UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Arthur Prince de Almeida

Características de uma implementação em paralelo do método SPH com o uso da tecnologia de GLSL(OpenGL Shading Language)

São Paulo

2022

Arthur Prince de Almeida

Características de uma implementação em paralelo do método SPH com o uso da tecnologia de GLSL(OpenGL Shading Language)

Monografia apresentada à Escola de Artes, Ciências e Humanidades da Universidade de São Paulo, como parte dos requisitos exigidos na disciplina ACH 2018 – Projeto Supervisionado ou de Graduação II, para obtenção do título de Bacharelado em Sistemas de Informação.

Modalidade: TCC curto (1 semestre) – individual.

Orientador: Helton Hideraldo Bíscaro

São Paulo

2022

Resumo

As técnicas de implementação de fluidos são abordagens utilizadas para simular o comportamento realista de fluidos em ambientes virtuais. Essas técnicas têm aplicações em uma variedade de campos, incluindo filmes, jogos, simulações de engenharia e efeitos visuais. Uma das técnicas comuns é a Simulação de Partículas Suavizadas (SPH), que modela o fluido como uma coleção de partículas que interagem com suas vizinhas. Neste artigo será abordado uma nova forma de implementação dessa técnica usando uma malha, o qual tem como grande vantagem a possibilidade de ser completamente paralelizada. Por essa razão, será abordado a implementação com a linguagem de shader OpenGL para programar o algoritmo em GPU(Unidade de Processamento Gráfico). Em resumo, primeiro o artigo introduzirá sobre algoritmos de geração de fluidos. Depois tratará de aspectos teóricos do método SPH como as equações de Navier-Stokes e as equações espaciais. Em seguida tratará da implementação do SPH com malha usando um algoritmo de integração de partículas. Por fim, abordaremos sobre um experimento para medir a eficiência dessa técnica e serão discutidos os resultados obtidos.

Palavras chaves: SPH, OpenGL, Navier-Stokes, GPU, malha de partículas

Lista de figuras

Lista de tabelas

Lista de Algoritmos

Sumário

1 - Introdução

Os algoritmos de geração de fluidos são técnicas computacionais que visam simular e reproduzir o comportamento realista de fluidos, como água, fogo, fumaça e outros elementos. Atualmente existem dois métodos que se destacam: a Euleriana e a Lagrangiana.

O método de Euleriano utiliza uma geometria de malha, ou fixa ou adaptativa(conhecido como AMR or Adaptive Mesh Refinement), com os parâmetros do fluido avaliado nas próprias células da malha. Sendo assim, locais focais com maior resolução serão mais precisos do que áreas de menor interesse, portanto, possibilitando a otimização de recursos na renderização do fluido.

Por outro lado, o método Lagrangiano não utiliza pontos fixos no espaço, na verdade as propriedades físicas e equações que envolvem o fluido se movimentam junto com uma partícula. Esse método tem um foco na segunda derivada, o que significa que a partir das propriedades hidrodinâmicas da partícula(como posição, velocidade, massa, etc…) e da influência que as outras partículas têm sobre aquela(as forças de viscosidade e pressão) que se calcula seu comportamento. Por essa razão se é dito que cada partícula é “suavizada” sobre um volume finito em um contexto onde a massa é fixa, ou seja, manuseando a densidade da partícula. Esse método baseado em partículas é conhecido como SPH e foi criado na década de 1970 por Gingold e Monaghan (1977) e por Lucy (1977) num contexto de simulação de fluidos incompressíveis em astrofísica (formação de galáxias e supernovas).

Cada uma dessas técnicas tem seus prós e contras. De forma geral, AMR tem uma maior resolução para um número igual de células na malha de partículas devido ao cálculo de vizinhança, mesmo sendo feito de qualquer parâmetro de fluído. Entretanto o SPH se adapta principalmente pela densidade o que permite a formação de vácuos, diferentemente do AMR que previne que o fluido desapareça de arestas. Por fim, com o aumento da entropia do sistema, o método AMR tende a ter problemas frequentes com renderização de fenômenos não físicos. E já que o SPH é um método Lagrangeano, pode-se implementá-lo com conservação de massa, momento, energia e similaridade, a menos que isso não seja desejado, como a própria conservação da entropia.

O objetivo deste trabalho é estudar e desenvolver um algoritmo semi-lagrangiano inspirado no SPH(Smoothed Particle Hydrodynamics) usando malhas para o cálculo da vizinhança, de forma que sua execução ocorra quase que totalmente em GPU. Para isso, foi usado a tecnologia GLSL com a API do OpenGL desenvolvida pelo grupo Khronos, e a linguagem de programação C++ para intermediar a comunicação entre a CPU e a GPU.

2 - Aspectos teóricos

O movimento e escoamento dos fluidos seguem as leis da física, assim como qualquer outro corpo. No entanto, devido à natureza complexa do cálculo da aceleração e velocidade de fluidos em escoamento, criar modelos que os representem de maneira apurada não é uma tarefa trivial. Para abordar essa questão, utiliza-se as equações de Navier-Stokes, que, apesar de muito úteis, apresentam um alto grau de dificuldade para serem solucionadas, exigindo a aplicação de métodos numéricos (SOUZA; BÍSCARO, 2018).
Seguem as equações:

$$\frac{dp}{dt} = -\ p\,\nabla.\,v$$

$$\frac{dv}{dt} = -\ \frac{1}{\rho}\nabla p\ +\ \frac{\mu}{\rho}\nabla^2 v\ +\ g$$

Onde t representa o tempo, ρ a densidade, p a pressão, g a aceleração da gravidade, μ a viscosidade do fluido, v a velocidade e ∇ o operador gradiente. Essas equações são utilizadas para definir o estado em um tempo futuro, t+1, a partir de seu estado atual, em tempo t.

O método SPH, ou Smoothed Particle Hydrodynamics, é uma abordagem que se destaca entre os métodos numéricos utilizados para resolver as equações de Navier-Stokes em simulações de fluidos. Ao discretizar o domínio de simulação em um conjunto de partículas, o SPH consegue contornar as dificuldades associadas ao uso de malhas convencionais. Em vez disso, a interpolação suavizada entre as partículas permite calcular as grandezas físicas em qualquer ponto do domínio, facilitando a simulação de fenômenos complexos.

Ele consiste na interpolação das propriedades físicas de uma partícula em uma posição arbitrária, bem como aproximar as derivadas espaciais a partir de um número finito de partículas adjacentes.

O método é baseado no conceito de representação integral de uma função contínua $f : \Omega \subset \mathbb{R}^d \rightarrow \mathbb{R}$, onde Ω é um conjunto aberto contido em $\mathbb{R}^d$. Essa representação é feita a partir da convolução de f com a distribuição delta de Dirac, na qual, para todo $x \in \Omega$, a função f pode ser escrita na forma:

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

onde:

$$\delta(x - x') = 0, \text{ se } x \neq x' \text{ e } \delta(x - x') = \infty \text{ se } x = x'$$

Uma propriedade importante de $\delta(x)$ é que qualquer função $(f * \delta)(x)$ definida como $\int_{-\infty}^{\infty} \delta(x - y)f(y)dy$ segue a propriedade:

$$f(x) = (f * \delta)(x) = \int_{-\infty}^{\infty} \delta(x - y)f(y)dy$$

Na prática, a distribuição de Dirac ($\delta(x - x')$) é definida como limite de funções suaves W(x, h), que são conhecidas como funções núcleo suave ou simplesmente núcleo, onde o fator h define o raio de influência de W e é conhecido como comprimento suave. Tipicamente, esses núcleos devem satisfazer as seguintes propriedades:

1. Não negativo: $W(x, h) \geq 0$
2. Suavidade: $W(x, h) \in C^k(\Omega)$ com k >1
3. suporte compacto $W(x - x', h) = 0$ para todo$|x - x'| \geq kh$
4. Partição da unidade: $\int_{\Omega} W(x, h)dx = 1$
5. Convergência: $W(x, h) \to \delta$ quando h→0

De modo geral, a interpolação utilizada no método SPH aproxima uma determinada quantidade A_i em uma posição arbitrária x_i a partir de um número finito de quantidades conhecidas A_j em posições x_j localizadas na vizinhança de x_i.

$$A_i = \sum_{j=1}^{k} \frac{m_j}{\rho_j} A_j W_{ij}$$

Onde A_i é a quantidade de uma determinada propriedade na partícula i, ρ_j e m_j são, respectivamente, a densidade e a massa locais. Já W_{ij} é a função núcleo avaliada em$\|x_i - x_j\|$. É importante ressaltar que cada partícula i carrega seus atributos físicos, tais como a densidade ρ_i, massa m_i, pressão p_i e volume ΔV_i. Além disso, ao longo do tempo t, a posição das partículas x_i e outros atributos são "transportados" segundo a velocidade v_i da partícula.
As equações espaciais são:

$$\nabla A_i = \rho_i \sum_{j=1}^{k} m_i \left(\frac{A_i}{\rho_i^2} + \frac{A_j}{\rho_j^2}\right) \nabla W_{ij}$$

$$\nabla . A_i = -\frac{1}{\rho_i} \sum_{j=1}^{k} m_j A_{ij} . \nabla W_{ij}$$

$$\nabla^2 A_i = 2 \sum_{j=1}^{k} \frac{m_j}{\rho_j} A_{ij} \frac{x_{ij} \nabla W_{ij}}{x_{ij} x_{ji} + 0{,}001h^2}$$

onde $A_{ij} = A_i - A_j$, $x_{ij} = x_i - x_j$

3 - Metodologia

O objetivo deste projeto foi implementar o método SPH sobre uma malha de pixels a fim de a usá-la para o cálculo da vizinhança, que é a parte mais custosa do algoritmo SPH tradicional. Para isso foi usado a tecnologia GLSL, ou OpenGL Shading Language, que é uma linguagem de programação de alto nível desenvolvida especificamente para trabalhar com a API OpenGL. Sendo uma parte essencial do pipeline gráfico moderno, o GLSL permite aos desenvolvedores criar shaders personalizados, que são programas executados diretamente na GPU. Esses shaders têm o papel fundamental de controlar a aparência visual e o comportamento dos objetos renderizados em uma cena 3D ou 2D.

O código é dividido em duas partes, uma para reintegrar as partículas e a outra para o cálculo do SPH que será chamado de simulação. Sendo que ambas são executadas em shaders diferentes e a entrada de um é a saída do outro.

A reintegração tem a função de integrar todas as partículas com a malha de pixels para que o fluido tenha sua massa conservada. Assim, soma-se as massas das partículas vizinhas sobrepostas e ponderando suas velocidades e posições de acordo com a massa.

Já a parte de Simulação é onde realmente está o que caracteriza esse algoritmo como uma variante do método SPH. Ou seja, todas as equações de densidade, forças de viscosidade, gravidade entre outras definições do SPH estão contidas nesta parte. A grande vantagem é que a vizinhança já está calculada na grade de pixels. Em resumo é como se a reintegração mantivesse a estrutura coerente e a simulação é o próprio SPH com a vizinhança e algumas propriedades da partícula já calculadas. Esse método permite tratar cada partícula individualmente e de forma paralela o que faz esse algoritmo ser muito rápido.

O algoritmo desenvolvido neste trabalho, embora modele um fluido com uma fronteira(que é a própria tela de pixel), não usa partículas diretamente, como no SPH, mas é na verdade um algoritmo baseado em grade semi-Lagrangiana.

3.1 - Simulação

São diversos os aspectos práticos e computacionais de uma implementação de escoamento de fluídos baseada em SPH como, por exemplo, a definição correta dos parâmetros, a estimativa da pressão, as condições de estabilidade que influenciam no passo de tempo, entre outros. A seguir serão discutidos os aspectos necessários para uma simulação aceitável.

3.1.1 - Algoritmo básico

Uma vez que a massa é fixa em todas as partículas, usaremos a definição apresentada no trabalho de Schechter e Bridson (2012), a qual estabelece $m_i = \frac{2}{3} h^3 \rho_0$. Onde h é o espaçamento entre as partículas e ρ_0 é a densidade de referência do fluido. A pressão p_i é calculada a partir da densidade ρi de cada partícula. Uma escolha comum para o cálculo da pressão é a equação de Tait (MONAGHAN, 1994):

$$p_i = k\left(\left(\frac{\rho_i}{\rho_0}\right)^{\lambda} - 1\right)$$

Na qual ρ_0 é a densidade do fluido em repouso, k e λ são parâmetros relacionados às flutuações de densidade do fluido.

O Algoritmo 1 representa os passos básicos de uma iteração do método SPH em CPU. Detalharemos cada um dos laços a seguir.

Algoritmo 1 - SPH

1. **Para cada** *partícula i* **faça:**
   // Atualização da vizinhança
2. *Encontre os k vizinhos mais próximos da partícula i ;*
3. **Para cada** *partícula i* **faça:**
   // Atualização da densidade e pressão
4. $\rho_i = \Sigma_j^k (m_i W_{ij})$;
5. Calcule p_i usando ρ_i
6. **Para cada** *partícula i* **faça:**
   // Atualização das forças atuantes
7. $\mathbf{F}_i^{\text{pressão}} = -m_i/\rho_i \nabla p_i$
8. $\mathbf{F}_i^{\text{viscosas}} = -m_i \eta \nabla^2 \boldsymbol{v}_i$
9. $\mathbf{F}_i^{\text{externas}} = -m_i g$
10. $\mathbf{F}_i(t) = \mathbf{F}_i^{\text{pressão}} + \mathbf{F}_i^{\text{viscosas}} + \mathbf{F}_i^{\text{externas}}$
11. **Para cada** *partícula i* **faça:**
    // Integração temporal
12. Atualize a velocidade $v_i(t + \Delta t)$
13. Atualize a posição $x_i(t + \Delta t)$

O primeiro laço do Algoritmo 1 executa uma das tarefas mais importantes em uma simulação de SPH: a atualização das relações de vizinhança das partículas. A literatura descreve principalmente duas abordagens para resolver esse problema: estruturas hierárquicas, como kd-Trees (ADAMS et al., 2007; SIN; BARGTEIL; HODGINS, 2009), e grades regulares (TESCHNER et al., 2003). Uma grande vantagem das estruturas hierárquicas é a sua adaptabilidade, evitando desperdício de memória, pois as células são construídas apenas quando necessário. No entanto, o tempo de construção dessas estruturas é da ordem de O(n log(n)), enquanto o acesso é feito em O(log(n)). Por outro lado, as estruturas baseadas em grades utilizam uma função hash e são construídas em tempo O(n), permitindo acesso em tempo constante. O trabalho de Paiva et al. (2009) recomenda o uso de pelo menos 20 vizinhos por partícula em simulações 2D e 56 em simulações 3D.

A vantagem da estrutura com grade é que só precisa ser construída uma vez e após isso pode-se acessá-la de forma paralela, diferentemente da kd-Tree em que deve ser atualizada em cada ciclo. Assim, com a tecnologia GLSL, foi implementado aquela estrutura para que o acesso à vizinhança seja feito em O(1) de forma paralelizada em GPU.

O segundo laço, que começa na linha 4 do Algoritmo 1, realiza a atualização da densidade e da pressão em cada partícula com base na densidade. Isso é conhecido na literatura como equação de estado (equação 9). Existem vários esquemas alternativos na literatura para avaliar com maior precisão as forças de pressão atuantes na partícula (SOLENTHALER; PAJAROLA, 2009; HE et al., 2012; CHENTANEZ; MÜLLER, 2011). Embora não seja o foco do nosso estudo, vale mencionar que a ideia geral por trás desses esquemas é dividir o cálculo das forças de pressão em duas etapas: na primeira, uma velocidade intermediária é calculada a partir das forças independentes da pressão; na segunda etapa, a componente dependente da pressão é computada levando em consideração a velocidade intermediária.

O terceiro laço, iniciado na linha 8 do mesmo algoritmo, atualiza todas as componentes das forças atuantes em cada partícula. A força resultante no instante t, $F_i(t)$, será a aceleração incidente em cada partícula e será utilizada para a última etapa de integração no tempo.

No Algoritmo implementado neste trabalho a vizinhança é definida na própria malha de pixels. O que faz com que a simulação fique responsável pela atualização da densidade de forças atuantes, a qual impacta apenas na velocidade e direção da partícula. A reintegração por sua vez faz a integração temporal de forma que mantenha a estrutura coerente como será mostrado a seguir

3.2 - Reintegração

Neste algoritmo o estado de cada célula é definido pelos atributos da partícula na célula e em qual direção ela está se movendo. No nosso caso

armazenamos a posição, velocidade e a massa da partícula o que são 5 floats. Para combater o problema de perda de massa das partículas e deixar o fluido mais suavizado foi usado uma função de distribuição que divide a massa da partícula sobre uma área. Para encontrar a quantidade de massa e seu centro que cada partícula depositada na célula em questão, é preciso integrar a distribuição dentro dos limites da célula atual (E é aí que vem o nome - reintegrando as distribuições das partículas vizinhas). Ou seja, para calcular a massa de cada célula nos próximos frames basta somar as intersecções das distribuições de massas das partículas vizinhas com a célula.

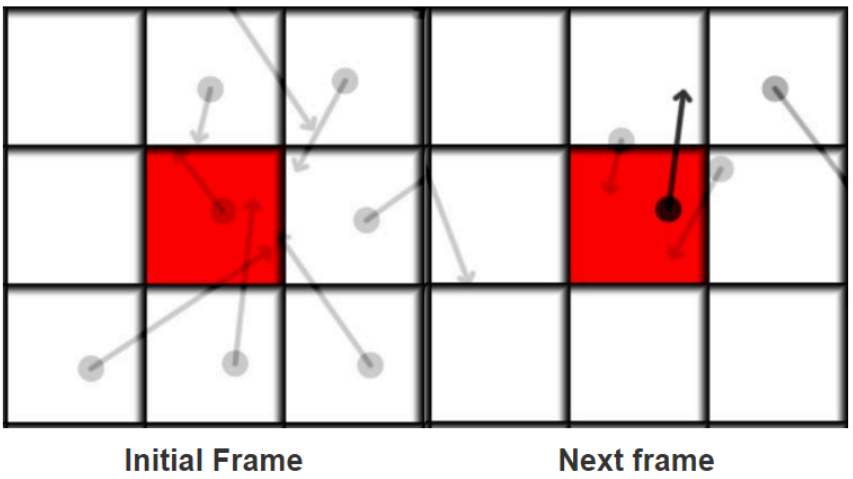

Figura 1 - integração de partículas
Fonte: Mykhailo Moroz(2020)

A ideia principal da reintegração é percorrer todos os vizinhos da célula atual (incluindo ela mesma), integrar a posição de cada partícula vizinha e adicioná-la se ela acabar nesta célula. Na figura 1 está uma visualização de como isso se parece. Cada célula possui uma partícula com uma massa representada pela opacidade. A célula vermelha é a célula atual para a qual queremos encontrar seu estado futuro, e a posição futura das partículas é mostrada pela seta. Contudo, há mais de uma partícula se movendo para a célula vermelha. No final, todas as partículas que acabam na mesma célula são somadas, e suas posições e velocidades são calculadas pela média ponderada pela massa da partícula. Em forma matemática, isso pode ser escrito da seguinte maneira:

$$M_i^{t+1} = \sum_j^{vizinhos} K_i(X_j^{\rightarrow t} + \Delta t\, V_j^{\rightarrow t})\, m_j^t$$

$$X_i^{\rightarrow t+1} = \frac{1}{M_i^{t+1}} \sum_j^{vizinhos} K_i(X_j^{\rightarrow t} + \Delta t\, V_j^{\rightarrow t})(X_j^{\rightarrow t} + \Delta t\, V_j^{\rightarrow t})\, m_j^t$$

$$V_i^{\rightarrow t} = \frac{1}{M_i^{t+1}} \sum_j^{vizinhos} K_i(X_j^{\rightarrow t} + \Delta t\, V_j^{\rightarrow t}) V_i^{\rightarrow t}\, m_j^t$$

Onde M_i^{t+1} é a massa da partícula na célula i no passo de tempo t, $\vec{X_j}^t$ é a posição da partícula e $\vec{V_i}^t$ é a velocidade. A função $K_i(\vec{X})$ é igual a 1 se o ponto $\vec{X}$ está dentro da célula i e zero caso contrário. Δt é o intervalo de tempo.

Para uma célula quadrada, a função K_i é simplesmente:

$$K_i(\vec{X}) = H\left(\vec{X} - \vec{C}_i + 0,5\right)H\left(\vec{C}_i - \vec{X} + 0,5\right)$$

Onde H é a função de passo de Heaviside multivariada e $\vec{C}_i$ é o centro da célula i.

$$H(x) := \{1\ se\ x \geq 0;\ 0\ se\ x < 0\}$$

Apesar de ser um algoritmo bastante simples, existe uma limitação. Para garantir que contamos todas as partículas vizinhas possíveis que possam incidir nesta célula, teríamos que percorrer toda a grade, o que é muito caro computacionalmente, ou limitar a velocidade máxima das partículas para tornar o raio de busca finito.

Obviamente, a segunda opção é muito melhor em nosso contexto e caso seja desejado, é possível limitar a velocidade de forma que as partículas atravessem a grade em apenas 1 célula por frame. Dessa forma é mais econômico fazer muitos passos menores com um vizinhança de uma célula(de forma completamente paralelizada em GPU) em vez de contar todas as células em um único passo, já que $9\sqrt{N} < N$, onde N é o número de células, e a raiz quadrada é porque precisamos aplicar a operação apenas um número linear de vezes, em vez de aplicá-la para cada célula. E o 9 é o número de vizinhos. Em 3D, seria equivalente a $27N^{1/3} < N$, o que é ainda mais eficiente.

Há diversas distribuições que é possível de usar, como por exemplo distribuição normal. O problema de como calculá-la, já que não há uma solução analítica para essa integral, e seria preciso integrar numericamente a distribuição. Também pode-se tentar uma distribuição circular uniforme ou uma distribuição circular não uniforme a distribuição é igual a $1 - |\vec{X_0} - \vec{X}|^2$ para $|\vec{X_0} - \vec{X}| \leq 1$ e igual a 0 para $|\vec{X_0} - \vec{X}| > 1$, onde $\vec{X_0}$ é a posição da partícula. A distribuição mais simples que fornece uma solução analítica que é na verdade um quadrado uniforme alinhada aos eixos, para a qual a massa e o centro de massa são facilmente computadas de forma analítica com um raio de difusão relativo ao tamanho da célula como mostrado na figura 2.

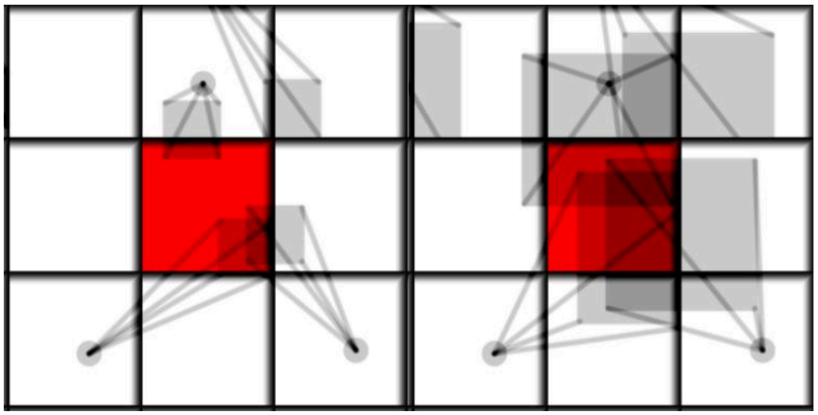

Figura 2 - raio de difusão
Fonte: Mykhailo Moroz(2020)

Nas figura 3 mostra uma exemplificação da distribuição de massa das partículas incidindo sobre a célula vermelha. Assim, a massa resultante da célula vermelha no próximo frame é a soma das massas das relativas áreas de cada partícula vizinha, como mostrado na figura 3.

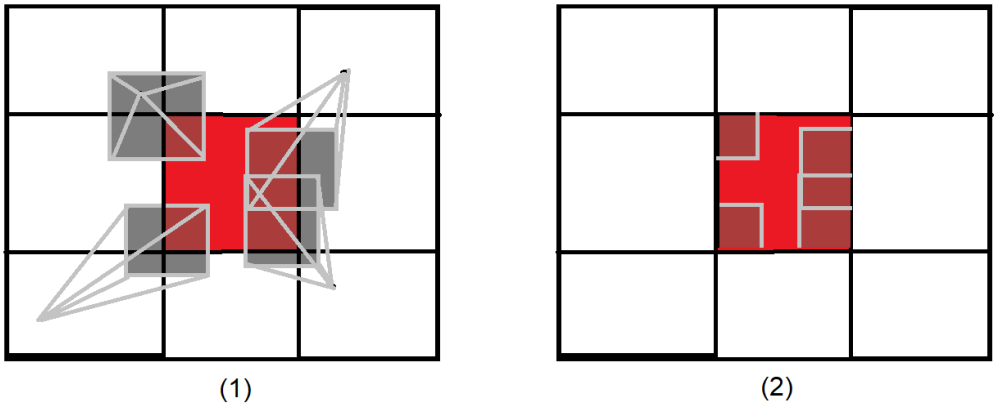

Figura 3 - distribuição de massa

A área relativa será a massa relativa e seu centro será o centro de massa. Isso pode ser implementado da seguinte forma:

Algoritmo 2 - Reintegração

1. massa = 0.
2. **Para cada** *partícula k vizinha da célula* i_i **faça:**
   // Integração da posição da partícula
3. $P_k.X = P_k.V\ \Delta t$
4. CellBox = ($\vec{c}$ - 0.5, $\vec{c}$ + 0.5); //caixa da célula
   // a caixa da partícula
5. P_kBox =(x - diffusion_radius, x + diffusion_radius)
   // caixa de intersecção
6. P_{ik}Box =(max(CellBox, PkBox), min(CellBox, PkBox))
7. centro = 0.5*(P_{ik}Box.x + P_{ik}Box.y) //centro de massa
8. largura = P_{ik}Box.y - P_{ik}Box.x //somente positivo
   //área relativa
9. m = size.x*size.y/$(diffusion\_radius)^2$;
   //adiciona atributos com o peso de cada partícula
10. massa += m
11. P_i.X += centro * m
12. P_i.V += P_k.V * m
13. //normalização
14. P_i.X /= massa
15. P_i.V /= massa

Vale notar que após essas operações é necessário limitar tanto a posição da partícula para que ela não ultrapasse $\vec{C} \pm 0.5$ e a velocidade para que não seja maior que uma célula. Com essas restrições apenas a transferência de massa ocorre entre as células como ilustrado na figura 6:

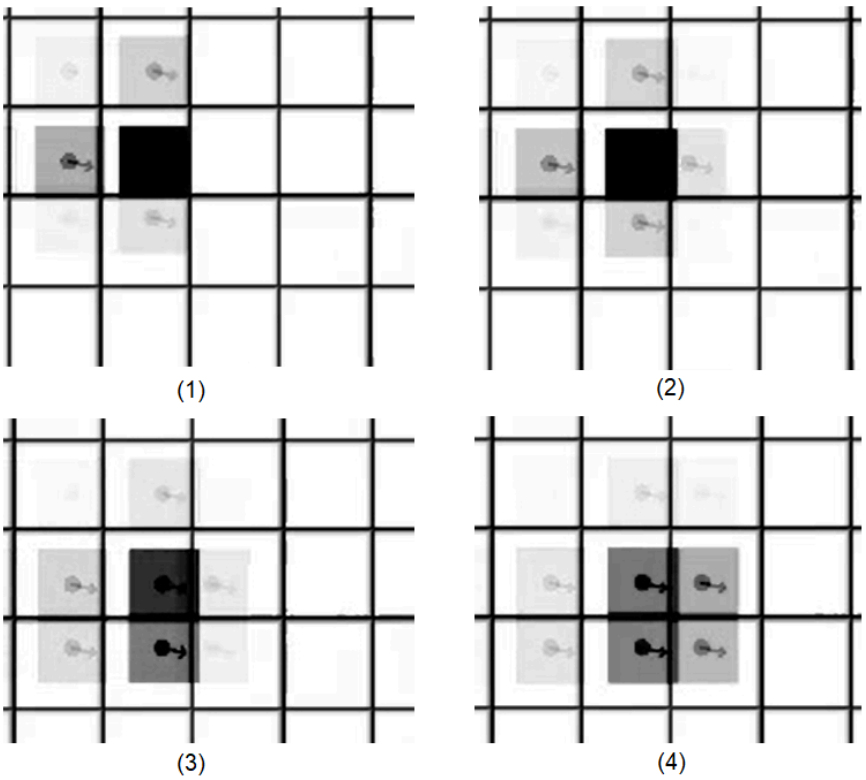

Figura 4 - demonstração dos funcionamento da reintegração

Em (1) se vê a caixa da partícula tocando na extremidade de sua célula. Em seguida começa a transferência de massa representada pelo grau de transparência da partícula. O limite da posição da partícula para que ela não saia da célula faz com que a transferência não seja imediata. Dessa forma se vê que começam aparecer outras partículas com massa na estrutura.

4 - Resultados

Para realizar o experimento, utilizamos um computador com as seguintes especificações: CPU Intel Core i5 8250U e GPU Intel UHD Graphics 620, executando o sistema operacional Linux Ubuntu 22.04. A linguagem de programação C++ e o compilador g++, com apenas uma thread, foram empregados para a integração com a biblioteca OpenG, a qual foi usada para implementar o método SPH.

O experimento realizado envolve duas fontes que inicializam as partículas com uma determinada velocidade e massa, uma em verde e a outra azul. Essas fontes farão com que seja possível visualizar a fluidez do líquido e como as partículas interagem como mostrado na figura 5.

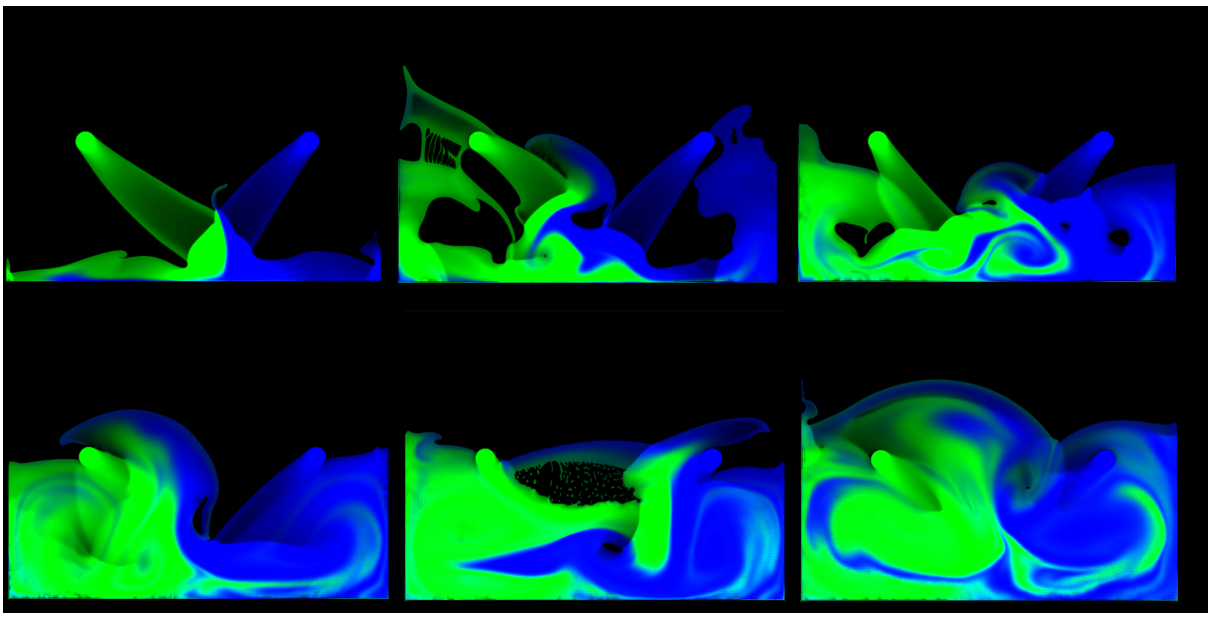

Figura 5 - Exemplo do funcionamento do algoritmo

A tabela 1 por sua vez mostra os resultados da execução do experimento citado até o momento em que todos os pixels apresentam massa diferente de zero. Nela é medido a variação de quadros por segundo, e o tempo médio de execução da reintegração e da simulação em diferentes resoluções.

Resolução	Pixels	Quadros por segundo médio	Tempo de reintegração médio	Tempo de simulação médio
200x200	40.000	290-310	1.13968 ms	1.15013 ms
500x500	250.000	80-90	5.26449 ms	4.20479 ms
800x600	480.000	45-50	9.44331 ms	7.21845 ms
1.000x1.000	1.000.000	28-33	17.9473 ms	13.6682 ms
1500x1000	1.500.000	24-27	21.3748 ms	15.0436 ms

Tabela 1 - resultado do experimento

Os resultados evidenciam que à medida que a resolução aumenta, a implementação desse método se torna mais dispendiosa, uma consequência previsível, pois a GPU deve calcular um maior número de partículas. No entanto, o uso da malha de pixels para o cálculo de vizinhança permite que o método seja completamente paralelizado, conferindo uma vantagem significativa em relação aos métodos tradicionais que realizam tal cálculo na CPU.

A diferença entre o tempo de reintegração e de simulação se dá pois o algoritmo não realiza a simulação em partículas que tenham massa igual a zero. Diferentemente da reintegração, que precisa verificar o destino de todas as partículas. Em cenários em que todos os pixel começam com alguma massa nota-se um aumento significativo de até 6ms, ao mesmo tempo em que sempre que se inicia o experimento há um pico de quadros por segundos. Isso mostra que

apesar da reintegração ter um tempo constante a simulação interfere no tempo de acordo com a quantidade de partículas com massa.
Vale ressaltar que a simplicidade do modelo SPH implementado pode ter contribuído para a fluidez da simulação, pois foram considerados apenas cálculos básicos, como força de viscosidade, pressão e gravidade. Essa abordagem pode ter influenciado positivamente no desempenho do método.

Ademais, a capacidade de renderizar um grande número de "partículas" em um tempo aceitável mostra a eficiência do método em comparação com outras abordagens tradicionais de SPH devido principalmente ao cálculo de vizinhança ocorrer na GPU.

Figura 6 - Animando o fluido em uma imagem

5 - Conclusão

O método SPH é altamente flexível e eficiente para simular escoamentos complexos de fluidos. Ao representar o fluido como um conjunto de partículas, ele consegue lidar de forma mais precisa com colisões, ondas e interações com sólidos. O uso de malha fez com que houvesse uma restrição no espaço de atuação das partículas, no entanto ganhou significativo desempenho com o cálculo de vizinhança.

A tecnologia GLSL possibilitou o desenvolvimento de shaders que são programas executados em GPU. Dessa forma o algoritmo implementado neste artigo foi completamente paralelizado, o que possibilitou a renderização de milhões de partículas em tempo real.

A grande vantagem deste algoritmo é a velocidade de execução devido a paralelização. No entanto, há ainda a desvantagem de que o espaço de atuação do fluido é limitado pela própria grade, e a velocidade da partícula também não pode ser maior do que o raio de vizinhança, caso contrário haverá perda de massa.

Além disso, a possibilidade de colocar mais propriedades na partículas faz com que seja possível criar simulações mais complexas e precisas de fluídos em tempo real. É incrível imaginar como seriam os limites desse algoritmo em placas de vídeo modernas.

6 - Referências bibliográficas

ADAMS, B. et al. Adaptively sampled particle fluids. ACM Trans. Graph., ACM, New York, NY, USA, v. 26, n. 3, jul. 2007. ISSN 0730-0301. Disponível em: ⟨http://doi.acm.org/10.1145/1276377.1276437⟩.

CHENTANEZ, N.; MÜLLER, M. Real-time eulerian water simulation using a restricted tall cell grid. ACM Trans. Graph., ACM, New York, NY, USA, v. 30, n. 4, p. 82:1–82:10, jul. 2011. ISSN 0730-0301. Disponível em: ⟨http://doi.acm.org/10.1145/2010324.1964977⟩.

COSSINS, P. Smoothed Particle Hydrodynamics. arXiv, Sat, 2 Oct 2010 19:10:31 UTC disponível em: ⟨https://doi.org/10.48550/arXiv.1007.1245⟩.

FRISCH, U.; HASSLACHER, B.; POMEAU, Y. Lattice-Gas Automata for the Navier-Stokes Equation. In: Phys. Rev. Lett., v. 56, n. 14, p. 1505-1508, 1986. Disponível em: https://link.aps.org/doi/10.1103/PhysRevLett.56.1505.

GINGOLD, R. A.; MONAGHAN, J. J. Smoothed particle hydrodynamics - Theory and application to non-spherical stars. Monthly Notices of the Royal Astronomical Society, v. 181, p. 375–389, nov. 1977.

MOROZ, M. Reintegration tracking, Mykhailo Moroz 's blogs, aug. 2020, disponível em: ⟨https://michaelmoroz.github.io/Reintegration-Tracking/⟩.

MONAGHAN, J. Simulating free surface flows with sph. J. Comput. Phys., Academic Press Professional, Inc., San Diego, CA, USA, v. 110, n. 2, p. 399–406, fev. 1994. ISSN 0021-9991. Dispon´ıvel em: ⟨http://dx.doi.org/10.1006/jcph.1994.1034⟩.

PAIVA, A. et al. Simulação de Fluidos sem Malha:Uma introdução ao método SPH. [S.l.]: 27th Colóquio Brasileiro de Matemática, IMPA, 2009. ISBN 9788524403026.

Schechter, Hagit and Bridson, Robert. "Ghost SPH for animating water." ACM Transactions on Graphics 31 (2012): 1-8. DOI: 10.1145/2185520.2335412.

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.

TESCHNER, M. et al. Optimized spatial hashing for collision detection of deformable objects. In: ERTL, T. et al. (Ed.). Vision, modeling, and visualization 2003. Berlin: AKA, 2003. p. 47–54.