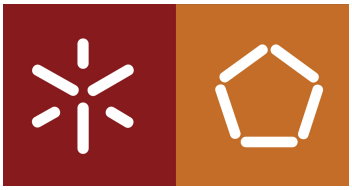


Universidade do Minho

Mestrado em Engenharia Informática

Cloth Simulation

Visualização em Tempo Real

Grupo 6
Gonçalo Gonçalves Barroso - PG58980

26 de janeiro de 2026

Conteúdo

1 Introdução

No âmbito da unidade curricular de Visualização em Tempo Real, escolhemos como tema Cloth Simulation, que simula o comportamento de tecidos, permitindo representar de forma realista a sua deformação e movimento sob a ação de forças externas, como a gravidade, bem como a interação com objetos do ambiente.

Neste relatório, focamo-nos no estudo de um algoritmo específico de Cloth Simulation, descrevendo o seu funcionamento e justificando as opções por nós adotadas. Adicionalmente analisamos as vantagens e limitações desta abordagem.

2 Escolha do algoritmo

Após a análise de diferentes abordagens existentes, optámos pela utilização do ***Mass-Spring Model*** para a implementação de *Cloth Simulation* neste trabalho. Esta escolha deve-se principalmente ao equilíbrio que este modelo oferece entre simplicidade de implementação, desempenho computacional e qualidade visual.

2.1 Funcionamento do Mass-Spring Model

No *Mass-Spring Model*, o pano é representado como uma grid regular de partículas (*mass points*), onde cada partícula possui propriedades físicas como posição, velocidade e massa. As partículas são interligadas por springs (molas), que modelam as forças elásticas responsáveis por manter a estrutura do tecido.

O comportamento de cada *spring* é descrito pela Lei de Hooke, que define a força exercida por uma mola em função da sua deformação relativamente ao comprimento de repouso. De forma simplificada, esta lei afirma que a força aplicada por uma *spring* é proporcional à diferença entre o comprimento atual da mola e o seu comprimento de repouso, atuando no sentido oposto à deformação. Esta relação pode ser expressa como:

$$\vec{F} = -k\,(l - l_0)\,\hat{d} \tag{2.1}$$

onde k representa a *stiffness* da *spring*, l o comprimento actual, l_0 o comprimento de repouso e $\hat{d}$ a direção da força. Esta força faz com que as partículas tendam a regressar à sua posição de equilíbrio sempre que o pano é esticado ou comprimido.

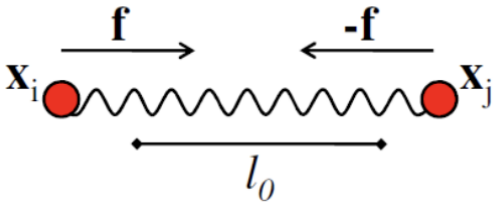


Figura 2.1: Representação da Lei de Hooke aplicada a uma spring entre duas partículas.

Para além da força elástica, é normalmente introduzido um termo de amortecimento (*damping*), proporcional à velocidade relativa entre partículas ligadas pela mesma *spring*. Este termo tem como objetivo reduzir oscilações excessivas e dissipar energia, contribuindo para a estabilidade da simulação.

As ligações entre partículas não são todas iguais. De forma a simular de forma mais realista o comportamento do tecido, são utilizados diferentes tipos de *springs*, cada um com uma função específica na malha:

- **Structural Springs**: ligam partículas adjacentes na horizontal e vertical da *grid*. Estas ligações são responsáveis por manter a forma base do pano e resistir a estiramentos excessivos.
- **Shear Springs**: ligam partículas na diagonal. Estas *springs* ajudam a preservar a área da malha e evitam deformações irrealistas em forma de losango.
- **Bending Springs**: ligam partículas não adjacentes, normalmente separadas por uma partícula intermédia. Estas ligações permitem controlar a flexão do pano, evitando dobras demasiado abruptas e conferindo maior suavidade visual à simulação.

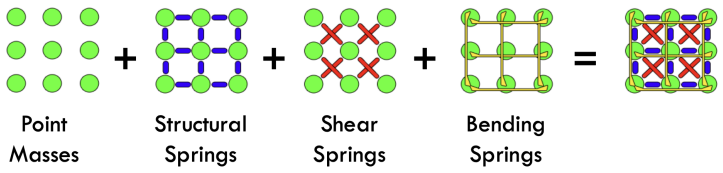


Figura 2.2: Mass-Spring system for cloth simulation

As partículas do sistema estão ainda sujeitas a forças externas, como a gravidade, que provocam o movimento global e as deformações naturais do pano. A soma de todas as forças internas (*springs* e *damping*) e externas resulta na força total aplicada a cada partícula.

A evolução do sistema ao longo do tempo é calculada recorrendo a um método de *numerical integration*, que permite actualizar a velocidade e a posição das partículas em cada passo temporal. Este processo é repetido iterativamente, originando o movimento contínuo e deformável do pano.

2.2 Vantagens do Mass-Spring Model

Uma das principais vantagens do *Mass-Spring Model* é a sua simplicidade conceptual. O modelo é intuitivo e permite compreender facilmente a relação entre forças, movimento e deformação.

Do ponto de vista do desempenho, trata-se de um modelo computacionalmente eficiente, especialmente quando comparado com abordagens mais complexas, como os *Finite Element Methods* (FEM).

Outra vantagem relevante é a sua flexibilidade pois através do ajuste dos parâmetros das *springs*, como *stiffness* e *damping*, é possível simular diferentes tipos de tecidos.

2.3 Limitações do *Mass-Spring Model*

Apesar das suas vantagens, o *Mass-Spring Model* apresenta algumas limitações. Uma delas é a forte dependência da calibração dos parâmetros, sendo necessário ajustar cuidadosamente os valores de *stiffness* e *damping* para evitar comportamentos instáveis ou pouco realistas.

Além disso, este modelo constitui uma aproximação discreta do comportamento físico do tecido, não garantindo uma simulação fisicamente exata. Em situações que exigem elevado rigor físico, como simulações científicas ou industriais, métodos mais avançados podem ser mais adequados.

3 Implementação

Este capítulo descreve a implementação prática do algoritmo de Cloth Simulation baseado no *Mass-Spring Model* explicado anteriormente. A simulação foi desenvolvida recorrendo maioritariamente a *compute shaders*, permitindo paralelizar o cálculo das forças e a integração do movimento diretamente na GPU, garantindo melhor desempenho e escalabilidade.

3.1 Justificação Arquitetural (Engenharia de Software)

As decisões arquiteturais da implementação foram guiadas por princípios de engenharia de software, visando modularidade, desempenho e manutenibilidade.

3.1.1 Separação Compute/Render

A arquitetura segue o princípio de **separação de responsabilidades**, dividindo claramente a lógica de simulação física (compute pass) da renderização visual (render pass). Esta separação permite:

- Modificar a física sem alterar o código de renderização e vice-versa
- Testar e otimizar cada componente independentemente
- Reutilizar os shaders de renderização com diferentes simulações

3.1.2 Utilização de Shader Storage Buffer Objects (SSBO)

A escolha de *SSBOs* em vez de texturas ou *Uniform Buffer Objects* justifica-se por:

- **Leitura e escrita**: SSBOs permitem operações de read/write no mesmo buffer, essencial para atualizar posições e velocidades in-place
- **Partilha entre shaders**: O mesmo buffer é acedido pelo compute shader (física) e pelos shaders de renderização, evitando cópias de memória

3.2 Estrutura da Malha

O tecido é representado como uma malha regular de 25×25 vértices, totalizando 625 partículas. Cada partícula é processada em paralelo na GPU através de *compute shaders*.

A malha é gerada através de um script Python que produz:

- Ficheiro OBJ com a geometria da malha
- Buffers de posições, velocidades, normais e forças
- Buffers de adjacências para as *Structural*, *Shear* e *Bending Springs*

3.3 Pipeline de Renderização

A simulação segue um pipeline com duas fases principais por *frame*:

1. **Compute Pass**: Executa o *compute shader* que calcula todas as forças físicas, atualiza velocidades e posições, e resolve colisões.
2. **Render Pass**: Desenha o tecido e a esfera utilizando os *vertex*, *geometry* e *fragment shaders*.

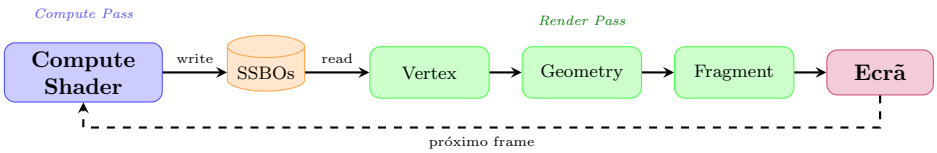


Figura 3.1: Pipeline de renderização da simulação de tecido

3.3.1 Compute Pass (Física)

O *compute shader* é executado primeiro e realiza:

- Cálculo das forças de mola (Lei de Hooke) e amortecimento
- Aplicação da gravidade
- Integração numérica (Euler) para atualizar velocidades e posições
- Deteção e resposta a colisões (esfera e auto-colisão)
- Cálculo das normais para iluminação

Os resultados são escritos nos *SSBOs* partilhados.

3.3.2 Render Pass (Visualização)

A fase de renderização utiliza três shaders em sequência:

- **Vertex Shader** (`cloth.vert`): Lê as posições atualizadas dos SSBOs e passa os índices dos vértices.

- **Geometry Shader** (`cloth.geom`): Recebe triângulos e emite **duas faces** — frontal e traseira com normais invertidas — permitindo que o tecido seja visível de ambos os lados.
- **Fragment Shader** (`cloth.frag`): Aplica iluminação difusa e textura ao tecido.

3.4 Implementação das Forças

3.4.1 Lei de Hooke

A força elástica entre duas partículas conectadas por uma mola é calculada pela seguinte função GLSL:

Listing 3.1: Implementação da Lei de Hooke em GLSL

```
vec3 hookes_law(vec3 p1, vec3 p2, float stiffness, float
    edge_distance) {
    vec3 v = p1 - p2;
    float l = length(v);
    vec3 vec_dir = normalize(v);
    vec3 x = (l - edge_distance) * vec_dir;
    return -stiffness * x;
}

```

3.4.2 Força de Amortecimento

O amortecimento é proporcional à velocidade relativa entre partículas:

Listing 3.2: Implementação da força de amortecimento

```
vec3 damping_force(vec3 p1, vec3 p2, vec4 vel1, vec4 vel2,
    float damping_coeff) {
    vec3 vec = p1 - p2;
    vec3 vel = vel1.xyz - vel2.xyz;
    vec3 vec_dir = normalize(vec);
    vec3 x = ((vel * vec) / length(vec)) * vec_dir;
    return -damping_coeff * x;
}

```

3.4.3 Gravidade

A força gravitacional é aplicada uniformemente a todas as partículas:

$$\vec{F}_{gravidade} = M \times \vec{g} = 0.1 \times (0, -9.8, 0) \tag{3.1}$$

3.5 Integração Numérica

Para atualizar as posições e velocidades das partículas, utilizamos a **integração de Euler**, um método simples mas eficaz para simulações em tempo real:

$$\vec{a} = \frac{\vec{F}_{total}}{M} \tag{3.2}$$

$$\vec{v}_{novo} = \vec{v} + \vec{a} \times \Delta t \tag{3.3}$$

$$\vec{p}_{novo} = \vec{p} + \vec{v}_{novo} \times \Delta t \tag{3.4}$$

O passo temporal $\Delta t = 0.001s$ foi escolhido suficientemente pequeno para garantir estabilidade numérica.

3.6 Sistema de Colisões

3.6.1 Colisão com a Esfera

Quando um vértice penetra o volume da esfera, é reposicionado na superfície:

Listing 3.3: Deteção e resposta à colisão com a esfera

```
if (length(new_pos.xyz - sphereCenter) < sphereRadius) {
    vec3 collision_normal = normalize(new_pos.xyz -
    sphereCenter);
    new_pos.xyz = sphereCenter + collision_normal * (
    sphereRadius + 0.01);

    float vel_towards_sphere = dot(new_vel, collision_normal);
    if (vel_towards_sphere < 0.0) {
        new_vel -= vel_towards_sphere * collision_normal * 1.5;
    }
    new_vel *= 0.85; // Fricao
}

```

3.6.2 Auto-Colisão

Para evitar que o tecido atravesse a si próprio, é implementada deteção de colisão entre vértices não adjacentes. Cada vértice é tratado como uma esfera com raio `collision_radius`, e quando duas esferas se sobrepõem, os vértices são separados e as suas velocidades ajustadas.

O algoritmo verifica todos os pares de vértices, excluindo:

- O próprio vértice ($i \neq index$)

- Vértices adjacentes (ligados por molas), pois estes têm forças de mola a controlá-los

Listing 3.4: Deteção e resposta à auto-colisão

```
for (int i = 0; i < height * width; i++) {
    if (i != index && !is_adjacent(index, i) &&
        length(new_pos.xyz - pos[i].xyz) < 2 * collision_radius
    ) {

        // Calcular normal de colisao (direcao de separacao)
        vec3 n = normalize(new_pos.xyz - pos[i].xyz);

        // Empurrar vertice para fora da zona de colisao
        new_pos.xyz = pos[i].xyz + n * 2 * collision_radius;

        // Ajustar velocidade: remover componente em direcao ao
     outro vertice
        float vel_towards_grid = dot(new_vel, n);
        if (vel_towards_grid < 0.0) {
            new_vel -= vel_towards_grid * n * 1.2; // Fator de
    bounce
        }
    }
}

```

Explicação do algoritmo:

1. **Deteção**: Se a distância entre dois vértices for menor que $2 \times r$ (onde r é o raio de colisão), há sobreposição.

2. **Separação**: O vértice é reposicionado na superfície da esfera de colisão do outro vértice.

3. **Resposta de velocidade**: Se o vértice se move em direção ao outro (velocidade negativa na direção normal), essa componente é invertida com um fator de 1.2, simulando um pequeno *bounce*.

3.7 Pontos Fixos e Animação

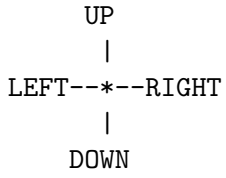
Os dois cantos superiores do tecido estão fixos e oscilam no eixo Z para criar movimento dinâmico.

3.8 Cálculo de Normais

As normais dos vértices são calculadas dinamicamente no *compute shader* para iluminação correta (*smooth shading*). Para cada vértice, a normal é calculada como a **média das normais das faces adjacentes**, usando o produto vetorial das arestas.

3.8.1 Conceito

Cada vértice interior pode ter até 4 quadrantes (faces) adjacentes. A normal do vértice é a soma normalizada das normais dessas faces:



Os 4 quadrantes são percorridos em sentido anti-horário para manter consistência na orientação das normais.

3.8.2 Implementação

Listing 3.5: Verificação de vizinhos disponíveis

```
// Verificar se o vertice nao esta na borda
bool hasLeft = (x > 0);
bool hasRight = (x < width - 1);
bool hasUp = (z > 0);
bool hasDown = (z < height - 1);

// Obter posicoes dos vizinhos (ou posicao atual se na borda)
vec3 left = hasLeft ? pos[index - 1].xyz : currentPos;
vec3 right = hasRight ? pos[index + 1].xyz : currentPos;
vec3 up = hasUp ? pos[index - width].xyz : currentPos;
vec3 down = hasDown ? pos[index + width].xyz : currentPos;

```

Listing 3.6: Cálculo da normal por produto vetorial

```
vec3 vertexNormal = vec3(0.0);

// Quadrante NW: cross(left - current, up - current)
if (hasLeft && hasUp) {
    vertexNormal += cross(left - currentPos, up - currentPos);
}
// Quadrante NE: cross(up - current, right - current)
if (hasUp && hasRight) {
    vertexNormal += cross(up - currentPos, right - currentPos);
}
// Quadrante SE: cross(right - current, down - current)
if (hasRight && hasDown) {
    vertexNormal += cross(right - currentPos, down - currentPos
    );
}
// Quadrante SW: cross(down - current, left - current)
if (hasDown && hasLeft) {
```

```
17      vertexNormal += cross(down - currentPos, left - currentPos)
        ;
18  }
19
20  // Normalizar resultado (com fallback para (0,1,0) se
        degenerado)
21  if (length(vertexNormal) > 0.001) {
22      normals[index] = vec4(normalize(vertexNormal), 0.0);
23  } else {
24      normals[index] = vec4(0.0, 1.0, 0.0, 0.0);
25  }
26
```

3.8.3 Porquê Produto Vetorial?

O **produto vetorial** de duas arestas adjacentes produz um vetor perpendicular ao plano definido por essas arestas — ou seja, a normal da face. A ordem dos operandos (sentido anti-horário) garante que todas as normais apontam para o mesmo lado do tecido.

4 Resultados Obtidos

A simulação implementada apresenta um comportamento fisicamente plausível do tecido, com as seguintes características observadas:

- O pano deforma-se naturalmente sob a ação da gravidade
- As *Structural Springs* mantêm a integridade da malha, evitando esticamento excessivo
- As *Bending Springs* conferem suavidade às dobras do tecido
- O amortecimento reduz oscilações, estabilizando a simulação ao longo do tempo
- A colisão com a esfera funciona corretamente, com o tecido a envolver o objeto
- A auto-colisão previne que o pano atravesse a si próprio

A interface ImGui permite ajustar em tempo real a posição da esfera e os parâmetros físicos (rigidez e massa), possibilitando experimentar diferentes comportamentos do tecido.

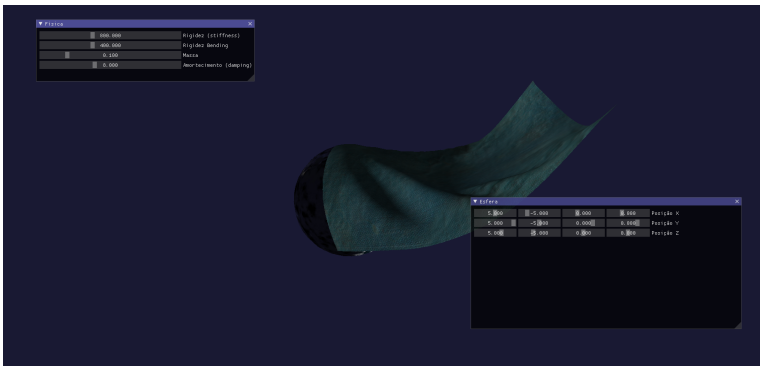


Figura 4.1: Mass-Spring system for cloth simulation

5 Análise Crítica e Comparação

5.1 Pontos Fortes da Implementação

- **Desempenho**: A utilização de *compute shaders* permite processar os vértices em paralelo, garantindo execução em tempo real.
- **Interatividade**: Os parâmetros ajustáveis permitem experimentação e demonstração de diferentes comportamentos.
- **Robustez**: O sistema de colisões previne artefactos visuais como interpenetrações.

5.2 Limitações

- **Integração de Euler**: Embora funcional, métodos como Verlet ofereceriam maior estabilidade para passos temporais maiores.
- **Auto-colisão**: A verificação local pode falhar em casos de deformação extrema.

5.3 Comparação com Alternativas

Existem várias abordagens alternativas para simulação de tecidos, cada uma com características distintas. Nesta secção, comparamos o *Mass-Spring Model* com as principais alternativas.

5.3.1 Position Based Dynamics (PBD)

O método *Position Based Dynamics*, introduzido por Müller et al. (2007), difere do *Mass-Spring Model* ao manipular diretamente as posições das partículas em vez de calcular forças.

Funcionamento: Em cada iteração, PBD aplica *constraints* (restrições) que corrigem as posições das partículas para satisfazer condições como distância fixa entre partículas ou limites de volume. O algoritmo resolve iterativamente estas restrições através de projeções.

Vantagens do PBD:

- Incondicionalmente estável – não sofre de explosões numéricas independentemente do timestep
- Controlo intuitivo através de restrições geométricas
- Excelente para aplicações interativas e jogos

Desvantagens do PBD:

- Comportamento dependente do número de iterações e timestep
- Não conserva energia de forma fisicamente correta
- Rigidez aparente varia com a resolução da malha

Porquê escolhemos Mass-Spring: Embora PBD seja popular em jogos, o *Mass-Spring Model* oferece uma base física mais clara (Lei de Hooke) e permite relacionar diretamente os parâmetros com propriedades físicas reais do tecido.

5.3.2 Finite Element Method (FEM)

O *Finite Element Method* é a abordagem mais rigorosa fisicamente, modelando o tecido como um meio contínuo discretizado em elementos finitos.

Funcionamento: O tecido é tratado como um material elástico contínuo, com propriedades definidas por tensores de stress e strain. A malha é dividida em elementos (tipicamente triângulos) e as equações de elasticidade são resolvidas numericamente.

Vantagens do FEM:

- Precisão física elevada – baseado em mecânica dos meios contínuos
- Comportamento independente da resolução da malha
- Permite modelar materiais anisotrópicos (diferentes propriedades em diferentes direções)
- Adequado para simulações científicas e industriais (design de vestuário, airbags)

Desvantagens do FEM:

- Complexidade de implementação significativamente maior
- Custo computacional elevado – requer resolução de sistemas lineares grandes
- Mais difícil de paralelizar eficientemente na GPU
- Overkill para aplicações puramente visuais

Porquê escolhemos Mass-Spring: Para uma demonstração em tempo real com controlo interativo, a complexidade adicional do FEM não se justifica. O *Mass-Spring Model* atinge resultados visualmente convincentes com uma fração do esforço de implementação.

5.3.3 Métodos de Integração: Euler vs Verlet vs RK4

A escolha do método de integração numérica é ortogonal à escolha do modelo físico, mas afeta significativamente a estabilidade e precisão.

Euler Explícito (utilizado neste projeto):

$$\vec{v}_{n+1} = \vec{v}_n + \vec{a}_n \cdot \Delta t, \quad \vec{x}_{n+1} = \vec{x}_n + \vec{v}_{n+1} \cdot \Delta t \tag{5.1}$$

Simples mas requer timesteps pequenos para estabilidade.

Verlet Integration:

$$\vec{x}_{n+1} = 2\vec{x}_n - \vec{x}_{n-1} + \vec{a}_n \cdot \Delta t^2 \tag{5.2}$$

Não armazena velocidade explicitamente, melhor conservação de energia, mais estável para o mesmo timestep. Requer buffer adicional para posições anteriores.

Runge-Kutta 4 (RK4): Avalia a derivada em 4 pontos por timestep, oferecendo precisão de 4ª ordem. Excelente para simulações precisas mas 4× mais caro computacionalmente.

Justificação da escolha: Optámos por Euler pela simplicidade de implementação em GLSL e porque, com $\Delta t = 0.001s$, a estabilidade é garantida. Para trabalho futuro, Verlet seria a melhoria mais imediata.

5.3.4 Tabela Comparativa

Critério	**Mass-Spring**	**PBD**	**FEM**
Complexidade de implementação	Baixa	Média	Alta
Desempenho computacional	Alto	Alto	Baixo
Precisão física	Média	Baixa	Alta
Estabilidade numérica	Média	Alta	Média
Paralelização GPU	Fácil	Fácil	Difícil
Controlo de parâmetros	Intuitivo	Geométrico	Físico
Aplicação típica	Jogos, demos	Jogos, VR	CAD, ciência

Tabela 5.1: Comparação entre métodos de simulação de tecidos

5.3.5 Conclusão da Comparação

O *Mass-Spring Model* representa o melhor compromisso para os objetivos deste projeto: demonstração visual em tempo real com interatividade. Oferece base física suficiente para comportamento realista, permite implementação eficiente em *compute shaders*, e os parâmetros são intuitivos de ajustar.

Para aplicações que exijam maior precisão física, FEM seria preferível. Para jogos com muitos objetos deformáveis simultâneos, PBD poderia ser mais adequado pela sua estabilidade incondicional.

6 Conclusão

Este projeto demonstra com sucesso a implementação de uma simulação de tecido em tempo real utilizando o modelo *Mass-Spring*. A combinação de *compute shaders* para física na GPU com uma interface interativa permite visualizar e experimentar o comportamento do tecido sob diferentes condições.

Os objetivos principais foram alcançados: o tecido reage realisticamente à gravidade, interage corretamente com a esfera de colisão, e mantém-se estável ao longo do tempo graças ao amortecimento implementado.

Como trabalho futuro, seria interessante explorar:

- Métodos de integração mais avançados (Verlet, RK4)
- Aumento da resolução da malha com otimizações de desempenho
- Adição de forças externas como vento
- Técnicas de *constraint solving* para maior realismo

Bibliografia

[1] Fisher, M. (2014). *Cloth Simulation*. Stanford University. `https://graphics.stanford.edu/~mdfisher/cloth.html`

[2] Provot, X. (1995). *Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior*. Graphics Interface.

[3] Baraff, D. & Witkin, A. (1998). *Large Steps in Cloth Simulation*. SIGGRAPH '98.

[4] Müller, M., Heidelberger, B., Hennix, M. & Ratcliff, J. (2007). *Position Based Dynamics*. Journal of Visual Communication and Image Representation.