5
Implementação SPH

No Capítulo 4, mostramos com detalhes a aproximação SPH das equações de Navier–Stokes. Nesse capítulo, discutiremos os aspectos computacionais de implementação de um simulador de fluidos baseado no método SPH, onde o ciclo de simulação de um sistema SPH segue cada uma das etapas mostradas na Figura 5.1. Essas etapas serão apresentadas nas seções desse capítulo.

Na Seção 5.1, apresentaremos algumas estruturas de dados de busca de pares de interação entre as partículas. Na Seção 5.2, abordaremos os esquemas de integração numérica para resolver as equações que descrevem a dinâmica do sistema. Nas Seções 5.3 e 5.4, mostraremos a forma em que as condições de fronteira do sistema são usualmente modeladas e a representação da superfície livre do fluido, respectivamente. E finalmente, apresentaremos na Seção 5.5 um exemplo de uma simulação SPH.

5.1
Busca de partículas vizinhas

No método SPH como todas as aproximações são realizadas através de convoluções discretas com núcleos que possuem suporte compacto de raio κh, somente um número finito de partículas estarão dentro da área de influência de uma partícula dada. Essas partículas são conhecidas como sendo *partículas vizinhas* de uma determinada partícula. Ao contrário dos métodos eulerianos,

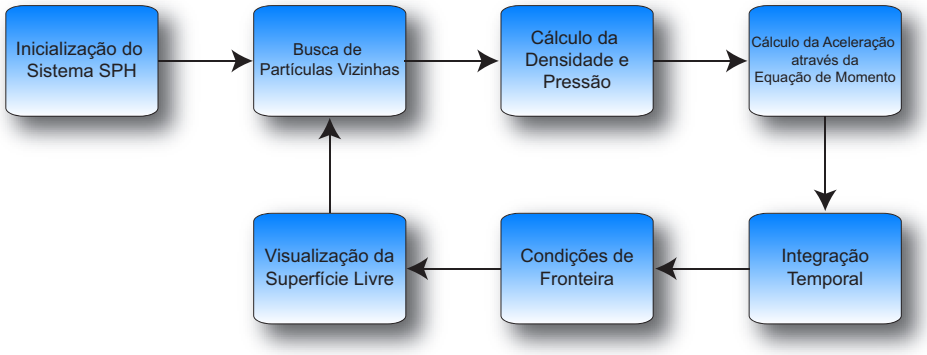

Figura 5.1: Uma visão geral do ciclo de simulação de um sistema SPH.

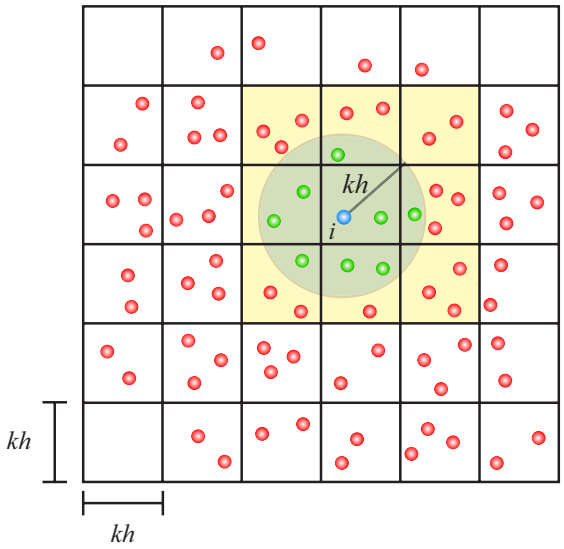

Figura 5.2: Busca das partículas vizinhas utilizando um grid uniforme bidimensional com o espaçamento sendo o raio de influência κh. A região amarela representa as células do grid onde será realizada a busca.

onde as posições das células vizinhas do grid são bem definidas, no SPH as partículas vizinhas geralmente variaram com o tempo.

A busca de partículas vizinhas em SPH é uma tarefa árdua e requer estruturas de dados eficientes que possam realizar consultas de partículas que distam entre si uma distância menor do que o raio de influência do núcleo. Exemplos dessas estruturas de dados são: *grids uniformes*, *tabela hash*, *octrees*, *kd–trees* e *bsp–trees* [34]. Essas estruturas de dados se diferem na complexidade computacional de sua construção ou atualização, como também na consulta de vizinhança de uma partícula e no consumo de memória. Estruturas de dados não hierárquicas tais como grids podem ser construídas e atualizadas eficientemente, enquanto estruturas de dados hierárquicas como octrees, são geralmente mais caras de construir e de atualizar, mas podem ser bastante eficientes quando o comprimento suave do núcleo é variável.

5.1.1 Algoritmo baseado em grid

Esse algoritmo é recomendado para casos em que o núcleo possui comprimento suave constante. Nessa implementação um grid uniforme (células alinhadas com os eixos) é construído sobre o domínio do problema (Figura 5.2). Para núcleos que possuem raio de influência κh, o espaçamento do grid também será κh e uma célula pode ser identificada através do seu vértice mínimo, ou seja, o vértice que possui os valores mínimos nas três coordenadas. Cada célula do grid está associada a uma lista simplesmente encadeada contendo todas

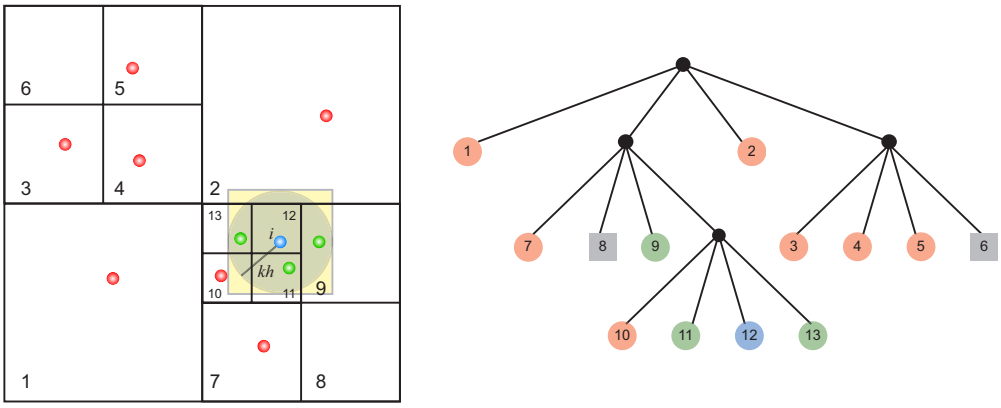

Figura 5.3: Estrutura de árvore utilizada no algoritmo de busca e a subdivisão hierárquica do espaço bidimensional. A busca é realizada através de um teste de intersecção do cubo envolvente da partícula i (região em amarelo) com as células representadas pelos nós da árvore.

as partículas que ocupam a porção de espaço delimitada pela própria célula. Dado um grid 3D, um ponto $(x, y, z) \in \mathbb{R}^3$ é mapeado numa célula do grid com índice (I, J, K) da seguinte maneira

$$(I, J, K) = \left(\left\lfloor \frac{x}{kh} \right\rfloor, \left\lfloor \frac{y}{kh} \right\rfloor, \left\lfloor \frac{z}{kh} \right\rfloor \right), \tag{5-1}$$

onde $\lfloor\ \rfloor$ representa a menor parte inteira. Através do mapeamento (5-1) podemos armazenar as partículas na listas associadas às células do grid. Perceba que a construção do grid é muito eficiente, pois as partículas são inseridas nas listas em tempo constante $\mathcal{O}(1)$, resultando numa complexidade de construção e atualização de $\mathcal{O}(N)$ sendo N o número de partículas do sistema.

Após a construção do grid, dada uma partícula i as suas partículas vizinhas só podem estar na mesma célula ocupada pela partícula i ou em suas células diretamente adjacentes. Assim, a busca por partículas que possuem uma distância menor do que κh a partir da partícula i é restrita a 3^d células, onde d é a dimensão do espaço. Se a média do número de partículas por células for suficientemente pequena, a complexidade do algoritmo baseado em grid uniforme é $\mathcal{O}(N)$. O algoritmo se torna menos eficiente quando o comprimento suave varia em relação ao tempo, logo o espaçamento do grid pode não ser ótimo para todas as partículas.

5.1.2 Algoritmo baseado em octree

Esse algoritmo funciona muito bem em simulações onde o comprimento suave do núcleo varia entre cada partícula do sistema. A estratégia do algoritmo é a de criar uma árvore octária (octree) de acordo com a posição das partículas. A octree subdivide recursivamente o domínio do problema em octantes que contenham pelo menos uma partículas. Cada octante corresponde a um nó da árvore, a recursão termina quando cada folha da árvore possuir apenas uma partícula (Figura 5.3). Se a árvore estiver balanceada, a complexidade da construção e atualização da octree é na média $\mathcal{O}(N \log N)$ e no pior caso $\mathcal{O}(N^2)$, sendo N o número de partículas do sistema. Após a octree ser gerada, podemos usar essa estrutura para realizar a busca de partículas vizinhas.

Dada uma partícula i, a busca por suas partículas vizinhas é realizada através de um cubo centrado na posição da partícula i de lado $2\kappa h_i$ e que envolve a própria partícula. Em cada nível da árvore é feito o teste de quais células da octree são intersectadas pelo cubo envolvente da partícula. Se caso o cubo não intersectar uma determinada célula da octree, a descida ao próximo nível da octree no caminho determinado por essa célula é interrompida. Se o cubo intersectar a célula, o processo de descida ao próximo nível da octree continua recursivamente até encontrar uma folha da árvore que contenha apenas uma partícula. Finalmente, é verificada se essa partícula está dentro do suporte compacto da partícula i. Caso esteja, ela é marcada como sendo uma partícula vizinha. A complexidade da busca baseada em octree é na média $\mathcal{O}(N \log N)$ e $\mathcal{O}(N^2)$ no pior caso.

5.2 Integração numérica temporal

Após o cálculo da versão SPH das equações de Navier–Stokes (Seção 4.2), para atualizar a posição, velocidade e outras propriedades das partículas num certo instante de tempo, precisamos integrar a equação de momento (4-38) em relação ao tempo. Existem vários métodos de integração de equações diferenciais ordinárias, veja o livro de Eberly [18]. Dependendo de como são calculadas as variáveis desconhecidas do problema, os esquemas de integração podem ser classificados em *métodos implícitos* e *métodos explícitos*. Nos métodos implícitos as variáveis desconhecidas são dadas implicitamente como a solução de um sistema de equações envolvendo o estado atual e o desconhecido do sistema, enquanto que nos métodos explícitos elas são calculadas considerando apenas o estado atual. As vantagens dos métodos explícitos são: podem ser implementados facilmente, podem ser calculados de maneira rápida e eficiente.

Porém a estabilidade do método depende de como é feita a escolha do passo de tempo Δt, para que haja a convergência o passo de tempo Δt deve obedecer ao critério de estabilidade numérica conhecido como *condição Courant–Friedrichs–Lewy* (CFL) [14]. Já os métodos implícitos são estáveis em passos de tempo arbitrários, mas para isso requer resolver sistemas algébricos que são na sua maioria não–lineares. Nessa tese, optamos em usar métodos explícitos como *método de Euler* e o *integrador leap–frog* em nossa implementação.

5.2.1 Método de Euler

O método explícito mais simples é conhecido como método de Euler. Nesse método, a velocidade $\mathbf{v}_i$ e a posição $\mathbf{x}_i$ são atualizadas paralelamente da seguinte maneira:

$$\begin{aligned} \mathbf{v}_i\left(t+\Delta t\right) &= \mathbf{v}_i\left(t\right)+\Delta t\,\mathbf{a}_i(t) \\ \mathbf{x}_i\left(t+\Delta t\right) &= \mathbf{x}_i\left(t\right)+\Delta t\,\mathbf{v}_i\left(t+\Delta t\right), \end{aligned} \tag{5-2}$$

onde $\mathbf{a}_i$ é aceleração da partícula num instante de tempo t, obtida a partir da equação (4-38).

Infelizmente, o método de Euler possui uma precisão de primeira ordem e pode se tornar bastante instável em certas circunstâncias – uma propriedade indesejável para qualquer integrador. Essa instabilidade pode ser remediada tomando um passo de tempo Δt muito pequeno.

5.2.2 Integrador leap–frog

Um integrador mais atraente e tão simples quanto o método de Euler, mas com uma precisão de segunda ordem, é o integrador leap–frog [18]. Nesse esquema de integração a velocidade de cada partícula é calculada nos pontos médio dos intervalos de tempo (Figura 5.4). Assim, posição e velocidade são avaliadas intercaladamente da seguinte forma:

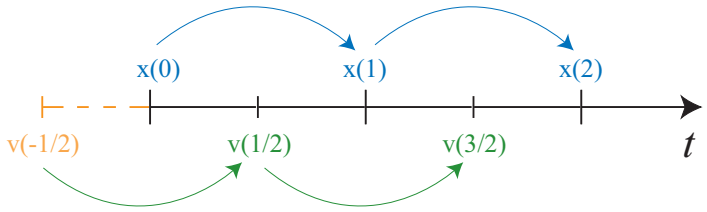

Figura 5.4: Esquema de integração leap–frog: posição $\mathbf{x}_i$ e velocidade $\mathbf{v}_i$ numa partícula i são avaliadas de maneira intercalada em relação ao tempo t.

$$\begin{aligned} \mathbf{v}_i\left(t+\tfrac{1}{2}\Delta t\right) &= \mathbf{v}_i\left(t-\tfrac{1}{2}\Delta t\right)+\Delta t\,\mathbf{a}_i(t) \\ \mathbf{x}_i\left(t+\Delta t\right) &= \mathbf{x}_i\left(t\right)+\Delta t\,\mathbf{v}_i\left(t+\tfrac{1}{2}\Delta t\right). \end{aligned} \tag{5-3}$$

Note que a velocidade no tempo t não é dada de forma explícita, logo ela é calculada através da média entre a velocidade anterior e posterior

$$\mathbf{v}_i\left(t\right)=\frac{1}{2}\left[\mathbf{v}_i\left(t+\frac{1}{2}\Delta t\right)+\mathbf{v}_i\left(t-\frac{1}{2}\Delta t\right)\right]. \tag{5-4}$$

Além disso, a inicialização do método requer uma etapa adicional no cálculo da velocidade $\mathbf{v}_i\left(-\frac{1}{2}\right)$ através do método de Euler

$$\mathbf{v}_i\left(-\frac{1}{2}\right)=\mathbf{v}_i(0)-\frac{1}{2}\Delta t\,\mathbf{a}_i(0). \tag{5-5}$$

O integrador leap–frog é reversível no tempo devido à forma simétrica na qual ele é definido. Métodos explícitos clássicos como o método Euler, método do ponto–médio e os métodos de Runge–Kutta não possuem essa propriedade, pois todos eles avaliam as derivadas de maneira assimétrica. A reversibilidade no tempo é uma propriedade importante, pois ela garante conservação de energia do sistema.

5.3 Condição de fronteira

O tratamento de fronteiras sólidas no SPH foi proposto por Monaghan em [47], a sua idéia era a de representar as fronteiras do domínio através de partículas virtuais conhecidas como *partículas fantasmas* (veja Figura 5.5). O objetivo das partículas fantasmas é gerar forças altamente repulsivas e assim prevenir a interpenetração de partículas do fluido nas fronteiras sólidas do problema. Essa força de repulsão é calculada usando uma expressão matemática

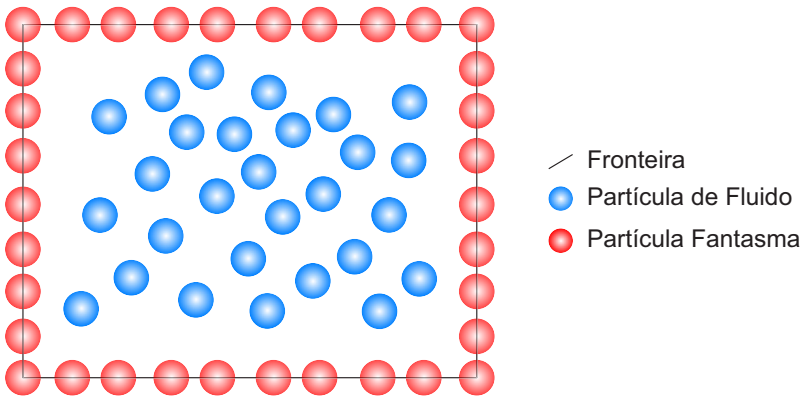

Figura 5.5: As partículas fantasmas (vermelho) são utilizadas para simular as condições de fronteira sólidas, evitando que as partículas de fluido (azul) escapem do domínio do problema.

semelhante à do potencial de Lennard–Jones utilizado em dinâmica molecular [4]. Portanto, a força de repulsão sobre uma partícula de fluido i que colide com uma partícula fantasma g é

$$\Gamma_{ig} = \begin{cases} D\left[\left(\frac{r_0}{r_{ig}}\right)^{12} - \left(\frac{r_0}{r_{ig}}\right)^{4}\right]\frac{\mathbf{x}_{ig}}{r_{ig}^2}, & \frac{r_0}{r_{ig}} \leq 1 \\ 0, & \frac{r_0}{r_{ig}} > 1 \end{cases}, \tag{5-6}$$

com $\mathbf{x}_{ig} = \mathbf{x}_i - \mathbf{x}_g$, $r_{ig} = \|\mathbf{x}_i - \mathbf{x}_g\|$. A constante D depende do problema e deve ser da mesma ordem de grandeza do quadrado da maior velocidade esperada no escoamento. O raio de interação r_0 é importante na simulação. Pois se for escolhido um valor muito alto para r_0, a força de repulsão pode causar uma grande perturbação no estado inicial das partículas e conseqüentemente arruinar toda a simulação. Se for escolhido um valor pequeno, a força de repulsão não será suficiente para evitar a interpenetração de partículas na fronteira. Na maioria dos casos, é recomendado r_0 ser um valor próximo da distância inicial das partículas do sistema.

Uma das maiores dificuldades da abordagem acima é a de modelar fronteiras que possuem geometria complexa através de partículas fantasmas. Para simular a interação entre corpos deformáveis com fluidos utilizando SPH, Müller *et al.* em [53] utilizaram quadratura Gaussiana para gerar partículas fantasmas sobre a malha triangular de um objeto deformável. No Capítulo 6, mostraremos uma nova abordagem no tratamento de fronteiras sólidas de natureza puramente geométrica.

5.4 Representação implícita da superfície livre

Uma superfície implícita de um *isovalor* c é dada como $S = f^{-1}(c) = \{\mathbf{x} \in \mathbb{R}^3 : f(\mathbf{x}) = c\}$ onde $f : \mathbb{R}^3 \to \mathbb{R}$ é uma função escalar. A função f é definida tal que um ponto $\mathbf{x}$ está no interior da superfície S temos $f(\mathbf{x}) - c < 0$ e quando $\mathbf{x}$ está no exterior de S temos $f(\mathbf{x}) - c > 0$. Logo, para verificar se um ponto $\mathbf{x}$ está dentro ou fora de uma superfície S basta observar o sinal de $f(\mathbf{x})$. Por outro lado, fazer amostragem de pontos sobre uma superfície é um processo laborioso, pois se torna um problema de achar raízes de uma função arbitrária. Assim a visualização de superfícies implícitas se torna mais difícil do que a visualização de superfícies paramétricas.

Geralmente, a função $f : \mathbb{R}^3 \to \mathbb{R}$ é definida como sendo a *função distância com sinal* da superfície S, daí a amostragem de um campo distância se torna uma tarefa fácil. A visualização de superfícies implícitas é feita tradicionalmente com malhas triangulares obtidas através do algoritmo de *marching cubes* (MC) [40]. No método MC a função implícita é armazenada

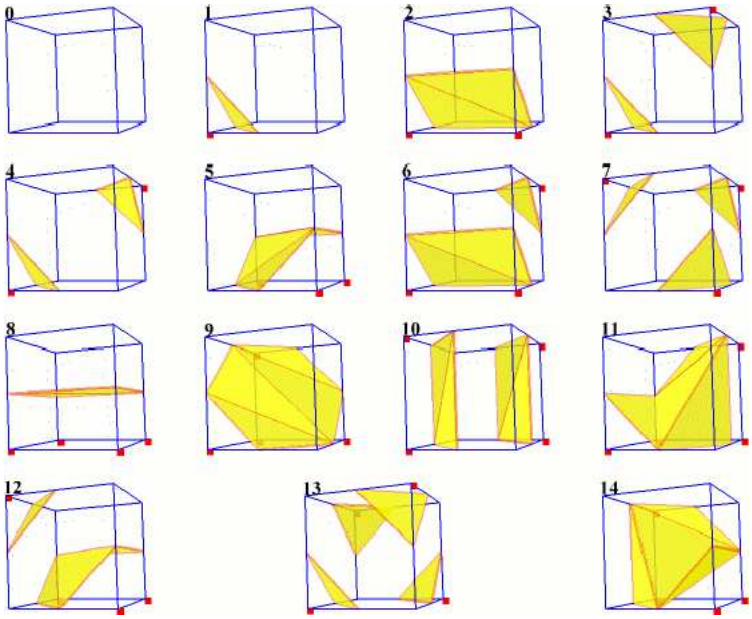

Figura 5.6: Tabela do algoritmo de Marching Cubes (MC) que codifica a triangulação de cada cubo do grid dependendo da configuração dos sinais em seus vértices.

num campo escalar amostrado num grid tridimensional. O MC classifica os vértices do grid como positivo ou negativo, de acordo com o isovalor dado. Após cada vértice ser classificado, o algoritmo faz uma varredura nos cubos do grid verificando a configuração dos sinais dos vértices de cada cubo. Se um cubo possuir mudança de sinal, parte da triangulação da superfície está no seu interior e é dada através de uma tabela pré–definida de acordo com sua configuração de sinais, veja a Figura 5.6. Finalmente, as posições dos vértices dos triângulos no interior do cubo são obtidas através de uma interpolação tri–linear.

Entretanto, visualizar superfícies com singularidades como pontas e quinas, requer informações adicionais tais como o gradiente da superfície (normal da superfície). O algoritmo *dual marching cubes* (DMC) [64] é uma extensão do algoritmo de MC devido ao uso da informação do gradiente para detectar e reconstruir singularidades através de um grid adaptativo. A vantagem das superfícies implícitas é a sua consistência no ponto de vista de não possuir auto–intersecções e a de lidar intrinsecamente com as mudanças topológicas. Entretanto, o algoritmo de MC possui ambigüidades topológicas que podem aparecer nas faces ou no interior do cubo. Essas ambigüidades podem gerar mudanças topológicas incorretas, por exemplo, duas componentes conexas de uma superfície podem se unir pelo fato de estarem próximas e formar apenas uma única componente (Figura 5.7).

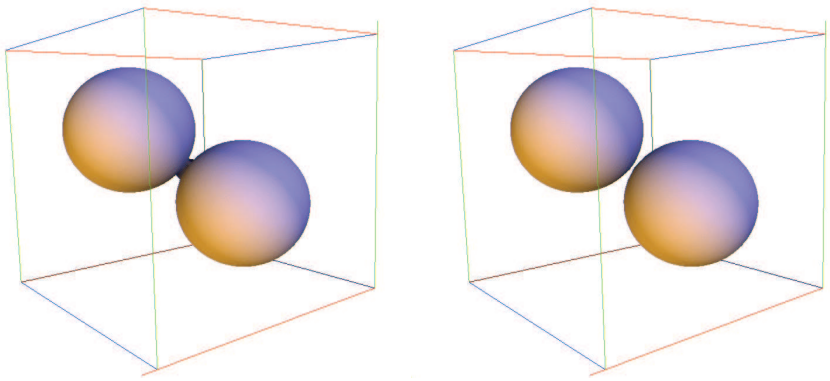

Figura 5.7: Superfície implícita de duas esferas próximas uma da outra. À esquerda a superfície é gerada através do algoritmo de MC original. À direita a superfície é gerada utilizando o algoritmo de MC com garantias topológicas.

A captura da superfície livre do fluido é dada implicitamente através da aproximação SPH da função característica dada por Morris em [49]:

$$\chi(\mathbf{x}) = \sum_{j \in N(\mathbf{x})} \frac{m_j}{\rho_j} W(\mathbf{x} - \mathbf{x}_j, h), \tag{5-7}$$

o isovalor é tomado no intervalo $[0, 1]$, isso se deve ao fato do núcleo W possuir suporte compacto e ser partição da unidade. A geração da malha triangular da superfície livre do fluido é feita através de uma implementação eficiente do MC e com garantias topológicas [36]. Usamos a mesma estrutura de dados utilizada na busca de partículas vizinhas (Seção 5.1) para acelerar a avaliação da função

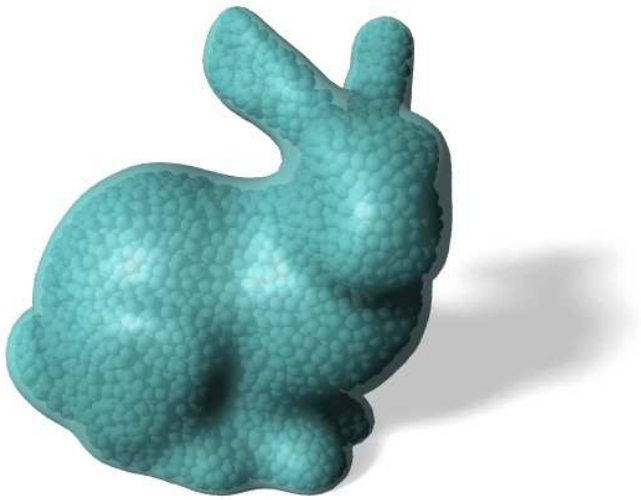

Figura 5.8: Visualização da superfície implícita extraída do modelo Stanford Bunny constituído de partículas.

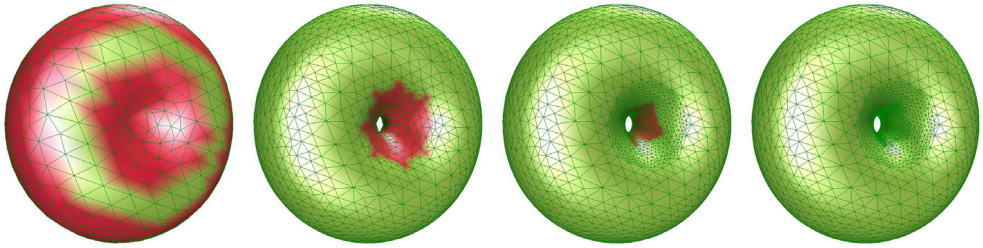

Figura 5.9: Triangulação adaptativa da isosuperfície do torus utilizando um algoritmo robusto de DMC. As regiões em verde representam as regiões da superfície que possuem garantias topológicas e geométricas, enquanto as regiões com ambigüidades (vermelho) são resolvidas com um número pequeno de refinamentos.

característica (5-7) no grid. A Figura 5.8 mostra a superfície livre do fluido gerada a partir de modelo volumétrico formado por partículas. Uma classe especial de superfícies implícitas usadas na representação da superfície livre de fluidos são as *level–sets* [56]. A superfície level–set $F(\mathbf{x}, t)$ acompanha a evolução da superfície livre do fluido ao longo de sua normal de acordo com um campo escalar de velocidade $\mathbf{v}(\mathbf{x}, t)$ e é representada através de uma equação diferencial parcial

$$\frac{\partial F}{\partial t} = -\mathbf{v} \left\| \nabla F \right\| . \tag{5-8}$$

Paiva *et al.* [58] criaram um algoritmo robusto de DMC com garantias topológicas utilizando ferramentas numéricas auto–validadas como aritmética intervalar e diferenciação automática (Figura 5.9). Um grande desafio na visualização de superfície livre dada de forma implícita é a síntese de textura dinâmica, devido ao fato de geralmente a superfície não possuir uma parametrização explícita. Todas as animações apresentadas nessa tese foram geradas utilizando o *ray–tracer* de código aberto POV–Ray (`http://www.povray.org`).

5.5 Exemplo numérico

Um exemplo clássico da aplicação do método SPH em superfície livre é a simulação da quebra de uma barragem [47]. O problema consiste num volume de água confinado numa região cúbica de um canal onde é instantaneamente retirada a barragem da represa. O método SPH é utilizado para simular o comportamento da queda da coluna de água através das equações de Navier–Stokes (4-37) – (4-38). O problema é esquematizado na Figura 5.10.

Inicialização numérica do problema. O sistema é iniciado com densidade inicial da água sendo $\rho_0 = 1000\text{kg/m}^3$, a viscosidade $\mu = 10^{-3}\text{Ns/m}^2$ e a gravidade $\mathbf{g} = 9.8\text{m/s}^2$. A velocidade característica do escoamento $\mathbf{v}_e$ é estimada

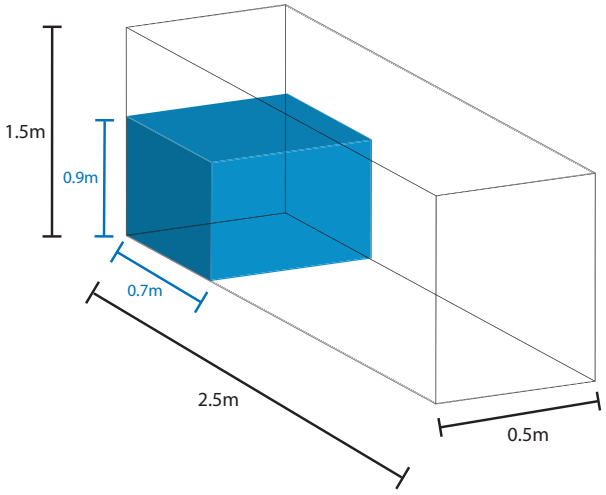

Figura 5.10: Configuração inicial do problema.

a partir da conservação de energia mecânica do sistema (transformação de energia potencial em cinética), segue que

$$\frac{m\mathbf{v}_e^2}{2} = m\mathbf{g}H, \tag{5-9}$$

como a coluna de água possui altura $H = 0.9\text{m}$, logo $\mathbf{v}_e = \sqrt{2\mathbf{g}H} = 4.2\text{m/s}$. Assim, podemos estimar a velocidade do som $c \approx 42\text{m/s}$ e obter a constante $B = 252\text{kPa}$ da equação (4-39).

Dado o volume total do fluido V, a massa de uma partícula i é determinada da forma

$$m_i = \rho_0 \frac{V}{N}, \tag{5-10}$$

sendo N o número total de partículas do sistema. Cada partícula do sistema é representada por uma esfera, logo o raio de uma partícula i é

$$r_i = \sqrt[3]{\frac{3}{4}\frac{m_i}{\rho_0\,\pi}}. \tag{5-11}$$

Na simulação do problema utilizamos 10^4 partículas (Figura 5.11), portanto a massa de cada partícula vale $m_i = 3.15 \times 10^{-2}\text{kg}$. As partículas são inicialmente geradas num grid uniforme onde o raio de influência κh é tomado como sendo 1.2 vezes o tamanho do espaçamento do grid, recomenda-se que em simulações SPH que o número de partículas vizinhas seja no mínimo 20 em simulações 2D e 56 em simulações 3D [38]. O critério de estabilidade numérica da integração temporal respeita a condição CFL

$$\Delta t = 0.1 \min_i \left\{ \frac{h}{\|\mathbf{v}_i\| + c} \right\}. \tag{5-12}$$

A atualização do estado de um sistema SPH convencional é realizada seguindo o Algoritmo 1. Em todas as simulações dessa tese, adotamos o

comprimento suave h constante, logo a busca de partículas vizinhas é feita eficientemente utilizando um grid uniforme (Seção 5.1.2).

Algoritmo 1 Sistema SPH convencional

repeat
  Faça a busca das partículas vizinhas. (Seção 5.1.2)
  for cada partícula i **do**
    Atualize a pressão p_i. (equação (4-39))
  end for
  for cada partícula i **do**
    Calcule a derivada da densidade. (equação (4-37))
    Calcule a aceleração. (equação (4-38))
  end for
  for cada partícula i **do**
    Atualize $\mathbf{v}_i$ e ρ_i com o integrador leap–frog. (Seção 5.2.2)
    Aplique a condição de fronteira. (Seção 5.3)
  end for
  Atualize Δt usando a condição CFL. (equação (5-12))
  $tempo = tempo + \Delta t$
until $tempo < tempo_{total}$

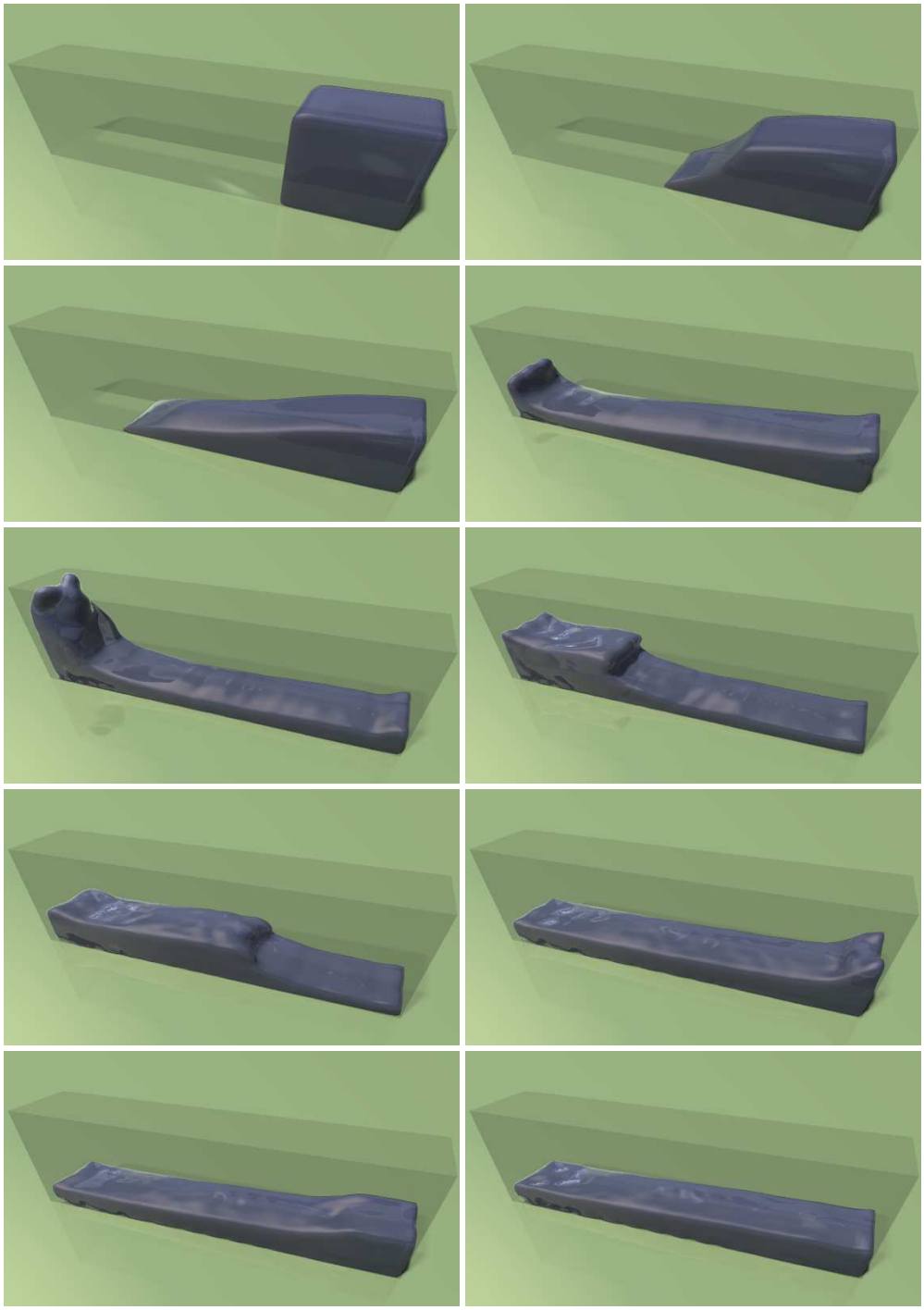

Figura 5.11: Simulação SPH da quebra de uma barragem utilizando 10^4 partículas.