Um sistema simplificado para animação física usando malhas tetraedrais

Guina Sotomayor Alzamora Claudio Esperança
University Federal of Rio of Janeiro
Computer Graphics Laboratory
Rio de Janeiro - Brasil
gsa,esperanc@lcg.ufrj.br

Abstract

We present a simplified approach for animation of geometrically complex deformable objects represented as tetrahedral meshes. Our prototype system detects and responds to collisions of objects subject to elastic deformations of variable stiffness. The proposed approach combines several techniques, namely, collision detection using a Spatial Hashing, collision response through a contact surface that use a consistent penetration depth using propagation, an estimate for displacement vector of the deformation region and binary search to separate objects. The dynamics is based on shape matching and a modal analysis scheme, using an Euler explicit-implicit integrator. Preliminary results show that collisions between objects containing several hundreds tetrahedra can be animated in real-time.

Resumo

Apresenta-se uma abordagem simplificada para animação de objetos deformáveis geometricamente complexos, representados como malhas tetraedrais. O sistema detecta e responde a colisões de objetos sujeitos a deformações elásticas de rigidez variável. A abordagem combina várias técnicas, como a detecção de colisões usando Hashing espacial, resposta às colisões através do cômputo da superfície de contato, que usa o cálculo da profundidade de penetração por propagação, a estimativa dos vetores de deslocamento dos vértices da região de deformação e busca binária para separar os objetos. A dinâmica está baseada no casamento de formas e na análise modal, na integração do sistema é usado um esquema de Euler explícito-implícito. Resultados preliminares mostram a interação entre objetos constituídos de várias centenas de tetraedros em tempo real.

1. Introdução

O que se entende pelo termo "animação baseada em física" (*physically based animation*, em inglês) é um processo computacional que visa obter animação de objetos com plausibilidade física. Isto contrasta com o termo "animação física", empregado usualmente para designar animações que visam replicar processos físicos com alto grau de acurácia, embora por vezes os dois termos sejam usados indistintamente. Este trabalho insere-se mais no contexto do primeiro termo, uma vez que trata de abordagens onde a preocupação com desempenho leva a um tratamento simplificado de determinadas interações entre objetos.

Embora os modelos baseados em física não pretendam reproduzir a realidade, eles tentam produzir movimentos com base nos mesmos princípios físicos. A parte da física que estuda o movimento dos corpos é a dinâmica, que é baseada nas *Leis de Movimento de Newton*.

A animação de objetos rígidos e deformáveis frequentemente se baseia em sistemas de partículas, sendo que a animação de objetos deformáveis deve adotar algum modelo de deformação físico, que permita deformações elásticas ou plásticas. Tais deformações podem ser obtidas usando métodos baseados em malhas como os sistemas massa-mola e elementos finitos ou métodos sem malhas [14, 13, 16]. Mas, objetos deformáveis têm normalmente representação complexa quando comparados com corpos rígidos, já que podem mudar de forma no tempo pela interação com ele mesmo ou com outros agentes no cenário, pesquisas recentes nessa área vêm adaptando diversas técnicas para animação de corpos rígidos.

Este trabalho enfoca o uso de uma metodologia simples para animação de objetos deformáveis, geometricamente complexos, que são representados por malhas tetraedrais. O objetivo é obter uma simulação dinâmica estável, com comportamento físico plausível. Ressalve-se, entretanto, que as propriedades do material não foram modeladas neste

protótipo. O método apresentado combina várias técnicas empregadas para a modelagem de objetos deformáveis, a saber:

- A detecção de colisões usa uma abordagem de volumes limitantes para diminuir o número de colisões potenciais e uma abordagem baseada em um *Hashing* espacial (usando subdivisão espacial) para encontrar as colisões reais [23]).
- A resposta às colisões envolve o cômputo da profundidade de penetração por propagação [7], o cálculo de uma região de deformação e seus vetores de deslocamento [7], e a resolução das colisões assimétricas [8] dos vértices envolvidos na colisão. Finalmente, a separação dos objetos é efetuada usando Busca Binária [19] encontrando a superfície de contato.
- A dinâmica está baseada numa técnica de casamento de formas na qual não é necessário o uso de malhas, embora estas sejam usadas no processamento de colisões e na integração do sistema, é usado um esquema de Euler explícito-implícito [13].

Resultados preliminares mostram a interação entre objetos constituídos de várias centenas de tetraedros em tempo real.

O resto do texto está organizado na seguinte forma: a seguinte seção apresenta os trabalhos relacionados, a seção 3 descreve o pré-processamento necessário, a seção 4 apresenta como a detecção de colisão é realizada em duas etapas, a seção 5 trata o processo de resposta às colisões, a seção 6 descreve o modelo dinâmico usado, a seção 7 apresenta o método de integração usado, a seção 8 mostra os resultados obtidos, e finalmente a seção 9 apresenta as conclusões e trabalhos futuros.

2. Trabalhos relacionados

O processo de detecção de colisão usualmente é dividido em duas etapas, uma etapa que determina grupos de objetos em colisão potencial (*broad phase*, em inglês), e a segunda etapa que executa verificações de colisão exatos entre os objetos em colisão potencial (*narrow* phase, em inglês).

2.1. Detecção de colisão grosseira

Esta etapa tipicamente é realizada usando um esquema de varredura e poda (sweep and prune, em inglês) [5]. Esta técnica mantém uma lista ordenada para cada eixo principal, onde os elementos na lista são as projeções dos objetos nos eixos. Estas listas são atualizadas freqüentemente usando as formas atuais dos objetos. Assim, um par de objetos pode intersectar apenas se os intervalos de suas projeções intersectam nos três eixos.

2.2. Detecção de colisão exata

As técnicas de detecção de colisão são classificadas em três grandes grupos: baseadas em volumes limitantes, baseadas em subdivisão espacial e as baseadas no espaço-imagem. As técnicas que usam hierarquias de volumes limitantes, tem mostrado maior eficiência para detecção de colisão entre objetos rígidos [6, 11] do que para objetos deformáveis [25, 9], já que precisam ser atualizados freqüentemente. Os volumes limitantes permitem aproximar um objeto complexo por outro de geometria mais simples, ajustando o objeto original da melhor forma possível. Assim, testes de colisões são realizados apenas entre pares de objetos cujos volumes limitantes se intersectam. Já as técnicas baseadas em subdivisão espacial são mais adequadas para objetos deformáveis, já que o espaço do mundo é subdividido em células, onde cada célula contem primitivas do objeto que a intersectam. Uma dificuldade é a escolha da estrutura de dados que represente o espaço 3D. Teschner apresentou uma estrutura de dados baseada numa tabela de dispersão (*hash*), que é eficiente em memória e tempo computacional [23, 24]. Em contraste com as técnicas anteriores, as técnicas baseadas no espaço-imagem fazem uso de funcionalidades das placas gráficas (GPUs). Estas técnicas são apropriadas para aplicações interativas já que não requerem de pré-processamentos nem de atualizações de estruturas de dados complexas [24] , mas em geral são restritas a objetos convexos [2, 17].

2.3. Resposta a colisões

Existem dois esquemas para obter a resposta a colisões: os métodos baseados em restrições e os métodos baseados em penalidades.

Os métodos baseados em restrições evitam a interpenetração entre objetos e são particularmente interessantes na dinâmica de corpos rígidos ou semi-rígidos, já que estes fornecem menos graus de liberdade [10, 16]. Já os métodos baseados em penalidades computam uma força de resposta para cada ponto colidido, cujo valor está relacionado com uma medida de interpenetração. Consequentemente, o esforço numérico cresce com a intensidade da penetração [12], o que torna necessária uma computação robusta da profundidade de penetração dos pontos colididos.

Pode-se dizer que os métodos baseados em penalidades são mais rápidos, enquanto que os métodos baseados em restrições são mais robustos, permitindo intervalos de tempo maiores. Entretanto, os sistemas de restrições são mais custosos e inadequados para resolver colisões em tempo real.

2.4. Animação

Na dinâmica, pode-se empregar métodos como [15]: os métodos de elementos finitos (FEM) são usados para discretizar o objeto em um conjunto de células disjuntas (malha), e resolve comportamentos elásticos para cada partícula. Nos sistemas massa-mola [4, 3, 22] os modelos consistem de partículas conectadas entre si por uma rede de molas, onde o alongamento das molas gera forças elásticas em cada massa. São intuitivos e simples de se implementar. Entretanto, eles não são necessariamente exatos porque não são construídos sobre elasticidade contínua. Os métodos sem malhas se originam no FEM, mas não requerem nenhuma informação de conectividade, uma vez que os objetos são tratados como sistemas de partículas [13].

3. Pré-processamento

As estruturas de dados são iniciadas e os parâmetros iniciais da tabela de dispersão (*hash*) são calculados [23] (veja Figura 1):

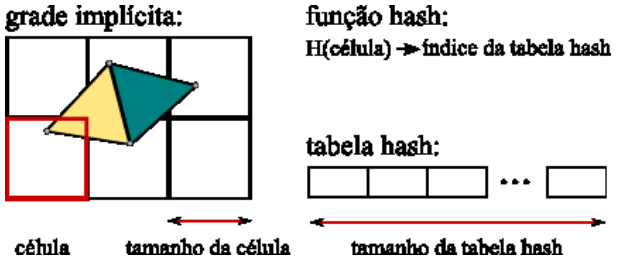


Figura 1. parâmetros da tabela de dispersão (*hash*).

tamanho da tabela: o tamanho ótimo tem relação direta com o número de primitivas a serem avaliadas; tipicamente é um número primo grande e influi no desempenho do algoritmo;

tamanho da célula: influi diretamente no número de primitivas que intersectam uma célula da grade. O sistema usa a média do comprimento das arestas dos tetraedros;

função *"hash"*: serve para encontrar um índice que distribua as células arbitrariamente na tabela de dispersão:

$$h = hash(i, j, k) = (i \cdot \alpha \oplus j \cdot \beta \oplus k \cdot \gamma) \,\%\, n, \quad (1)$$

onde i, j, k são coordenadas do vértice nas coordenadas da grade, α, β e γ são números primos grandes e n é o tamanho da tabela de dispersão.

4. Detecção de colisões

A detecção de colisões é um componente fundamental na simulação física e um problema amplamente pesquisado nas últimas décadas. Seu objetivo não é apenas saber quais objetos colidem, mas também os pontos exatos de colisão e assim permitir computar uma resposta realista à colisão. Em se tratando de objetos deformáveis, problemas como auto-colisões, busca de informação de colisão (profundidade de penetração, pontos de contato, etc.) e eficiência no desempenho têm que ser avaliados. Entre as abordagens para tratar os problemas mencionados temos os métodos de partição do objeto, que empregam hierarquias de volume limitante para estruturar o processo geométrico dos objetos envolvidos em colisões e os métodos de subdivisão espacial, que repartem o espaço de forma explícita ou implícita, a fim de detectar regiões ocupadas por mais de um objeto.

Enquanto que abordagens de subdivisão espacial mapeiam todo o espaço do universo da simulação, o método proposto mapeia somente as regiões onde existem colisões potenciais, evitando assim atualizar desnecessariamente a tabela, para tanto primeiro se usa uma fase de filtragem grosseira (em inglês, *broad phase*), que trata colisões entre os volumes limitantes dos objetos diminuindo o número de colisões, e subsequentemente a fase de filtragem exata, para encontrar as colisões reais nessas regiões, usando o *Hashing* espacial, baseado em subdivisão espacial.

4.1. Filtragem grosseira

O objetivo dessa fase é descartar regiões onde não há colisões. Para tanto, é efetuado um teste simples de colisão entre cada par de objetos, considerando apenas seus volumes limitantes. Para um par de objetos A e B, existe colisão se a distância entre os centros de suas esferas é menor que a soma de seus raios (ver equação). Assim, para cada par de objetos, são identificados os vértices envolvidos na colisão potencial ($v \subset A \cap B$), usando apenas consultas para verificar se um vértice v do objeto A está dentro da esfera do objeto B e vice-versa.

$$(c_1 - c_2)(c_1 - c_2) < (r_1 + r_2)^2 \quad (2)$$

4.2. Uso da subdivisão espacial

Após ter encontrado as regiões de colisão potencial, encontramos as colisões reais, isto é, encontrarmos os vértices que realmente colidem. Para encontrar tais vértices usamos um *Hashing* espacial [23], que subdivide implicitamente o espaço do mundo numa grade uniforme de células regulares, onde cada célula é uma caixa alinhada com os eixos coordenados e contem uma lista das primitivas de objetos que a intersectam, armazenando-as num endereço da

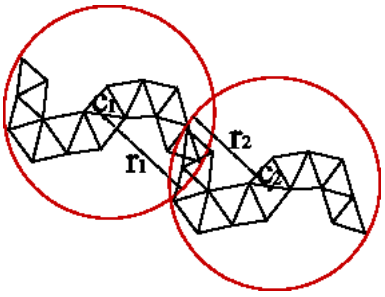


Figura 2. intersecção entre duas esferas.

tabela de dispersão obtido usando a função "*hash*"$= h = hash(i, j, k)$. Cada célula tem três coordenadas inteiras i, j e k que a associam a uma posição na tabela de dispersão. Este esquema atenua o desperdício de memória mantendo acesso rápido às células.

A tabela é atualizada somente nas células onde existe colisão potencial. Também são mapeados os tetraedros e as faces incidentes nesses vértices e inseridos na tabela. Para evitar inserções duplicadas, é armazenado em cada célula um inteiro que identifica em qual quadro da animação esta foi alterada. Este identificador é conhecido como *timestamp*. Assim, cada célula tem uma lista de tetraedros e faces que a intersectam e vice-versa. As células são acessadas por um índice inteiro como mostra a Figura 3.

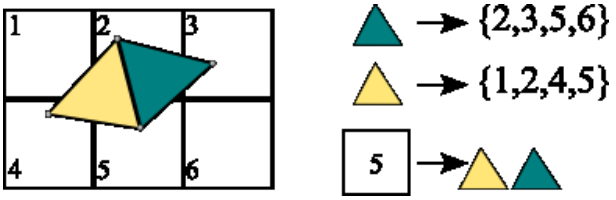


Figura 3. a célula 5 contém as faces verde e amarela, do mesmo modo, estas faces contêm as células $2, 3, 5, 6$ e $1, 2, 4, 5$, respectivamente.

A seguir, a inserção das primitivas:

vértices: cada vértice é armazenado na célula correspondente da forma:

$$(i, j, k) := \lfloor x/l \rfloor, \lfloor y/l \rfloor, \lfloor z/l \rfloor.$$

Logo, uma função *hash* insere a célula (i, j, k) num índice $h = hash(i, j, k)$ da tabela de dispersão como mostra a Figura 4;

tetraedros: para simplificar o armazenamento do tetraedro nas suas células correspondentes é utilizado o *AABB* do tetraedro. Assim, os vértices mínimo e máximo que descrevem o *AABB* representam o intervalo das células que intersectam o *AABB*. Finalmente, todos os índices h encontrados são inseridos na tabela de dispersão, como mostra a Figura 4;

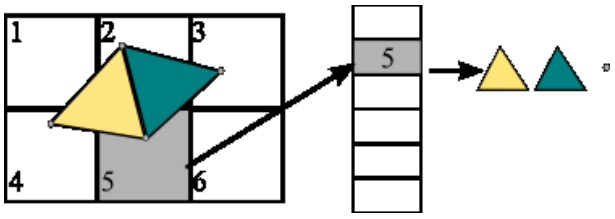


Figura 4. exemplo de mapeamento de primitivas de objetos. As faces amarela e verde são armazenadas na célula 5. Por outra parte, a célula 5 é mapeada num índice arbitrário da tabela de dispersão.

faces: para mapear as faces, segue-se o mesmo esquema usado para inserir tetraedros, usando as *AABB* das faces;

arestas: usa-se uma técnica [1] que permite armazenar somente as células cruzadas por arestas de intersecção, avaliando a vizinhança da aresta na partição do espaço 3D.

A cada quadro da animação, os tetraedros de cada objeto são inseridos nas células que os intersectam, indistintamente do objeto. Numa segunda etapa, um teste de colisão é efetuado para encontrar os *vértices colididos*, que são os vértices que penetram tetraedros de outros objetos ou do mesmo objeto (auto-colisão). Logo, para cada entrada da tabela é executado um teste de intersecção entre os vértices que pertencem à região em colisão potencial e os tetraedros contidos nessa célula, encontrando assim os vértices colididos.

Para cada vértice v é realizado um teste de intersecção com cada tetraedro t da célula. Para acelerar o processo, primeiro se verifica se o vértice está dentro do *AABB* do tetraedro ($v \subset t.AABB()$). Em caso afirmativo, se verifica se está dentro do tetraedro. Se a verificação resulta verdadeira, o vértice é marcado como vértice colidido. Este processo também detecta auto-intersecção, como mostra a Figura 5. A verificação ponto-tetraedro pode ser feita usando coordenadas baricêntricas.

5. Resposta às colisões

O processo de detecção de colisão, descrito na seção anterior, fornece uma lista de vértices colididos. Estes são usados na resposta às colisões. O processo começa com o cálculo da profundidade de penetração dos vértices colididos, seguido do cômputo da região de deformação, a

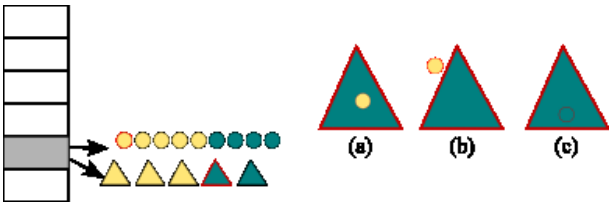


Figura 5. para cada entrada não vazia da tabela de dispersão, testa-se intersecção entre vértices e tetraedros, para verificar se existe: (a) colisão, (b) não colisão ou (c) autocolisão.

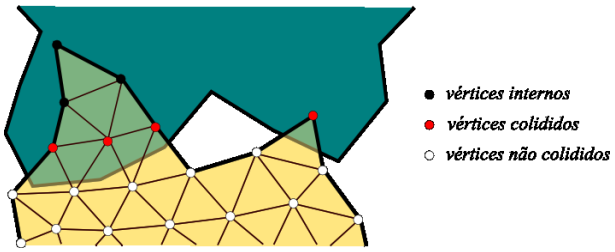


Figura 6. vértices colididos: da borda ou internos.

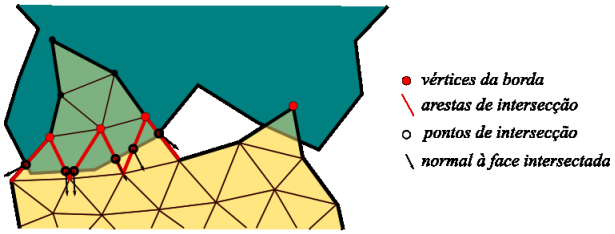


Figura 7. vértices da borda, arestas de intersecção, pontos exatos de intersecção e normais às faces intersectadas.

qual contém todos os vértices envolvidos na colisão. Finalmente, os objetos são separados usando uma técnica de Busca Binária.

5.1. Profundidade de penetração

A profundidade de penetração entre dois objetos é a translação mínima necessária para separá-los.

O *Algoritmo baseado em malhas tetraedrais* [7] computa a profundidade e a direção de penetração para cada vértice colidido fazendo uso das malhas tetraedrais. Tal profundidade pode ser usada para computar forças de penalidade que forneçam respostas realistas às colisões.

A idéia é classificar os vértices colididos em relação a sua profundidade de penetração. Assim, primeiro são avaliados os vértices mais próximos à superfície. Essa informação é então propagada aos vértices que possuem maior penetração, até que todos os vértices colididos sejam processados. No final, cada vértice colidido possui uma profundidade de penetração d e um vetor de direção de penetração $\vec{r}$.

Finalmente, um triângulo de contato é calculado para cada vértice colidido. Tal triângulo de contato é uma face na superfície do objeto penetrado que intersecta com o vetor de direção de profundidade de penetração.

Classificação: se um vértice colidido tem um ou mais vértices incidentes não colididos este é um *vértice da borda*, caso contrário é um *vértice interno* (Figura 6).

Profundidade de penetração dos vértices da borda: primeiro são identificadas as *arestas de intersecção*, ou seja, arestas que possuem um vértice da borda e um vértice não colidido. A seguir, procura-se a face da superfície mais próxima que intersecta a aresta. O teste pode ser feito usando coordenadas baricêntricas, já que permitem computar a normal da face intersectada n_{srf} e obter o ponto exato de intersecção p_{int}. Estes dados são armazenados na aresta e servem para encontrar a profundidade de penetração dos vértices da borda (Figura 7). Para cada vértice da borda v, é computada sua profundidade de penetração $d(v)$ e seu vetor de direção de penetração $\vec{r}(v)$ da seguinte forma: para cada aresta de intersecção incidente em v, computa-se um peso ponderado w entre o ponto de intersecção da aresta p_{int} e v:

$$w(p_{int}, v) = \frac{1}{\|p_{int} - v\|^2}, \tag{3}$$

logo, são computados $d(v)$ e $\vec{r}(v)$ através das fórmulas:

$$d(v) = \frac{\sum_{i=1}^{k} (w(p_i, v) \cdot (p_i - v) \cdot n_i)}{\sum_{i=1}^{k} (p_i, v)}, \tag{4}$$

$$\vec{r}(v) = \frac{\sum_{i=1}^{k} (w(p_i, v) \cdot n_i)}{\sum_{i=1}^{k} (p_i, v)}, \tag{5}$$

onde k é o número de arestas de intersecção incidentes em v, p_i e n_i representam o i-ésimo p_{int} e n_{srf} a serem avaliados. A Figura 8 ilustra a profundidade de penetração para os vértices da borda v. No final, todos os vértices da borda são inseridos numa lista de vértices processados.

Profundidade de penetração dos vértices internos: após o processamento de todos os vértices da borda, a informação da profundidade de penetração obtida é propagada para os vértices internos v_i (Figura 9).

Encontra-se uma lista de vértices a processar, isto é, dada uma lista de vértices processados, é obtida uma lista de vértices a processar. Desta forma, vê-se que o cômputo da

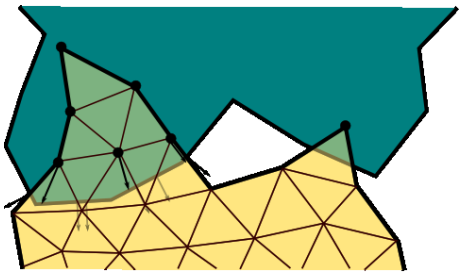


Figura 8. profundidade de penetração de vértices da borda.

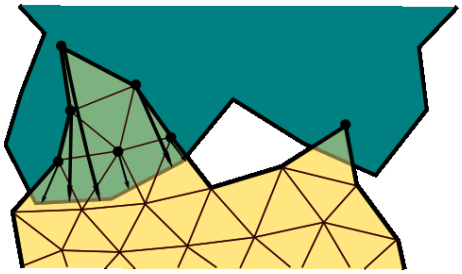


Figura 9. profundidade de penetração de vértices internos usando propagação.

profundidade de penetração é feito por níveis, até que todos os vértices internos sejam computados.

Computa-se a profundidade de penetração $d(v)$ e a direção de penetração $\vec{r}(v)$ dos vértices internos v, enquanto existam vértices internos a serem processados. Primeiro, para cada vértice processado v_{inc} incidente no vértice v, computa-se um peso ponderado μ:

$$\mu(v_{inc}, v) = \frac{1}{\|v_{inc} - v\|^2},$$

depois, computa-se $d(v)$ e $\vec{r}(v)$ segundo as fórmulas:

$$d(v) = \frac{\sum_{j=1}^{k} (\mu(v_j, v) \cdot ((v_j - v) \cdot r(v_j) + d(v_j)))}{\sum_{j=1}^{k} \mu(v_j, v)}, \tag{6}$$

$$\vec{r}(v) = \frac{\sum_{j=1}^{k} \mu_j r(v_j)}{\sum_{j=1}^{k} \mu_j}, \tag{7}$$

onde k é o número de vértices incidentes no vértice v, e v_j e o j-ésimo v_{inc} avaliado.

5.2. Região de deformação

5.2.1. Resolução de colisões assimétricas Após o cálculo da profundidade de penetração, os objetos devem ser separados, a fim de fornecer um estado fisicamente correto. Para tal encontra-se uma *região de deformação* dos vértices envolvidos na colisão. Esta região de deformação considera os vértices colididos e os vértices dos triângulos de contato destes, que não necessariamente colidem (x_i, x_j e x_k na Figura 10, colisões com essa configuração são chamadas de *colisões assimétricas*). Para todos os vértices da região de deformação é computado um vetor de deslocamento, que será usado no processo de separação. Nesta etapa também são resolvidas as colisões assimétricas, usando o método de projeção de Jakobsen [8]. Este método projeta os vértices colididos para fora do obstáculo, movendo-os até que fiquem livres da intersecção.

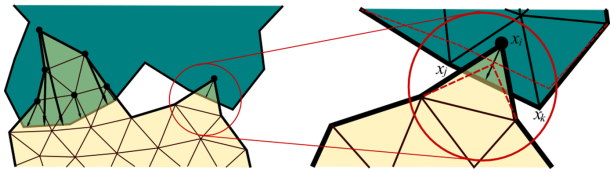


Figura 10. solução de colisões assimétricas.

Por exemplo, na Figura 10, o vetor de deslocamento de x_i é sua profundidade de penetração, e os vetores de deslocamento dos vértices não colididos x_j e x_k são calculados da forma:

$$\vec{s_j} = \frac{\alpha_1}{\alpha_1^2 + \alpha_2^2}(x_i' - x_i),$$

$$\vec{s_k} = \frac{\alpha_2}{\alpha_1^2 + \alpha_2^2}(x_i' - x_i),$$

onde x_i é o vértice colidido, x_i' sua projeção na aresta de contato (x_j e x_k), s_j e s_k são os vetores de deslocamento de x_j e x_k, respectivamente, os valores α representam a proporção de deslocamento dos vértices da aresta de contato, sendo que $\alpha_1 + \alpha_2 = 1$.

5.2.2. Busca binária para computar a superfície de contato Após ter computado os vetores de deslocamento, é estimada uma superfície de contato implícita que permite a separação dos objetos. Este deslocamento deve ser algo em torno da metade do comprimento dos vetores de deslocamento, para tal pode ser usado um esquema de busca binária [19]:

$$x^{i+1} \leftarrow x^i \pm \frac{1}{2^{(i+2)}} s, \tag{8}$$

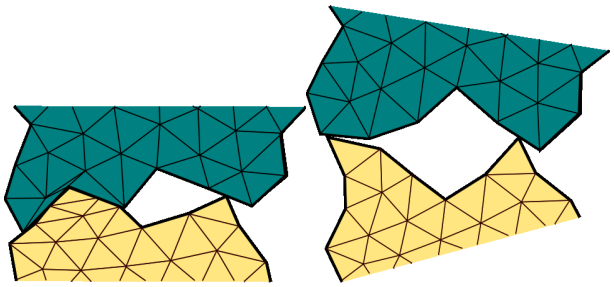
onde s é o vetor de deslocamento, x é a posição do vértice de deslocamento original. Em outras palavras, para cada iteração, o intervalo de deslocamento é dividido por dois.

6. Dinâmica do sistema

Na animação dos objetos é aplicado o método de casamento de formas usando mínimos quadrados [13], onde

os nós da malha são tratados como partículas e animados como um sistema de partículas simples, sem conectividade. Esta técnica permite encontrar uma correspondência entre os vértices no estado inicial $\mathbf{x}_i^0$ e estado atual $\mathbf{x}_i$. Ou seja, a correspondência com o menor erro quadrático.

O objetivo é encontrar a melhor matriz de transformação afim, que permita efetuar deformações plausíveis e que permita que o objeto volte à sua forma original (Figura 11).



(a) (b)

Figura 11. (a)superfície de contato, objeto deformado e (b) a possível solução da deformação.

O algoritmo tem dois componentes principais: (1) encontrar uma transformação rígida ótima que aproxime uma nova posição e a orientação do objeto (problema de correspondência) e (2) mover as partículas para as posições alvo aplicando um modelo de deformação linear.

Considerando os pesos das partículas, uma transformação linear composta de uma translação $\mathbf{t}$ e uma rotação $\mathbf{R}$ sobre o ponto $\mathbf{t}_0$ pode ser encontrada minimizando:

$$\sum_i w_i(\mathbf{R}(\mathbf{x}_i^0 - \mathbf{t}_0) + \mathbf{t} - \mathbf{x}_i)^2.$$

Podemos estabelecer para o problema $w_i = m_i$, isto é, que a ponderação das partículas é representada por suas massas, e que os vetores de translação ótimos são o *centro de massa* da forma inicial e o centro de massa da forma atual. Assim, temos

$$\mathbf{t}_0 = \mathbf{x}_{cm}^0 = \frac{\sum_i m_i \mathbf{x}_i^0}{\sum_i m_i}, \qquad e \qquad \mathbf{t} = \mathbf{x}_{cm} = \frac{\sum_i m_i \mathbf{x}_i}{\sum_i m_i}.$$

Para encontrar um $\mathbf{R}$ ótimo é necessário encontrar uma matriz de transformação linear $\mathbf{A}$. Observe que $\mathbf{A}$ não necessariamente é orto-normal. Para encontrar $\mathbf{A}$ define-se as posições relativas

$$\mathbf{q}_i = \mathbf{x}_i^0 - \mathbf{x}_{cm}^0 \qquad e \qquad \mathbf{p}_i = \mathbf{x}_i - \mathbf{x}_{cm}.$$

Estas posições definem o campo de deformação das partículas, tratando deformações elásticas de forma implícita.

O problema de correspondência é resolvido com a técnica de mínimos quadrados, que é um método clássico usado para estabelecer a melhor correspondência com o menor erro quadrático. Assim, a matriz de transformação linear ótima $\mathbf{A}$ é encontrada minimizando o termo:

$$\sum_i m_i(\mathbf{A}\mathbf{q}_i - \mathbf{p}_i)^2.$$

Onde:

$$\mathbf{A} = (\sum m_i \mathbf{p}_i \mathbf{q}_i^T)(\sum m_i \mathbf{q}_i \mathbf{q}_i^T)^{-1} = \mathbf{A}_{pq}\mathbf{A}_{qq}, \qquad (9)$$

na equação $\mathbf{A}_{pq}$ é uma matriz de correlação e $\mathbf{A}_{qq}$ é uma matriz simétrica que pode conter escala mas não rotações. Conclui-se portanto que a estimativa de $\mathbf{A}$ requer que se estime o componente de rotação da matriz $\mathbf{A}_{pq}$, o que pode ser realizado através de algum método de decomposição. Em particular, o trabalho de Müller et al. [13] emprega decomposição polar de matrizes [18] e o algoritmo Jacobi [20], que é um algoritmo simples e estável, usado para a diagonalização de matrizes e o cômputo de auto-vetores. O funcionamento do algoritmo se caracteriza pela aplicação de sucessivas operações elementares chamadas de *Rotações de Jacobi*. Geralmente são realizadas de 5 a 10 *Rotações de Jacobi* para diagonalizar uma matriz 4×4.

Assim, a matriz de rotação ótima é:

$$\mathbf{R} = \mathbf{A}_{pq}(\sqrt{\mathbf{A}_{pq}^T \mathbf{A}_{pq}})^{-1}.$$

Finalmente, havendo encontrado a matriz de rotação é realizada a seguinte transformação linear para rotação e translação do objeto, a fim de encontrar a melhor posição alvo de cada partícula:

$$\mathbf{g}_i = \mathbf{R}(\mathbf{x}_i^0 - \mathbf{x}_{cm}^0) + \mathbf{x}_{cm}.$$

Os pontos são movidos para as posições $\mathbf{g}_i$, exatamente a cada intervalo de tempo.

7. Integração física

Nosso esquema deriva de um método de integração de Euler, que inclui uma parte explícita (a atualização da velocidade) e uma parte implícita (a atualização da posição).

Para computar as posições dos objetos, as acelerações e velocidades são integradas numericamente para cada intervalo de tempo utilizando as seguintes equações:

$$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \alpha \frac{\mathbf{g}_i(t) - \mathbf{x}_i(t)}{\Delta t} + \frac{\Delta t}{m_i} f_{ext}(t), \quad (10)$$

$$\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t), \qquad (11)$$

onde $\mathbf{v}_i(t + \Delta t)$ é a velocidade no próximo intervalo de tempo, $\mathbf{v}_i(t)$ é a velocidade no intervalo de tempo atual, α

é o parâmetro que simula rigidez, $\mathbf{g}_i(t)$ é a melhor posição do vértice $\mathbf{x}_i(t)$ (para resolver a deformação), Δt é a magnitude do intervalo de tempo, m_i é a massa da partícula e $\mathbf{f}_{ext}(t)$ são as forças externas.

8. Resultados

O protótipo destinado a servir como prova de conceito para o sistema foi desenvolvido utilizando a linguagem C++, a API OpenGL e a biblioteca Glut. Uma observação importante é o uso do Vertex Buffer Object (VBO), extensão do OpenGL, para a obtenção de uma boa taxa de renderização dos objetos. Para avaliar o protótipo desenvolvido foram conduzidos experimentos envolvendo objetos deformáveis de formas variadas. Todos os experimentos foram executados numa estação de trabalho com sistema operacional Linux (Fedora 8) com processador Intel Core 2 Duo a 2.4Ghz e 1 GB de memória.

Os experimentos visam avaliar o desempenho do sistema em função do número de objetos envolvidos na simulação e da complexidade desses objetos. A Tabela 1 mostra as resoluções em vértices, faces e tetraedros dos objetos usados.

Objeto	Vértices na superfície	vértices	faces	tetraedros
coelho	436	510	868	1750
pato	424	519	846	1819
tubo	220	340	436	1270
esfera	386	729	768	2560

Tabela 1. objetos com resoluções diferentes.

Todas as simulações usaram coeficiente de rigidez $\alpha = 0.8$ nos objetos (quase rígidos).

Foram coletados alguns indicadores em termos de quadros por segundo, quantidade de primitivas em colisão (vértices em colisão potencial, faces, tetraedros e vértices em colisão real) processadas a cada intervalo de tempo e a porcentagem gasta por cada sub-processo (detecção de colisões potenciais, detecção exata de colisões, profundidade de penetração, casamento de formas) em milisegundos.

Um primeiro experimento consiste na simulação física de uma cena com diferentes tipos de objetos (veja a Figura 12): 3 patos, 2 coelhos e 3 esferas (Tabela 1). Este resultado mostra que o protótipo lida com objetos de geometria arbitrária desde que estejam triangulados adequadamente.

Do experimento podemos obter algumas informações, por exemplo, a Figura 13 mostra o tempo gasto em milisegundos nos sub-processos mais importantes: detecção de colisões potenciais, detecção exata de colisões, computação

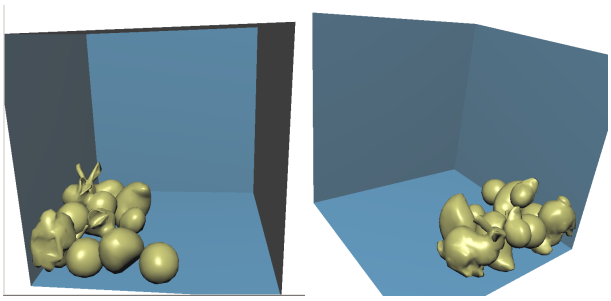


Figura 12. oito objetos em contato: 3 patos, 2 coelhos e 3 esferas. A cena contém 2952 vértices e 9917 tetraedros animados a 32 fps.

da profundidade de penetração e casamento de formas a cada intervalo de tempo. Pode-se ver que a detecção de colisões, nas duas fases, toma a maior parte do tempo, sendo que o cálculo da profundidade de penetração se mantém quase constante e o casamento de formas, usando todos os vértices da superfície dos objetos, toma um tempo insignificante em relação aos outros sub-processos.

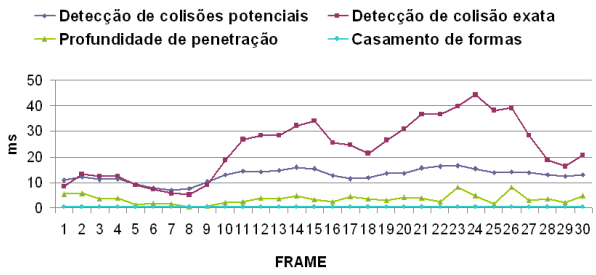


Figura 13. tempo gasto em milisegundos para cada sub-processo a cada intervalo de tempo.

A Figura 14 mostra, para o mesmo experimento, a quantidade de primitivas em colisão (vértices em colisão potencial, faces, tetraedros e vértices em colisão) por intervalo de tempo. Note que a filtragem grosseira detecta uma quantidade de vértices em colisão potencial significativamente inferior ao total de vértices na cena. Em média, a simulação registra 487 vértices em colisão potencial, sendo que as colisões reais envolvem 37 faces, 82 tetraedros e 28 vértices em média.

Foram também conduzidos outros experimentos usando quantidades variáveis de esferas de forma a avaliar a escalabilidade do método. A Figura 15, mostra a simulação de

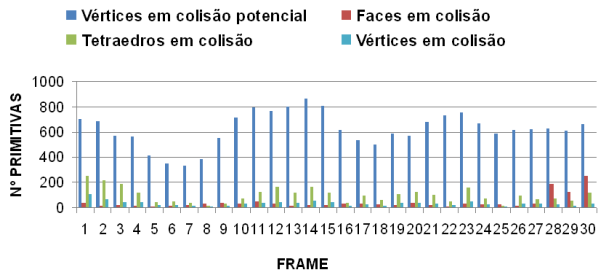


Figura 14. número de primitivas em colisão, a cada intervalo de tempo.

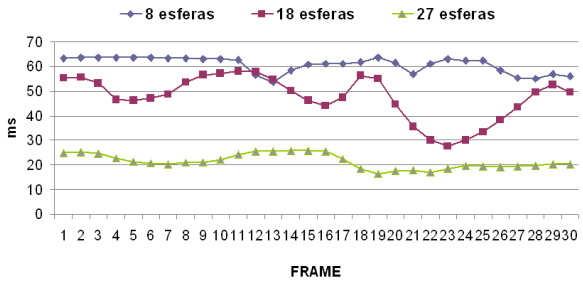


Figura 16. tempo gasto em milisegundos a cada intervalo de tempo de esferas com 8, 18 e 27 objetos.

cenas com 8,18 e 27 esferas (Tabela 1). As cenas contêm 5832 vértices e 20480 tetraedros (8 esferas), 13122 vértices e 46080 tetraedros (18 esferas) e 19683 vértices e 69120 tetraedros (27 esferas). O desempenho do método para esses experimentos foi de 62, 41 e 22 quadros por segundo em média, respectivamente. Um gráfico com a taxa de quadros por segundo a cada intervalo de tempo é mostrado na Figura 16.

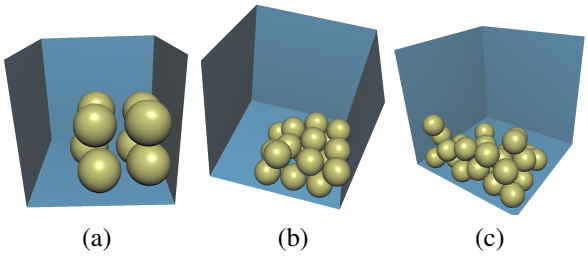


Figura 15. experimentos com 8 (a), 18 (b) e 27 (c) esferas.

9. Conclusões e Trabalhos Futuros

Foi apresentado um protótipo de sistema de animação baseada em física, que usa várias técnicas, como a detecção de colisões usando um *Hashing* espacial [21] e volumes limitantes, a resposta às colisões através do cômputo da superfície de contato, que utiliza o cálculo da profundidade de penetração por propagação [7], a estimativa dos vetores de deslocamento [7] e a resolução das colisões assimétricas [8] dos vértices da região de deformação, e Busca Binária para separar os objetos [19], a animação é feita usando uma técnica de casamento de formas e um integrador de Euler explícito-implícito [13].

A estrutura original de detecção de colisões foi estendida usando-se um mecanismo de detecção de regiões de colisão potencial, para minimizar as regiões atualizadas pela tabela de dispersão, onde se efetuam as colisões reais. Também, no esquema de resposta a colisões, é usado o método de projeção de Jakobsen [8] para resolver colisões assimétricas.

A detecção e resposta às colisões permitem identificar e resolver colisões dos objetos baseados em malhas tetraedrais, em objetos rígidos e deformáveis.

O sistema oferece simulações plausíveis, produzindo respostas às colisões de forma convincente e próxima às metodologias tradicionais, conseguindo manipular objetos complexos e objetos empilhados.

Para trabalhos futuros, planeja-se estender o protótipo do sistema apresentado, na animação, da deformação (agora linear) à quadrática, e plástica; na detecção de colisões o uso de uma hierarquia de esferas limitantes, e finalmente, para aperfeiçoar o código e alcançar um melhor desempenho, desenvolver o sistema usando *GPU*.

Referências

[1] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *Eurographics '87*, pages 3–10. Elsevier Science Publishers, Amsterdam, North-Holland, 1987.

[2] G. Baciu, W. Wong, and H. Sun. Recode: An image-based collision detection algorithm. *The Journal of visualization and Computer Animation*, 10(4):181–192, 1998.

[3] D. Baraff and A. Witkin. Large steps in cloth simulation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA, 1998. ACM.

[4] D. Breen, D. House, and M. Wozny. Predicting the drape of woven cloth using interacting particles. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Compu-*

ter graphics and interactive techniques, pages 365–372, New York, NY, USA, 1994. ACM.

[5] J. D. Cohen, M. C. Lin, D. Manocha, and M. Ponamgi. I-collide: an interactive and exact collision detection system for large-scale environments. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 189–ff., New York, NY, USA, 1995. ACM.

[6] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: a hierarchical structure for rapid interference detection. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, New York, NY, USA, 1996. ACM.

[7] B. Heidelberger, M. Teschner, R. Keiser, and M. Müller. Consistent penetration depth estimation for deformable collision response. In *Proceedings of Vision, Modeling, Visualization VMV'04*, pages 157–164, Stanford, USA, 2004.

[8] T. Jakobsen. Advanced character physics. In *Proceedings, Game Developer's Conference 2001*, SJ, USA, 2001. GDC Press.

[9] D. L. James and D. K. Pai. BD-Tree: Output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23(3), Aug. 2004.

[10] D. Kaufman, T. Edmunds, and D. Pai. Fast frictional dynamics for rigid bodies. *ACM Trans. Graph.*, 24(3):946–956, 2005.

[11] J. T. Klosowski, M. Held, J. S. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 04(1):21–36, 1998.

[12] M. Moore and J. Wilhelms. Collision detection and response for computer animationr3. *SIGGRAPH Comput. Graph.*, 22(4):289–298, 1988.

[13] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. In *Proceedings of SIGGRAPH'05*, pages 471–478, New York, NY, USA, 2005.

[14] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 141–151, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[15] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson. Physically based deformable models in computer graphics. In *Computer Graphics Forum*, pages 809–836. Blackwell Publishing Ltd., 2006.

[16] M. Pauly, D. Pai, and L. Guibas. Quasi-rigid objects in contact. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 109–119, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[17] M. Shinya and M.-C. Forgue. Interference detection through rasterization. In *The Journal of Visualization and Computer Animation*, volume 2, pages 132–134, 1991.

[18] K. Shoemake and T. Duff. Matrix animation and polar decomposition. In *Proceedings of the conference on Graphics interface '92*, pages 258–264, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[19] J. Spillmann and M. Teschner. Contact surface computation for coarsely sampled deformable objects. In *Proceedings of Vision, Modeling, Visualization VMV'05*, pages 16–18, Stanford, USA, 2005.

[20] H. Sumali. *A New Adaptive Array of Vibration Sensors*. PhD thesis, Mechanical Engineering Virginia Polytechnic Institute and State University, Virginia, USA, 1992.

[21] M. Teschner, B. Heidelberger, D. Manocha, N. Govindaraju, G. Zachmann, S. Kimmerle, J. Mezger, and A. Fuhrmann. Collision handling in dynamic simulation environments. In *Eurographics Tutorial # 2*, pages 1–4, Dublin, Ireland, 29 August 2005. Eurographics Association.

[22] M. Teschner, B. Heidelberger, M. Müller, and M. Gross. A versatile and robust model for geometrically complex deformable solids. In *CGI '04: Proceedings of the Computer Graphics International (CGI'04)*, pages 312–319, Washington, DC, USA, 2004. IEEE Computer Society.

[23] M. Teschner, B. Heidelberger, M. Müller, and D. Pomeranets. Optimized spatial hashing for collision detection of deformable objects. In *Proceedings of Vision, Modeling, Visualization VMV'03Proceedings of SPM 2005*, pages 47–54, 2003.

[24] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnetat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. In *Computer Graphics Forum*, pages 61–81. Eurographics Association, Eurographics Association and Blackwell Publishing, 2005.

[25] G. van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools*, 2(4):1–13, 1997.