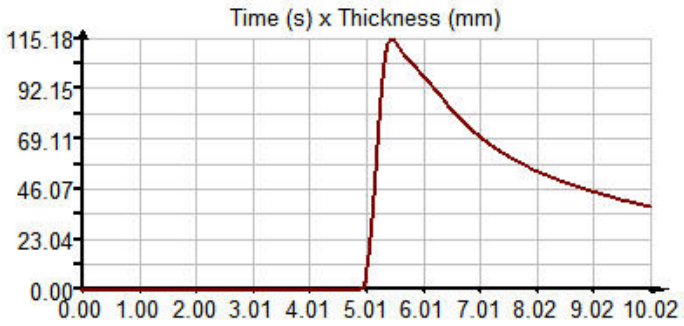

Figura 3.10 – Gráfico das espessuras coletadas na simulação

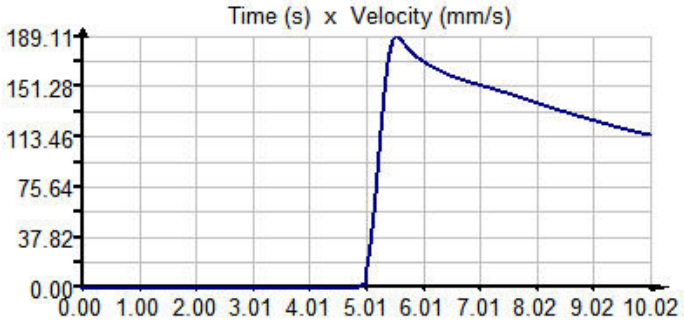

Figura 3.11 – Gráfico das velocidades coletadas na simulação

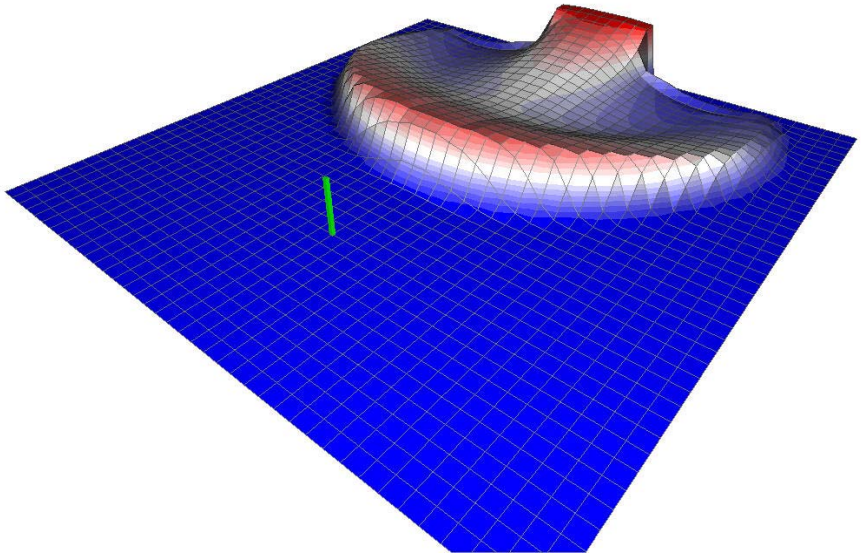

Figura 3.12 – Simulação numérica de correntes de turbidez

A execução do simulador 2D é trivial. Uma vez iniciado o programa, todos os parâmetros e atributos da simulação são carregados com valores pré-definidos através de diversos testes do programa. Então, basta o usuário pressionar o botão "*play*" para que a simulação comece. Porém, se os resultados não forem satisfatórios, é possível ajustar os parâmetros desejados e rodar uma nova simulação para se obter novos resultados. Para comparar os resultados numéricos com os experimentais, criou-se uma ferramenta capaz de comparar graficamente os dois: basta pressionar o botão à direita do botão "*play*" com o ícone de comparação de gráficos. Esta ferramenta será melhor explicada no capítulo 4 de análise dos dados.

3.3. Algoritmo

Conforme mencionado anteriormente, desenvolveu-se todo o algoritmo usando linguagem de programação orientada a objetos, C++. Com o objetivo de manter o código o mais simples possível, foram criadas apenas cinco classes:

- *Main_Dialog*
- *List*
- *Draw*
- *Current_Flow*
- *Comparison*

De uma meneira geral, as classes foram distribuídas de forma que cada uma ficasse com um papel específico dentro do programa. A classe "*Main_Dialog"* é responsável pelo gerenciamento do programa, incluindo a interface com o usuário, a entrada de dados e a própria simulação. Já a classe "*List"* foi criada com o objetivo de organizar os resultados do programa utilizando uma estrutura de dados do tipo lista encadeada (Celes, 2004). O papel designado para a classe "*Draw"*, como o nome sugere, foi o de gerenciar as rotinas de desenho da parte gráfica dos resultados da simulação 2D. Enquanto isso, a classe "*Current_Flow"*, ficou responsável por não somente armazenar informações da simulação, mas como também calcular a própria. Por último, vem a classe "*Comparison"*, que tem como objetivo gerenciar as funções de comparação dos resultados numéricos com os resultados experimentais.

A função mais importante do simulador para essa pesquisa é a "*Simulation*", a qual é disparada assim que o botão "*Play*" da interface é pressionado. Esta função é o cérebro do modelo computacional, pois nela se encontra todo o cálculo das espessuras e velocidades da corrente para cada passo da simulação, concentrando praticamente todo o esforço matemático do programa. Além disso, a "*Simulation*" também é responsável por chamar as rotinas de desenho, para a visualização da simulação.

O algoritmo da função "*Simulation*" busca solucionar numericamente as equações diferencias parciais, descritas na seção 1.3, utilizando o método das diferenças finitas de forma explícita, ou seja, partindo de valores conhecidos

para se chegar a valores desconhecidos. A idéia desse método consiste em substituir, de forma aproximada, as derivadas das EDP`s por diferenças finitas. Então, foram consideradas três formas, "*forward diference*" (14), "back*ward diference*" (15) e "*central diference*" (16), sendo suas aplicações orientadas por Dr. Waltham. Nas equações a seguir, $f`(x)$ é a derivada da função no ponto x e h é um valor não nulo, porém tendendo a zero.

$$f`(x) \cong \frac{\{f(x+h) - f(x-h)\}}{2h} \tag{14}$$

$$f`(x) \cong \frac{\{f(x+h) - f(x)\}}{h} \tag{15}$$

$$f`(x) \cong \frac{\{f(x) - f(x-h)\}}{h} \tag{16}$$

Dois outros pontos importantes no algoritmo dessa função são: as condições de contorno e a suavização dos resultados. Para as condições de contorno, foi adotata uma metodologia que consiste em utilizar a variação do penúltimo valor no último valor a ser calculado. A suavização dos resultados a cada passo da simulação se fez muito importante para a estabilidade do simulador, pois sem ela o programa se tornava instável. Se trata de uma suavização simples de três pontos, calculando a média aritimética desses pontos e multiplicando por um fator arbitrário de suavização.

Um ponto interessante é que no lugar da suavização havia sido utilizado um termo de difusão, utilizando o método de Crank-Nicholson para resolvê-lo (Press et al, 1997), porém foi constatado que a suavização, além de apresentar um esforço computacional menor, também mostrou dar mais estabilidade ao algoritmo.

No esquema a seguir, mostra-se o passo a passo do algoritmo desenvolvido para a função "*simulation*".

Passo 1:
- Inicialização geral das variáveis
- Inicialização dos vetores e matrizes de velocidade e de espessura como o valor 0.0

Passo 2:
- Criação do "*loop*" para o tempo de simulação

…
Loop (tempo de simulação)
{

Passo 2.1:
- Incrementar variável do tempo de simulação

Passo 2.2:
- Cálculo dos novos vetores de velocidade e espessura do canal

…
{

Passo 2.2.1:
- Cálculo do termo de atrito
- Cálculo do termo de pressão hidrostática → [*"Central Difference"*]
- Cálculo do termo de advecção → ["*Backward Difference*"]

Passo 2.2.2:
- Cálculo do vetor de velocidade

Passo 2.2.3:
- Cálculo do vetor de espessuras → ["*Backward Difference*"]

Passo 2.2.4:
- Aplicar condições de contorno
- Aplicar suavização

Passo 2.2.5:
- Plotar resultados

}
…

Passo 2.3:
- Conexão entre o canal e a plataforma

Passo 2.4:
- Cálculo das novas matrizes de velocidade e espessura da plataforma

…

{

Passo 2.4.1:
- Cálculo dos termos de advecção para as componentes X e Y

…
{

If (vel > 0.0) → ["*Backward Difference*"]

Else if (vel < 0.0) → ["*Forward Difference"*]

Else → [*"Central Difference"*]

}
…

Passo 2.4.2:
- Cálculo dos termos de pressão hidrostática para as componentes X eY → [Central Difference]

Passo 2.4.3:
- Cálculo dos termos de atrito para as componentes X e Y

Passo 2.4.4:
- Cálculo da matriz de velocidade

Passo 2.4.5:
- Cálculo da matriz de espessuras

…
{

If (vel > 0.0) → ["*Backward Difference*"]

Else if (vel < 0.0) → ["*Forward Difference*"]

Else → [*"Central Difference"*]

}
…

Passo 2.4.6:
- Aplicar condições de contorno

Passo 2.4.7:
- Armazenar as matrizes finais de velocidade e espessura para análise pontual

Passo 2.4.8:
- Desenhar a corrente atualizada na plataforma

}
…

}
…

Todos os passos desse esquema são importantes. Porém, podemos destacar como passo essencial a ligação do canal com a plataforma do tanque T. Conforme mostrado, para cada passo de tempo da simulação é necessário calcular, primeiramente, os parâmetros da corrente dentro do canal para depois se calcular fora do canal, ou seja, na plataforma. Essa metodologia é totalmente pertinente, pois o canal funciona como um provedor para o fluxo sobre a plataforma, alimentando o mesmo com novas velocidades e espessuras ao longo da simulação. Como resultado, o fluxo do canal influencia diretamente as condições de contorno do modelo da plataforma.

Vale ressaltar a importância da função de desenho do simulador 2D, pois se a mesma não for implementada de maneira eficaz, poderá comprometer significantemente o desempenho do algoritmo. Por isso, procurou-se utilizar funções do OpenGL que otimizassem o processamento do algoritmo, por exemplo ao passar os pontos para a placa gráfica, optou-se por passar vetores com todos os pontos a serem desenhados, ao invés de se passar ponto a ponto.

Com esta rotina de desenho, é possível não somente visualizar a simulação, mas como também fornecer opções de visualização ao usuário, como malha de discretização e mapa de contorno. A malha de discretização da plataforma auxilia o usuário com relação ao posicionamento, ou seja, saber o que acontece com cada elemento da malha ao longo da simulação. A implementação desta malha na rotina de desenho foi feita de forma muito simples, utilizando um método chamado "*Polygon Offset"*. Tal método consiste em renderizar primeiramente o plano da plataforma, com uma profundidade imperceptível, e depois renderizar as linhas da malha, sem profundidade.

Quanto ao mapa de contorno, tem-se que sua finalidade é fazer um mapeamento de propriedade, auxiliando o usuário a interpretar o que está ocorrendo com a corrente em tempo real da simulação. O mapa de contorno pode ser calculado de duas maneiras, em função da espessura do fluxo e em função da velocidade média do fluxo. Para cada passo da simulação é calculado e desenhado um novo mapa de contorno, mostrando, por exemplo, a espessura máxima da corrente no passo. Para implementação desse mapa de propriedades, fez-se uso de uma técnica de visualização do OpenGL conhecido

como aplicação de textura 1D. Essa técnica consiste, basicamente, em criar uma textura com uma escala de cores, por exemplo a figura 3.13 com 16 cores, e uma coordenada de textura, no caso S, para o mapeamento nas primitivas do OpenGL. Uma vez definido o valor máximo e mínimo da propriedade, se faz uma interpolação de 0 a 1 definindo as coordenadas de textura S, e, depois se passa todas as coordenada, de textura e do objeto, para o OpenGL fazer a associação com as cores da escala e desenhar na tela.

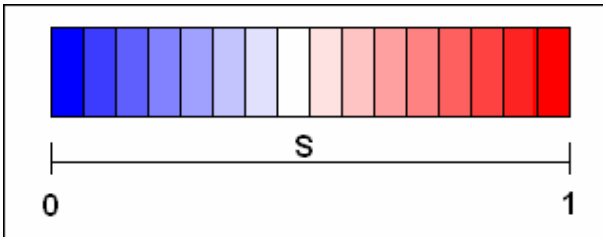

Figura 3.13 – Escala de cores da textura 1D e sua coordenada S