UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

PEDRO HENRIQUE PIANTONI BOSZCZOVSKI

DESENVOLVIMENTO DE SOFTWARE DE SIMULAÇÃO DE FLUIDOS UTILIZANDO O MÉTODO *SMOOTHED PARTICLE HYDRODYNAMICS*

CAMPO MOURÃO

2022

PEDRO HENRIQUE PIANTONI BOSZCZOVSKI

DESENVOLVIMENTO DE SOFTWARE DE SIMULAÇÃO DE FLUIDOS UTILIZANDO O MÉTODO *SMOOTHED PARTICLE HYDRODYNAMICS*

Development of fluid simulation software using the Smoothed Particle Hydrodynamics method

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Ambiental do Curso de Engenharia Ambiental da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Adilandri Mércio Lobeiro

CAMPO MOURÃO

2022

PEDRO HENRIQUE PIANTONI BOSZCZOVSKI

DESENVOLVIMENTO DE SOFTWARE DE SIMULAÇÃO DE FLUIDOS UTILIZANDO O MÉTODO *SMOOTHED PARTICLE HYDRODYNAMICS*

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Ambiental do Curso de Engenharia Ambiental da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 14/junho/2022

Adilandri Mércio Lobeiro
Doutorado
Universidade Tecnológica Federal do Paraná

Flavia Aparecida Reitz Cardoso
Doutorado
Universidade Tecnológica Federal do Paraná

Clicia Giovane Alves Pereira
Mestrado
Universidade Estadual de Maringá

CAMPO MOURÃO
2022

AGRADECIMENTOS

Agradeço a meus pais Henrique Boszczovski e Lucimara Piantoni Boszczovski, pela paciência que tiveram comigo durante o desenvolvimento deste trabalho e nova graduação, e aos meus amigos do grupo de estudo dos pês, além dos amigos de trajetória.

Ao meu professor Dr. Adilandri Mércio Lobeiro (Querido Pê) também pela paciência, durante estes cinco anos de graduação. Ele sempre me forçou, de uma forma ou outra a melhorar, e sempre abriu portas para mim. Ao professor Dr. Eudes José Arantes, agradeço muito novamente, tanto por ser meu professor como pela sabedoria dividida.

Ao querido professor Dr. Marcelo Galeazzi Caxambú pelo estágio oferecido, por ser mais que um mestre, também um amigo e por todo esforço em me melhorar como estudante e profissional.

Por fim ao professor Dr. Liu Gui-Rong por ter cedido gentilmente seu código original para este trabalho.

“Sou artista o suficiente para desenhar livremente em minha imaginação. Imaginação é mais importante que conhecimento. O conhecimento é limitado. A imaginação envolve o mundo” - Albert Einstein. Adaptado de Einstein (1929).

RESUMO

Métodos numéricos são utilizados em grande parte para resolução de problemas e fenômenos físicos naturais complexos que não possuem solução analítica. Sua abordagem de aproximação possibilita a resolução de problemas técnicos e científicos com grande fidelidade de resultados. O método “Smoothed Particle Hydrodynamics” (SPH) ou Hidrodinâmica de Partículas Suavizadas é um método cuja origem deriva do estudo de problemas astrofísicos com comportamento semelhante à de fluidos. Por isso, atualmente é uma ferramenta poderosa capaz de solucionar grandes problemas relacionados a dinâmica de fluidos, utilizado pela academia e em funções comerciais. O método é Lagrangeano e não depende de uma malha numérica, que é responsável pela leitura de informações a cada instante de tempo, mas sim de que pequenas partículas, discretizadas no domínio carreguem suas próprias informações físicas como velocidade, posição, temperatura e etc, durante o tempo necessário. Um programa foi codificado em Fortran com base no método numérico, sua validação ocorreu por meio da fórmula analítica do fluxo de Poiseuille e obteve resultados acima de noventa por cento de precisão no geral.

Palavras-chave: sph; hidrodinâmica de partículas suavizadas; método computacional; dinâmica dos fluidos computacional; fortran.

ABSTRACT

Numerical methods are largely used to solve complex natural physical problems and phenomena that do not have an analytical solution. Its approach approach enables the resolution of technical and scientific problems with high fidelity of results. The “Smoothed Particle Hydrodynamics” (SPH) method or Smoothed Particle Hydrodynamics is a method whose origin derives from the study of astrophysical problems with behavior similar to that of fluids. Therefore, it is currently a powerful tool capable of solving major problems related to fluid dynamics, used by academia and commercial functions. The method is Lagrangian and does not depend on a numerical mesh, which is responsible for reading information at each instant of time, but on the fact that small particles, discretized in the domain, carry their own physical information such as velocity, position, temperature, etc., during the necessary time. A program was coded in Fortran based on the numerical method, its validation took place using the Poiseuille flow analytical formula, and it obtained results above ninety percent of accuracy in general.

Keywords: sph; smoothed particle hydrodynamics; numerical method; computational fluid dynamics; fortran.

LISTA DE FIGURAS

LISTA DE ALGORITIMOS

*

SUMÁRIO

1 INTRODUÇÃO

O método numérico Smoothed Parcicle Hydrodynamics (SPH) tem sua origem em estudos astrofísicos e a relação e influência que corpos no espaço causam entre si. O método foi adaptado para o estudo de partículas menores, sendo nesta vertente proposta, o estudo de fluidos por meio da metodologia de Liu e Liu (2003). Matematicamente a ideia do método é não ser dependente de uma malha numérica responsável pela leitura das informações que ocorrem em cada instante de tempo em uma simulação computacional ou cálculo, mas sim que cada partícula do fluido carregue consigo todas as informações computacionais necessárias. Este método resolve e facilita vários problemas numéricos além da alta capacidade em simular deformações, (LIU; LIU, 2003).

O método é utilizado em diversos problemas, entre alguns exemplos mais comuns acadêmicos, estão listados problemas de hidráulica e fenômenos de transportes como dissipação de calor em uma placa quadrada, tanque hidrostático, quebra de barragem, mecânica quântica, colisões de estrelas, formação e colapso de galáxias, vazamentos de fluidos e etc. Há também simulações de bioengenharia como células sanguíneas, fluxo de sangue, micro-fluídos, válvulas do coração, aplicação de vacinas sem agulha, simulação de cirurgias entre outros campos (SHADLOO; OGER; TOUZÉ, 2016).

Em empresas privadas Shadloo, Oger e Touzé (2016), citam como exemplos de engenharia civil a utilização do SPH em simulação de estruturas como pontes, em empresas de engenharia aeronáutica como resistência de fuselagem de aeronaves, em indústrias automobilísticas no estudo do equilíbrio de cargas liquidas em veículos tanques e simulação de aquaplanagem, na indústria energética e petroquímica em simulações de extração de petróleo e comportamento de mistura de líquidos como água, gás e óleo bruto ou simulações de barragens.

O SPH já tem décadas de existência (JIAN *et al.*, 2015), porém sua aplicação em escala é recente devido ao aumento de capacidade computacional disponível aos pesquisadores e para aplicação industrial.

2 OBJETIVOS

Neste capitulo são apresentados o objetivo geral e os objetivos específicos ou cada etapa necessária para o sucesso deste trabalho.

2.1 Objetivo Geral

Desenvolver um software de simulação de fluidos através do método numérico SPH (Smoothed Particle Hydrodynamics) efetuando seu cálculo, modelagem matemática e programação do seu código.

2.2 Objetivos Específicos

- Levantar e estudar os dados do método, conceitos e fundamentos.
- Criar a estrutura matemática do método.
- Criar a estrutura matemática de validação.
- Programar os cálculos efetivos e suas variáveis pertinentes
- Simulação do método e testes.

3 JUSTIFICATIVA

A inovação conforme o Oslo (2005) é introdução de bens e/ou serviços novos ou a introdução de um novo processo de marketing, ou metodologia organizacional nas práticas de negócios, organização do ambiente de trabalho e relacionamentos externos. Tais inovações são dependentes das atividades de inovação ou as atividades cientificas, tecnológicas, organizacionais, financeiras e comerciais no qual uma empresa ou organização dedica seus recursos e esforços com objetivo de adquirir vantagens competitivas em relação a seus concorrentes.

O Oslo (2005), define que as fontes de informação e possíveis parcerias necessárias para que empresas alcancem seus objetivos são os conhecimentos e avanços em pesquisas, artigos acadêmicos, ambientes universitários e centros tecnológicos. Sendo a Universidade Tecnológica Federal do Paraná um dos centros de conhecimento disponíveis a sociedade e considerando sua cultura e missão em desenvolver a formação tecnológica de excelência, justifica-se esta futura pesquisa como sendo uma inovação tecnológica no qual beneficiará o aluno, sociedade e a academia.

Entre as possibilidades inovativas atuais nos ramos de engenharia, encontra-se a simulação numérica que, segundo Shadloo, Oger e Touzé (2016), graças aos avanços de capacidade de processamento dos computadores, é utilizada pelas empresas e organizações como meio de inovação e resolução de problemas. O autor ainda explica que não há método e modelo matemático universal sendo necessário o desenvolvimento de métodos exclusivos a cada necessidade e problemas encontrados.

Considerando que um dos núcleos da Engenharia Ambiental é o estudo de fluidos e relacionados como: hidrologia, hidráulica, solos, gases atmosféricos tratamento de afluentes e efluentes, sistemas de drenagem e distribuição, suas consequências, parâmetros e tecnologias necessárias (DAVIS; MASTEN, 2016), justifica-se o desenvolvimento do método como meio de auxiliar futuras pesquisas na área da engenharia ambiental além de contribuir com seu papel social de oferecer tecnologia e informações a sociedade.

4 REVISÃO DE LITERATURA

Este capitulo descreve os princípios básicos do método, da sua ideia de concepção e base, até sua forma discretizada em termos matemáticos. Há também suas características numéricas, descrição das formulas de Navier-Stokes e seu formato em SPH. Por fim uma breve descrição da história e funcionalidades básicas da linguagem Fortran. Os principais autores do método utilizados são Liu e Liu (2003).

4.1 Conceitos Básicos

A simulação numérica se trata da tradução de aspectos físicos da realidade em equações de descrição matemáticas, que quando aplicadas a computação, são capazes de, ao invés do modo tradicional experimentalista, recriar situações de forma virtual e suas soluções, considerando todos os problemas e variáveis envolvidas. É uma alternativa, a investigação cientifica através do experimentalismo real, com menor custo e maior segurança, sendo uma ponte entre teorias e fenômenos reais, (MONAGHAN, 1992).

A simulação de fluidos em superfícies livres, móveis, com geometria complexa e deformação de topografia com precisão e qualidade se torna um problema, demandando recursos e tempo, além de ser extremamente difícil. Segundo Fraga Filho (2016), para que se execute de forma consistente e com o menor uso de recursos computacionais possíveis é necessário um método numérico livre de redes e não euleriano.

Os autores Liu e Liu (2003) explicam que no método SPH o estado de um sistema é determinado por um conjunto de partículas, que possuem individualmente dados sobre suas propriedades materiais e que se movem de acordo com as equações de conservação. Sua invenção, inicialmente, tinha como objetivo o estudo de problemas astrofísicos em espaços tridimensionais abertos.

Ainda de acordo com os autores Liu e Liu (2003), o método é lagrangiano e sem malha, possuindo vantagens entre os métodos tradicionais, sendo a principal sua grande capacidade de adaptação. Tal adaptação é consequência da aproximação da variável de campo, que ocorre em cada instante de tempo baseada na atual configuração ou distribuição de partículas do momento, livrando assim que sua formulação seja influenciada pelas arbitrariedades da distribuição de partículas, sendo capaz de lidar com problemas com grande deformação.

Sua natureza sem malha também se deve à formulação adaptativa e o uso de partículas para simular o domínio, que por consequência age como uma estrutura, sem que seja necessário prover uma malha conectiva pré-determinada, as partículas, durante o processo computacional. Há uma grande harmonia entre o método lagrangiano e a aproximação de partículas, em outros métodos, livres de malha, se exige a presença de pontos usados para interpolação, já no SPH, cada partícula carrega em si informações sobre suas propriedades materiais, movendo-se

apenas com a interação de suas forças internas e externas, agindo como pontos de aproximação e componentes materiais (WANG *et al.*, 2016).

Conforme os autores Liu e Liu (2003), Shadloo, Oger e Touzé (2016), Liu e Liu (2010), o método é utilizado principalmente para solucionar problemas de hidrodinâmica, cuja as variáveis de campo, como densidade, energia, velocidade e etc, estão na forma de equações diferenciais ordinárias parciais (EDPs). Quando uma solução analítica não é possível, torna-se necessário a aplicação de métodos numéricos, no qual a primeira necessidade é a discretização do domínio do problema, sendo o método posteriormente utilizado para a aproximação dos valores das funções de campo e de suas derivadas em qualquer ponto. Posteriormente, é aplicada a função de aproximação que transforma as EDPs, em um conjunto de equações diferenciais ordinárias (EDOs), que podem então, serem solucionadas, utilizando alguma rotina padrão do método das diferenças finitas.

Ideias chave, explanadas por Liu e Liu (2003):

1. ***Meshfree***: o domínio do problema é representado por partículas arbitrariamente distribuídas, sem a necessidade de conexão entre as mesmas. Isso determina a natureza sem malha do método.

2. **Representação da função integral**: a representação integral do método é utilizada para a aproximação da função de campo, ou como é chamado no SPH, a aproximação pelo Kernel ou função núcleo. Tal ideia providencia a estabilidade matemática necessária ao método.

3. **Suporte compacto**: a aproximação pela função núcleo é executada posteriormente usando-se partículas. Isso é feito substituindo a integração, na representação da integral da função de campo e seus derivados, com somatórios em todos os valores correspondentes das partículas vizinhas em um domínio local ou domínio de suporte, como é chamado no método. O resultado final é a produção de um sistema de matriz entrelaçados ou espaçados extremamente importantes em termos de desempenho computacional.

4. **Adaptativo**: a aproximação das partículas é feita e executada em todo passo de tempo, e o uso de cada uma depende da distribuição local das mesmas. Isso é o que proveem o método a sua adaptabilidade. A cada instante de tempo a aproximação de partículas em seus respectivos domínios de suporte em estágios iniciais, auxilia de modo que sua formulação não seja alterada posteriormente.

5. **Lagrangiano**: a aproximação é performada em todos os termos com relação as funções de campo, nas EDPs com formato discretizado em relação apenas ao tempo.

6. **Dinâmico**: as EDOs são resolvidas usando um algorítimo explícito de integração, o que permite atingir uma execução rápida e assim um histórico de todas as variáveis das partículas.

4.2 Formulação Básica do Método

Segundo Paiva (2007) o método baseia-se na interpolação das propriedades físicas de um sistema hidrodinâmico e da aproximação das derivadas espaciais com campos contínuos, com base em um conjunto discreto de partículas. Sua principal ideia é a representação integral de uma função contínua $f : \mathbb{R}^d \to \mathbb{R}$, que é definido em um domínio aberto $\Omega \subset \mathbb{R}^d$ tal que $x \in \Omega$. A representação é feita com a convolução de f com a distribuição Delta de Dirac.

A formulação dividi-se em duas etapas, a primeira é a representação integral ou a aproximação das funções de campo pela função núcleo. Neste passo a integração da multiplicação de uma função arbitrária e uma função núcleo de suavização, gera a aproximação pela função núcleo em sua representação integral. Posteriormente, a representação integral da função é aproximada pelo somatório dos valores das variáveis de suas partículas vizinhas, produzindo assim a aproximação de partículas da função em determinado ponto ou partícula em si, equação (1) (LIU; LIU, 2003):

$$f(x) = \int_{\Omega} f(x')\delta(x - x')dx'. \tag{1}$$

No qual f é a função do vetor posição tridimensional x, e $\delta(x - x')$ é função delta de Dirac. Dada pela equação (2),

$$\delta(x - x') = \begin{cases} 0 & x \neq x', \\ \infty & x = x'. \end{cases} \tag{2}$$

Na equação (1), o símbolo Ω é o volume de controle da integral contendo x. A equação (1) implica que uma função pode ser representada pela forma integral. A partir do ponto em que a função Delta de Dirac é utilizada na equação, a representação integral na mesma é exata e rigorosa, contanto que $f(x)$ seja definida e contínua em Ω (LIU; LIU, 2003).

Quando a função núcleo Delta de Dirac $\delta(x - x')$ é substituída por uma função de suavização $W(x - x',h)$, a representação integral de $f(x)$ é resolvida pela equação (3), no qual W é chamada de função núcleo de suavização (Kernel de suavização, função de suavização ou função kernel). O termo h é o comprimento de suavização que determina o comprimento da área de influência da função de suavização. Por W não ser mais a função Dirac, sua representação integral passa a ser apenas uma aproximação, por isso o termo Núcleo de Aproximação ou Kernel de Aproximação.

$$f(x) = \int_{\Omega} f(x')W(x - x',h)dx'. \tag{3}$$

Por convenção o operador do núcleo de aproximação é marcado por $<>$, conforme equação (4).

$$< f(x) >= \int_{\Omega} f(x')W(x - x',h)dx'. \tag{4}$$

Os autores Liu e Liu (2003) explicitam que a função W normalmente é escolhida para ser uma função par e deve obedecer a três condições. A primeira relacionada ao núcleo de suavização no qual a integral dela deve ser igual ao valor um, ou a chamada “condição de normalização”, ou "condição de unidade" já que a integração da função de suavização gera a unidade. Equação (5).

$$\int_{\Omega} W(x - x',h)dx' \quad = \quad 1. \tag{5}$$

A segunda condição é a “propriedade da função delta” no qual, conforme o comprimento de suavização (*h*) do núcleo tende a zero ela se iguala a função Delta de Dirac. Equação (6).

$$\lim_{h \to 0} W(x - x',h)dx' \quad = \quad \delta(x - x'). \tag{6}$$

A terceira condição é “condição compacta”, demonstrado na equação (7). Onde k é uma constante relacionada a função de suavização para um ponto em x e que define sua área efetiva (não zero). A área efetiva é chamada de suporte de domínio para a função de suavização do ponto x. Utilizando-se do suporte compacto, a integração de um domínio de um problema inteiro é localizado como uma integração em um suporte de domínio de uma função de suavização, logo a integração do domínio Ω pode ser a mesma como a do suporte de domínio (LIU; LIU, 2003).

$$W(x-x',h) \;=\; 0, \qquad quando\ |x-x'|>kh\ . \tag{7}$$

Comumente discutido na literatura sobre SPH, a aproximação pelo núcleo é dita como tendo uma precisão de h^2 ou precisão de segunda ordem. A razão é a seguinte: na equação (7) pode-se notar que o suporte de domínio da função de suavização é $|x'-x| \leq kh$, os erros da integral podem assim, serem calculados usando uma série de expansão em Taylor de $f(x')$, de segunda ordem, em volta de x, onde $f(x)$ é diferenciável, apresentado nas equações (8) e (9). (LIU; LIU, 2003; ZIENKIEWICZ; TAYLOR; ZHU, 2005; OGER *et al.*, 2007),

$$< f(x) >= \int_\Omega [f(x)+f'(x)(x'-x)+r((x'-x)^2)]W(x-x',h)dx', \tag{8}$$

$$= f(x)\int_\Omega W(x-x',h)dx' + f'(x)\int_\Omega (x'-x)W(x-x',h)dx' + r(h^2), \tag{9}$$

onde r refere-se ao residual. A função W é par em relação a x, assim $(x'-x)W(x-x',h)$ deve ser uma função ímpar. Logo tem-se a equação (10)

$$\int_\Omega (x'-x)W(x-x',h)dx' = 0. \tag{10}$$

Usando as equações (5) e (10), a equação (9) se transforma na equação (11)

$$< f(x) >= f(x)+r(h^2). \tag{11}$$

Da equação (11), pode-se observar que no método SPH a representação integral da função núcleo de aproximação, é de precisão de segunda ordem, porém o núcleo de aproximação não necessariamente mantem a mesma precisão, caso a função de suavização não seja uma função par ou se a condição de normalização não é satisfeita.

4.2.1 Representação Integral da Derivada de uma Função

A derivada espacial $\nabla \cdot f(x)$, pode ser representada substituindo-se $f(x)$ por $\nabla \cdot f(x)$ na equação (12), (LIU; LIU, 2003)

$$< \nabla \cdot f(x) >= \int_{\Omega} [\nabla \cdot f(x')]W(x - x',h)dx', \tag{12}$$

onde se opera, em relação a coordenada inicial, o divergente na integral. Quando se aplica a propriedade de operador divergente, $(\nabla \cdot f)W = \nabla \cdot (Wf) - (\nabla W) \cdot f$, (RABELLO; VIEIRA, 2020), assim tem-se a equação (13)

$$[\nabla \cdot f(x')]W(x - x',h) = \nabla \cdot f(x')W(x - x',h)] - f(x') \cdot \nabla W(x - x',h). \tag{13}$$

A partir da equação (12) é obtida a equação (14)

$$< \nabla \cdot f(x) >= \int_{\Omega} \nabla \cdot [f(x')W(x - x',h)]dx' - \int_{\Omega} f(x') \cdot \nabla W(x - x',h)dx'. \tag{14}$$

Pode-se converter a primeira integral do lado direito da equação (14), utilizando-se do teorema de divergência, em uma integral sobre a superfície S do domínio de integração (Ω), equação (15) (LIU; LIU, 2003)

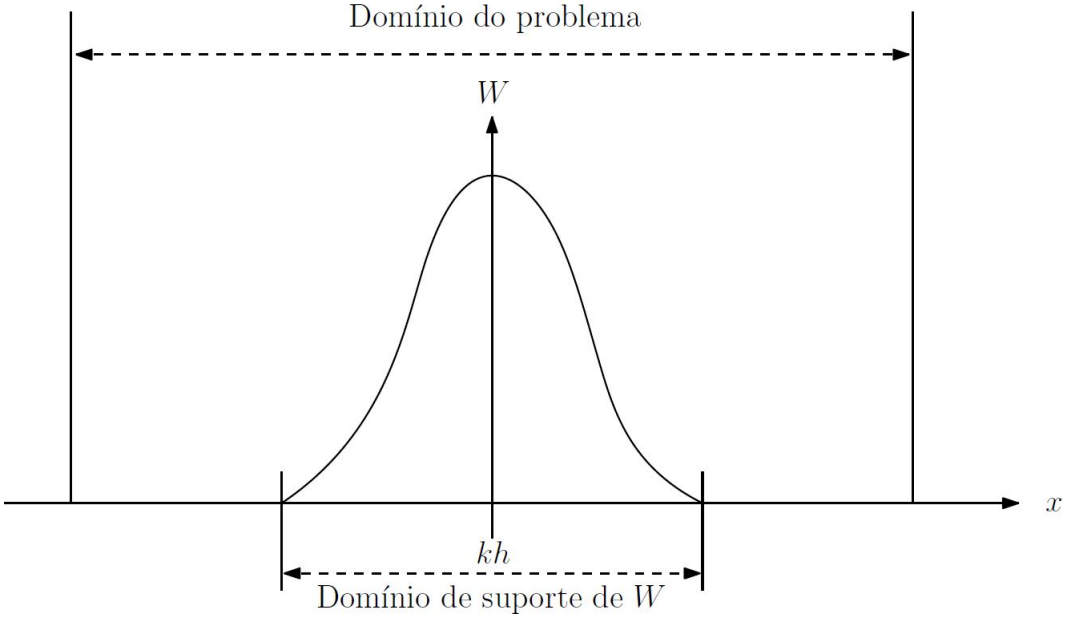
$$< \nabla \cdot f(x) >= \int_S f(x')W(x-x',h) \cdot \vec{n} dS - \int_\Omega f(x') \cdot \nabla W(x-x',h)dx'. \quad (15)$$

A variável $\vec{n}$ é o vetor normal em relação a superfície S. A função W é definida para possuir suporte compacto, quando o domínio de suporte é localizado no meio do domínio do problema. A integral de superfície, no lado direito da equação (15), é zero, representado na Figura 1. Caso o domínio se sobreponha ao domínio do problema, como representado na Figura 2, a função de suavização W é truncada pela borda ou limite, e a integral da superfície não é mais zero, equação (15). Logo para todos os pontos em que o domínio de suporte é contido dentro do domínio do problema, a equação (15) se simplifica como a equação (16) (LIU; LIU, 2003)

$$< \nabla \cdot f(x) >= -\int_\Omega f(x') \cdot \nabla W(x-x',h)dx'. \quad (16)$$

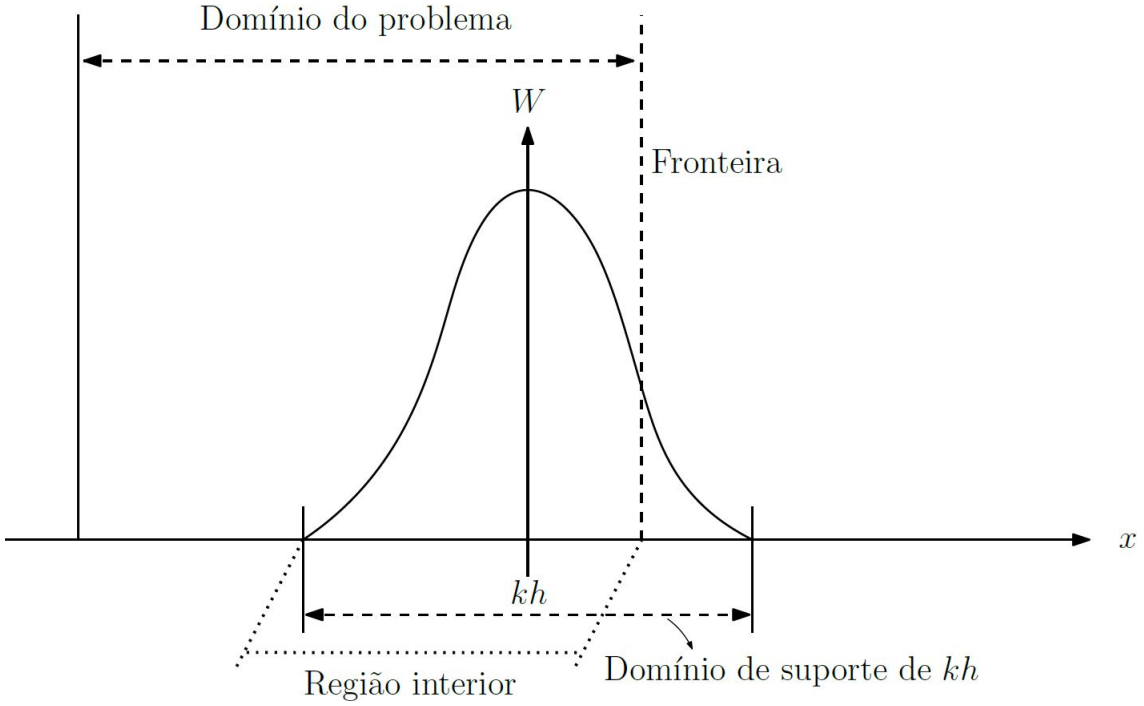
Logo a equação (16) demonstra que a representação integral SPH, da derivada de uma função de campo, permite que o gradiente espacial seja determinado dos valores da função e das derivadas da função de suavização W, ao invés das derivadas em si (LIU; LIU, 2003).

Figura 1 – O domínio de suporte da função de suavização W, contida dentro do domínio do problema.



Fonte: Autoria própria (2021), adaptado de Liu, Liu e Lam (2003), p. 40.

Figura 2 – O domínio de suporte da função de suavização W sobreposto do domínio do problema.

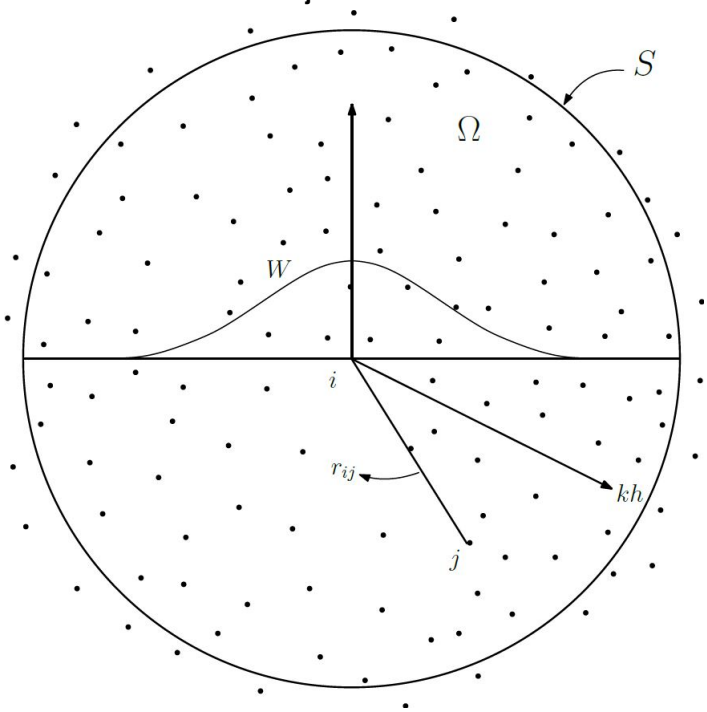


Fonte: Autoria própria (2021), adaptado de Liu, Liu e Lam (2003), p.40.

4.2.2 Aproximação de partículas

Um fator chave no SPH é que sua representação é feita por meio de uma número finito de partículas, que carregam cada uma sua massa individual e ocupam um espaço individual. A representação contínua da integral, em relação ao núcleo SPH, expresso nas equações (4) e (16), podem ser convertidas em formas discretizadas de somatório, tal processo é conhecido por aproximação de partículas, demonstrado na Figura 3.

Figura 3 – Aproximação de partículas contidas dentro do domínio de suporte da função de suavização W, cujo raio é kh.



Fonte: Autoria própria (2021), adaptado de Liu, Liu e Lam (2003), p.41.

Substituindo o volume infinitesimal dx' nas integrações, no local da partícula j, por um volume finito da partícula ΔV_j que está relacionado a massa da partícula M_j pela equação (17), onde ρ_j é a densidade das partículas $j(= 1, 2, ..., N)$ e N é o número de partículas contidas dentro do domínio de suporte da partícula j, então a forma integral contínua SPH para $f(x)$ pode ser escrita em sua forma discretizada de aproximação de partícula nas equações (18), (19), (20) e (21) ou equação (22)

$$m_j = \Delta V_j \rho_j, \tag{17}$$

$$f(x) = \int_\Omega f(x')W(x - x', h)dx' \tag{18}$$

$$\approx \sum_{j=1}^{n} f(x_j)W(x - x_j, h)\Delta V_j \tag{19}$$

$$= \sum_{j=1}^{n} f(x_j)W(x - x_j, h)\frac{1}{\rho_j}(\rho_j \Delta V_j) \tag{20}$$

$$= \sum_{j=1}^{n} f(x_j)W(x - x_j, h)\frac{1}{\rho_j}(m_j) \tag{21}$$

ou ainda,

$$f(x) = \sum_{j=1}^{n} \frac{m_j}{\rho_j} f(x_j) W(x - x_j, h). \tag{22}$$

Logo a “aproximação por partícula” para uma função na partícula i, pode ser finalmente escrita como a equação (23)

$$< f(x_i) >= \sum_{j=1}^{n} \frac{m_j}{\rho_j} f(x_j) W_{ij}. \tag{23}$$

Onde

$$W_{ij} = W(x_i - x_j, h). \tag{24}$$

Atesta-se na equação (23), que o valor de uma função na partícula i é aproximado, utilizando-se a média dos valores da função em todas as partículas, contidas no domínio de suporte da partícula i, mensuradas pela função. Assim, utilizando o mesmo argumento, a “aproximação por partícula” para a derivada espacial da função é a equação (25)

$$< \nabla \cdot f(x) >= -\sum_{j=1}^{n} \frac{m_j}{\rho_j} f(x_j) \nabla W(x - x_j, h), \tag{25}$$

onde o gradiente ∇W refere-se a partícula j. Logo pode-se escrever a “aproximação por partícula” em uma partícula i como equação a (26)

$$< \nabla \cdot f(x_i) >= -\sum_{j=1}^{n} \frac{m_j}{\rho_j} f(x_j) \nabla W_{ij}, \tag{26}$$

em que,

$$\nabla_i W_{ij} = \frac{x_i - x_j}{r_{ij}} \frac{\partial W_{ij}}{\partial r_{ij}} = \frac{x_{ij}}{r_{ij}} \frac{\partial W_{ij}}{\partial r_{ij}}. \tag{27}$$

O mesmo raciocínio é aplicado na equação (26), o valor do gradiente de uma função na partícula i é aproximado pela média dos valores de todas as partículas em seu domínio de suporte, medidos pelo gradiente da função de suavização.

Nota-se que um dos fatores principais do método não possuir malha está justamente no fato da representação integral ser substituída por um somatório de partículas arbitrárias. Ao mesmo tempo se introduz as variáveis de massa e densidade, que são importantes para problemas de hidrodinâmica. Caso utilizado em problemas de sólidos mecânicos é necessário um tratamento especial.

Porém a "aproximação por partículas" sofre de alguns problemas numéricos, como inconsistências de partículas e instabilidade elástica. Resumindo, para uma dada partícula i, o valor de uma função e sua derivada para partícula i é aproximado como:

$$< f(x_i) >= \sum_{j=1}^{N} \frac{m_j}{\rho_j} f(x_j) W_{ij} \tag{28}$$

$$< \nabla \cdot f(x_i) >= \sum_{j=1}^{N} \frac{m_j}{\rho_j} f(x_j) \cdot \nabla_i W_{ij} \tag{29}$$

$$W_{ij} = W(x_i - x_j, h) = W(|x_i - x_j|, h) \tag{30}$$

$$\nabla_i W_{ij} = \frac{x_i - x_j}{r_{ij}} \frac{\partial W_{ij}}{\partial r_{ij}} = \frac{x_{ij}}{r_{ij}} \frac{\partial W_{ij}}{\partial r_{ij}}, \tag{31}$$

onde r_{ij} é a distancia entre a partícula i e j. Percebe-se que $\nabla_i W_{ij}$ diz respeito a partícula i, logo o sinal de negativo da equação (25) é removido na equação (29).

Propriedade de simetria. Observando a equação (30), e aplicando a propriedade de simetria $W(x) = W(|x|)$ (PEREIRA, 2019):

$$\begin{aligned} W_{ij} &= W(x_i - x_j, h) \\ &= W(|x_i - x_j|, h) \\ &= W(|x_{ij}|, h) \\ &= W(r_{ij}, h). \end{aligned}$$

Assim a partir da equação (31), para calcular $\nabla_i W_{ij}$ utiliza-se o conceito de derivada direcional. Então, calcula-se o vetor unitário U,

$$\begin{aligned} x_{ij} &= x_i - x_j \\ |x_{ij}| &= r_{ij} \\ U &= \frac{x_{ij}}{r_{ij}}. \end{aligned}$$

Assim,

$$\begin{aligned} \nabla_i W_{ij} &= \nabla W(x_i - x_j, h) = \nabla(r_{ij}, h) \\ &= \frac{x_i - x_j}{r_{ij}} \cdot \nabla W(r_{ij}, h) \\ &= \frac{x_{ij}}{r_{ij}} \cdot \frac{\partial W}{\partial r_{ij}}. \end{aligned}$$

Substituindo $f(x)$ com a função densidade na equação (28), sua aproximação na forma SPH é obtida como equação (32), sendo uma das formas mais populares de se obter a densidade no SPH (LIU; LIU, 2003).

$$\begin{aligned} \rho(x_i) &= \sum_{j=1}^{N} \frac{m(x_j)}{\rho(x_j)} \rho(x_j) W_{ij} \\ \rho_i &= \sum_{j=1}^{N} m_j W_{ij}. \end{aligned} \tag{32}$$

4.3 Funções Núcleo

4.3.1 Construção de Funções de Suavização

Um dos problemas de métodos sem malha, é a aproximação por meio de uma função de forma efetiva, quando se trata de um conjunto de nós espalhados de forma arbitrária, sem utilizar uma malha pré-definida ou grade que promova a conectividade necessária. Os autores Liu e Liu

(2003), classificam de 3 formas o método de aproximação: representação integral, representação em série e representação diferencial.

O método SPH utiliza a representação integral utilizando uma função de suavização, ou também chamada de função suavizadora, função núcleo (kernel) ou função núcleo de suavização. A função de suavização determina o padrão de aproximação, a dimensão do suporte de domínio das partículas, assim como sua consistência, precisão e eficiência computacional (LIU; LIU, 2003).

4.3.1.1 Propriedades básicas de funções de suavização

De forma sumarizada:

- A função de suavização deve ser normalizada (condição de unidade) em seu domínio de suporte. Assim, fica assegurado que a integral da função de suavização seja unitária sobre o domínio de suporte,

$$\int_{\Omega} W(x - x', h)dx' = 1. \tag{33}$$

- A função de suavização deve possuir suporte compacto. Essa condição garante que a aproximação SPH seja transformada de uma função global para uma função local. Isso gera um sistema de matrizes discretizadas, influenciando o esforço computacional.

$$W(x - x') = 0, \quad para \quad |x - x'| > kh. \tag{34}$$

  Sendo que a dimensão do suporte compacto é definido pelo comprimento de suavização h, e seu fator de escala k, que determina o alcance ou largura da função de suavização em especifico. A função $|x - x'| \leq kh$ define o domínio de suporte em qualquer ponto x.

- O termo $W(x - x') \geq 0$ para qualquer ponto x' inserido no domínio de suporte de uma partícula no ponto x (Positividade). Determina-se assim que a função não pode ser negativa em seu suporte de domínio. Não sendo matematicamente necessário, porém determinante na estabilidade física, principalmente em problemas hidrodinâmicos.

- O valor de uma função de suavização para uma partícula deve decair monotomicamente com o aumento da distancia da partícula. O decaimento garante que as partículas mais próximas devem ser mais influentes que as partículas mais distantes.
- A função de suavização deve satisfazer a função de Delta de Dirac, conforme o comprimento de suavização se aproxima de zero

$$\lim_{h \to 0} W(x - x', h) \quad = \quad \delta(x - x'). \tag{35}$$

- A função de suavização deve ser par (Propriedade de Simetria). Isso garante que partículas com a mesma distância porém, posições diferentes, tenham a mesma influência em determinada partícula.
- A função de suavização deve ser suficientemente suave. Isso implica que sua continuidade irá contribuir para melhores aproximações, proporcionando melhores resultados para a função e suas derivadas. Assim, a função de suavização não será tão susceptível a um desordenamento de partículas.

4.3.1.2 Exemplos de funções

A função em formato de sino, proposta por Lucy (1977), descrito na equação (36):

$$W(x - x', h) = W(R, h) = \alpha_d \begin{cases} (1 + 3R)(1 - R)^3 & R \leq 1 \\ 0 & R > 1 \end{cases}, \tag{36}$$

onde $\alpha_d = 5/4h, 5/\pi^2$ e $105/16\pi^3$, para uma, duas e três dimensões respectivamente. Para que a condição de unidade seja correspondida, R na equação (36) é a distância entre duas partículas, no ponto x e x', R equivale a $\frac{r}{h} = \frac{|x-x'|}{h}$. A variável r é a distância entre dois pontos.

O autor Monaghan (1992), propõe uma função Gaussiana como a melhor forma de uma função de suavização. Os autores Gingold e Monaghan (1977) propõem a função da equação (37):

$$W(R,h) = \alpha_d \exp^{-R^2}. \tag{37}$$

Os termos acima são $\alpha_d = \frac{1/\pi^1}{2h}$, $\frac{1/\pi}{h^2}$ e $\frac{1/\pi^3}{2h^3}$, para uma, duas e três dimensões respectivamente. Extremamente suave para derivadas de alta ordem, estável, entretanto, não é compacta, R não vai há zero teoricamente, implicando em custo computacional.

A função *Cubic Spline* é a mais utilizada, similar a função Gaussiana mas possuindo um suporte compacto mais afunilado, enquanto sua segunda derivada é composta por funções lineares por partes, o que pode gerar problemas de estabilidade, exibido na equação (38) (MONAGHAN; LATTANZIO, 1985).

$$W(R,h) = \alpha_d x \begin{cases} \frac{2}{3} - R^2 + \frac{1}{2}R^3 & 0 \leq R < 1 \\ \frac{1}{6}(2-R)^3 & 1 \leq R < 2 \\ 0 & R \geq 2, \end{cases} \tag{38}$$

onde $\alpha_d = 1/h, 15/7\pi h^2$ e $3/2\pi h^3$, para uma, duas e três dimensões respectivamente.

O autor Morris (1996), Morris (1994), apresentou *Splines* de ordem maior, sendo a quártica, equação (39) e quíntica, equação (40), mais próximas da forma Gaussiana e mais estáveis,

$$W(R,h) = \alpha_d x \begin{cases} (R+2{,}5)^4 - 5(R+1{,}5)^4 + 10(R+0{,}5)^4 & 0 \leq R < 0.5 \\ (2{,}5-R)^4 - 5(1{,}5-R)^4 & 0.5 \leq R < 1.5 \\ (2{,}5-R)^4 & 1{,}5 \leq R < 2.5 \\ 0 & R > 2.5 \end{cases} \tag{39}$$

$\alpha_d = 1/24h$ em uma espaço de uma dimensão.

$$W(R,h) = \alpha_d x \begin{cases} (3-R)^5 - 6(2-R)^5 + 15(1-R)^5 & 0 \leq R < 1 \\ (3-R)^5 - 6(2-R)^5 & 1 \leq R < 2 \\ (3-R)^5 & 2 \leq R < 3 \\ 0 & R > 3 \end{cases} \tag{40}$$

$\alpha_d = 120/h, 7/478\pi h^2$ e $3/359\pi h^3$, para uma, duas e três dimensões respectivamente.

Os autores Johnson, Stryk e Beissel (1996) utilizaram a função quadrática, equação (41), em um problema de impacto de alta velocidade,

$$W(R,h) = \alpha_d(\frac{3}{16}R^2 - \frac{3}{4}R + \frac{3}{4}) \quad 0 \leq R \leq 2, \tag{41}$$

$\alpha_d = 1/h, 2/\pi h^2$ e $5/4\pi h^3$, para uma, duas e três dimensões respectivamente.

Contrário as outras funções de suavização, a derivada desta função sempre aumenta conforme as partículas se movem próximas e diminui conforme elas se distanciam, o que para o autor é uma vantagem sobre a função *Cubic Spline*, já que ajuda em problemas com instabilidade de compressão.

4.4 Aspectos Numéricos Do Método SPH

A seção a seguir descreve alguns aspectos fundamentais para o método SPH, sua estabilidade e controle de fenômenos físicos, com viscosidade, propriedades da fronteira e movimentação de partículas.

4.4.1 Termo de Viscosidade

De acordo com Paiva (2007) a equação do momento pode ser definida como:

$$\frac{Dv_i}{Dt} = -\frac{1}{\rho_i}\nabla P_i + \frac{1}{\rho_i}\nabla \cdot S_i + g. \tag{42}$$

Para o cálculo de forças viscosas, calcula-se primeiro o tensor campo de velocidade ∇v_i, para cada partícula i usando a aproximação SPH (PAIVA, 2007), equação (43):

$$\nabla v_i \quad = \quad \sum_{j \in N(xi)} \frac{m_j}{\rho_j}(v_j - v_i) \otimes \nabla_i W_{ij}. \tag{43}$$

Com o cálculo da velocidade, calcula-se o tensor de extra-tensão:

$$S_i = \mu_i D_i. \tag{44}$$

As variáveis μ_i equivalem a viscosidade, D_i é o tensor taxa de deformação:

$$D_i = \nabla v_i + (\nabla v_i)^T. \tag{45}$$

Com a atualização do tensor S_i, o termo de viscosidade da equação (42) pode ser definido como:

$$\frac{1}{\rho_i}\nabla \cdot S_i = \sum_{j \in N(x_i)} \frac{m_j}{\rho_i \rho_j}(S_i + S_j) \cdot \nabla_i W_{ij}. \tag{46}$$

Com base nos autores Liu e Liu (2003) de forma a simular problemas de hidrodinâmica são necessários tratamentos especiais ou métodos para que os algorítimos sejam capazes de modelar ondas de choque, ou a simulação irá desenvolver oscilações não físicas nos resultados numéricos ao redor da região em choque.

A formulação de viscosidade artificial mais comum e proposta por Monaghan (1992), é obtida escrevendo a equação de momento como (47):

$$\frac{Dv_i}{Dt} = \sum_b m_j \left(\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2} + \Pi_{ij} \right) \nabla_i W_{ij} + g \tag{47}$$

$$\Pi_{ij} \begin{cases} -\frac{\alpha \bar{c}_{ij} \mu_{ij} + \beta \mu_{ij}^2}{\overline{\rho_{ij}}} & se \quad v_{ij} x_{ij} < 0 \\ 0 & se \quad v_{ij} x_{ij} > 0. \end{cases} \tag{48}$$

Sendo:

$$\mu_{ij} = \frac{h_{ij} v_{ij} x_{ij}}{x_{ij}^2 + n^2} \quad (49)$$

$$\overline{c}_{ij} = \frac{1}{2}(c_i + c_j) \quad (50)$$

$$\overline{\rho}_{ij} = \frac{1}{2}(\rho_i + \rho_j) \quad (51)$$

$$h_{ij} = \frac{1}{2}(h_i + h_j). \quad (52)$$

As variáveis nas equações acima, são conforme Monaghan (1992) α_π e β_π são constantes tipicamente próximas a 1. A variável α_π, produz uma viscosidade massiva, enquanto β_π faz a supressão de interpenetração de partículas a grandes velocidades com alto número de Mach. O valor n^2 equivale a $0{,}1h_{ij}$ e é colocado de modo a prever divergências numéricas enquanto duas partículas se aproximam uma da outra. O termo c representa a velocidade do som, v o vetor velocidade da partícula, em que $v_{ij} = v_i - v_j; x_{ij} = x_i - x_j$.

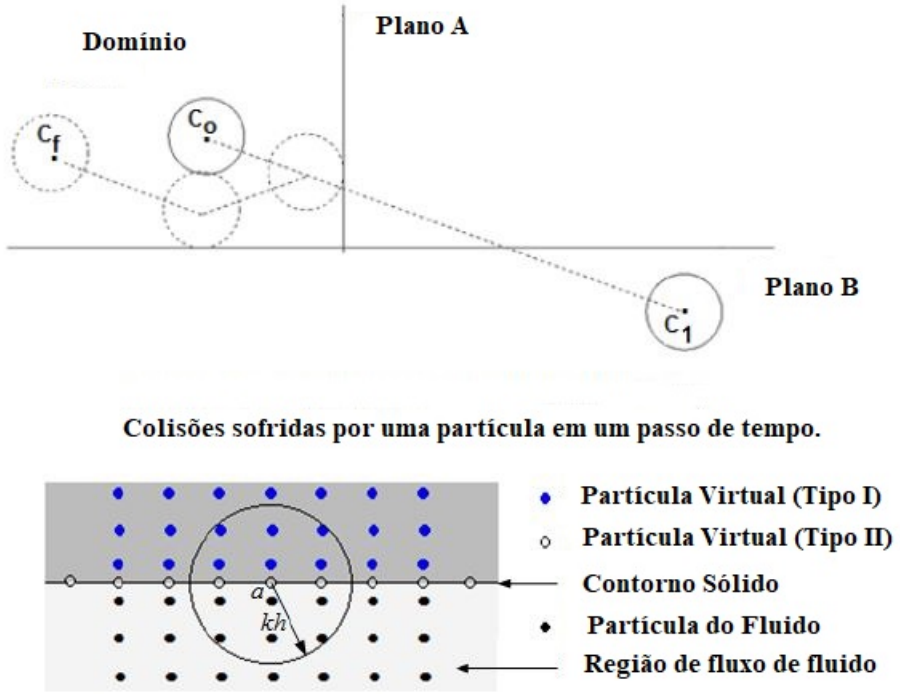
4.4.2 Tratamento de Fronteira

O aproveitamento máximo do método SPH, conforme Liu e Liu (2003), tem sido limitado por problemas com a deficiência do número de partículas próximas ou sobre a fronteira que ocorre por conta do truncamento da integral. Com partículas próximas ou sobre uma fronteira, somente aquelas no interior da fronteira geram uma contribuição ao somatório da interação de partículas, sem nenhuma contribuição das exteriores, gerando soluções incorretas. Essa contribuição de apenas um lado, possui o defeito de que enquanto a velocidade em uma superfície sólida é zero, as demais variáveis de campo como a densidade não necessariamente reduz a zero.

Por conta disso, explica os autores Liu e Liu (2003) inúmeras soluções tem sido buscadas de forma a melhorar os resultados. O autor Monaghan (1994) criou linhas de partículas fantasmas ou virtuais, localizadas próximas a borda do sólido, de modo a produzir uma grande força repulsiva e ao mesmo tempo, prevenir a penetração de partículas através da borda em si.

O autor Fraga Filho (2016) explica que uma situação análoga a dinâmicas moleculares também e usada em um tratamento no SPH. O tratamento de fronteira usa uma sequência de partículas congeladas (partículas virtuais), localizadas no contorno de um solido para produzir uma força repulsiva forte. Esse tipo de partícula é conhecida como tipo I. A prevenção de penetração nos contornos ocorre por meio de uma força de repulsão similar a força molecular Lennard-jones, que simula condições de fronteira com velocidade normal 0 (Nula). De acordo com Liu e Liu (2003) em alguns, casos partículas do tipo II podem ser localizadas fora do contorno, para cada partícula i localizada próxima a fronteira com a distância kh_i, uma partícula tipo II será colocada simetricamente fora da fronteira, com valores de densidade e pressão iguais, porém com velocidade oposta, conforme na Figura 4:

Figura 4 – Ilustração da região de contorno sólida. Ordenamento das partículas virtuais tipo I (Linha de contorno) e das partículas tipo II (Área estendida além do domínio)



Fonte: Autoria própria (2021), adaptado de Fraga Filho (2016).

Libersky *et al.* (1993) introduziram partículas fantasmas para refletir uma condição de superfície de fronteira simétrica, assim como Randles e Libersky (1996) que propuseram uma condição de tratamento no qual se assume o mesmo valor de fronteira, de uma variável de campo, para todas as partículas fantasmas, que posteriormente são interpolados suavemente assim como os valores calculados das partículas interiores.

4.4.3 Movimentação de Partículas

Conforme proposto por Monaghan (1989), para que ocorra o movimento de partículas foi criado o método XSPH. O método tenta garantir que as partículas se movam com a média de velocidade de suas partículas vizinhas, buscando criar um fluxo mais ordenado e a não penetração entre as mesmas.

$$\frac{Dr_i}{Dt} = v_i + \epsilon \sum_j \frac{m_j}{\rho_{ij}} v_{ji} W_{ij},$$

onde o termo ϵ é um parametro que varia de 0 até 1 e $\rho_{ij} = 0{,}5(\rho_i + \rho_j)$.

4.5 Equações de Navier-Stokes na forma SPH

Em mecânica dos fluidos, trabalha-se com as equações de Navier-Stokes, elas são as equações diferenciais que definem o comportamento de fluidos, suas variáveis pertinentes e seu comportamento físico, sendo divididas em Continuidade, Momento e Energia (GRAEBEL, 2007; NASA, 2020). Nas formas tri-dimensionais as equações são;

Equação de Continuidade (53):

$$\frac{\partial \rho}{\partial_t} + \frac{\partial(\rho u)}{\partial_x} + \frac{\partial(\rho v)}{\partial_y} + \frac{\partial(\rho w)}{\partial_z} = 0. \tag{53}$$

Equação de Momento em (*x*), (*y*) e (*z*), equações (54), (55), (56):

$$\frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2)}{\partial x} + \frac{\partial(\rho uv)}{\partial y} + \frac{\partial(\rho uw)}{\partial z} = -\frac{\partial P}{\partial x} + \frac{1}{Re}\left(\frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z}\right) \tag{54}$$

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho uv)}{\partial x} + \frac{\partial(\rho v^2)}{\partial y} + \frac{\partial(\rho vw)}{\partial z} = -\frac{\partial P}{\partial y} + \frac{1}{Re}\left(\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z}\right) \tag{55}$$

$$\frac{\partial(\rho w)}{\partial t} + \frac{\partial(\rho uw)}{\partial x} + \frac{\partial(\rho vw)}{\partial y} + \frac{\partial(\rho w^2)}{\partial z} = -\frac{\partial P}{\partial z} + \frac{1}{Re}\left(\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z}\right). \tag{56}$$

Equação de energia (57):

$$\frac{\partial(E_T)}{\partial T}+\frac{\partial(uE_T)}{\partial x}+\frac{\partial(vE_T)}{\partial y}+\frac{\partial(wE_T)}{\partial z}=-\frac{\partial(up)}{\partial x}+\frac{\partial(vp)}{\partial y}+\frac{\partial(wp)}{\partial z}$$
$$-\frac{1}{RePr}\left(\frac{\partial q_x}{\partial x}+\frac{\partial q_y}{\partial y}+\frac{\partial q_z}{\partial z}\right)+\frac{1}{Re}\left[\frac{\partial}{\partial x}(u\tau_{xx}+v\tau_{xy}+vw\tau_{xz})\right.$$
$$\left.+\frac{\partial}{\partial y}(u\tau_{xy}+v\tau_{yy}+vw\tau_{yz})+\frac{\partial}{\partial z}(u\tau_{xz}+v\tau_{yz}+vw\tau_{xz})\right]. \quad (57)$$

No qual (x,y,z) são coordenadas espaciais, (u,v,w) são componentes do vetor velocidade, t = tempo, ρ = densidade, μ = viscosidade, p = pressão, T = temperatura (contida na equação de energia sob o termo E_T ou energia total), τ = tensão ou estresse, q = fluxo de calor, *Re* = Número de Reynolds e *Pr* = Número de Prandtl.

Os termos i definem as partículas de origem e os termos j definem a partícula secundária ou a que está sendo comparada a origem. Aplicando o método SPH, as suas variáveis dependentes substituídas se transformando assim em Equações Diferenciais Ordinárias (EDOs), em SPH se convertem em, (LIU; LIU, 2003):

Equação da continuidade:

$$\frac{D\rho_i}{Dt} = \sum_{j=1}^{N} m_j v_{ij}^{\beta}\frac{\partial W_{ij}}{\partial x_i^{\beta}}. \quad (58)$$

Conservação de massa:

$$\frac{Dv_i^{\alpha}}{Dt} = \sum_{j=1}^{N} m_j\left(\frac{\sigma_i^{\alpha\beta}}{\rho_i^2}+\frac{\sigma_j^{\alpha\beta}}{\rho_j^2}\right)\frac{\partial W_{ij}}{\partial x_i^{\beta}}. \quad (59)$$

Equação da energia:

$$\frac{De_i}{Dt} = \frac{1}{2}\sum_{j=1}^{N} m_j\left(\frac{p_i+p_j}{\rho_{i^2}+\rho_{j^2}}\right)v_{ij}^{\beta}\frac{\partial W_{ij}}{\partial x_i^{\beta}}+\frac{\mu_i}{2\rho_j}\frac{\partial W_{ij}}{\partial x_i^{\beta}}\varepsilon_i^{\alpha\beta}\varepsilon_j^{\alpha\beta}. \quad (60)$$

4.6 Fluxo de Poiseuille

O fluxo descrito por Poiseuille ocorre entre duas placas planas paralelas, em duas dimensões, no qual uma diferença potencial de pressão desloca um fluido viscoso em uma placa infinita. O gradiente de pressão no eixo vertical y e sua velocidade desaparecem, assim o fluxo fica dependente apenas da velocidade no eixo x e seu gradiente de pressão (SIGALOTTI *et al.*, 2003). A equação de Poiseuille por ser analítica, pode determinar valores para comparação com o método numérico SPH para validação, por exemplo:

$$v_x(y,t) = \frac{|F|}{2\upsilon}(y^2 - d^2) + \sum_{n=0}^{\infty} \frac{16(-1)^n d^2 |F|}{\upsilon \pi^3 (2n+1)^3} \cos\left[\frac{(2n+1)\pi y}{2d}\right] \exp\left[-\frac{(2n+1)^2 \pi^2 \upsilon t}{4d^2}\right]. \quad (61)$$

Para tempos longos, tendendo ao infinito, a expansão em série desaparece e a velocidade chega a um estado desenvolvido:

$$v_x(y) = \frac{|F|}{2\upsilon}(y^2 - d^2), \quad (62)$$

$$= v_0 \left(1 - \frac{Y^2}{d^2}\right), \quad (63)$$

onde:

$$v_0 = -\frac{d^2}{2\rho\upsilon}\left(\frac{\Delta p}{L}\right) \quad (64)$$

$$|F| = \frac{1}{\rho}\left(\frac{\Delta p}{L}\right). \quad (65)$$

O termo v_x refere-se a velocidade no eixo horizontal x ou no sentido do fluxo, υ é o termino para viscosidade cinemática, d equivale a separação exata ao meio das placas, y refere-se ao eixo vertical, e possui valor de 0, ja que é o centro, Δp ao gradiente de pressão existente entre dois pontos separados por L, v_o é uma velocidade assintótica constante e F uma força por unidade de massa.

Os autores Liu e Liu (2003) utilizam uma versão semelhante simplificada, onde o valor de y é variado além do ponto 0.

$$v_x(y,t) = \frac{|F|}{2\upsilon} d(d - y). \quad (66)$$

4.7 Fortran

Com o objetivo de facilitar a programação durante os primórdios da computação na década de 50 (CAMPINAS, 2012), a empresa IBM criou uma linguagem de alto-nível denominada *Formula Translation*, que tinha como objetivo facilitar a programação, sem a necessidade de se dar instruções a unidades centrais de processamento (CPUs), ou a complicação de codigos-fonte ou código do tipo *Assembler*, código em notação numérica (GAELZER, 2012).

O Fortran, segundo Gaelzer (2012), na forma como foi concebido, era extremamente simples de ser usado, programar fórmulas matemáticas eram programadas pensadas em forma simbólica, isso garantiu uma alta eficiência e agilidade na criação de códigos com apenas uma pequena perda na eficiência do processamento, porque agora o foco ficava no programa em si e uma pequena parte ao compilador, que consumia muito tempo anteriormente. Além de não precisar *Assemblers*, e focar diretamente nas soluções. Dessa forma a linguagem se disseminou em campos como matemática, engenharia e física.

As gerações da linguagem Fortran seguem (CAMPINAS, 2012):

- **Fortran I** - (1954-1957). Manteve o recorde de traduzir um código por mais de 20 anos.
- **Fortran II** - 1958.
- **Fortran III** - 1958, limitado a seu laboratório de criação.
- **Fortran IV ou Fortran66** - 1966.
- **Fortran77** - 1977. Padronizado com o conceito de programação estruturada. Padrão: ANSI X3 e ISO/IECJTC1/SC22/WG5.
- **Fortran90** - 1980. Atualização do Fortran77 que demorou 12 anos para ser criada. Nessa versão vários recursos da linguagem se assemelham daquelas existentes na linguagem em C como alocação dinâmica de memória, apontadores e orientações ao objeto.
- **HPF - *High Performance Fortran*** - 1990. Fortran90 para ambientes de memória distribuída.
- **Fortran95** - 1995.

O Fortran77 é considerado obsoleto para atualidade, pois possui formato fixo, limitação de caracteres, incapacidade de lidar com operações paralelas, impossibilidade de alocar memória dinamicamente, forçando o programador a declarar vetores grandes com o maior numero de informações possíveis; não possui recursão explicita, ou seja não é possível chamar uma função dentro de outra; entre outros fatores (CAMPINAS, 2012).

Já o Fortran90 possui formato livre, maior número de caracteres para se trabalhar e comandos por linha, novas instruções, alocação dinâmica de memória, maior flexibilidade e precisão numérica de variáveis, recursividade de funções e rotinas, entre outras características benéficas (CRISTO, 2003).

Para se escrever em Fortran existem duas maneiras a primeira em *Fixed Form* ou formato fixado e a segunda em *Free Form* ou formato livre, sendo este apenas disponível para compiladores mais recentes e que suportam programação em Fortran90.

O formato fixo deve seguir as premissas (CRISTO, 2003):

- **Colunas 1 a 5** - Nestas colunas só se podem escrever rótulos ou *labels* que são números de comando, estes devem ser inteiros e contidos nestas colunas, não podem ser repetidos e não precisam estar em formato crescente. São usados por comandos para identificar aquela linha.
- **Coluna 6** - Aceita qualquer caractere diferente de 0 e indica que o que vem a seguir faz parte da linha anterior ou da ultima linha que não seja comentário, um comando pode ter até 19 linhas de código com linhas em branco ou intermediários.
- **Coluna 7 a 72** - Comandos ou comentários.
- **Colunas 73 a 80** - Campos de identificação que são usados pelo compilador não podem ser escritas.

O formato livre ou F90, presente no Fortran90 aceita ser escrito em qualquer posição, desde que este modo esteja ativado.

- A indicação de continuação de linhas é feita pelo simbolo “&” ao final da sentença, assim a próxima linha abaixo que não seja comentário será considerada a sua continuação.
- Os rótulos devem ser os primeiros caracteres da linha mas podem estar em qualquer coluna.

5 MATERIAL E MÉTODOS

A seguir são discutidos aspectos relacionados a criação do programa, programas utilizados e suas configurações, assim como o *Hardware*.

5.1 *Hardware* Utilizado

O computador utilizado para simulações é um HP Pavilion Gaming Laptop 15-cx0xxx, com a versão de Windows 10 Home 64 bits Versão: 18363.1198. O processador é um Intel (R) Core (TM) i5-8300H CPU @ 2.30GHz, com memória de 8GB Micron 2667MHz. O modelo de disco rigido é HGST HTS721010A9E630 - 931,51 GB. A placa de video é uma Gforce GTX 1050Ti com 4096Mb de GDDR5, driver 445.75.

5.2 Programas Utilizados

O programa utilizado para codificação é o *Microsoft Visual Studio 2022*, para compilar o código em Fortran foi utilizado o *Intel Fortran Compiler(Beta) & Intel Fortran Compiler Classic (2022)*, uma extensão instalada do *Visual Studio*.

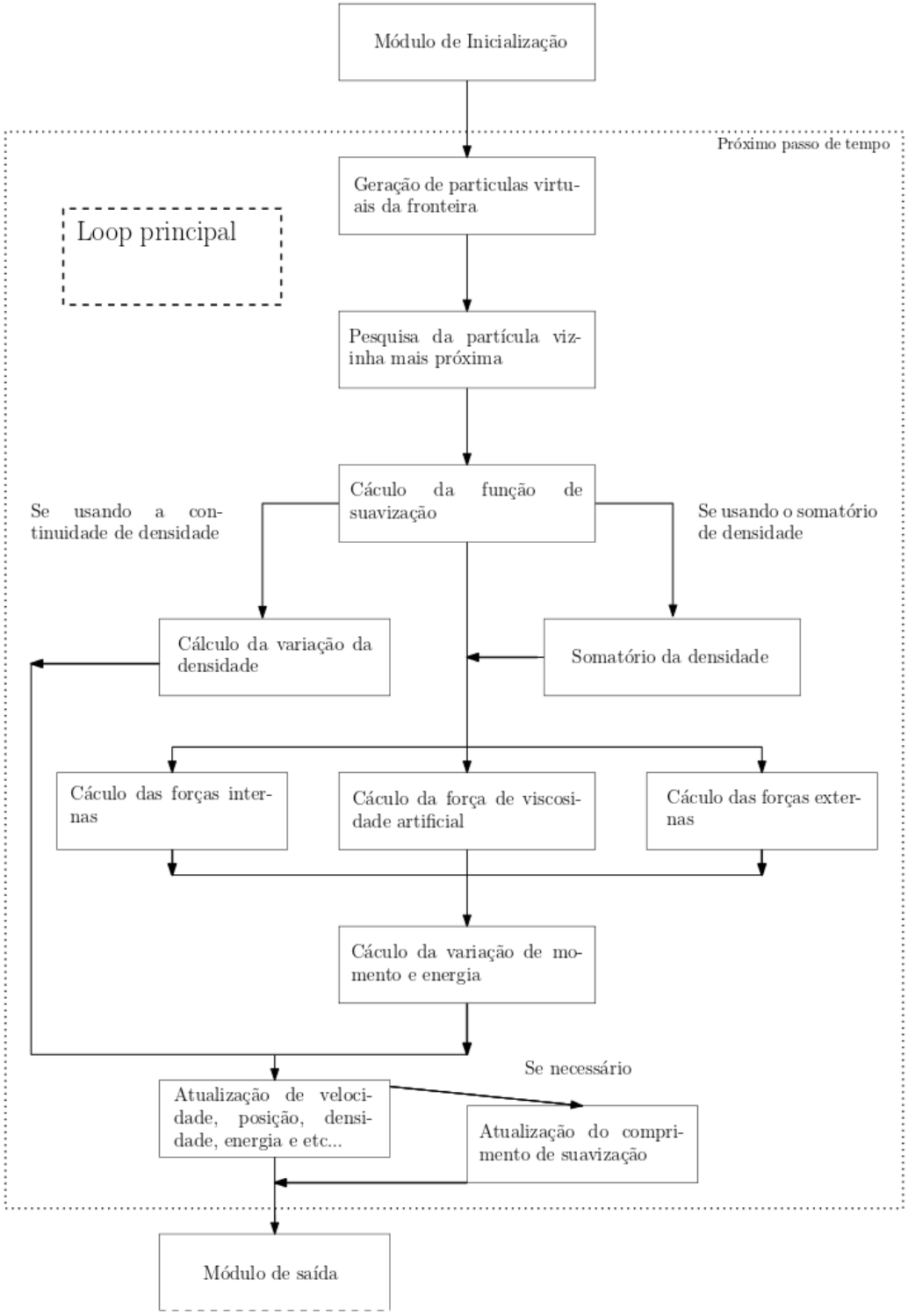
A visualização do resultado se deu por meio do *Microsoft Office 2016 - Excel*, no qual os dados foram organizados e plotados em gráficos.

5.3 O Programa

A criação do programa foi feita com base no código original criado por Liu e Liu (2003). O professor Dr. Liu Gui-Rong por meio de seu site e iniciativa Gui-Rong (2022), fornece o código, entretanto o acesso acontece somente com autorização do autor, Anexo - A. O código cedido em Fortran77 foi atualizado para Fortran90 e teve suas partes removidas, modificadas e/ou usadas na integra assim como outras foram adicionadas para o propósito deste trabalho. Os termos e sintaxe original foram mantidos para facilitar o trabalho.

De forma geral, um programa baseado no método SPH segue o seguinte fluxograma em seu código, demostrado na Figura 5 segundo Liu e Liu (2003):

Figura 5 – Fluxograma da execução de um código usando o método Smoothed Particles Hydrodynamics.



Fonte: Autoria própria (2021), adaptado de Liu e Liu (2003), p. 369.

O código começa sendo executado pelo arquivo “SPH.F”, que representa o modulo de inicialização da Figura 5 juntamente do arquivo “*single_step*”, o primeiro arquivo é responsável por determinar o tamanho dos passos de tempo, usados na integração de tempo e a quantidade utilizada para determinar o tempo da simulação. Neste caso o passo de tempo é de $1{,}0^{-3}$

multiplicado pelo valor de 5000 passos de tempo, o que da aproximadamente 5 segundos de simulação.

O segundo é responsável por chamar todas as funções que serão carregadas na memoria e que foram escolhidas no arquivo “param.inc.txt”.

Os arquivos “*Input.f*” e “virt_part.f” representam a segunda etapa de "Geração de partículas virtuais da fronteira". Estes dois arquivos carregam as partículas com coordenadas e formam a geometria do problema, juntamente com informações básicas de cada uma, como velocidade, densidade, pressão, energia, viscosidade e etc.

As partículas virtuais representam apenas uma barreira invisível, de modo que se comportam como uma barreira física real, neste caso, são duas camadas, acima e abaixo do problema. Um diferencial de pressão empurra qualquer partícula a uma velocidade de $1{,}0^{-4}$ metros por segundo (m/s).

Logo a seguir o arquivo “*direct_find*”, representando a fase de "Cálculo da função de suavização"é inicializado, nesta parte do código a distancia entre partículas é comparada com o comprimento de suavização de cada partícula. Se a mesma estiver dentro do comprimento é formado um par, no qual são computados os valores necessários.

As informações são repassadas ao passo “Cálculo da função de suavização” ou o arquivo "*Kernel.f"*. Neste arquivo a função de suavização é escolhida com base nos parâmetros iniciais e calculada.

As informações recolhidas são enviadas a etapa “Somatório de densidade” ou no programa “*Density.f*”, que calcula as informações relativas a densidade para alimentar os próximos passos "Cálculo das forças internas"ou arquivo “*Internal_force.f*”, “Cáculo da força de viscosidade artificial” ou “art_visc.f” e “Cálculo das forças externas” ou o arquivo “*external_force.f*”.

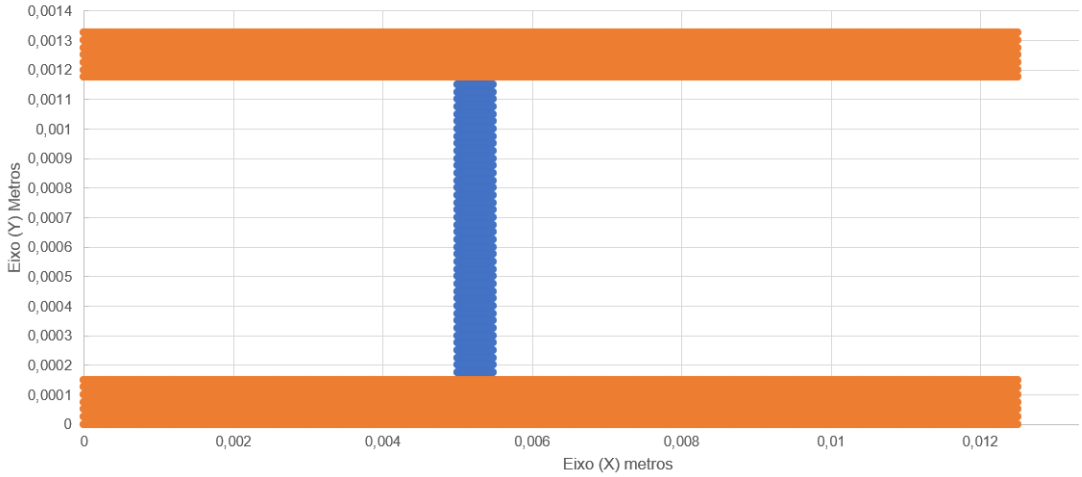
As informações produzidas ainda são organizadas pelo arquivo “*Single_step*", o mesmo faz o somatório dos resultados dos três últimos passos e converte em forças por unidade de massa e variação de força/tempo. Além de guardar os resultados das etapas anteriores. Isso equivale ao passo do “Cálculo da variação de energia”. O passo de tempo ocorre no arquivo “*time_integration*”, que atualiza as informações para o próximo passo ou a fase “ Atualização de velocidade, posição, densidade, energia e etc”. Por fim o “Módulo de de saída”, representa os arquivos que fazem o “*print*” das informações como “*time_print.f*” e “*time_elapsed.f*”.

5.3.1 Modelo Executado

O modelo executado, Figura 6, foi montado com sete camadas de partículas virtuais, representando as placas infinitas no qual o fluxo se desloca, abaixo e acima do problema. Cada partícula é colocada a uma distancia de $2{,}5^{-5}$ metros. O lado superior e inferior são compostos de 500(x) * 7(y) partículas. Junto a estas há um bloco composto de 20(x) * 40(y) partículas representando uma massa de água que fluí entre as placas.

A distância (y) entre as placas é de 0,001 metros. O ponto de origem (x) do bloco é de aproximadamente 5.0^{-3}, dentro do tubo. O centro do tubo é de aproximadamente $6,75^{-4}$ metros, a placa de baixo começa em 0 e vai até o valor de $1,75^{-4}$ metros, a placa superior começa em $1,175^{-3}$ e termina em $1,350^{-3}$ metros.

Figura 6 – Geometria do método SPH com a simulação de Fluxo de Poiseuille.



Fonte: Autoria própria (2022).

Para o bloco representando a água os dados são, para cada partícula:

- **Velocidade no eixo (x)** : 0 m/s
- **Velocidade no eixo (y)** : 0 m/s
- **Gradiente de Pressão** : 2^{-4} m/s
- **Variável Densidade (ρ)** : 1000 kg/m^3
- **Energia interna** : 357,1 J/kg
- **Comprimento de suavização (h)** : 3^{-5} metros
- **Viscosidade Cinemática (υ)** : 1^{-6} m/s^2
- **Viscosidade Dinâmica** : 1^{-3} $kg/m * s$

Para o bloco representando as placas, para cada partícula:

- **Velocidade no eixo (x)** : 0 m/s
- **Velocidade no eixo (y)** : 0 m/s
- **Gradiente de Pressão** : 0 m/s
- **Variável ρ (Densidade)** : 1000 kg/m^3

- **Energia interna** : 357,1 J/kg
- **Comprimento de suavização** : 3^{-5} metros
- **Viscosidade Cinemática (υ)** : $1^{-6}\ m/s^2$
- **Viscosidade Dinâmica** : $1^{-3}\ kg/m*s$
- **Distancia entre as placas (y)** : 1^{-3} metros
- **Centro das placas (d)** : 5^{-4} metros

5.3.2 Códigos fonte e algoritmos

Neste capitulo os códigos fonte utilizados são apresentados, baseados em Liu e Liu (2003).

Algoritimo 1 – Código Fonte "av_vel.f90" (2022); adaptado de Liu e Liu (2003).

```
!ARQUIVO DE CACULO DE DENSIDADE POR SOMATORIO E CONTINUIDADE!

!Arquivo para calcular a densidade com o algoritimo SPH de somatório.

! ntotal : Número de partículas                          [in]
! hsml   : Cumprimento de suavização                     [in]
! mass   : Massa das partículas                          [in]
! niac   : Número de pares de interação                  [in]
! pair_i : Lista do primeiro parceiro de interação       [in]
! pair_j : Lista do primeiro parceiro de interação       [in]
! w      : Núcleo para todos os pares de interação       [in]
! itype  : Tipo de partícula                             [in]
! x      : Coordenadas de todas as particulas            [in]
! rho    : Densidade                                    [out]

subroutine sum_density(ntotal,hsml,mass,niac,pair_i,pair_j,w, &
itype,rho)

implicit none
include 'param.f90'

integer ntotal, niac, pair_i(max_interaction), &
pair_j(max_interaction), itype(maxn)
```

```
double precision hsml(maxn),mass(maxn), w(max_interaction), &
rho(maxn)
integer i, j, k, d
double precision selfdens, hv(dim), r, wi(maxn)

! wi(maxn)--- Integração do próprio núcleo

do d=1,dim
  hv(d) = 0.e0
enddo

! Densidade própria de cada partícula: Wii (Núcleo para distancia 0)
! e computar a contribuição da própria particula:

r=0.

! 1 -Primeiro calculo é a integração do núcleo sobre o espaço.

do i=1,ntotal
  call kernel(r,hv,hsml(i),selfdens,hv)
  wi(i)=selfdens*mass(i)/rho(i)
enddo

do k=1,niac
  i = pair_i(k)
  j = pair_j(k)
  wi(i) = wi(i) + mass(j)/rho(j)*w(k)
  wi(j) = wi(j) + mass(i)/rho(i)*w(k)
enddo}
```

Algoritimo 2 – Código Fonte “density.f90” (2022); adaptado de Liu e Liu (2003).

```
!ARQUIVO DE CACULO DE DENSIDADE POR SOMATORIO E CONTINUIDADE!

!Arquivo para calcular a densidade com o algoritimo SPH de somatório.

! ntotal : Número de partículas                              [in]
! hsml   : Cumprimento de suavização                         [in]
! mass   : Massa das partículas                              [in]
! niac   : Número de pares de interação                      [in]
! pair_i : Lista do primeiro parceiro de interação           [in]
! pair_j : Lista do primeiro parceiro de interação           [in]
! w      : Núcleo para todos os pares de interação           [in]
! itype  : Tipo de partícula                                 [in]
! x      : Coordenadas de todas as particulas                [in]
! rho    : Densidade                                        [out]

subroutine sum_density(ntotal,hsml,mass,niac,pair_i,pair_j,w, &
itype,rho)

implicit none
include 'param.f90'

integer ntotal, niac, pair_i(max_interaction), &
pair_j(max_interaction), itype(maxn)
double precision hsml(maxn),mass(maxn), w(max_interaction), &
rho(maxn)
integer i, j, k, d
double precision selfdens, hv(dim), r, wi(maxn)

! wi(maxn)--- Integração do próprio núcleo

do d=1,dim
  hv(d) = 0.e0
enddo

! Densidade própria de cada partícula: Wii (Núcleo para distancia 0)
! e computar a contribuição da própria particula:

r=0.
```

```
! 1 -Primeiro calculo é a integração do núcleo sobre o espaço.

do i=1,ntotal
  call kernel(r,hv,hsml(i),selfdens,hv)
  wi(i)=selfdens*mass(i)/rho(i)
enddo

do k=1,niac
  i = pair_i(k)
  j = pair_j(k)
  wi(i) = wi(i) + mass(j)/rho(j)*w(k)
  wi(j) = wi(j) + mass(i)/rho(i)*w(k)
enddo

! 2 - Segundo calculo é a integração de rho sobre o espaço.

do i=1,ntotal
  call kernel(r,hv,hsml(i),selfdens,hv)
  rho(i) = selfdens*mass(i)
enddo

! 2.1 - Calculo do somatório SPH de rho
  do k=1,niac
  i = pair_i(k)
  j = pair_j(k)
  rho(i) = rho(i) + mass(j)*w(k)
  rho(j) = rho(j) + mass(i)*w(k)
enddo

! 3 - Calculo do rho normalizado rho=sum(rho)/sum(w)

if (nor_density) then
do i=1, ntotal
  rho(i)=rho(i)/wi(i)
enddo
endif

end
```

```
subroutine con_density(ntotal,mass,niac,pair_i,pair_j, &
dwdx,vx,itype,x,rho, drhodt)

!------------------------------------------------------------------
!Subroutine to calculate the density with SPH continuiity approach.


! ntotal : Número de partículas                              [in]
! hsml   : Cumprimento de suavização                         [in]
! mass   : Massa das partículas                              [in]
! niac   : Número de pares de interação                      [in]
! pair_i : Lista do primeiro parceiro de interação           [in]
! pair_j : Lista do primeiro parceiro de interação           [in]
! w      : Núcleo para todos os pares de interação           [in]
! itype  : Tipo de partícula                                 [in]
! x      : Coordenadas de todas as particulas                [in]
! rho    : Densidade
! dwdx   : Derivação do núcleo para todos os pares de interaçã  [in]
! vx     : Velocidade de todas as particulas                 [in]
! drhodt : Variação de densidade de cada partícula           [out]

implicit none
include 'param.f90'

integer ntotal,niac,pair_i(max_interaction), &
pair_j(max_interaction), itype(maxn)
double precision mass(maxn), dwdx(dim, max_interaction), &
vx(dim,maxn), x(dim,maxn), rho(maxn), drhodt(maxn)
integer i,j,k,d
double precision    vcc, dvx(dim)

do i = 1, ntotal
drhodt(i) = 0.
enddo

do k=1,niac
  i = pair_i(k)
  j = pair_j(k)
```

```
  do d=1,dim
    dvx(d) = vx(d,i) - vx(d,j)
  enddo
vcc = dvx(1)*dwdx(1,k)
do d=2,dim
  vcc = vcc + dvx(d)*dwdx(d,k)
enddo
  drhodt(i) = drhodt(i) + mass(j)*vcc
  drhodt(j) = drhodt(j) + mass(i)*vcc
enddo

end
```

Algoritimo 3 – Código Fonte "direct_find.f90" (2022); adaptado de Liu e Liu (2003).

```
! ARQUIVO PARA LOCALIZACAO DIRETA DE PARTICULAS !

! Arquivo para o calculo da função de suavização para cada particula
! e parametros usados no algoritimo SPH. Os pares de interacao são determinados
! por comparação direta da distancia com o cumprimento de suavização


!     itimestep : Atual passo de tempo                          [in]
!     ntotal    : Número de partículas                          [in]
!     hsml      : cumprimento de suavização.                    [in]
!     x         : Coordenadas de todas as partículas            [in]
!     niac      : Número de pares de interação                  [out]
!     pair_i    : Lista do primeiro parceiro de interação do par [out]
!     pair_j    : Lista do segundo parceiro de interação do par [out]
!     w         : Núcleo para todos os pares de interação       [out]
!     dwdx      : Derivação do núcleo com respeito a x,y e z    [out]
!     countiac  : Número de particulas vizinhas                 [out]


subroutine direct_find(itimestep, ntotal,hsml,x,niac,pair_i, &
pair_j,w,dwdx,countiac)

implicit none
include 'param.f90'
```

```
integer itimestep, ntotal,niac,pair_i(max_interaction), &
pair_j(max_interaction), countiac(maxn)
double precision hsml(maxn), x(dim,maxn), w(max_interaction), &
dwdx(dim,max_interaction)
integer i, j, d,  sumiac, maxiac, miniac, noiac, &
maxp, minp, scale_k
double precision dxiac(dim), driac, r, mhsml, tdwdx(dim)

if (skf.eq.1) then
  scale_k = 2
else if (skf.eq.2) then
  scale_k = 3
else if (skf.eq.3) then
  scale_k = 3
endif

do i=1,ntotal
  countiac(i) = 0
enddo

niac = 0

do i=1,ntotal-1
  do j = i+1, ntotal
  dxiac(1) = x(1,i) - x(1,j)
  driac    = dxiac(1)*dxiac(1)
    do d=2,dim
    dxiac(d) = x(d,i) - x(d,j)
    driac    = driac + dxiac(d)*dxiac(d)
    enddo
    mhsml = (hsml(i)+hsml(j))/2.
    if (sqrt(driac).lt.scale_k*mhsml) then
      if (niac.lt.max_interaction) then

! Lista de pares vizinhos e número total de interações,
! assim como interações totais de cada particula.

        niac = niac + 1
        pair_i(niac) = i
```

```
          pair_j(niac) = j
          r = sqrt(driac)
          countiac(i) = countiac(i) + 1
          countiac(j) = countiac(j) + 1

! Núcleo e derivação do núcleo.

          call kernel(r,dxiac,mhsml,w(niac),tdwdx)
          do d=1,dim
            dwdx(d,niac) = tdwdx(d)
          enddo
        else
          print *, &
' >>> ERRO <<< : Muitas interacoes'
          stop
        endif
      endif
    enddo
enddo

!  Estatistica para as interações

sumiac = 0
maxiac = 0
miniac = 1000
noiac  = 0
do i=1,ntotal
  sumiac = sumiac + countiac(i)
  if (countiac(i).gt.maxiac) then
    maxiac = countiac(i)
    maxp = i
  endif
  if (countiac(i).lt.miniac) then
    miniac = countiac(i)
      minp = i
  endif
    if (countiac(i).eq.0)       noiac  = noiac + 1
enddo
```

```
if (mod(itimestep,print_step).eq.0) then
  if (int_stat) then
    print *,' >> Statistics: interactions per particle:'
    print *,'**** Particle:',maxp, ' maximal interactions:',maxiac
    print *,'**** Particle:',minp, ' minimal interactions:',miniac
    print *,'**** Average :',real(sumiac)/real(ntotal)
    print *,'**** Total pairs : ',niac
    print *,'**** Particles with no interactions:',noiac
  endif
endif

end
```

Algoritimo 4 – Código Fonte “EOS.f90” (2022); adaptado de Liu e Liu (2003).

```
      ! ARQUIVO PARA EQUACAO DE ESTADO ARTIFICIAL !

!-----------------------------------------------------------------------
!   Gamma law EOS: rotina para calulo de pressão e som

!     rho     : Densidade                                      [in]
!     u       : Energia Interna                                [in]
!     p       : Pressão                                        [out]
!     c       : Velocidade do som                              [out]

subroutine p_gas(rho, u, p, c)
implicit none
double precision rho, u, p, c
double precision gamma

! Para o Ar (gás ideal)

gamma=1.4
p = (gamma-1) * rho * u
c = sqrt((gamma-1) * u)

end
!-----------------------------------------------------------------------
!   Equação de estado artificial para a compressibilidade artificial
```

```
!     rho     : Density                                          [in]
!     u       : Internal energy                                  [in]
!     p       : Pressure                                         [out]
!     c       : sound velocity                                   [out]

subroutine p_art_water(rho, p, c)
implicit none
double precision rho, u, p, c

! EOS Artificial , (Morris, 1997)
c = 0.000035
p = c**2 * rho
! c = 0.01  (Padrão)
end
```

Algoritimo 5 – Código Fonte "external_force.f90" (2022); adaptado de Liu e Liu (2003).

```
!  ARQUIVO PARA O CALCULO DE GRAVIDADE PROPRIA E FORCAS REPULSIVAS


! Rotina para calculo de forças externas como a gravidade,
! diferencial de pressão e interação com particulas virtuais

!     ntotal  : Número de particulas                          [in]
!     mass    : Massa das partículas                          [in]
!     x       : Coordenadas de todas as particulas            [in]
!     pair_i  : Lista do primeiro parceiro de interação       [in]
!     pair_j  : Lista do primeiro parceiro de interação       [in]
!     itype   : Tipo de Particula                             [in]
!     hsml    : Cumprimento de suavização                     [in]
!     dvxdt   : Aceleração com respeito a  x, y e z           [out]

subroutine ext_force(ntotal,mass,x,niac,pair_i,pair_j, &
itype,hsml,dvxdt)
implicit none
include 'param.f90'

integer ntotal, itype(maxn), niac, &
pair_i(max_interaction), pair_j(max_interaction)
double precision mass(maxn), x(dim,maxn), hsml(maxn),    &
```

```
dvxdt(dim,maxn)
integer i, j, k, d
double precision dx(dim), rr, f, rr0, dd, p1, p2

do i = 1, ntotal
  do d = 1, dim
    dvxdt(d, i) = 0
  enddo
enddo

!     Considerar gravidade própria ?

if (self_gravity) then
  do i = 1, ntotal
    dvxdt(1, i) = 2.0e-4
  enddo
endif

!     Força da particula de barreira e penalidade para repulsão.
rr0 = 2e-5 ! Distancia de referencia para considerar o raio de atuação
dd = 6.25e-10 !Valor de referencia livro do liu o quadrado
! da velocidade máx
p1 = 12 !Valor de referencia livro do liu
p2 = 4  !Valor de referencia livro do liu

do  k=1,niac
i = pair_i(k)
j = pair_j(k)
  if(itype(i).gt.0.and.itype(j).lt.0) then
    rr = 0.
    do d=1,dim
      dx(d) =  x(d,i) -  x(d,j)
      rr = rr + dx(d)*dx(d)
    enddo
    rr = sqrt(rr)
    if(rr.lt.rr0) then
      f = ((rr0/rr)**p1-(rr0/rr)**p2)/rr**2
      do d = 1, dim
        dvxdt(d, i) = dvxdt(d, i) + dd*dx(d)*f
```

```
        enddo
      endif
    endif
enddo

end
```

Algoritimo 6 – Código Fonte "input.f90" (2022); adaptado de Liu e Liu (2003).

```
      !GERA O PROBLEMA DE POISEUILLE

!     x-- Coordenadas das partículas                    [out]
!     vx-- Velocidade das partículas                    [out]
!     mass-- Massa das partículas                       [out]
!     rho-- Densidade das partículas                    [out]
!     p-- Pressão das partículas                        [out]
!     u-- Energia Interna das partículas                [out]
!     itype-- Tipo das particulas (2 = Agua)            [out]
!     h-- Cumprimento de suavização das partículas      [out]
!     ntotal-- Número total de partículas               [out]

subroutine input(x, vx, mass, rho, p, u, itype, hsml, ntotal)
implicit none
include 'param.f90'

integer itype(maxn), ntotal
double precision x(dim, maxn), vx(dim, maxn), mass(maxn), &
rho(maxn), p(maxn), u(maxn), hsml(maxn)
integer i, j, d, m, n, mp, np, k
double precision xl, yl, dx, dy, yy

! Configurando dados e outros valores de geometria

m = 21
n = 41
mp = m-1
np = n-1
ntotal = mp * np
xl = 5.e-4
yl = 1.e-3
```

```
dx = xl/mp
dy = yl/np
yy = 1.75e-4

do i = 1, mp
  do j = 1, np
    k = j + (i-1)*np
    x(1, k) = 5e-3 + (i-1)*dx
    x(2, k) = yy + (j-1)*dy
  enddo
enddo

do i = 1, mp*np
  vx(1, i) = abs(0.)
  vx(2, i) = 0.
  rho (i) = 1000.
  mass(i) = dx*dy*rho(i)
  p(i)= 0.
  u(i)= 357.1
  itype(i) = 2
  hsml(i) = 3e-5
enddo

end
```

Algoritimo 7 – Código Fonte “internal_force.f90” (2022); adaptado de Liu e Liu (2003).

```
!CALCULA AS FORCAS INTERNAS

!   Rotina para o calculo das forças internas do lado direito da equação
!   de Navier Stokes. Tensores de estresse viscoso e Gradientes de pressão
!   usados na integração de tempo. produção de entropia (tds/dt) e alteração
!   de energia interna por massa (de/dt).


!     itimestep: Atual passo de tempo                    [in]
!     dt       :   Passo de tempo                        [in]
!     ntotal   : Número de particulas                    [in]
!     hsml     : Cumprimento de suavização               [in]
```

```
!     mass   : Massa das partículas                              [in]
!     vx     : Velocidade de todas as partículas                 [in]
!     niac   : Número de pares de interação                      [in]
!     rho    : Densidade                                         [in]
!     eta    : Densidade dinâmica                                [in]
!     pair_i  : Lista do primeiro parceiro de interação          [in]
!     pair_j  : Lista do primeiro parceiro de interação          [in]
!     dwdx   : Derivada do núcleo com respeito a x, y e z        [in]
!     itype  : Tipo de patícula (Tipos materiais)                [in]
!     u      : Energia interna da particula                      [in]
!     x      : Coordenada da particula                           [in]
!     itype  : tipo da particula                                 [in]
!     t      : Temperatura da particula                       [in/out]
!     c      : Velocidade do som da particula                   [out]
!     p      : Pressãoa da particula                            [out]
!     dvxdt  : Aceleração com respeito a  x, y e z              [out]
!     tdsdt  : Produção de entropia viscosa                     [out]
!     dedt   : Alteração na energia interna especifica          [out]

subroutine int_force(itimestep,dt,ntotal,hsml,mass,vx,niac,rho, &
eta,pair_i,pair_j,dwdx,u,itype,x,t,c,p,dvxdt,tdsdt,dedt)
implicit none
include 'param.f90'

integer itimestep, ntotal,niac,pair_i(max_interaction), &
pair_j(max_interaction), itype(maxn)
double precision dt, hsml(maxn), mass(maxn), vx(dim,maxn), &
rho(maxn), eta(maxn), dwdx(dim,max_interaction), u(maxn), &
x(dim,maxn), t(maxn), c(maxn), p(maxn), dvxdt(dim,maxn), &
tdsdt(maxn),dedt(maxn)
integer i, j, k, d
double precision  dvx(dim), txx(maxn), tyy(maxn), &
tzz(maxn), txy(maxn), txz(maxn), tyz(maxn), vcc(maxn), &
hxx, hyy, hzz, hxy, hxz, hyz, h, hvcc, he, rhoij

!     Inicialização do tensor de cisalhamento, divergência de velocidade
!     energia viscosa, energia interna e aceleração

do i=1,ntotal
```

```
  txx(i) = 0.e0
  tyy(i) = 0.e0
  tzz(i) = 0.e0
  txy(i) = 0.e0
  txz(i) = 0.e0
  tyz(i) = 0.e0
  vcc(i) = 0.e0
  tdsdt(i) = 0.e0
  dedt(i) = 0.e0
  do d=1,dim
    dvxdt(d,i) = 0.e0
  enddo
enddo

!     Calculando o somatório SPH para o tensor de cisalhamento
!     Tab = va,b + vb,a - 2/3 delta_ab vc,c

if (visc) then
  do k=1,niac
    i = pair_i(k)
    j = pair_j(k)
    do d=1,dim
      dvx(d) = vx(d,j) - vx(d,i)
    enddo
    if (dim.eq.1) then
      hxx = 2.e0*dvx(1)*dwdx(1,k)
    else if (dim.eq.2) then
      hxx = 2.e0*dvx(1)*dwdx(1,k) -  dvx(2)*dwdx(2,k)
      hxy = dvx(1)*dwdx(2,k) + dvx(2)*dwdx(1,k)
      hyy = 2.e0*dvx(2)*dwdx(2,k) - dvx(1)*dwdx(1,k)

    endif
    hxx = 2.e0/3.e0*hxx
    hyy = 2.e0/3.e0*hyy
    hzz = 2.e0/3.e0*hzz
    if (dim.eq.1) then
      txx(i) = txx(i) + mass(j)*hxx/rho(j)
      txx(j) = txx(j) + mass(i)*hxx/rho(i)
      else if (dim.eq.2) then
```

```
      txx(i) = txx(i) + mass(j)*hxx/rho(j)
      txx(j) = txx(j) + mass(i)*hxx/rho(i)
      txy(i) = txy(i) + mass(j)*hxy/rho(j)
      txy(j) = txy(j) + mass(i)*hxy/rho(i)
      tyy(i) = tyy(i) + mass(j)*hyy/rho(j)
      tyy(j) = tyy(j) + mass(i)*hyy/rho(i)
    else if (dim.eq.3) then
      txx(i) = txx(i) + mass(j)*hxx/rho(j)
      txx(j) = txx(j) + mass(i)*hxx/rho(i)
      txy(i) = txy(i) + mass(j)*hxy/rho(j)
      txy(j) = txy(j) + mass(i)*hxy/rho(i)
      txz(i) = txz(i) + mass(j)*hxz/rho(j)
      txz(j) = txz(j) + mass(i)*hxz/rho(i)
      tyy(i) = tyy(i) + mass(j)*hyy/rho(j)
      tyy(j) = tyy(j) + mass(i)*hyy/rho(i)
      tyz(i) = tyz(i) + mass(j)*hyz/rho(j)
      tyz(j) = tyz(j) + mass(i)*hyz/rho(i)
      tzz(i) = tzz(i) + mass(j)*hzz/rho(j)
      tzz(j) = tzz(j) + mass(i)*hzz/rho(i)
    endif

!    Calculando o somatório SPH para vc, c = dvx/dx + dvy/dy + dvz/dz:


    hvcc = 0.
    do d=1,dim
      hvcc = hvcc + dvx(d)*dwdx(d,k)
    enddo
    vcc(i) = vcc(i) + mass(j)*hvcc/rho(j)
    vcc(j) = vcc(j) + mass(i)*hvcc/rho(i)
  enddo
endif

do i=1,ntotal

!     Entropia da viscosidade Tds/dt = 1/2 eta/rho Tab Tab

  if (visc) then
    if (dim.eq.1) then
```

```
      tdsdt(i) = txx(i)*txx(i)
    else if (dim.eq.2) then
      tdsdt(i) = txx(i)*txx(i) + 2.e0*txy(i)*txy(i)  &
           + tyy(i)*tyy(i)
    else if (dim.eq.3) then
      tdsdt(i) = txx(i)*txx(i) + 2.e0*txy(i)*txy(i)  &
      + 2.e0*txz(i)*txz(i)  &
      + tyy(i)*tyy(i) + 2.e0*tyz(i)*tyz(i)  &
      + tzz(i)*tzz(i)
    endif
    tdsdt(i) = 0.5e0*eta(i)/rho(i)*tdsdt(i)
  endif

!     Pressão da equação de estado

  if (abs(itype(i)).eq.1) then
    call p_gas(rho(i), u(i), p(i),c(i))
  else if (abs(itype(i)).eq.2) then
    call p_art_water(rho(i), p(i), c(i))
  endif

enddo

!      Calculo do somatório SPH para força de pressão -p,a/rho
!      e força viscosa (eta Tab),b/rho
!      e a mudança de energia interna de de/dt due para -p/rho vc,c

do k=1,niac
  i = pair_i(k)
  j = pair_j(k)
  he = 0.e0

!     para algoritimo SPH 1

  rhoij = 1.e0/(rho(i)*rho(j))
  if(pa_sph.eq.1) then
    do d=1,dim

!     Parte da Pressão
```

```
      h = -(p(i) + p(j))*dwdx(d,k)
      he = he + (vx(d,j) - vx(d,i))*h

!     Força viscosa

      if (visc) then

        if (d.eq.1) then

!     x-coordenada de aceleração

          h = h + (eta(i)*txx(i) + eta(j)*txx(j))*dwdx(1,k)
          if (dim.ge.2) then
            h = h + (eta(i)*txy(i) + eta(j)*txy(j))*dwdx(2,k)

          endif
        elseif (d.eq.2) then

!     y-coordenada de acelração

          h = h + (eta(i)*txy(i) + eta(j)*txy(j))*dwdx(1,k) &
              + (eta(i)*tyy(i) + eta(j)*tyy(j))*dwdx(2,k)

        elseif (d.eq.3) then

!     z-coordenada de aceleração

          h = h + (eta(i)*txz(i) + eta(j)*txz(j))*dwdx(1,k) &
            + (eta(i)*tyz(i) + eta(j)*tyz(j))*dwdx(2,k)

          endif
        endif
        h = h*rhoij
        dvxdt(d,i) = dvxdt(d,i) + mass(j)*h
        dvxdt(d,j) = dvxdt(d,j) - mass(i)*h
      enddo
      he = he*rhoij
      dedt(i) = dedt(i) + mass(j)*he
```

```
      dedt(j) = dedt(j) + mass(i)*he

!     Para algoritimo SPH 2

    else if (pa_sph.eq.2) then
      do d=1,dim
        h = -(p(i)/rho(i)**2 + p(j)/rho(j)**2)*dwdx(d,k)
          he = he + (vx(d,j) - vx(d,i))*h

!     Força viscosa

        if (visc) then
          if (d.eq.1) then

!     x-coordenada de aceleração

        h = h + (eta(i)*txx(i)/rho(i)**2 + &
          eta(j)*txx(j)/rho(j)**2)*dwdx(1,k)
          if (dim.ge.2) then
            h = h + (eta(i)*txy(i)/rho(i)**2 + &
              eta(j)*txy(j)/rho(j)**2)*dwdx(2,k)

          endif
        elseif (d.eq.2) then

!     y-coordenada de aceleração

          h = h + (eta(i)*txy(i)/rho(i)**2  &
            +  eta(j)*txy(j)/rho(j)**2)*dwdx(1,k) &
            + (eta(i)*tyy(i)/rho(i)**2   &
            +  eta(j)*tyy(j)/rho(j)**2)*dwdx(2,k)

        elseif (d.eq.3) then

!     z-coordenada de aceleração

          h = h + (eta(i)*txz(i)/rho(i)**2 + &
            eta(j)*txz(j)/rho(j)**2)*dwdx(1,k) &
            + (eta(i)*tyz(i)/rho(i)**2 + &
```

```
            eta(j)*tyz(j)/rho(j)**2)*dwdx(2,k)

        endif
      endif
      dvxdt(d,i) = dvxdt(d,i) + mass(j)*h
      dvxdt(d,j) = dvxdt(d,j) - mass(i)*h
    enddo
    dedt(i) = dedt(i) + mass(j)*he
    dedt(j) = dedt(j) + mass(i)*he
    endif
enddo

!     Mudança na energia especifica interna de/dt = T ds/dt - p/rho vc,c:

do i=1,ntotal
  dedt(i) = tdsdt(i) + 0.5e0*dedt(i)
enddo

end
```

Algoritimo 8 – Código Fonte "kernel.f90" (2022); adaptado de Liu e Liu (2003).

```
 ! CALCULO DA FUNCAO NUCLEO
!   Rotina para calcula a função núcleo de suavização (wij) e suas de-
! rivadas(dwdxij).
!     r    : Distancia entre as partículas i e j              [in]
!     dx   : Distancia x-, y- e z-entre as i e j              [in]
!     hsml : Cumprimento de suavização                        [in]
!     w    : Núcleo para todos os pares de interação          [out]
!     dwdx : Derivada do núcleo com respeito a  x, y e z      [out]

subroutine kernel(r,dx,hsml,w,dwdx)
implicit none
include 'param.f90'

double precision r, dx(dim), hsml, w, dwdx(dim)
integer i, j, d
double precision q, dw, factor
```

```
q = r/hsml
w = 0.e0
do d=1,dim
dwdx(d) = 0.e0
enddo

if (skf.eq.1) then      !Função nucleo 1

  if (dim.eq.1) then    !Fatores
    factor = 1.e0/hsml
  elseif (dim.eq.2) then
    factor = 15.e0/(7.e0*pi*hsml*hsml)
  elseif (dim.eq.3) then
    factor = 3.e0/(2.e0*pi*hsml*hsml*hsml)
  else
    print *,' >>> Erro <<< : Dimensão errada: Dim =',dim
  stop
  endif
  if (q.ge.0.and.q.le.1.e0) then
    w = factor * (2./3. - q*q + q**3 / 2.)
    do d = 1, dim
      dwdx(d) = factor * (-2.+3./2.*q)/hsml**2 * dx(d)
    enddo
  else if (q.gt.1.e0.and.q.le.2) then
    w = factor * 1.e0/6.e0 * (2.-q)**3
    do d = 1, dim
      dwdx(d) =-factor * 1.e0/6.e0 * 3.*(2.-q)**2/hsml * (dx(d)/r)
    enddo
  else
    w=0.
      do d= 1, dim
        dwdx(d) = 0.
      enddo
  endif

else if (skf.eq.2) then !Função nucleo 2

  factor = 1.e0 / (hsml**dim * pi**(dim/2.))
```

```
  if(q.ge.0.and.q.le.3) then
    w = factor * exp(-q*q)    ! Fatores
      do d = 1, dim
        dwdx(d) = w * ( -2.* dx(d)/hsml/hsml)
      enddo
  else
    w = 0.
    do d = 1, dim
      dwdx(d) = 0.
    enddo
  endif

else if (skf.eq.3) then ! Função nucleo 3

  if (dim.eq.1) then
    factor = 1.e0 / (120.e0*hsml) !Fatores
  elseif (dim.eq.2) then
    factor = 7.e0 / (478.e0*pi*hsml*hsml)
  elseif (dim.eq.3) then
    factor = 1.e0 / (120.e0*pi*hsml*hsml*hsml)
  else
    print *,' >>> Error <<< : Wrong dimension: Dim =',dim
    stop
  endif
  if(q.ge.0.and.q.le.1) then
    w = factor * ( (3-q)**5 - 6*(2-q)**5 + 15*(1-q)**5 )
    do d= 1, dim
      dwdx(d) = factor * ( (-120 + 120*q - 50*q**2) &
      / hsml**2 * dx(d) )
    enddo
  else if(q.gt.1.and.q.le.2) then
    w = factor * ( (3-q)**5 - 6*(2-q)**5 )
    do d= 1, dim
      dwdx(d) = factor * (-5*(3-q)**4 + 30*(2-q)**4)  &
        / hsml * (dx(d)/r)
    enddo
  else if(q.gt.2.and.q.le.3) then
    w = factor * (3-q)**5
    do d= 1, dim
```

```
        dwdx(d) = factor * (-5*(3-q)**4) / hsml * (dx(d)/r)
      enddo
    else
      w = 0.
      do d = 1, dim
      dwdx(d) = 0.
      enddo
    endif

endif

end
```

Algoritimo 9 – Código Fonte "output.f90" (2022); adaptado de Liu e Liu (2003).

```
!     IMPRIME(SALVA) OS DADOS DO PROCESSO NO DISCO

!       x-- Coordenadas das partículas                    [in]
!       vx-- Velocidade das partículas                    [in]
!       mass-- Massa das partículas                       [in]
!       rho-- Densidade das partículas                    [in]
!       p-- Pressão das partículas                        [in]
!       u-- Energia interna das partículas                [in]
!       !-- ~Velocidade do som das partículas            [in]
!       itype-- Tipo das partículas                       [in]
!       hsml-- Cumprimento de suavização das partículas   [in]
!       ntotal-- Número total de part~'iculas             [in]

subroutine output(x, vx, mass, rho, p, u, c, itype, hsml, ntotal)
implicit none
include 'param.f90'

integer itype(maxn), ntotal
double precision x(dim, maxn), vx(dim, maxn), mass(maxn), &
rho(maxn),p(maxn), u(maxn), c(maxn), hsml(maxn)
integer i, d, npart

open(1,file="f_xv.dat", status = 'replace')
!open(2,file="f_state.dat", status = 'new')
```

```
!open(3,file="f_other.dat", status = 'new')

write(1,*) ntotal
do i = 1, ntotal
  write(1,1001) i, (x(d, i), d=1,dim), (vx(d, i), d = 1, dim)
  write(2,1002) i, mass(i), rho(i), p(i), u(i)
  write(3,1003) i, itype(i), hsml(i)
enddo

1001  format(1x, I6, 6(2x, e14.8))
1002  format(1x, I6, 7(2x, e14.8))
1003  format(1x, I6, 2x, I4, 2x, e14.8)

close(1)
close(2)
close(3)

end
```

Algoritimo 10 – Código Fonte "param.f90" (2022); adaptado de Liu e Liu (2003).

```
      !------------------------------------------------------------
! Arquivo para parametros e constantes usadas no programa todo
!
!------------------------------------------------------------

!dim : Dimensão do problema (1, 2 or 3)
integer dim
parameter ( dim = 2)

! maxn : Número máximo de particulas
! max_interation : Número máximo de pares de interação
integer maxn,max_interaction
parameter ( maxn     = 12000     , &
max_interaction = 100 * maxn )

! Parametros para geometria computacional,
!x_maxgeom : Upper limit of allowed x-regime
!x_mingeom : Lower limit of allowed x-regime
```

```
!y_maxgeom : Upper limit of allowed y-regime
!y_mingeom : Lower limit of allowed y-regime
!z_maxgeom : Upper limit of allowed z-regime
!z_mingeom : Lower limit of allowed z-regime
double precision x_maxgeom,x_mingeom,y_maxgeom, &
y_mingeom,z_maxgeom,z_mingeom
parameter ( x_maxgeom =  1000.e0      , &
x_mingeom = -1000.e0      , &
y_maxgeom =  1000.e0      , &
y_mingeom = -1000.e0      , &
z_maxgeom =  1000.e0      , &
z_mingeom = -1000.e0      )

! Algoritimo SPH para aproximação de partículas (pa_sph)
! pa_sph = 1 : (e.g. (p(i)+p(j))/(rho(i)*rho(j))
!          2 : (e.g. (p(i)/rho(i)**2+p(j)/rho(j)**2)
integer pa_sph
parameter(pa_sph = 2)

! Método de busca de partículas vizinhas
! nnps = 1 : Busca direta, dentro do raio estipulado.

integer nnps
parameter(nnps = 1 )

! Função núcleo de suavização
! skf = 1, Splice cúbica - Spline (Monaghan 1985)
!     = 2, Gauss   (Gingold and Monaghan 1981)
!     = 3, Quintica (Morris 1997)
integer skf
parameter(skf = 1)

! Ativadores de cógidos

! summation_density = .TRUE. : Usa o modelo de somatório de densidade.
!                     .FALSE.: Usa equação de continuidade
! average_velocity = .TRUE. : Com tratamento de média de velocidade de Monaghan
!                    .FALSE.: sem tratamento de média.
! config_input = .TRUE. : Carrega configuração inicial,
```

```
!                .FALSE.: Gera configuração inicial.
! virtual_part = .TRUE. : Usa particulas virtuais,
!                .FALSE.: Não usa particulas virtuais.
! visc = .true. : considera viscosidade,
!        .false.: Sem viscosidade.
! ex_force =.true. : Considera força externa,
!           .false.: Não considera força externa.
! self_gravity = .true. : Considera "própria gravidade",
!                .false.: Não considera "própria gravidade"

logical summation_density, average_velocity, config_input, &
virtual_part, vp_input, visc, ex_force, heat_artificial, &
visc_artificial, self_gravity, nor_density
parameter ( summation_density  = .true. )
parameter ( average_velocity  = .true. )
parameter ( config_input  = .true. )
parameter ( virtual_part  = .true. )
parameter ( visc  = .true.  )
parameter ( ex_force  = .true.)
parameter ( self_gravity  = .true. )

! Simetria do problema
! nsym = 0 : sem simetria,
!      = 1 : simetria no eixo,
!      = 2 : simetria centralizada.
integer    nsym
parameter ( nsym = 0)

! Parametros de controle para imprimir informações
! int_stat = .true. : Imprimi estatisticas sobre as interações
! de particulas SPH, inclusive virtuais.
! print_step: Imprime o passo de tempo (Na tela)
! save_step : Salva o passo de tempo    (Arquivo em disco)
! moni_particle: O número de particulas a serem monitoradas.
logical int_stat
parameter ( int_stat = .true. )
integer print_step, save_step, moni_particle
parameter ( print_step = 100 , &
save_step = 6000,  &
```

```
moni_particle = 8000    )

double precision pi
parameter ( pi = 3.14159265358979323846 )

! Caso de simulação Poiseuille
! Poiseuille = .true. : Simula o fluxo de Poiseuille
logical poiseuille
parameter ( poiseuille  = .true. )
```

Algoritimo 11 – Código Fonte “single_step.f90” (2022); adaptado de Liu e Liu (2003).

```
    ! PASSO DE TEMPO/INTEGRACAO DE TEMPO


!   Rotina para determinar o lado direito de uma equação diferencial
!   em um passo de tempo para performar a integração de tempo. Nessa rotina
!   o algoritimo SPH é executado.

!     itimestep: Atual passo de tempo                          [in]
!     dt       : Passo de tempo                                [in]
!     ntotal   :  Número de particulas                         [in]
!     hsml     :  Cumprimento de suavização                    [in]
!     mass     :  Massa das partículas                         [in]
!     x        :  Posição das partículas                       [in]
!     vx       :  Velocidade das partículas                    [in]
!     u        :  Energia interna das partículas               [in]
!     s        :  Entropia das partículas  (Não é usada aqui)  [in]
!     rho      :  Densidade                                    [in/out]
!     p        :  Pressão                                      [out]
!     t        :  Temperatura                                  [in/out]
!     tdsdt    :  Produção de entropia viscosa t*ds/dt         [out]
!     dx       :  dx = vx = dx/dt                              [out]
!     dvx      :  dvx = dvx/dt, force per unit mass            [out]
!     du       :  du  = du/dt                                  [out]
!     ds       :  ds  = ds/dt                                  [out]
!     drho     :  drho =  drho/dt                              [out]
!     itype    :  Tipo de particula                            [in]
!     av       :  Monaghan velocidade média                    [out]
```

```
subroutine single_step(itimestep, dt, ntotal, hsml, mass, x, vx,  &
u, s,rho, p, t, tdsdt, dx, dvx, du, ds, drho,itype, av)
implicit none
include 'param.f90'

integer itimestep, ntotal, itype(maxn)
double precision dt, hsml(maxn), mass(maxn), x(dim,maxn), &
vx(dim,maxn), u(maxn), s(maxn), rho(maxn), p(maxn), &
t(maxn), tdsdt(maxn), dx(dim,maxn), dvx(dim,maxn), &
du(maxn), ds(maxn), drho(maxn), av(dim, maxn)
integer i, d, nvirt, niac, pair_i(max_interaction), &
pair_j(max_interaction), ns(maxn)
double precision w(max_interaction), dwdx(dim,max_interaction),&
indvxdt(dim,maxn),exdvxdt(dim,maxn),ardvxdt(dim,maxn),  &
avdudt(maxn), ahdudt(maxn), c(maxn), eta(maxn)

do  i=1,ntotal
  avdudt(i) = 0.
  ahdudt(i) = 0.
  do  d=1,dim
    indvxdt(d,i) = 0.
    ardvxdt(d,i) = 0.
    exdvxdt(d,i) = 0.
  enddo
enddo

!---  Posição das particulas virtuais (barreira)

nvirt = 0
if (virtual_part) then
  call virt_part(itimestep, ntotal,nvirt,hsml,mass,x,vx, &
  rho,u,p,itype)
endif

!---  Procura direta de particulas

if (nnps.eq.1) then
  call direct_find(itimestep, ntotal+nvirt,hsml,x,niac,pair_i, &
```

```
  pair_j,w,dwdx,ns)
endif

!---  Aproximação de densidade ou taxa de variação da densiade

if (summation_density) then
  call sum_density(ntotal+nvirt,hsml,mass,niac,pair_i,pair_j,w, &
  itype,rho)
else
  call con_density(ntotal+nvirt,mass,niac,pair_i,pair_j, &
  dwdx,vx, itype,x,rho, drho)
endif

!---  Viscosidade dinamica:

if (visc) call viscosity(ntotal+nvirt,itype,x,rho,eta)

!---  Forças internas:

call int_force(itimestep,dt,ntotal+nvirt,hsml,mass,vx,niac,rho, &
eta, pair_i,pair_j,dwdx,u,itype,x,t,c,p,indvxdt,tdsdt,du)

!---  Forças externas:

if (ex_force) call ext_force(ntotal+nvirt,mass,x,niac, &
pair_i,pair_j,itype, hsml, exdvxdt)

if (average_velocity) call av_vel(ntotal,mass,niac,pair_i, &
pair_j, w, vx, rho, av)

!---  Convertendo velocidade, força e energia para f e dfdt

do i=1,ntotal
  do d=1,dim
    dvx(d,i) = indvxdt(d,i) + exdvxdt(d,i) + ardvxdt(d,i)
  enddo
  du(i) = du(i) + avdudt(i) + ahdudt(i)
enddo
```

```
if (mod(itimestep,print_step).eq.0) then
  write(*,*)
  write(*,*) '**** Informacao para particula ****', &
        moni_particle
  write(*,101)'internal a ','artifical a=', &
        'external a ','total a '
  write(*,100)indvxdt(1,moni_particle),ardvxdt(1,moni_particle), &
        exdvxdt(1,moni_particle),dvx(1,moni_particle)
endif
101   format(1x,4(2x,a12))
100   format(1x,4(2x,e12.6))

end
```

Algoritimo 12 – Código Fonte “sph.f90” (2022); adaptado de Liu e Liu (2003).

```
      !CODIGO SPH TRIDIMENSIONAL

!     mass-- Massa das partículas                              [in]
!     ntotal-- Numero total de particulas                      [in]
!     dt--- Passo de tempo usado na integração de tempo        [in]
!     itype-- types of particles                               [in]
!     x-- Coordenadas das partículas                        [in/out]
!     vx-- Velocidade das partículas                        [in/out]
!     rho-- Densidade das partículas                        [in/out]
!     p-- Pressão das partículas                            [in/out]
!     u-- Energia interna das partículas                    [in/out]
!     hsml-- Cumprimento de suavização das partículas       [in/out]
!     c-- Velocidade do som das partículas                     [out]
!     s-- Entropia das partículas                              [out]
!     e-- Energia total das partículas                         [out]

program SPH
implicit none
include 'param.f90'

integer ntotal, itype(maxn), maxtimestep, d, m, i, yesorno
double precision x(dim,maxn), vx(dim, maxn), mass(maxn),rho(maxn), &
```

```
p(maxn), u(maxn), c(maxn), s(maxn), e(maxn), hsml(maxn), dt
double precision s1, s2

call time_print
call time_elapsed(s1)

if (poiseuille) dt = 1.e-4
call input(x, vx, mass, rho, p, u, itype, hsml, ntotal)
1     write(*,*)' ************************************************'
write(*,*)'     Por favor coloque o numero de passos maximo '
write(*,*)' ************************************************'
read(*,*) maxtimestep
call time_integration(x, vx, mass, rho, p, u, c, s, e, itype, &
hsml, ntotal, maxtimestep, dt )
call output(x, vx, mass, rho, p, u, c, itype, hsml, ntotal)
write(*,*)' ************************************************'
write(*,*)       'Voce ira rodar mais passos ? (0=no, 1=yes)'
write(*,*)' ************************************************'
read (*,*) yesorno
if (yesorno.ne.0) go to 1
call time_print
call time_elapsed(s2)
write (*,*)'         Elapsed CPU time = ', s2-s1

end
```

Algoritimo 13 – Código Fonte “time_elapsed.f90” (2022); adaptado de Liu e Liu (2003).

```
   subroutine time_elapsed(s)

!================================================
!   The standard Fortran 90 routine RTC
!   is used to calculate the elapsed CPU
!================================================

use dfport
implicit none

integer, parameter :: output = 6
```

```
real(8) :: s

s = rtc()

end subroutine time_elapsed
```

Algoritimo 14 – Código Fonte “time_integration.f90” (2022); adaptado de Liu e Liu (2003).

```
    ! INTEGRACAO DE TEMPO

!-----------------------------------------------------------------------
!       x-- Coordenadas das partículas                  [input/output]
!       vx-- Velocidades das partículas                 [input/output]
!       mass-- Massa das partículas                     [input]
!       rho-- Densidade das partículas                  [input/output]
!       p-- Pressão das partículas                      [input/output]
!       u-- Energia Interna das partículas              [input/output]
!       c-- Velocidade do som das partículas            [output]
!       s-- Entropia das partículas (não usada aqui)    [output]
!       e-- Energia total das partículas                [output]
!       itype-- Tipo das partículas                     [input]
!            =1   gas ideal
!            =2   agua
!            =3   TNT (Explosivo)
!       hsml-- Cumprimento de suavização das partículas [input/output]
!       ntotal-- Número total das partículas            [input]
!       maxtimestep-- Número total de passos de tempo   [input]
!       dt-- Passo de tempo                             [input]

subroutine time_integration(x,vx, mass, rho, p, u, c, s, e, itype, &
hsml, ntotal, maxtimestep, dt )

implicit none
include 'param.f90'

integer itype(maxn), ntotal, maxtimestep
double precision x(dim, maxn), vx(dim, maxn), mass(maxn), &
rho(maxn), p(maxn), u(maxn), c(maxn), s(maxn), e(maxn), &
hsml(maxn), dt
```

```
integer i, j, k, itimestep, d, current_ts, nstart
double precision  x_min(dim, maxn), v_min(dim, maxn), u_min(maxn), &
rho_min(maxn), dx(dim,maxn), dvx(dim, maxn), du(maxn),  &
drho(maxn),  av(dim, maxn), ds(maxn), &
t(maxn), tdsdt(maxn)
double precision  time, temp_rho, temp_u

do i = 1, ntotal
  do d = 1, dim
    av(d, i) = 0.
  enddo
enddo

do itimestep = nstart+1, nstart+maxtimestep

  current_ts=current_ts+1
  if (mod(itimestep,print_step).eq.0) then
    write(*,*)'_____________________________________________'
    write(*,*)'  Atual passo de tempo =', &
    itimestep,'      Atual tempo=', real(time+dt)
    write(*,*)'_____________________________________________'
  endif

!     Se não for o primeiro passo de tempo, atualiza a energia termal
!     densidade e velocidade a metade do passo de tempo.

  if (itimestep .ne. 1) then

    do i = 1, ntotal
    u_min(i) = u(i)
    temp_u=0.
  if (dim.eq.1) temp_u=-nsym*p(i)*vx(1,i)/x(1,i)/rho(i)
    u(i) = u(i) + (dt/2.)* (du(i)+temp_u)
    if(u(i).lt.0)  u(i) = 0.

    if (.not.summation_density) then
      rho_min(i) = rho(i)
  temp_rho=0.
  if (dim.eq.1) temp_rho=-nsym*rho(i)*vx(1,i)/x(1,i)
```

```
      rho(i) = rho(i) +(dt/2.)*( drho(i)+ temp_rho)
    endif

    do d = 1, dim
      v_min(d, i) = vx(d, i)
      vx(d, i) = vx(d, i) + (dt/2.)*dvx(d, i)
    enddo
  enddo

endif

!---  Define as variaveis fora do vetor função.

call single_step(itimestep, dt, ntotal, hsml, mass, x, vx, u, s, &
rho, p, t, tdsdt, dx, dvx, du, ds, drho,itype, av)

if (itimestep .eq. 1) then

  do i=1,ntotal
    temp_u=0.
if (dim.eq.1) temp_u=-nsym*p(i)*vx(1,i)/x(1,i)/rho(i)
    u(i) = u(i) + (dt/2.)*(du(i) + temp_u)
    if(u(i).lt.0)  u(i) = 0.

    if (.not.summation_density ) then
  temp_rho=0.
  if (dim.eq.1) temp_rho=-nsym*rho(i)*vx(1,i)/x(1,i)
    rho(i) = rho(i) + (dt/2.)* (drho(i)+temp_rho)
  endif

  do d = 1, dim
    vx(d, i) = vx(d, i) + (dt/2.) * dvx(d, i) + av(d, i)
    x(d, i) = x(d, i) + dt * vx(d, i)
  enddo
enddo

else

  do i=1,ntotal
```

```
      temp_u=0.
if (dim.eq.1) temp_u=-nsym*p(i)*vx(1,i)/x(1,i)/rho(i)
  u(i) = u_min(i) + dt*(du(i)+temp_u)
  if(u(i).lt.0)  u(i) = 0.

  if (.not.summation_density ) then
    temp_rho=0.
  if (dim.eq.1) temp_rho=-nsym*rho(i)*vx(1,i)/x(1,i)
    rho(i) = rho_min(i) + dt*(drho(i)+temp_rho)
  endif

do d = 1, dim
  vx(d, i) = v_min(d, i) + dt * dvx(d, i) + av(d, i)
  x(d, i) = x(d, i) + dt * vx(d, i)
  enddo
enddo

endif

time = time + dt

if (mod(itimestep,save_step).eq.0) then
  call output(x, vx, mass, rho, p, u, c, itype, hsml, ntotal)
endif

if (mod(itimestep,print_step).eq.0) then
  write(*,*)
  write(*,101)'x(Posicao)','velocidade', 'dvx(Aceleracao)'
  write(*,100)x(1,moni_particle), vx(1,moni_particle), &
  dvx(1,moni_particle)
endif

101      format(1x,3(2x,a12))
100      format(1x,3(2x,e12.6))

enddo

nstart=current_ts
```

```
end
```

Algoritimo 15 – Código Fonte “time_print.f90” (2022); adaptado de Liu e Liu (2003).

```
   subroutine time_print

!=======================================================
!   TIME_PRINT      Print out the current date and time.
!
!   Notes:
!
!   The standard Fortran 90 routine
!   DATE_AND_TIME is used to get the current
!   date and time strings.
!
!=======================================================

implicit none
integer, parameter :: output = 6

! . local scalars.
character ( len =  8 ) :: datstr
character ( len = 10 ) :: timstr

! . Get the current date and time.
call date_and_time ( datstr, timstr )

! . Write out the date and time.
write ( output, "(/A)"  ) "         Date = " // datstr(7:8) // "/" // &
datstr(5:6) // "/" // &
datstr(1:4)
write ( output, "(A)"   ) "         Time = " // timstr(1:2) // ":" // &
timstr(3:4) // ":" // &
timstr(5:10)
write ( output, *)

end subroutine time_print
```

Algoritimo 16 – Código Fonte “virtu_part.f90” (2022); adaptado de Liu e Liu (2003).

```
    ! PARTICULAS VIRTUAIS DO FLUXO DE POISEUILLE
    !(PLACAS INFINITAS DE FRONTEIRA)

! Rotina para determinar a informação das particulas virtuais do
! tipo Monaghan para o problema de poiseuille

!     itimestep : Atual Passo de Tempo                          [in]
!     ntotal : Número de partículas                             [in]
!     nvirt  : Número de partículas virtuais                   [out]
!     hsml   : Cumprimento de suavização                     [in|out]
!     mass   : Massa das partículas                          [in|out]
!     x      : Coordenadas das partículas                    [in|out]
!     vx     : Velocidade das partículas                     [in|out]
!     rho    : Densidade                                     [in|out]
!     u      : Energia Interna                               [in|out]
!     itype   : Tipo de partícula                            [in|out]

subroutine virt_part(itimestep, ntotal,nvirt,hsml,mass,x,vx, &
rho,u,p,itype)

implicit none
include 'param.f90'
integer itimestep, ntotal, nvirt, itype(maxn)
double precision hsml(maxn),mass(maxn),x(dim,maxn),vx(dim,maxn), &
rho(maxn), u(maxn), p(maxn)
integer i, j, d, im, mp, np
double precision xl, dx, v_inf, yl, dy, yy

nvirt = 0
mp = 500
np = 40
xl = 1.25e-2
yl = 1.0e-3
yy = 1.175e-3
```

```
dx = xl / mp
dy = yl / np
v_inf = 0

!     Particulas Monaghan do topo (dx = 2,5e-5 dif)

do i = 1, mp+1
  nvirt = nvirt + 1
  x(1, ntotal + nvirt) = (i-1)*dx
    x(2, ntotal + nvirt) = yy
    vx(1, ntotal + nvirt) = v_inf
  vx(2, ntotal + nvirt) = 0.
enddo

do i = 1, mp+1
  nvirt = nvirt + 1
  x(1, ntotal + nvirt) = (i-1)*dx
    x(2, ntotal + nvirt) = yy + 2.5e-5
    vx(1, ntotal + nvirt) = v_inf
  vx(2, ntotal + nvirt) = 0.
enddo

do i = 1, mp+1
  nvirt = nvirt + 1
  x(1, ntotal + nvirt) = (i-1)*dx
    x(2, ntotal + nvirt) = yy + 5e-5
    vx(1, ntotal + nvirt) = v_inf
  vx(2, ntotal + nvirt) = 0.
enddo

do i = 1, mp+1
  nvirt = nvirt + 1
  x(1, ntotal + nvirt) = (i-1)*dx
    x(2, ntotal + nvirt) = yy + 7.5e-5
    vx(1, ntotal + nvirt) = v_inf
  vx(2, ntotal + nvirt) = 0.
enddo

do i = 1, mp+1
```

```
   nvirt = nvirt + 1
   x(1, ntotal + nvirt) = (i-1)*dx
     x(2, ntotal + nvirt) = yy + 1e-4
     vx(1, ntotal + nvirt) = v_inf
   vx(2, ntotal + nvirt) = 0.
enddo

do i = 1, mp+1
   nvirt = nvirt + 1
   x(1, ntotal + nvirt) = (i-1)*dx
     x(2, ntotal + nvirt) = yy + 1.25e-4
     vx(1, ntotal + nvirt) = v_inf
   vx(2, ntotal + nvirt) = 0.
enddo

do i = 1, mp+1
   nvirt = nvirt + 1
   x(1, ntotal + nvirt) = (i-1)*dx
     x(2, ntotal + nvirt) = yy + 1.5e-4
     vx(1, ntotal + nvirt) = v_inf
   vx(2, ntotal + nvirt) = 0.
enddo


!      Particulas Monaghan da base

do i = 1, mp+1
   nvirt = nvirt + 1
   x(1, ntotal + nvirt) = (i-1)*dx
     x(2, ntotal + nvirt) = 0.
     vx(1, ntotal + nvirt) = v_inf
   vx(2, ntotal + nvirt) = 0.
enddo

do i = 1, mp+1
   nvirt = nvirt + 1
   x(1, ntotal + nvirt) = (i-1)*dx
     x(2, ntotal + nvirt) = 0 + 2.5e-5
     vx(1, ntotal + nvirt) = v_inf
```

```
  vx(2, ntotal + nvirt) = 0.
enddo

do i = 1, mp+1
  nvirt = nvirt + 1
  x(1, ntotal + nvirt) = (i-1)*dx
    x(2, ntotal + nvirt) = 0 + 5e-5
    vx(1, ntotal + nvirt) = v_inf
  vx(2, ntotal + nvirt) = 0.
enddo

do i = 1, mp+1
  nvirt = nvirt + 1
  x(1, ntotal + nvirt) = (i-1)*dx
    x(2, ntotal + nvirt) = 0 + 7.5e-5
    vx(1, ntotal + nvirt) = v_inf
  vx(2, ntotal + nvirt) = 0.
enddo

do i = 1, mp+1
  nvirt = nvirt + 1
  x(1, ntotal + nvirt) = (i-1)*dx
    x(2, ntotal + nvirt) = 0 + 1e-4
    vx(1, ntotal + nvirt) = v_inf
  vx(2, ntotal + nvirt) = 0.
enddo

do i = 1, mp+1
  nvirt = nvirt + 1
  x(1, ntotal + nvirt) = (i-1)*dx
    x(2, ntotal + nvirt) = 0 + 1.25e-4
    vx(1, ntotal + nvirt) = v_inf
  vx(2, ntotal + nvirt) = 0.
enddo

do i = 1, mp+1
  nvirt = nvirt + 1
  x(1, ntotal + nvirt) = (i-1)*dx
    x(2, ntotal + nvirt) = 0 + 1.5e-4
```

```
    vx(1, ntotal + nvirt) = v_inf
  vx(2, ntotal + nvirt) = 0.
enddo

! Informações das partículas

do i = 1, nvirt
  rho (ntotal + i) = 1000.
  mass(ntotal + i) = rho (ntotal + i) * dx * dx
  p(ntotal + i) = 0.
  u(ntotal + i) = 357.1
  itype(ntotal + i) = -2
  hsml(ntotal + i) = 3e-5
enddo


if (mod(itimestep,save_step).eq.0) then
  open(1,file="!:\Users\pedro\Desktop\Parametro/xv_vp.dat")
  open(2,file="!:\Users\pedro\Desktop\Parametro/state_vp.dat")
  open(3,file="!:\Users\pedro\Desktop\Parametro/other_vp.dat")
  write(1,*) nvirt
  do i = ntotal + 1, ntotal + nvirt
    write(1,1001) i, (x(d, i), d=1,dim), (vx(d, i), d = 1, dim)
    write(2,1002) i, mass(i), rho(i), p(i), u(i)
    write(3,1003) i, itype(i), hsml(i)
  enddo
1001    format(1x, I6, 6(2x, e14.8))
1002    format(1x, I6, 7(2x, e14.8))
1003    format(1x, I6, 2x, I4, 2x, e14.8)
  close(1)
  close(2)
  close(3)
endif

if (mod(itimestep,print_step).eq.0) then
  if (int_stat) then
    print *,' >> Estatisticas, particulas virtuais de barreira:'
    print *,'          Numero de partículas virtuais:',NVIRT
  endif
```

```
endif

end
```

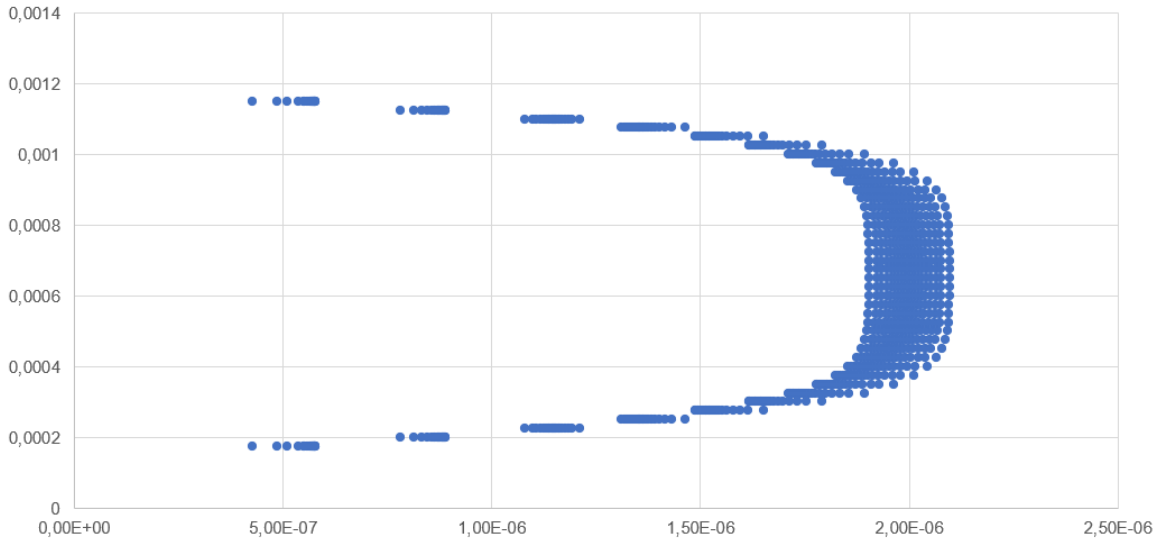
Algoritimo 17 – Código Fonte “viscosity.f90” (2022); adaptado de Liu e Liu (2003).

```
    ! VISCOSIDADE

!     ntotal  : Número de particulas                    [in]
!     itype   : Tipo de particula                       [in]
!     x       : Coordenada de todas as particulas       [in]
!     rho     : Densidade                               [in]
!     eta     : Viscosidade Dinãmica                    [out]

subroutine viscosity(ntotal,itype,x,rho,eta)
implicit none
include 'param.f90'

integer ntotal,i,itype(maxn)
double precision x(dim,maxn),rho(maxn),eta(maxn)

do i=1,ntotal
  if (abs(itype(i)).eq.1) then
    eta(i)=0.
  else if (abs(itype(i)).eq.2) then
    eta(i)=1.0e-3
  endif
enddo

end
```

6 RESULTADOS E DISCUSSÃO

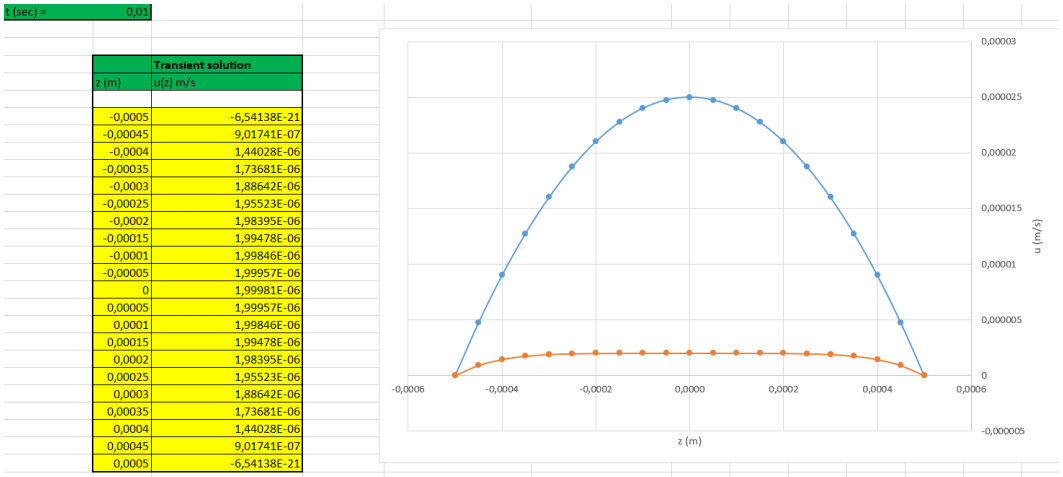
Após a simulação os resultados apresentados são para 0,01 segundos, 0,1 segundos e 0,5 segundos. Os três equivalem a respectivamente 100, 1000 e 5000 passos de tempo com tamanho de 1^{-4} segundos. O experimento foi modelado conforme o exemplo dos autores Liu e Liu (2003) e foi comparado a calculadora analítica de Poiseuille do projeto DualSPHysics dos autores Crespo *et al.* (2014).

Figura 7 – Resultado numérico do método SPH com a simulação de Fluxo de Poiseuille em transição de 0,01 segundos. Curva de velocidade, a simetria do eixo (y)(Vertical) = 0,00675 m. O eixo (x)(Horizontal) representa a velocidade em m/s.



Fonte: Autoria própria (2022).

Figura 8 – Resultado analítico do método SPH com a simulação de Fluxo de Poiseuille em transição de 0,01 segundos, calculadora DualSPHysics. O eixo (z) Horizontal representa a largura entre as placas, (u) a velocidade em m/s.



t (sec) = 0,01

	Transient solution
z (m)	u(z) m/s
-0,0005	-6,54138E-21
-0,00045	9,01741E-07
-0,0004	1,44028E-06
-0,00035	1,73681E-06
-0,0003	1,88642E-06
-0,00025	1,95523E-06
-0,0002	1,98395E-06
-0,00015	1,99478E-06
-0,0001	1,99846E-06
-0,00005	1,99957E-06
0	1,99981E-06
0,00005	1,99957E-06
0,0001	1,99846E-06
0,00015	1,99478E-06
0,0002	1,98395E-06
0,00025	1,95523E-06
0,0003	1,88642E-06
0,00035	1,73681E-06
0,0004	1,44028E-06
0,00045	9,01741E-07
0,0005	-6,54138E-21

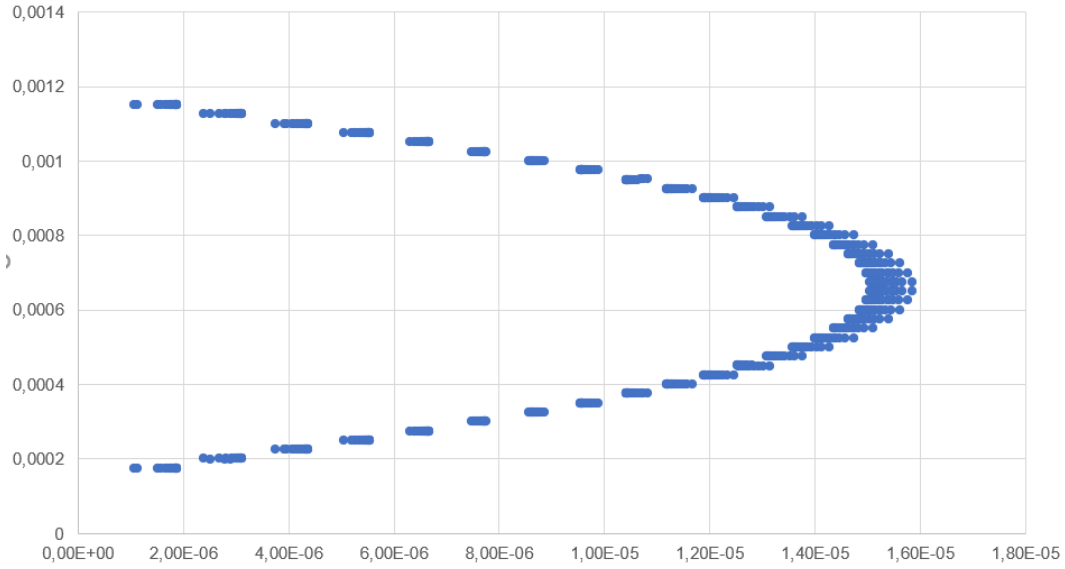
Fonte: (CRESPO *et al.*, 2014) .

Para o calculo de erro relativo a formula (67) foi utilizada (SOUZA; BÍSCARO, 2018), comparando as 50 partículas com maior velocidade ao valor analítico final para aquele espaço de tempo.

$$Errorelativo = \sqrt{\frac{\sum_{i=1}^{n}(V_iSPH - V_iAnalitico)^2}{\sum_{i=1}^{n}(V_iAnalitico)^2}} \tag{67}$$

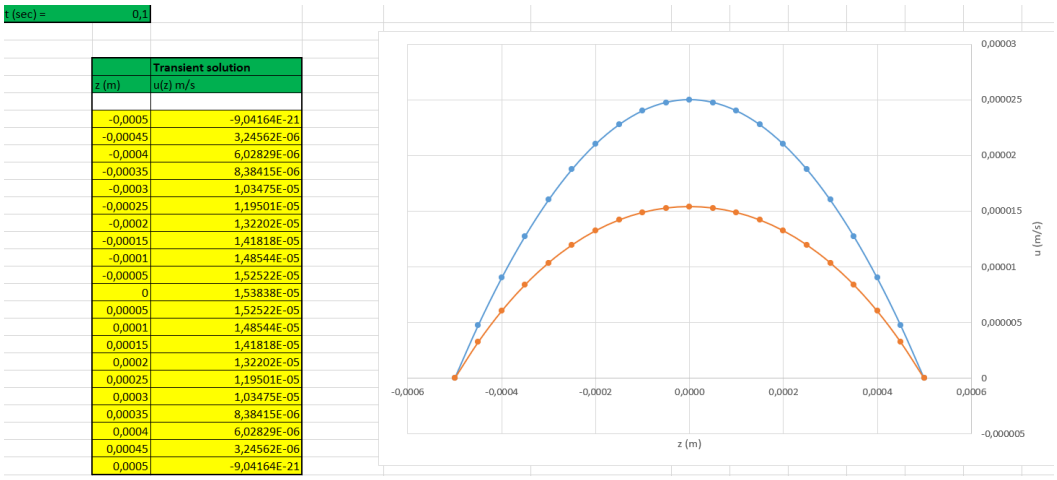
Neste caso, com 0,01 segundos a variação final foi de aproximadamente 0.039%, indicando grande precisão.

Figura 9 – Resultado numérico do método SPH com a simulação de Fluxo de Poiseuille em transição de 0,1 segundos. Curva de velocidade, a simetria do eixo (y)(Vertical) = 0,00675 m. O eixo (x)(Horizontal) representa a velocidade em m/s.



Fonte: Autoria própria (2022).

Figura 10 – Resultado analítico do método SPH com a simulação de Fluxo de Poiseuille em transição de 0,1 segundos, calculadora DualSPHysics. O eixo (z) Horizontal representa a largura entre as placas, (u) a velocidade em m/s.

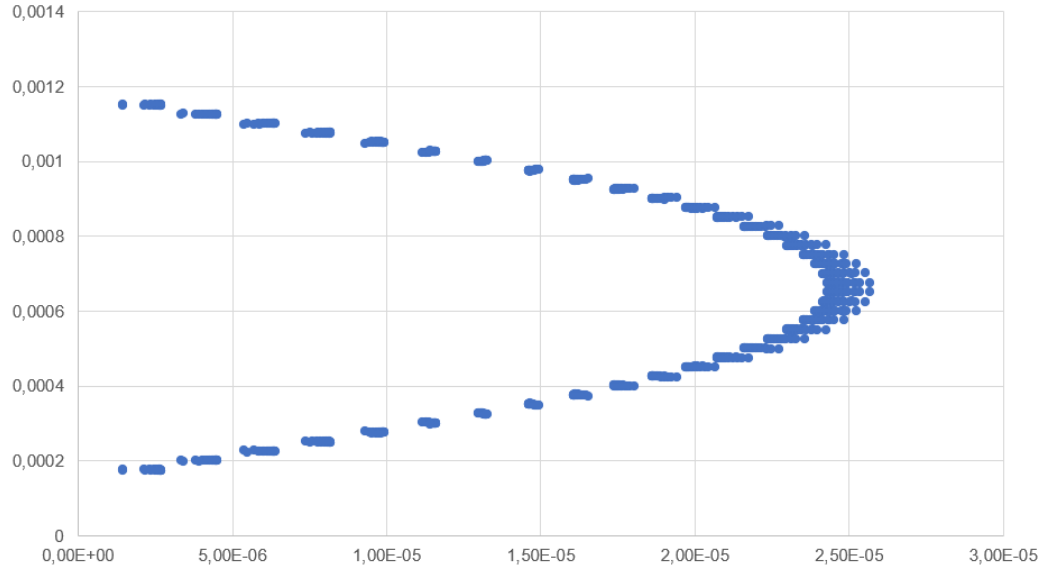


t (sec) =	0,1
	Transient solution
z (m)	u(z) m/s
-0,0005	-9,04164E-21
-0,00045	3,24562E-06
-0,0004	6,02829E-06
-0,00035	8,38415E-06
-0,0003	1,03475E-05
-0,00025	1,19501E-05
-0,0002	1,32202E-05
-0,00015	1,41818E-05
-0,0001	1,48544E-05
-0,00005	1,52522E-05
0	1,53838E-05
0,00005	1,52522E-05
0,0001	1,48544E-05
0,00015	1,41818E-05
0,0002	1,32202E-05
0,00025	1,19501E-05
0,0003	1,03475E-05
0,00035	8,38415E-06
0,0004	6,02829E-06
0,00045	3,24562E-06
0,0005	-9,04164E-21

Fonte: (CRESPO *et al.*, 2014) .

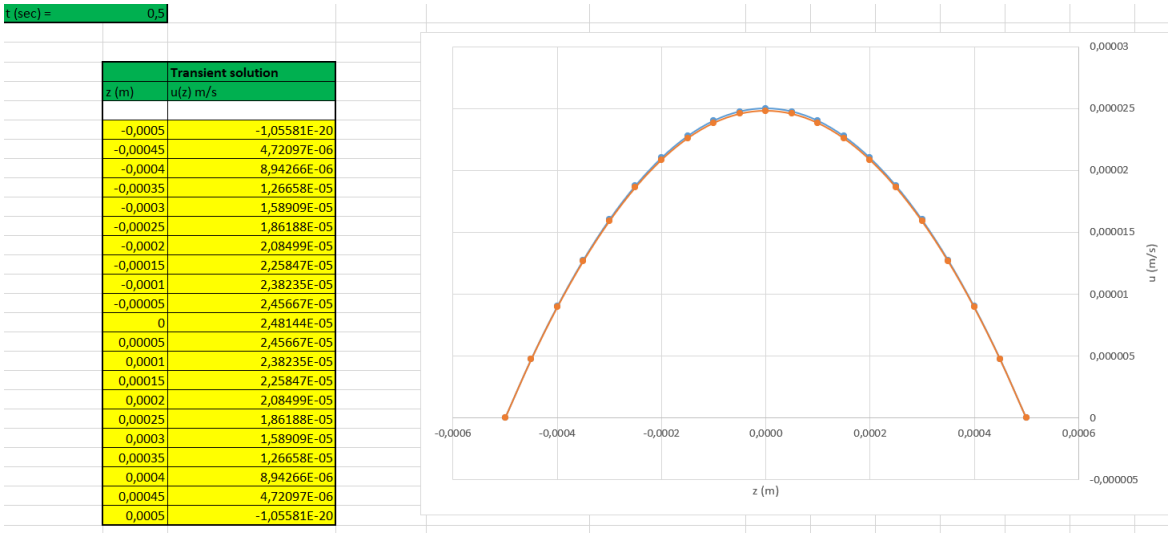
Para a execução com 0,1 segundos a variação final foi de aproximadamente 1,12%, indicando grande precisão.

Figura 11 – Resultado numérico do método SPH com a simulação de Fluxo de Poiseuille desenvolvido de 0,5 segundos. Curva de velocidade, a simetria do eixo (y)(Vertical) = 0,00675 m. O eixo (x)(Horizontal) representa a velocidade em m/s.



Fonte: Autoria própria (2022).

Figura 12 – Resultado analítico do método SPH com a simulação de Fluxo de Poiseuille desenvolvido de 0,5 segundos, calculadora DualSPHysics. O eixo (z) Horizontal representa a largura entre as placas, (u) a velocidade em m/s.



t (sec) =	0,5
	Transient solution
z (m)	u(z) m/s
-0,0005	-1,05581E-20
-0,00045	4,72097E-06
-0,0004	8,94266E-06
-0,00035	1,26658E-05
-0,0003	1,58909E-05
-0,00025	1,86188E-05
-0,0002	2,08499E-05
-0,00015	2,25847E-05
-0,0001	2,38235E-05
-0,00005	2,45667E-05
0	2,48144E-05
0,00005	2,45667E-05
0,0001	2,38235E-05
0,00015	2,25847E-05
0,0002	2,08499E-05
0,00025	1,86188E-05
0,0003	1,58909E-05
0,00035	1,26658E-05
0,0004	8,94266E-06
0,00045	4,72097E-06
0,0005	-1,05581E-20

Fonte: (CRESPO *et al.*, 2014) .

Por fim para a simulação desenvolvida o resultado de erro relativo foi de 0,013%.

Todos os resultados também são condizentes com os apresentados por Liu e Liu (2003) 13, o que garante, pelo menos neste cenário uma grande capacidade de simulação por parte do programa e do método em si.

Figura 13 – Resultado analítico do método SPH com a simulação de Fluxo de Poiseuille desenvolvido de 0,01; 0,1; e 0,5 segundos.

Figure 4.19 Velocity quiver for the Poiseuille flow obtained using the SPH method at the 6000-th step. It is seen that some particles that are originally located in the downstream are wraparounded to the upstream after imposing the periodic boundary condition.

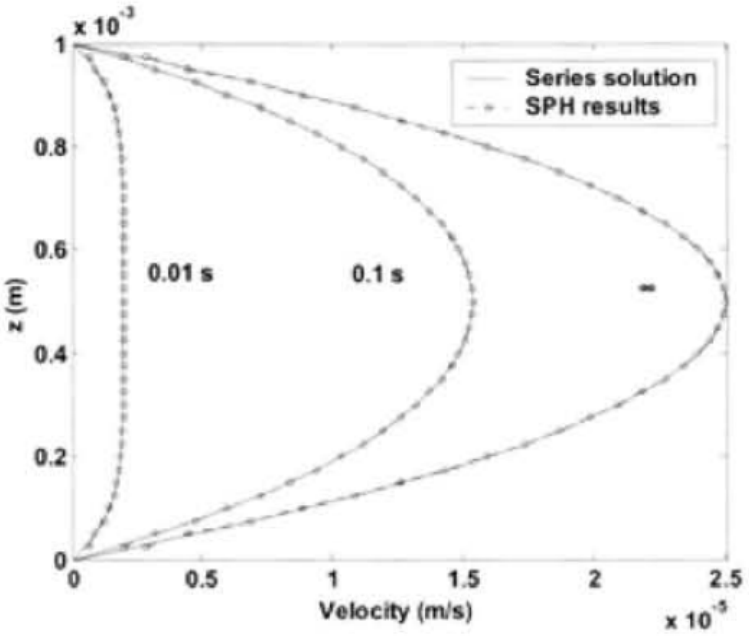


Figure 4.20 Velocity profiles for the Poiseuille flow.

Fonte: (LIU; LIU, 2003), p. 153.

7 CONCLUSÃO

O projeto teve interesse pessoal, acadêmico e profissional. Seu desenvolvimento apresentou grande dificuldade, porém seu resultado é um conjunto de aprendizados ao acadêmico, conhecimentos e uma ferramenta de simulação de fluidos que irá ficar disponível a comunidade, favorecendo a missão da Universidade Tecnológica Federal do Paraná.

O conteúdo organizado e estruturação matemática poderá servir de base a futuros trabalhos acadêmicos que podem evoluir o assunto ou aprendê-lo para benefício próprio em outros projetos. O simulador pode ser futuramente ampliado e utilizado em pesquisas relacionados a dinâmica de fluidos no curso de Engenharia Ambiental. Alguns pontos críticos relacionados podem ser o estudo de fronteira ou tratamento de fronteira, fator extremamente sensível na simulação ou paralelização de execução do programa para melhor aproveitamento dos processadores atuais.

O método proposto vem sendo evoluído a décadas, sendo limitado apenas pela necessidade computacional, porém com a grande capacidade disponível atualmente, o mesmo vem sendo amplamente pesquisado em busca de novas soluções para diferentes problemas e situações, assim como meios de otimizá-lo e torná-lo mais eficiente e eficaz, entre outros acadêmicos e a indústria. A busca para implementá-lo ajuda a universidade a estar na vanguarda neste campo do conhecimento.

REFERÊNCIAS

CAMPINAS, U. E. de. Apostila de treinamento:introdução ao fortran90. **Centro Nacional de Processamento de Alto Desempenho São Paulo**, Universidade Estadual de Campinas, SP, 2012. Disponível em: https://www.cenapad.unicamp.br/treinamentos/apostilas/apostila_fortran90.pdf. Acesso em: 05 de janeiro de 2022.

CRESPO, A. J. *et al.* Dualsphysics: Open-source parallel cfd solver based on smoothed particle hydrodynamics (sph). **Computer Physics Communications**, Elsevier, v. 187, p. 204–216, 2014.

CRISTO, H. P. Programação em linguagem fortran. **Arquivo Livre: Fortran. pdf**, 2003.

DAVIS, M. L.; MASTEN, S. J. **Princípios de engenharia ambiental**. [*S.l.*]: McGraw Hill Brasil, 2016.

EINSTEIN, A. What life means to einstein: An interview by george sylvester viereck. **The Saturday Evening Post**, v. 26, p. 117, 1929.

FRAGA FILHO, C. A. D. Development of a computer code using the lagrangian smoothed particle hydrodynamics (sph) method for solution of problems in fluid dynamics and heat transfer. **Revista Interdisciplinar de Pesquisa em Engenharia**, v. 2, n. 27, 2016.

GAELZER, R. Introdução ao fortran 90/95. **Instituto de Fisíca e Matemática**, Universidade Federal de Pelotas, RS, 2012. Disponível em: https://wp.ufpel.edu.br/diehl/files/2016/10/Apostila_links.pdf. Acesso em: 05 de janeiro de 2022.

GINGOLD, R. A.; MONAGHAN, J. J. Smoothed particle hydrodynamics: theory and application to non-spherical stars. **Monthly notices of the royal astronomical society**, Oxford University Press Oxford, UK, v. 181, n. 3, p. 375–389, 1977.

GRAEBEL, W. **Advanced fluid mechanics**. [*S.l.*]: Academic Press, 2007.

GUI-RONG, L. Gr.lab. University of Cincinnati - School of Aerospace System, 2022. Disponível em: http://www.ase.uc.edu/~liugr/SPH.html. Acesso em: 04 de Abril de 2022.

JIAN, W. *et al.* Sph study of the evolution of water–water interfaces in dam break flows. **Natural Hazards**, Springer, v. 78, n. 1, p. 531–553, 2015.

JOHNSON, G. R.; STRYK, R. A.; BEISSEL, S. R. Sph for high velocity impact computations. **Computer methods in applied mechanics and engineering**, Elsevier, v. 139, n. 1-4, p. 347–373, 1996.

LIBERSKY, L. D. *et al.* High strain lagrangian hydrodynamics: a three-dimensional sph code for dynamic material response. **Journal of computational physics**, Elsevier, v. 109, n. 1, p. 67–75, 1993.

LIU, G.-R.; LIU, M. B. **Smoothed particle hydrodynamics: a meshfree particle method**. [*S.l.*]: World scientific, 2003.

LIU, M.; LIU, G. Smoothed particle hydrodynamics (sph): an overview and recent developments. **Archives of computational methods in engineering**, Springer, v. 17, n. 1, p. 25–76, 2010.

LIU, M.; LIU, G.; LAM, K. Constructing smoothing functions in smoothed particle hydrodynamics with applications. **Journal of Computational and applied Mathematics**, Elsevier, v. 155, n. 2, p. 263–284, 2003.

LUCY, L. B. A numerical approach to the testing of the fission hypothesis. **The astronomical journal**, v. 82, p. 1013–1024, 1977.

MONAGHAN, J. On the problem of penetration in particle methods. **Journal of Computational physics**, Academic Press Professional, Inc. San Diego, CA, USA, v. 82, n. 1, p. 1–15, 1989.

MONAGHAN, J. J. Smoothed particle hydrodynamics. **Annual review of astronomy and astrophysics**, Annual Reviews 4139 El Camino Way, PO Box 10139, Palo Alto, CA 94303-0139, USA, v. 30, n. 1, p. 543–574, 1992.

MONAGHAN, J. J. Simulating free surface flows with sph. **Journal of computational physics**, Elsevier, v. 110, n. 2, p. 399–406, 1994.

MONAGHAN, J. J.; LATTANZIO, J. C. A refined particle method for astrophysical problems. **Astronomy and astrophysics**, v. 149, p. 135–143, 1985.

MORRIS, J. P. A study of the stability properties of smooth particle hydrodynamics. **Publications of the Astronomical Society of Australia**, v. 13, p. 97–102, 1994.

MORRIS, J. P. **Analysis of smoothed particle hydrodynamics with applications**. [*S.l.*]: Monash University Australia, 1996.

NASA. Navier-stokes equations. National Aeronautics and Space Administration - Glenn Research Center, 2020. Disponível em: https://www.grc.nasa.gov/WWW/k-12/airplane/nseqs.html#. Acesso em: 05 de Set de 2020.

OGER, G. *et al.* An improved sph method: Towards higher order convergence. **Journal of Computational Physics**, Elsevier, v. 225, n. 2, p. 1472–1492, 2007.

OSLO, O. M. de. Diretrizes para coleta e interpretação de dados sobre inovação. **Organização para a Cooperação e Desenvolvimento Econômico**, 2005.

PAIVA, A. **Uma abordagem lagrangeana para simulaçao de escoamentos de fluidos viscoplásticos e multifásicos**. 2007. Tese (Doutorado) — Pontifıcia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2007.

PEREIRA, C. G. A. **Estudo matemático do método SPH para modelo Wet-Dam-Break**. 2019. Dissertação (Mestrado) — Universidade Estadual Paulista (UNESP), São Paulo, 2019.

RABELLO, T. N.; VIEIRA, M. C. d. C. O teorema da divergência ou teorema de gauss. **Divisão de Ciências Fundamentais**, Instituto Tecnológico da Aeronáutica, 2020. Disponível em: http://www.mat.ita.br/mat36/capi/integrais_de_superficie/6asem/pdf/pdf.pdf. Acesso em: 05 de Set de 2020.

RANDLES, P.; LIBERSKY, L. D. Smoothed particle hydrodynamics: some recent improvements and applications. **Computer methods in applied mechanics and engineering**, Elsevier, v. 139, n. 1-4, p. 375–408, 1996.

SHADLOO, M.; OGER, G.; TOUZÉ, D. L. Smoothed particle hydrodynamics method for fluid flows, towards industrial applications: Motivations, current state, and challenges. **Computers & Fluids**, Elsevier, v. 136, p. 11–34, 2016.

SIGALOTTI, L. D. G. *et al.* Sph simulations of time-dependent poiseuille flow at low reynolds numbers. **Journal of computational physics**, Elsevier, v. 191, n. 2, p. 622–638, 2003.

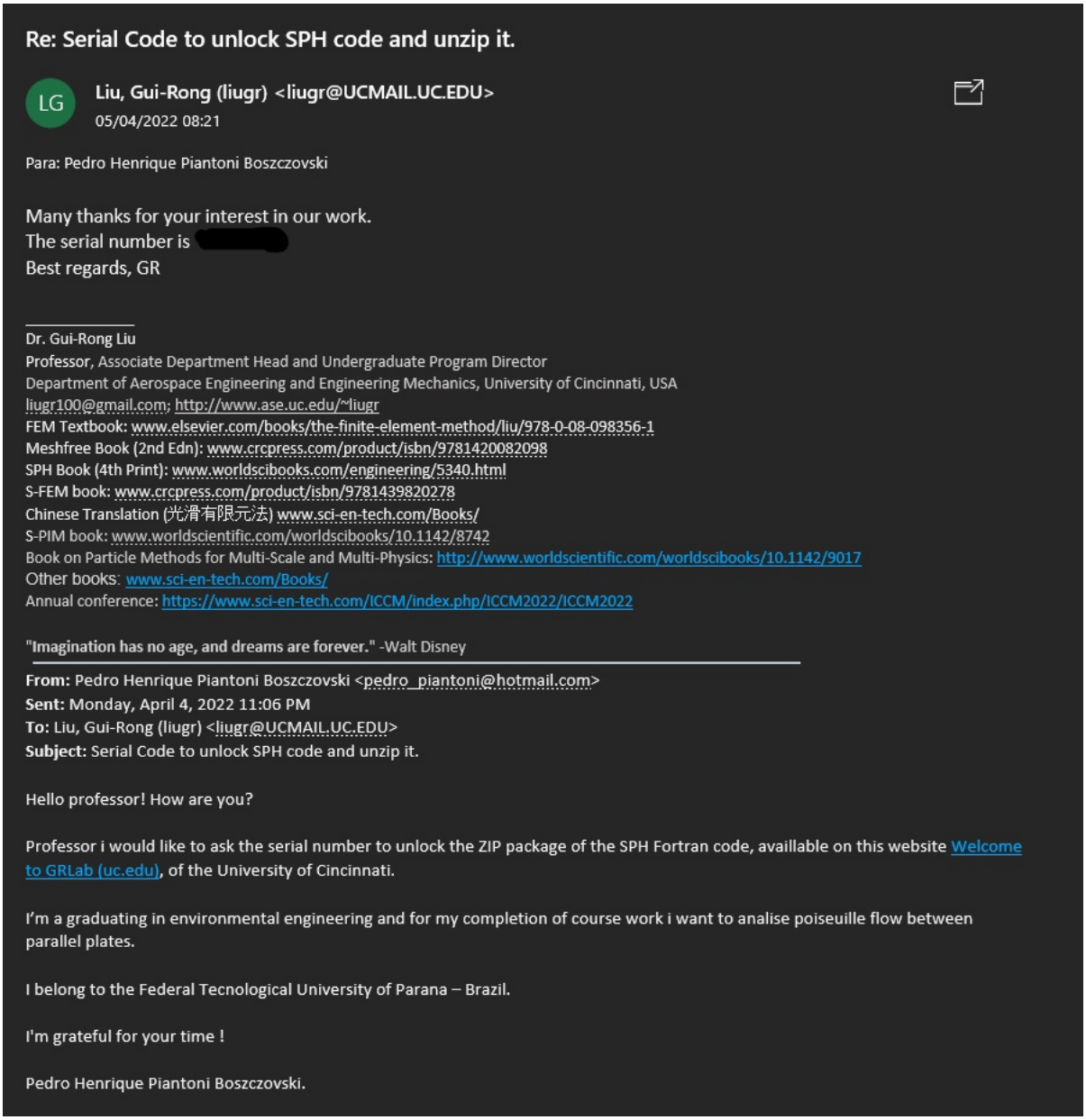
SOUZA, T.; BÍSCARO, H. Simulação computacional do sangue usando o método smoothed particle hydrodynamics (sph). In: SBC. **Anais Estendidos do XIV Simpósio Brasileiro de Sistemas de Informação**. [*S.l.*], 2018. p. 118–121.

WANG, Z.-B. *et al.* An overview of smoothed particle hydrodynamics for simulating multiphase flow. **Applied Mathematical Modelling**, Elsevier, v. 40, n. 23-24, p. 9625–9655, 2016.

ZIENKIEWICZ, O. C.; TAYLOR, R. L.; ZHU, J. Z. **The finite element method: its basis and fundamentals**. [*S.l.*]: Elsevier, 2005.

ANEXO A – Autorização de uso do código em Fortran77. Professor Dr. Liu Gui-Rong .

Figura 14 – Carta de autorização de uso do código do professor Dr. Liu Gui-Rong. O número serial está ocultado por se tratar de uma senha.



Re: Serial Code to unlock SPH code and unzip it.

LG Liu, Gui-Rong (liugr) <liugr@UCMAIL.UC.EDU>
05/04/2022 08:21

Para: Pedro Henrique Piantoni Boszczovski

Many thanks for your interest in our work.
The serial number is [redacted]
Best regards, GR

Dr. Gui-Rong Liu
Professor, Associate Department Head and Undergraduate Program Director
Department of Aerospace Engineering and Engineering Mechanics, University of Cincinnati, USA
liugr100@gmail.com; http://www.ase.uc.edu/~liugr
FEM Textbook: www.elsevier.com/books/the-finite-element-method/liu/978-0-08-098356-1
Meshfree Book (2nd Edn): www.crcpress.com/product/isbn/9781420082098
SPH Book (4th Print): www.worldscibooks.com/engineering/5340.html
S-FEM book: www.crcpress.com/product/isbn/9781439820278
Chinese Translation (光滑有限元法) www.sci-en-tech.com/Books/
S-PIM book: www.worldscientific.com/worldscibooks/10.1142/8742
Book on Particle Methods for Multi-Scale and Multi-Physics: http://www.worldscientific.com/worldscibooks/10.1142/9017
Other books: www.sci-en-tech.com/Books/
Annual conference: https://www.sci-en-tech.com/ICCM/index.php/ICCM2022/ICCM2022

"Imagination has no age, and dreams are forever." -Walt Disney

From: Pedro Henrique Piantoni Boszczovski <pedro_piantoni@hotmail.com>
Sent: Monday, April 4, 2022 11:06 PM
To: Liu, Gui-Rong (liugr) <liugr@UCMAIL.UC.EDU>
Subject: Serial Code to unlock SPH code and unzip it.

Hello professor! How are you?

Professor i would like to ask the serial number to unlock the ZIP package of the SPH Fortran code, availlable on this website Welcome to GRLab (uc.edu), of the University of Cincinnati.

I'm a graduating in environmental engineering and for my completion of course work i want to analise poiseuille flow between parallel plates.

I belong to the Federal Tecnological University of Parana – Brazil.

I'm grateful for your time !

Pedro Henrique Piantoni Boszczovski.

Fonte: Autoria própria (2022).