UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Integração de Sistemas de Partículas com Detecção de Colisões em Ambientes de Ray Tracing

por

MAURO STEIGLEDER

Dissertação submetida à avaliação, como requisito parcial para
a obtenção do grau de Mestre em
Ciência da Computação

Prof. Anatólio Laschuk
Orientador

Porto Alegre, agosto de 1997.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Steigleder, Mauro

Integração de sistemas de partículas com detecção de colisão em ambientes de *Ray Tracing* / por Mauro Steigleder. - Porto Alegre: CPGCC da UFRGS, 1997.

103f.: il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul. Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, BR-RS, 1997. Orientador: Laschuk, Anatólio.

1.Sistemas de Partículas. 2.Computação Gráfica. 3.Foto-realismo. 4.Técnicas de Aceleração. I. Laschuk, Anatólio. II. Título.

❝ No fim, tudo dá certo.
Se não deu, é porque
ainda não chegou ao fim. ❞

Fernando Sabino

Agradecimentos

Ao meu orientador, Anatólio Laschuk, cujas sugestões foram de grande valia para a elaboração deste trabalho.

À professora Carla Maria Dal Sasso Freitas, pelas conversas e pelo apoio dado.

Ao meu grande amigo, Christian Hofsetz, que desde a graduação tem me ajudado na obtenção de melhores resultados, seja qual for o trabalho.

Ao meu amigo Franz Figueroa pelas sugestões e por todo apoio dado.

Aos meus amigos e professores Marcelo Walter e Fernando Osório, por terem me introduzido ao fantástico mundo da Computação Gráfica.

Aos demais colegas, professores e funcionários que de alguma forma ajudaram na conclusão deste trabalho.

Ao CNPq pelo auxílio financeiro, sem o qual não haveria a possibilidade da conclusão deste trabalho.

E principalmente aos meus pais, Geraldo e Laura Steigleder, que sempre me apoiaram em todos os meus sonhos e desejos.

Sumário

Lista de Figuras

Resumo

Encontrar um modo de criar imagens fotorealísticas tem sido uma meta da Computação Gráfica por muitos anos [GLA 89]. Neste sentido, os aspectos que possuem principal importância são a modelagem e a iluminação.

Ao considerar aspectos de modelagem, a obtenção de realismo mostra-se bastante difícil quando se pretende, através de técnicas tradicionais de modelagem, modelar objetos cujas formas não são bem definidas. Dentre alguns exemplos destes tipos de objetos, podem-se citar fogo, fumaça, nuvens, água, etc. Partindo deste fato, Reeves [REE 83] introduziu uma técnica denominada sistemas de partículas para efetuar a modelagem de fogo e explosões.

Um sistema de partículas pode ser visto como um conjunto de partículas que evoluem ao longo do tempo. Os procedimentos envolvidos na animação de um sistema de partículas são bastante simples. Basicamente, a cada instante de tempo, novas partículas são geradas, os atributos das partículas antigas são alterados, ou estas partículas podem ser extintas de acordo com certas regras pré-definidas.

Como as partículas de um sistema são entidades dinâmicas, os sistemas de partículas são especialmente adequados para o uso em animação. Ainda, dentre as principais vantagens dos sistemas de partículas quando comparados com as técnicas tradicionais de modelagem, podem-se citar a facilidade da obtenção de efeitos sobre as partículas (como borrão de movimento), a necessidade de poucos dados para a modelagem global do fenômeno, o controle por processos estocásticos, o nível de detalhamento ajustável e a possibilidade de grande controle sobre as suas deformações.

Entretanto, os sistemas de partículas possuem algumas limitações e restrições que provocaram o pouco desenvolvimento de algoritmos específicos nesta área. Dentre estas limitações, as principais são a dificuldade de obtenção de efeitos realísticos de sombra e reflexão, o alto consumo de memória e o fato dos sistemas de partículas possuírem um processo de animação específico para cada efeito que se quer modelar. Poucos trabalhos foram desenvolvidos especificamente para a solução destes problemas, sendo que a maioria se destina à modelagem de fenômenos através de sistemas de partículas.

Tendo em vista tais deficiências, este trabalho apresenta métodos para as soluções destes problemas. É apresentado um método para tornar viável a integração de sistemas de partículas em ambientes de *Ray Tracing*, através do uso de uma grade tridimensional. Também, são apresentadas técnicas para a eliminação de efeitos de *aliasing* das partículas, assim como para a redução da quantidades de memória exigida para o armazenamento dos sistemas de partículas.

Considerando aspectos de animação de sistemas de partículas, também é apresentado uma técnica de aceleração para a detecção de colisões entre o sistema de partículas e os objetos de uma cena, baseada no uso de uma grade pentadimensional.

Aspectos relativos à implementação, tempo de processamento e fatores de aceleração são apresentados no final do trabalho, assim como as possíveis extensões futuras e trabalhos sendo realizados.

Palavras-chaves: Sistemas de partículas, fotorealismo, síntese de imagens, técnicas de aceleração.

TITLE: "INTEGRATION OF PARTICLE SYSTEMS WITH COLISION DETECTION IN RAY TRACING ENVIRONMENTS"

Abstract

Finding a way to create photorealistic images has been a goal of Computer Graphics for many years [GLA 89]. In this sense, the aspects that have main importance are modeling and illumination.

Considering aspects of modeling, the obtention of realism is very difficult when it is intended to model fuzzy objects using traditional modeling techniques. Among some examples of these types of objects, fire, smoke, clouds, water, etc. can be mentioned. With this fact in mind, Reeves [REE 83] introduced a technique named particle systems for modeling of fire and explosions.

A particle system can be seen as a set of particles that evolves over time. The procedures involved in the animation of particle systems are very simple. Basically, at each time instant, new particles are generated, the attributes of the old ones are changed, or these particles can be extinguished according to predefined rules.

As the particles of a system are dynamic entities, particle systems are specially suitable for use in animation. Among the main advantages of particle systems, when compared to traditional techniques, it can be mentioned the facility of obtaining effects such as motion blur over the particles, the need of few data to the global modeling of a phenomen, the control by stochastic processes, an adjustable level of detail and a great control over their deformations.

However, particle systems present some limitations and restrictions that cause the little development of specific algorithms in this area. Among this limitations, the main are the difficulty of obtention of realistic effects of shadow and reflection, the high requirement of memory and the fact that particle systems need a specific animation process for each effect intended to be modeled. Few works have been developed specifically for the solution of these problems; most of them are developed for the modeling of phenomena through the use of particle systems.

Keeping these deficiencies in mind, this work presents methods for solving these problems. A method is presented to make practicable the integration of particle systems and ray tracing, through the use of a third-dimensional grid. Also, a technique is presented to eliminate effects of aliasing of particles, and to reduce the amount of memory required for the storage of particle systems.

Considering particle systems animation, a technique is also presented to accelerate the collision detection between particle systems and the objects of a scene, based on the use of a fifth-dimensional grid.

Aspects related to the implementation, processing time and acceleration factors are presented at the end of the work, as well as the possible future extensions and ongoing works.

Keywords: Particle systems, photorealism, image synthesis, acceleration techniques.

1 Introdução

Segundo Amanatides [AMA 87], um dos principais e mais importantes objetivos a serem alcançados pela computação gráfica é criar imagens com o maior grau de realismo possível. Ainda, de acordo com Foley [FOL 90], o realismo é um requisito fundamental em áreas como simulação, CAD/CAM, entretenimento, etc. Contudo, a obtenção do realismo na criação de objetos cujas fronteiras são irregulares ou não são bem definidas é extremamente difícil e dispendiosa através de técnica convencionais de modelagem. Dentre os objetos que podem ser classificados neste grupo encontram-se boa parte dos fenômenos naturais, tais como, fogo, água, fumaça, neblina, explosões, etc.

Tendo tal fato em mente, Reeves [REE 83] introduziu uma técnica de modelagem conhecida por sistema de partículas, onde um objeto é modelado através de um conjunto de partículas que definem o seu volume, ao contrário das técnicas convencionais de modelagem que definem a fronteira do objeto. A primitiva básica de um sistema de partículas é denominada partícula e pode ser definida como uma primitiva simples (normalmente pontos tridimensionais) que possui propriedades e atributos comuns. Estas primitivas básicas ou partículas são dinâmicas, ou seja, são dependentes de um parâmetro temporal, de modo que podem modelar não somente a estrutura do objeto, mas também a mudança da sua forma ao longo do tempo. Além disto, partículas são normalmente controladas por processos estocásticos, de modo que a sua forma não é totalmente especificada em determinado instante de tempo.

Devido a estas características, sistemas de partículas possuem um grande potencial para uso em animações e modelagem de fenômenos naturais diversos. Porém, apesar do seu uso recente na modelagem de uma grande variedade de fenômenos naturais, pouco desenvolvimento tem sido realizado quanto aos problemas envolvendo aspectos como síntese e animação de sistemas de partículas com grandes quantidades de partículas. Dentre tais problemas, encontram-se o tempo de processamento, a quantidade de memória exigida, o uso de sombras e reflexões e a qualidade da imagem gerada.

1.1 Motivação e Objetivos

Apesar da modelagem através de sistemas de partículas apresentar desenvolvimentos significativos ultimamente, os desenvolvimentos restringem-se à utilização de sistemas de partículas para modelar um determinado tipo de fenômeno, sendo que pouco desenvolvimento foi realizado com o intuito de solucionar ou pelo menos amenizar os problemas inerentes à modelagem através de sistemas de partículas.

Este trabalho pretende apresentar uma explanação sobre sistemas de partículas, discorrer sobre as suas principais características, aplicações e problemas, assim como descrever os principais fenômenos modelados através de sistemas de partículas. Com base em uma análise dos problemas inerentes à modelagem de sistemas de partículas, o trabalho

apresenta possíveis soluções para a aceleração do processo de síntese de imagens de sistemas de partículas em um ambiente de *Ray Tracing*, assim como propõe uma solução para amenizar o problema da quantidade de memória exigida neste processo. Também, o trabalho propõe uma técnica de aceleração para a animação de sistemas de partículas onde existe a necessidade de se tratar colisões entre as partículas e os objetos sólidos da cena.

1.2 Organização Geral da Dissertação

O conteúdo deste trabalho é dividido em sete capítulos principais.

No capítulo dois é apresentado um estudo sobre a modelagem através de sistemas de partículas, bem como as suas principais características, vantagens, desvantagens e deficiências.

No capítulo três é apresentada uma breve explanação sobre a síntese de imagens através da técnica de *Ray Tracing* juntamente com algumas das principais técnicas para a aceleração de *Ray Tracing*. Estes dois capítulos possuem o principal objetivo de efetuar uma preparação conceitual para apresentação dos métodos desenvolvidos neste trabalho.

No capítulo quatro, o autor propõe métodos para efetuar a integração de sistemas de partículas em um ambiente de *Ray Tracing*, tendo por principais objetivos tratar os problemas do tempo de processamento, alto consumo de memória e obtenção de efeitos de reflexão, transparência e sombras.

O quinto capítulo, por sua vez, apresenta os problemas encontrados no processo de animação de sistemas de partículas em uma cena que contém objetos sólidos, assim como o desenvolvimento dos sistemas de partículas e, principalmente, o tratamento de colisões entre estes sistemas de partículas e o demais objetos da cena, sendo proposta pelo autor uma técnica para solucionar tais problemas.

No sexto capítulo, são apresentados os resultados obtidos através das implementações dos métodos desenvolvidos nos capítulos quatro e cinco, assim como a análise destes resultados.

Finalmente, no capitulo sete, são descritas as conclusões obtidas com base nas análises dos resultados, bem como as possíveis futuras extensões ao trabalho.

2 Sistemas de Partículas

2.1 Introdução e Conceitos Gerais

A modelagem de objetos cujas formas não são bem definidas, tais como nuvens, fogo, fumaça, provou ser difícil e dispendiosa se forem utilizadas técnicas convencionais de modelagem, como por exemplo, modelagem através de polígonos e superfícies curvas. O uso de sistemas de partículas como uma técnica de modelagem proporciona uma modelagem simples, eficaz e eficiente de tais objetos dispersos.

Inicialmente introduzida por William T. Reeves [REE 83] como uma técnica de modelagem de objetos, a modelagem através de sistemas de partículas pode ser definida como sendo a representação de um objeto através de um conjunto composto por inúmeras partículas que definem o volume deste objeto.

Tome-se, como exemplo, uma nuvem. Ela é composta por um conjunto de inúmeras gotículas de água suspensas no ar, sendo que à medida que estas gotículas se movimentam ao longo do tempo a forma da nuvem também se modifica. Estas gotículas de água podem ser imaginadas como partículas, sendo que estas partículas definem completamente a forma da nuvem em questão. Ainda, estas partículas possuem um conjunto de atributos com propriedades semelhantes. Isto demonstra que sistemas de partículas são especialmente adequados para a modelagem deste tipo de fenômeno.

Existe uma enorme gama de fenômenos que são bem modelados através de sistemas de partículas, sendo que eles possuem especialmente a característica de não apresentarem formas bem definidas e regulares. Dentre estes fenômenos, podem-se citar fumaça, fogo, água, nuvens, fluidos em geral, além de fenômenos mais específicos, como erupção vulcânica, arco-íris, chuva, fogos de artifícios, etc. Pode-se inferir, também, que sistemas de partículas são utilizados com freqüência para a modelagem de uma grande variedade de fenômenos naturais.

Os procedimentos para a animação de um sistema de partículas são bastante simples. Um modelo básico para a animação de sistemas de partículas, baseado em Reeves [REE 83], consiste de cinco passos elementares:

❶ A cada passo na animação de um sistema de partículas, novas partículas são geradas. Uma aproximação da quantidade de novas partículas a serem geradas por passo é normalmente especificada pelo usuário;

❷ Para cada partícula gerada são associados atributos próprios. Os valores aproximados dos atributos são, na sua maioria, também especificados pelo usuário;

❸ Cada partícula que atinge alguma condição de extinção é eliminada do sistema de partículas. Dentre as condições de extinção mais comuns encontram-se: a expiração do tempo de vida, uma transparência muito grande, uma colisão contra um objeto com uma velocidade muito elevada e a localização fora de um determinado limite. Tais condições devem ser especificadas pelo usuário e normalmente são dependentes do efeito que se que simular;

❹ Os atributos das partículas restantes são modificados de acordo com as regras dinâmicas (para atributos posicionais) e de acordo com as regras específicas para o efeito ou fenômeno que se quer modelar;

❺ As partículas ativas são utilizadas na criação da imagem final de um quadro da animação.

Tais passos são explicados em maiores detalhes nas seções subseqüentes.

2.2 Geração de Partículas

Nesta etapa, a cada instante de tempo correspondente a um quadro da animação, uma determinada quantidade de partículas é criada dentro do sistema de partículas, sendo que esta quantidade é controlada por processos estocásticos. O número de partículas geradas a cada quadro é importante na medida em que influencia a densidade do objeto. Pode-se, portanto, controlar tal etapa para obter previsões do comportamento global do sistema de partículas.

Basicamente, o processo de geração de partículas é especificado de duas maneiras. Na primeira, são especificados o número médio de partículas a serem geradas a cada passo e a sua variância. Deste modo, semelhantemente a Reeves [REE 83], o número de novas partículas a serem criadas a cada passo pode ser dado por

$$NP = \mu_{Part} + \Delta.\sigma_{Part}$$

onde NP é o número de novas partículas a serem geradas, μ_{Part} é a média de partículas a serem geradas, σ_{Part} é a variância de partículas a serem geradas e Δ é uma função de distribuição normalizada para o intervalo [-1,1]. Normalmente, é utilizada uma função com distribuição uniforme devido à sua simplicidade, porém nada impede que se utilize uma função com distribuição gaussiana ou outra qualquer.

Uma outra opção para a geração de partículas leva em consideração a área da tela ocupada pelo objeto modelado, sendo por isto dependente do dispositivo de saída. Neste modelo, a média e a variância de partículas a serem geradas não é absoluta, mas relativa à área do sistema de partículas, de modo que

$$NP = \left(\mu_{sa} + \Delta.\sigma_{sa}\right).SA$$

onde *NP* é o número de partículas a serem geradas, μ_{sa} é a média de partículas a serem geradas por unidade de área do dispositivo de saída, σ_{sa} é a variância de partículas a serem geradas, também por unidade de área do dispositivo de saída, Δ é uma função de distribuição normalizada para o intervalo [-1,1] e *SA* é a área do dispositivo de saída ocupada pelo sistema de partículas.

Ainda, os parâmetros μ_{Part} , μ_{sa} , σ_{Part} , σ_{sa} podem variar ao longo da animação proporcionando a geração de mais ou menos partículas à medida que a animação decorre. Tais parâmetros normalmente variam com base no período de tempo que o sistema de partículas está ativo.

2.3 Atributos das Partículas

Para cada nova partícula que é gerada no sistema de partículas, atributos iniciais devem ser especificados. Os atributos mais comuns são:

❶ Posição,

❷ Velocidade (direção e módulo),

❸ Tamanho,

❹ Cor,

❺ Transparência,

❻ Forma e

❼ Tempo de vida.

Porém, atributos alternativos e dependentes da aplicação e dos fenômenos que estão sendo simulados podem ser incorporados. Alguns exemplos de tais atributos incluem temperatura e índice de refração.

Um atributo das partículas que merece particular atenção é o atributo posição, dado que um determinado sistema de partículas apresenta uma posição central (origem), assim como um sistema de coordenadas próprio (determinado por dois ângulos de rotação). Ainda, um sistema de partículas apresenta uma forma que define a região do espaço onde as partículas serão geradas. Através da observação de diversos programas gráficos que utilizam animações de sistemas de partículas, assim como de algumas publicações como [REE 83] [SIM 90] [LOK 92] [EBE 94], pode-se destacar as seguintes formas como as mais utilizadas:

- Esférica, definida por um raio *r* e uma origem *C*, como na figura 2.1;
- Elíptica, definida por dois raios r_1 e r_2, uma origem *C* e um vetor normal $\overrightarrow{N}$, como na figura 2.2;

- Triangular, definida por três pontos P_1, P_2 e P_3, como na figura 2.3;
- Pontual, defina por um ponto *P*, como na figura 2.4;
- Retangular, definida pela largura *w*, altura *h*, uma origem *C* e um vetor normal $\vec{N}$, como na figura 2.5.

A forma de um sistema de partículas também influencia a direção que as partículas irão percorrer. Por exemplo, em sistemas de partículas com forma esférica ou pontual, as partículas se deslocam em todas as direções, ao passo que em sistemas de partículas triangulares, elipsoidais ou retangulares, as partículas se deslocam em direções derivadas daquelas do vetor normal, com uma variação dependente do ângulo de ejeção α.

Os demais atributos normalmente são estipulados através de uma média e uma variância, de modo que pode-se utilizar a seguinte equação para defini-los, semelhantemente a Amboni [AMB 93]:

$$< atributo >_{inicial} = \mu_{<atributo>} + \Delta.\sigma_{<atributo>}$$

onde <*atributo*> é o atributo que se está referenciando, como por exemplo, cor, transparência, velocidade, tamanho, temperatura. $\mu_{<atributo>}$ diz respeito à média do atributo, $\sigma_{<atributo>}$ é a variância do atributo e Δ é uma função de distribuição normalizada para o intervalo [-1,1].

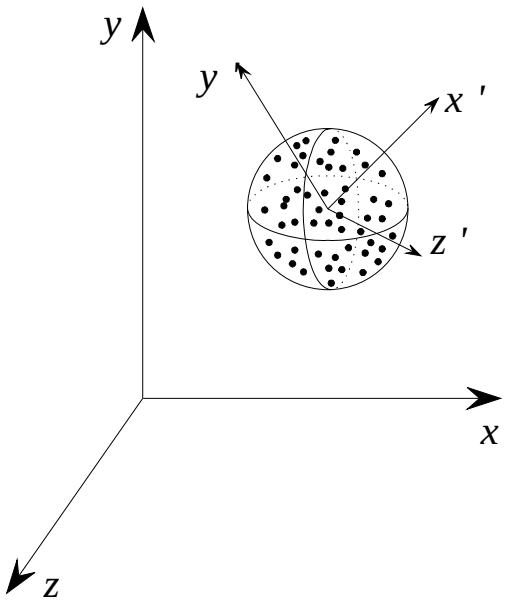

Figura 2.1 - Sistema de partículas com forma esférica.

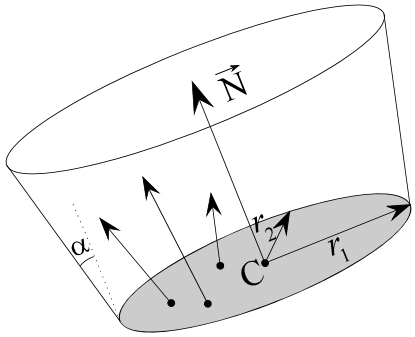

Figura 2.2 - Sistema de partículas com forma elíptica.

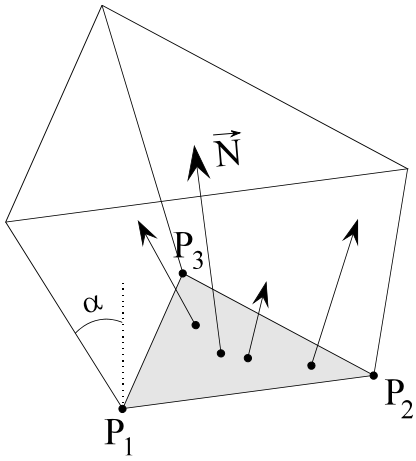

Figura 2.3 - Sistema de partículas com forma triangular.

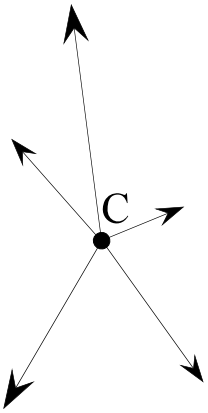

Figura 2.4 - Sistema de partículas com forma pontual.

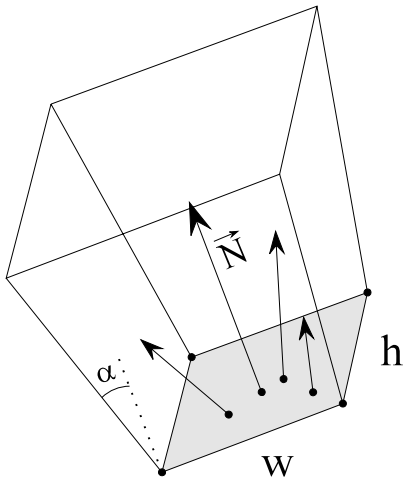

Figura 2.5 - Sistema de partículas com forma retangular.

2.4 Dinâmica das Partículas

Ao longo do seu tempo de vida, cada partícula apresenta variações em seus atributos, não somente no atributo posição, mas também nos atributos cor, transparência, tamanho e velocidade.

No que diz respeito ao atributo posição, o movimento de uma determinada partícula pode ser efetuado de diversas maneiras:

- Simplesmente adicionando o vetor velocidade (com perturbações na direção) ao vetor posição;
- Através de regras pré-definidas, tais como regras algorítmicas, ou baseadas em equações diferenciais ou mesmo através do uso de movimentos brownianos [THA 86].

Ainda, pode-se simular efeitos de gravidade, vento ou demais forças externas através da modificação correspondente na direção e/ou no módulo do vetor velocidade das partículas. O efeito de forças eólicas sobre um sistema de partículas, por exemplo, pode ser efetuado de acordo com as equações diferenciais apresentadas por Shinya [SHI 92].

2.5 Extinção das Partículas

A extinção de uma determinada partícula pode ocorrer devido a uma série de motivos. O motivo mais comum para a eliminação de uma partícula do sistema de partículas é a expiração do seu tempo de vida, porém outros motivos podem ser levados em

consideração, como por exemplo quando uma partícula apresentar uma transparência muito alta, quando estiver a uma determinada distância do centro do sistema de partículas, ou quando colidir com uma determinada velocidade com um determinado objeto.

Os motivos para a extinção de uma determinada partícula são normalmente dependentes da aplicação ou do fenômeno que se quer modelar.

2.6 Geração da Imagem Final com o Sistema de Partículas

A cada instante de tempo no processo de animação do sistema de partículas, com as posições e os atributos de suas partículas atualizados, uma imagem com o sistema de partículas e os demais objetos da cena é gerada. Pouca atenção tem sido dada a esta fase do processo de evolução de sistemas de partículas, especialmente no que tange à sua integração em ambientes para a geração de imagens realísticas, como *Ray Tracing* ou radiosidade.

2.7 Aplicações de Sistemas de Partículas

Desde a introdução de sistemas de partículas como uma técnica de modelagem por Reeves [REE 83], sistemas de partículas têm sido utilizados para simular uma grande gama de fenômenos naturais. Também, sistemas de partículas têm sido utilizados como técnica de controle de animações [REY 87] ou como técnica para a visualização de fenômenos astrofísicos [NAG 94].

Dentre as aplicações de sistemas de partículas mais comuns, encontram-se simulação de gases, fluidos, fogo, explosões, árvores e arbustos, lava vulcânica e arco-íris, etc. Ainda, sistemas de partículas têm sido utilizados na modelagem de objetos deformáveis, na modelagem da fusão da cera de velas e na simulação do movimento de nebulosas. Também, técnicas para a simulação de forças externas, como por exemplo vento, vórtices e gravidade, utilizam sistemas de partículas.

2.7.1 Modelagem de Fogo

Uma das primeiras aplicações de sistemas de partículas foi para a simulação de fogo. A técnica introduzida por Reeves [REE 83] é um procedimento simples que foi utilizado no desenvolvimento da seqüência da bomba *Genesis* no filme *Jornada nas Estrelas II: A Ira de Khan* [PAR 82].

A simulação da bomba *Genesis* utilizou um sistema de partículas hierárquico de dois níveis. Neste sistema de partículas hierárquico, cada partícula gerada no primeiro nível se transforma em um sistema de partículas de segundo nível. Deste modo, pode-se utilizar o sistema de partículas de primeiro nível para controlar o movimento global e o sistema de

partículas de segundo nível para controlar o movimento mais fino, de modo a reduzir a complexidade do processo de modelagem.

O processo de geração de sistemas de partículas hierárquico pode ser observado na figura 2.6, onde o sistema de partículas de primeiro nível tem origem no ponto de impacto da bomba e onde as partículas geradas sobre a superfície do planeta por este sistema de partículas são transformadas em sistemas de partículas de segundo nível, possibilitando assim a simulação da difusão do fogo sobre a superfície do planeta. Os sistemas de partículas de segundo nível apresentam forma circular e normal igual a normal da esfera no ponto central. Algumas imagens desta aplicação pode ser vistas na figura 2.7.

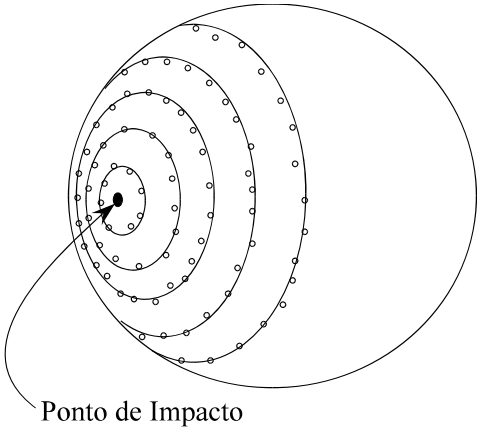

Figura 2.6 - Sistema de partículas hierárquico. [REE 83]

Também, Justin A. McCune [CUN 95] utilizou sistemas de partículas para simular não somente fogo, mas também a interação do fogo com água com a formação de fumaça. A simulação apresentada por McCune utiliza o conceito de sistemas de partículas interdependentes.

Em um sistema de partículas interdependente, as partículas podem mudar os seus atributos baseadas na presença, quantidade e proximidade das partículas vizinhas. Por exemplo, no mundo real, sistemas constituídos de moléculas tendem a se dispersar até atingir um estado em equilíbrio. Além disso, estas moléculas trocam energia entre si, sendo que tal energia interfere diretamente na cor que as moléculas emitem ou refletem. O uso de sistemas de partículas interdependentes possibilita a simulação destes fenômenos, tal como ocorre quando um sistema de partículas de água se mistura com um sistema de partículas de fogo produzindo um sistema de partículas de fumaça.

Considerando-se que o problema do cálculo das interferências das partículas de um sistema de partículas é de ordem $O(n^2)$, onde *n* é o número de partículas do sistema, McCune propôs o uso de uma grade tridimensional onde cada célula armazena apenas uma quantidade limitada de partículas, a temperatura ou outros dados essenciais. Desta forma as simulações de colisões, movimento e troca de temperatura são efetuadas através de

processos probabilísticos, de modo que a técnica passa a representar apenas uma aproximação do desenvolvimento real. Por exemplo, se uma partícula irá entrar em uma determinada célula e o número máximo de partículas desta determinada célula da grade tridimensional é de 30 partículas e existem 20 partículas nesta célula, existem uma probabilidade ⅔ da partícula colidir com as demais partículas.

Outro problema que deve ser levado em consideração é o problema de atualização das células à medida que a animação prossegue. Por exemplo, se uma partícula se move de uma célula para outra, o número de partículas nestas células deve ser alterado, assim como se uma partícula de água é movida para uma célula repleta de partículas de fogo, ela é transformada em uma partícula de fumaça e a temperatura da célula é diminuída.

Exemplificando tal procedimento, se três partículas de água são movidas para uma célula contendo 15 partículas de fogo e, considerando que são necessárias duas partículas de fogo para vaporizar uma partícula de água, ao final do processo restarão nove partículas de fogo e três partículas de fumaça na célula, uma vez que seis partículas de fogo foram utilizadas para transformar as três partículas de água em partículas de fumaça.

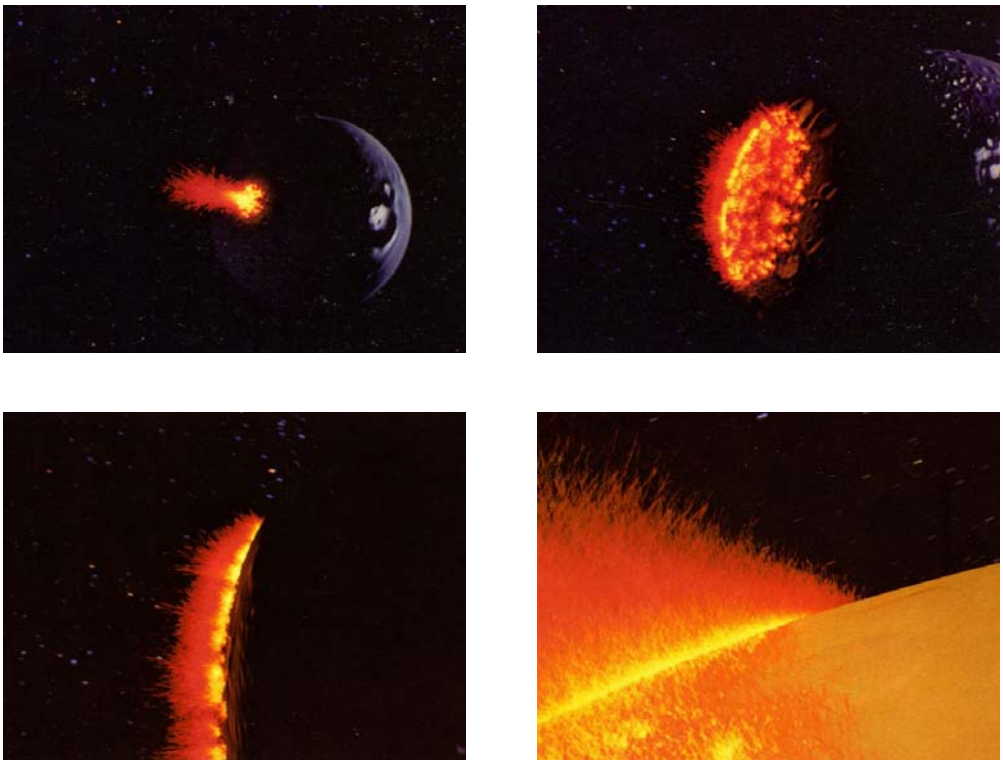

Figura 2.7 - Cenas da animação de fogo geradas por Reeves [REE 83].

2.7.2 Modelagem de Fluidos

David S. Ebert, Wayne E. Carlson e Richard E. Parent introduziram em [EBE 94] um sistema para o controle da animação de gases, líquidos, e outros volumes de funções de densidade. A técnica apresentada por Ebert *et al*. utiliza o conceito de *sistemas de partículas inversos*, assim como possui uma utilização extensiva de tabelas tridimensionais. Através do uso de tal técnica, é possível não somente efetuar um controle fisicamente embasado do movimento do fluido, mas também possibilita o controle do movimento do fluido pelo projetista da animação.

Diferentemente dos sistemas de partículas normais, onde uma partícula situada no universo tridimensional é projetada na tela, os *sistemas de partículas inversos* efetuam o procedimento inverso, de modo que cada posição da tela ocupada pelo fluido é projetada no universo tridimensional. Ou seja, cada ponto que um determinado fluido ocupa na janela de visualização é movido através do espaço tridimensional, sendo então determinada que parte do fluido é projetada sobre o ponto em questão da janela de visualização.

Quanto às tabelas utilizadas para a geração do movimento dos fluidos, são basicamente utilizadas duas tabelas. A primeira e principal é a *tabela de campos vetoriais*, que contém informações que definem a direção do movimento original do fluido. A outra tabela contém informação sobre que tipo de função deve ser evoluída para a definição do tipo de movimento do fluido, como por exemplo, movimentos helicoidais, movimentos de atração e repulsão, ou então movimentos de vórtices espirais. Esta tabela é denominada *tabela de campos de fluxos funcionais*. A primeira tabela permite que programas externos gerem os campos vetoriais que controlam o movimento básico do fluido, sendo possível então efetuar a simulação de fluido através de técnicas complexas envolvendo sistemas de equações diferencias e, após utilizar esta técnica para efetuar a visualização. Ainda, pode-se utilizar funções de atração/repulsão na segunda tabela para simular um fluido envolvendo um determinado objeto, possibilitando um tratamento simplificado de colisão do fluido com um objeto sólido.

Alguns exemplos da utilização desta técnica podem ser vistos nas figuras 2.8 e 2.9. Na figura 2.9, observe como o gás é desviado do peão devido ao uso de repulsores próximo à fronteira do peão.

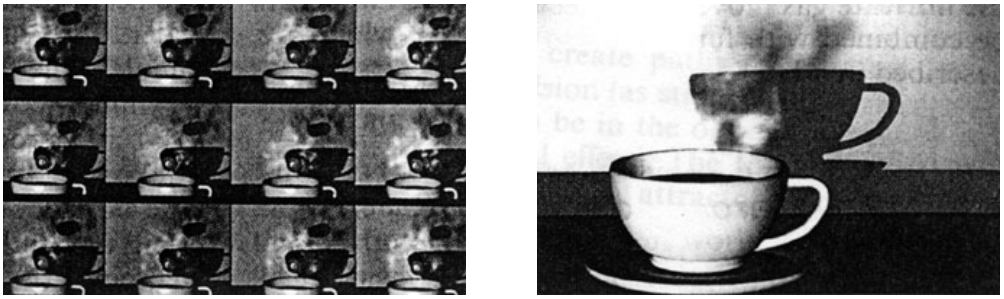

Figura 2.8 - Exemplos de resultados da técnica utilizada por Ebert *et al.* [EBE 94].

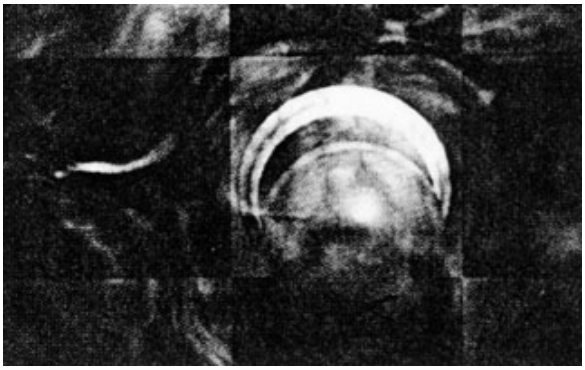

Figura 2.9 - Uso de repulsores para simulação de colisão entre fluido e objeto.

Também, Maziar H. Manesh utilizou sistemas de partículas bidimensionais para simular uma queda d'água, sendo necessário somente ao modelador especificar as restrições e propriedades do sistema de partículas. O maior problema é que, devido ao fato do sistema ser animado com base em propriedades físicas, ele é extremamente custoso computacionalmente, além de ser somente bidimensional. Um exemplo de uma imagem gerada neste sistema é apresentada na figura 2.10.

Ainda, Karl Sims [SIM 90] utilizou sistemas de partículas para simular uma queda d'água, porém em um ambiente tridimensional. No processo de criação da animação, à medida que partículas de água colidem com objetos sólidos, como esferas e planos, novas partículas são criadas para simular o desmembramento de uma gota d'água em gotículas menores. Alguns exemplos das imagens geradas por Karl Sims podem ser vistas na figura 2.11.

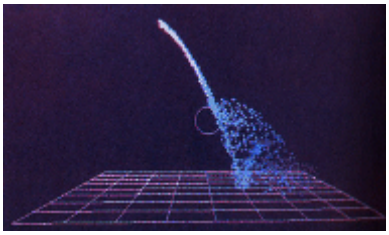

Figura 2.10 - Queda d'água modelada por Manesh.

Figura 2.11 - Queda d'água gerada por Karl Sims [SIM 90].

2.7.3 Modelagem de Fogos de Artifício

Uma aplicação para a qual sistemas de partículas são bastante adequados e muito utilizados é a simulação de fogos de artifícios. Teng-See Loke *et al.* [LOK 92] desenvolveu um sistema para a modelagem e exibição de fogos de artifícios utilizando sistemas de partículas. Também, Basia Taylor [TAY 96] realizou estudos na mesma linha de pesquisa.

Em síntese, o sistema desenvolvido por Loke *et al.* armazena as informações das partículas que compõem o fogo de artifício em duas listas: uma para as partículas ativas e outra para as partículas extintas. Uma vez que as partículas podem ser reutilizadas, salvar as partículas que foram extintas é muito útil, economizando, deste modo, tempo no processo de alocação e realocação de memória para partículas com a mesma estrutura.

As partículas ativas, por sua vez, são introduzidas em um *mecanismo de manipulação de sistemas de partículas* (*Particle System Rendering Engine*, ou então, *PSRE*). Este mecanismo consiste de vários módulos que manipulam as propriedades particulares de cada partícula, como por exemplo, cor, brilho, forma, rastro, entre outros. Cada módulo no *PSRE* também exibe as partículas na área de vídeo, onde a exibição de cada partícula está sujeita às projeções e transformações perspectivas. O *PSRE* consiste basicamente de oito componentes que manipulam os seguintes procedimentos:

1. *Controle de movimento*: Manipula a dinâmica das partículas. Este módulo move as partículas para a sua próxima posição e efetua os cálculos correspondentes para a aceleração ou desaceleração das partículas;

2. *Mudança de cor*: Modifica a cor e a transparência das partículas através do uso de um incremento RGBA, onde A representa um fator de transparência a ser aplicado sobre os termos R, G e B;

3. *Efeitos especiais*: Trata efeitos especiais, como o *mousing*;

4. *Contagem de estágios*: Controla a contagem regressiva do atributo relativo ao tempo de vida da partícula, mensurado em número de quadros;

5. *Recorte*: Remove as partículas que estão fora da área de interesse, assim como elimina as partículas que estão com pouco brilho;

6. *Cintilação*/*Exibição*: Manipula o efeito de cintilação das partículas e também exibe as partículas que estão ativas na área de vídeo. Também controla o efeito de *starring*;

7. *Salvamento do quadro*: Captura cada quadro individual que foi totalmente exibido e transfere para arquivo;

8. *Spawning*: Cria o rastro que a partícula deixa a medida que se move no espaço.

Quanto às características das partículas, o atributo de cor é associado a cada partícula individualmente e é representado através dos componentes RGB. Ainda, um quarto componente, denominado componente *alpha*, é introduzido de modo a controlar a transparência de cada partícula. O componente *alpha* é necessário devido ao fato que as partículas em fogos de artifícios desaparecem gradualmente, sendo tal procedimento modelado através do aumento da transparência das partícula à medida que o tempo passa.

Já o atributo de brilho das partículas modela a intensidade da partícula de um fogo de artifício, onde zero é equivalente à cor base da partícula e o valor unitário representa uma partícula extremamente brilhante. Deste modo, através de um interpolação linear, obtêm-se as seguintes equações para a modificação das cores, segundo [LOK 92]:

$$C_r = (1.0 - R_0).B + R_0$$

$$C_g = (1.0 - G_0).B + G_0$$

$$C_b = (1.0 - B_0).B + B_0$$

onde C_r, C_g e C_b são as componentes RGB da cor a ser exibida, R_0, G_0 e B_0 são as componentes RGB da cor base da partícula e *B* é o brilho da partícula. O brilho de uma partícula começa com um valor próximo ao valor unitário e decresce gradualmente até zero.

Um ponto luminoso, quando fotografado, normalmente deixa um rastro. O rastro das partículas de um fogo de artifício é modelado através da criação de partículas filhas idênticas à original, porém com maior transparência, de modo a modelar corretamente o desaparecimento do rastro.

O movimento das partículas, por sua vez, é definido através das seguintes equações básicas:

$$\Delta s = u(t).\Delta t$$

$$\Delta v = a(t).\Delta t$$

onde Δt, Δv e Δs são, respectivamente, os incrementos de tempo, velocidade e posição entre quadros. $u(t)$ e $a(t)$ são, respectivamente, a velocidade e a aceleração da partícula no instante de tempo *t*.

Ainda, efeitos como *mousing* e *starring*, são simulados através de procedimentos específicos realizados em módulos separados. O efeito de *mousing* normalmente ocorre quando as partículas estão prestes a desaparecer e começam a se movimentar desordenadamente. Uma pequena perturbação no vetor velocidade é introduzida para a simulação deste efeito. Já o efeito de *starring* ocorre quando uma partícula é extremamente brilhante e aparenta ser uma estrela, normalmente com quatro membros.

Alguns exemplos dos resultados obtidos por este sistema de manipulação de sistemas de partículas desenvolvido por Loke *et al.* podem ser visto na figura 2.12.

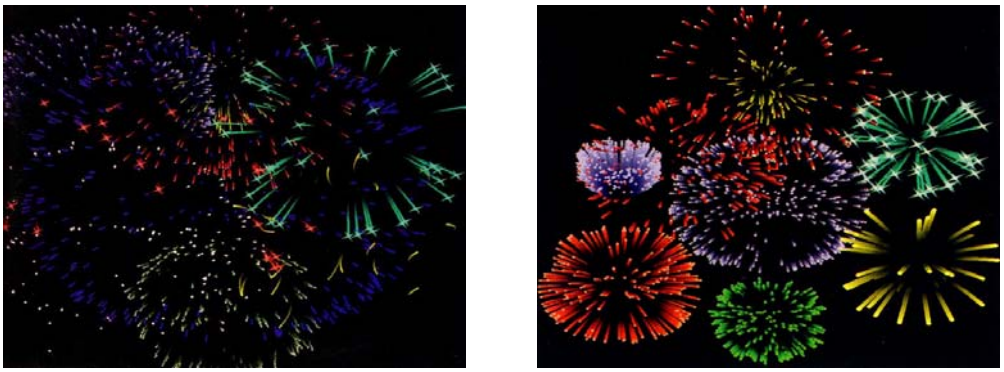

Figura 2.12 - Fogos de artifício modelados por Loke *et al.*. [LOK 92]

2.7.4 Modelagem de Fluxos de Lava e Erupções Vulcânicas

Pereira e Hsu [PER 95] demonstraram ser possível o uso de sistemas de partículas para a modelagem de um fenômeno natural bastante particular: fluxo de lava em uma erupção vulcânica. No modelo introduzido por estes autores, as partículas são geradas no topo do vulcão, sendo que a cada partícula são atribuídas uma temperatura e uma velocidade inicial de acordo com as características da erupção vulcânica.

O vulcão, por sua vez, é representado através de uma grade bidimensional, onde para cada célula da grade, o vulcão possui uma determinada altura e temperatura, semelhantemente a um campo de alturas (*height field*). Ainda, cada partícula é associada

com uma célula da grade de acordo com a sua posição. Tal procedimento pode ser verificado na figura 2.13.

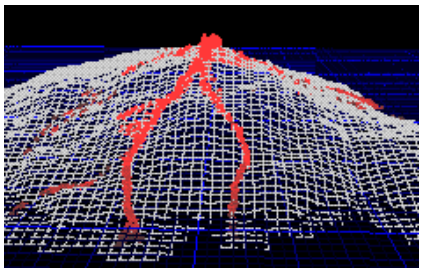

Figura 2.13 - Vulcão definido por uma grade bidimensional. [PER 95]

O movimento das partículas é definido por uma série de fatores, porém os fatores que são considerados para evoluir o movimento das partículas são *viscosidade*, *repulsão*, *atração*, *gravidade* e *fricção*. Estes fatores podem ser divididos em forças entre partículas (viscosidade, repulsão e atração) e forças externas (gravidade e fricção).

A viscosidade propicia a troca da quantidade de movimento entre partículas próximas, de modo que estas começam a se movimentar mais uniformemente. A viscosidade entre partículas aumenta à medida que a temperatura das partículas diminui.

A repulsão mantém as partículas a uma distância mínima, de modo que elas não fiquem demasiadamente próximas, assim como a atração mantém as partículas dentro de uma distância máxima, de modo que elas não fiquem demasiadamente afastadas, conseguindo-se com isso que a lava mantenha uma característica semelhante à de fluido. As forças de repulsão e atração são implementadas através da avaliação da densidade de partículas em uma determinada célula, sendo que quando a densidade de partículas torna-se muito grande, as partículas começam a se repelir e quando a densidade de partículas torna-se muito baixa, as partículas começam a atrair partículas de outras células. Este modelo, apesar de não ser tão exato quanto o que se obteria efetuando o cálculo de cada partícula separadamente, é muito mais rápido.

Quanto às forças externas, quando uma partícula encontra-se solta no ar, a gravidade a acelera para baixo. Uma vez que a partícula encontra-se em contato com a superfície do vulcão, a gravidade empurra a partícula de acordo com o vetor gradiente da superfície na célula em questão. Já a fricção diminui a velocidade da lava ao longo da superfície do vulcão, sendo que quando as partículas atingem o oceano, outro fator de fricção é introduzido de modo a simular a resistência da água.

O processo de eliminação de partículas é realizado quando a temperatura de uma determinada partícula decresce abaixo de um determinado limiar e a partícula, então, solidifica e torna-se parte do vulcão. Ao nível de implementação, isto é realizado aumentando-se a altura da célula do vulcão onde a partícula solidificou-se. O processo de

aumento da área do vulcão devido à solidificação da lava pode ser observado na figura 2.14.

As partículas de lava são sintetizadas através de aproximações de esferas, possuindo as mesmas propriedades difusa e especular de uma rocha, porém com a propriedade de emitir de luz. A quantidade de luz emitida é dependente da temperatura da partícula e é implementada variando-se a componente ambiente da esfera.

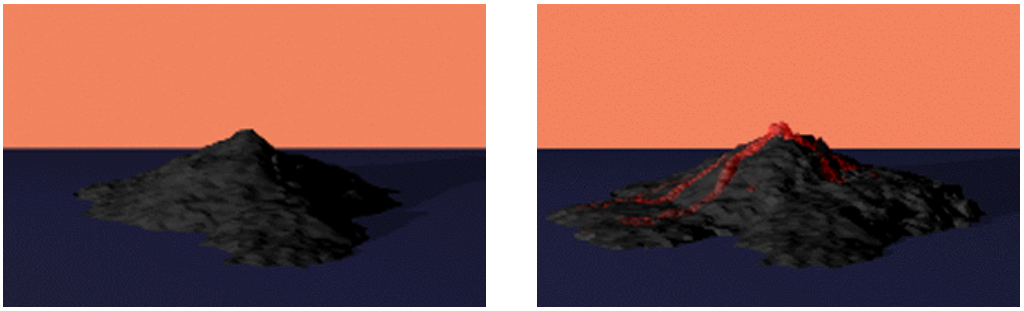

Figura 2.14 - Vulcão antes e depois de uma erupção vulcânica. [PER 95]

Outra característica considerada é que, a noite, a luminosidade da lava é suficiente para iluminar as regiões do vulcão próximas a ela. Para efetuar a simulação deste efeito, é adicionado um termo ambiente adicional para as células do vulcão que possuem partículas com alta temperatura nas proximidades. Um exemplo de uma simulação de uma erupção vulcânica noturna pode ser vista na figura 2.15.

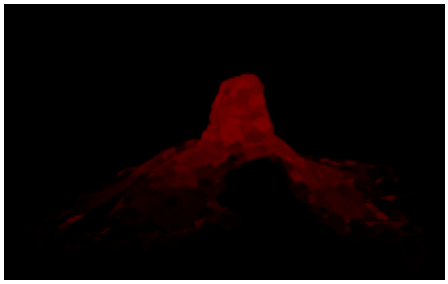

Figura 2.15 - Simulação de uma erupção vulcânica noturna. [PER 95]

2.7.5 Modelagem de Chuva e Arco-íris

Alice Tull e Helios Tsoi [TUL 95] desenvolveram um modelo para efetuar a simulação de chuva e arco-íris. O modelo desenvolvido considera o fato de que a modelagem de chuva é um problema típico de sistemas de partículas, uma vez que a chuva é constituída de pequenas gotículas de água com atributos similares. Deste modo,

semelhantemente ao modelo introduzido por Reeves [REE 83], partículas são criadas a cada instante de tempo com atributos próprios (como posição, velocidade, cor, etc.) e são extintas ao colidirem com o solo. O modelo desenvolvido por Alice Tull e Helios Tsoi também possui a capacidade de modelar efeitos de forças externas como, por exemplo, vento.

O arco-íris, por sua vez, é modelado através de um modelo fisicamente embasado. O modelo subdivide a cor dos raios de luz em uma parte de luz branca e uma parte de luz pura, sendo que uma tabela de consulta pré-calculada é utilizada para determinar a cor de determinada partícula no arco-íris. Os efeitos de luz branca e luz pura são consultados na tabela e suas cores combinadas no resultado final. Um exemplo deste modelo pode ser visto na figura 2.16. Observe que, como a tabela de associação entre freqüência e cor pode ser definida pelo usuário, pode-se conseguir arco-íris com somente algumas cores específicas, como ocorre no exemplo da figura 2.16.

Figura 2.16 - Arco-íris com o *Taj Mahal* ao fundo. [TUL 95]

2.7.6 Modelagem de Fusão de Cera de Vela

A modelagem de substâncias como cera é particularmente difícil, devido ao fato que, em um determinado instante, ela pode consistir tanto de componentes sólidos como de componentes líquidos que interagem entre si. Gary Herman e David Redkey [HER 95] propuseram um modelo para a modelagem do comportamento físico da cera.

Basicamente, o modelo proposto por Gary Herman e David Redkey decompõe o bloco de cera (no caso a vela) em um conjunto de esferas sólidas constituídas de partículas, sendo que estas partículas são mantidas agrupadas devido às forças de atração existentes entre elas.

Além destas forças interpartículas, as partículas transferem calor entre elas proporcionalmente às suas diferença de calor. Deste modo, à medida que as partículas aquecem, as forças interpartículas diminuem. No momento que as forças interpartículas

atingem um limiar mínimo, as partículas desmembram-se das esferas, caracterizando a fusão da cera.

Além das interações interpartículas, outros fatores externos foram introduzidos, tais como gravidade, arraste e calor proveniente da chama. O calor proveniente da chama é modelado através de uma distribuição quadrática inversa do calor com o ponto central no centro da chama. A força de arraste é proporcional à velocidade da partícula, porém na direção oposta e é introduzida para considerar os efeitos de fricção, resistência do ar, etc.

A implementação do modelo consiste, para cada quadro, de quatro etapas:

❶ Calcular as forças interpartículas;

❷ Calcular a transferência de calor entre as partículas;

❸ Atualizar a posição e a velocidade de cada partícula, baseadas nas forças interpartículas e forças externas;

❹ Ajustar a altura da chama de modo a simular o processo de fusão da cera da vela;

Alguns exemplos do modelo implementado por Gary Herman e David Redkey podem ser observados na figura 2.17.

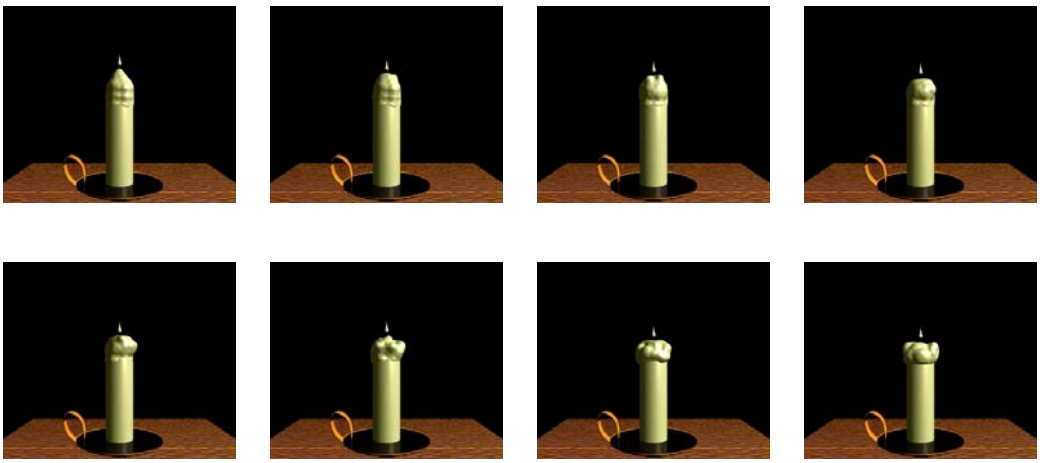

Figura 2.17 - Exemplos de fusão de vela modelado por Herman e Redkey. [HER 95]

2.7.7 Modelagem de Árvores e Arbustos

Os primeiros trabalhos no sentido de utilizar sistemas de partículas para modelar árvores e arbustos foram propostos por William T. Reeves e Ricki Blau [REE 85]. O modelo proposto por Reeves e Blau utiliza sistema de partículas estruturados para a modelagem de árvores com alto grau de detalhamento e onde o processo de sombreamento e detecção de superfícies visíveis é feito através de métodos probabilísticos. Também, processos estocásticos são utilizados para modelar movimentos complexos e que possuem bastante variação, como por exemplo, vento e brisa.

No processo de geração de uma árvore segundo este modelo, o algoritmo constrói a árvore a partir do tronco gerando galhos recursivamente até o nível de detalhamento desejado. Antes de começar a geração dos galhos, entretanto, o algoritmo estocasticamente atribui o conjunto inicial de parâmetros que definem a estrutura de uma determinada árvore. Os principais atributos de uma árvore são mostrados na figura 2.18.

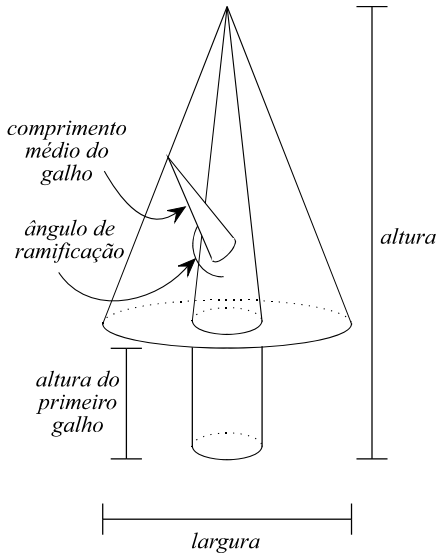

Figura 2.18 - Principais atributos no processo de geração de uma árvore.

Os galhos, por sua vez, são diminuídos através da associação da sua espessura com o tamanho que o galho possui. Desta forma, Reeves e Blau [REE 85] definiram a espessura de um determinado galho como sendo

$$\tau = \tau_b \sqrt{\frac{l-d}{l}}$$

onde τ é a espessura do galho a ser gerado, τ_b é a espessura da base do galho, ou seja, do primeiro galho gerado junto ao tronco, l é o comprimento total estimado do galho e d é o tamanho atual do galho.

Existe ainda uma herança entre os galhos gerados, ou seja, cada galho gerado herda muitos dos atributos do seu galho gerador ou pai. Porém, alguns atributos são

ajustados para as atuais dimensões do galho gerado ou filho, como por exemplo, o ângulo de ramificação diminui a medida que a espessura do galho diminui.

O sombreamento efetuado sobre as árvores é realizado através de métodos probabilísticos, como já foi mencionado. Isto quer dizer que a posição e a orientação de uma determinada partícula determinam a probabilidade de que esta partícula esteja na sombra, possua uma determinada cor difusa ou possua características especulares. Também, processos probabilístiscos são utilizados para a obtenção de efeitos de auto-sombreamento.

A obtenção das componentes especular, difusa e ambiente são obtidas basicamente através de duas distâncias. De acordo com a figura 2.19, a componente difusa é baseada na distância entre a superfície externa da árvore e a partícula em questão. A equação para a obtenção da intensidade difusa, segundo [REE 85], é dada por

$$I_d = e^{-\alpha d_d}$$

onde I_d é a intensidade difusa, d_d é distância entre o envelope cônico da árvore e a partícula e α é um parâmetro que controla a atenuação exponencial. A contribuição especular, por sua vez, é introduzida probabilisticamente à medida que a distância d_d torna-se pequena e que o co-seno do ângulo entre a direção do galho e a direção da fonte de luz é próxima de zero.

O efeito de auto-sombreamento é simulado através da manipulação da componente ambiente, sendo que esta é atenuada exponencialmente à medida que a distância do envelope cônico da árvore ao seu centro, d_a , aumenta. Esta distância é independente da posição da fonte de luz. Porém, deve-se considerar que mesmo nos pontos mais internos de uma árvore existe alguma intensidade luminosa, de modo que uma constante A_{min} é especificada para determinar este limiar mínimo. Deste modo, a equação que define a intensidade ambiente em uma determinada partícula no interior da árvore é

$$I_a = max\left(e^{-\beta d_a}, A_{min}\right)$$

onde I_a é a intensidade ambiente e β é um parâmetro que determinada o fator de atenuação exponencial da intensidade ambiente.

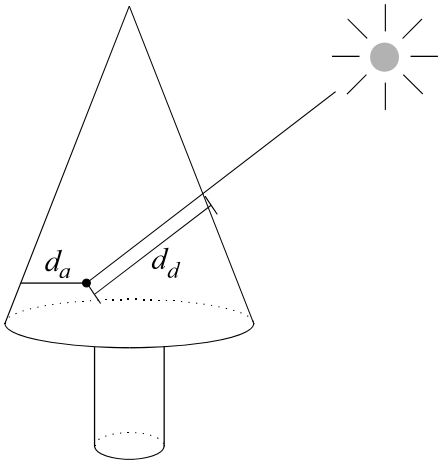

Figura 2.19 - Distâncias utilizadas para a obtenção do sombreamento.

Existe ainda o problema das sombras causadas pelas árvores vizinhas. Para obter tal efeito é utilizada uma aproximação na qual um plano é definido através do vértice da árvore vizinha e a fonte de luz e especificando-se sempre a mesma orientação, como pode ser visto na figura 2.20. Se a partícula em questão estiver abaixo deste plano, a partícula encontra-se na sombra e somente a componente ambiente é considerada. Já se a partícula encontra-se acima deste plano, a partícula é iluminada diretamente pela fonte de luz e possui também as componentes difusa e especular.

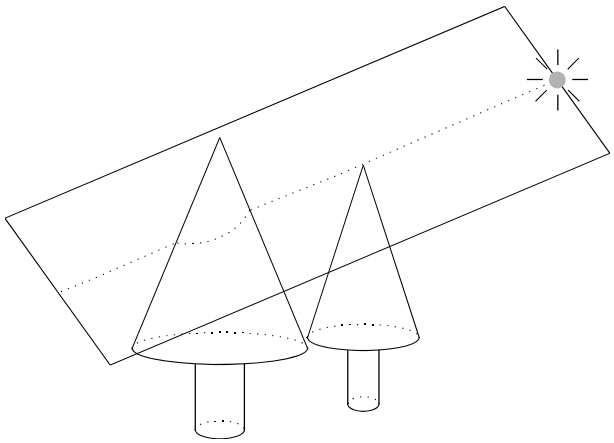

Figura 2.20 - Sombreamento de árvores vizinhas.

Alguns exemplos da modelagem de árvores através desta técnica podem ser vistos na figura 2.21.

Figura 2.21 - Exemplos de árvores e grama modeladas por Reeves e Blau. [REE 85]

2.7.8 Modelagem de Vento

A modelagem de forças externas (normalmente forças eólicas) sobre as partículas tem sofrido poucos desenvolvimentos. Tais desenvolvimentos são bastantes esparsos e dificilmente é enfocado a aplicação direta dos modelos sobre sistemas de partículas.

Os primeiros passos foram desenvolvidos por Reeves e Blau [REE 85] que modelam a influência de forças eólicas, como vento e brisa, através da criação e evolução de um sistema de partículas onde cada partícula define uma rajada de vento. Cada partícula de vento, por sua vez, influencia as partículas próximas de modo que os ramalhetes de grama modelados também por sistemas de partículas apresentavam um efeito perceptual de ação eólica. Também, Ebert, Carlson e Parent [EBE 94] desenvolveram um modelo perceptual baseado no uso de atratores e repulsores.

Ainda, Mikio Shinya e Alain Fournier [SHI 92] desenvolveram um modelo genérico para a modelagem de efeitos eólicos sobre uma variedade de técnicas de modelagem de fenômenos naturais, tais como, modelos estruturais, texturas tridimensionais, modelos impressionistas, assim como sistemas de partículas.

O modelo considera que os objetos, como árvores modeladas por Reeves e Blau [REE 85], são definidos pela trajetória de partículas em movimento e que, devido a este fato, o processo dinâmico do objeto não é tão bem definido como nos modelos estruturais.

Modelos estruturais, por sua vez, definem um objeto como uma estrutura em forma de árvore determinada através conexões entre troncos, nodos, galhos e dados geométricos. Contudo, movimentos realísticos de grama e arbustos sob a influência de forças eólicas são criadas através do controle das condições iniciais das partículas.

Assim sendo, Shinya e Fournier [SHI 92] utilizam equações diferenciais para a obtenção da perturbação das forças eólicas sobre as partículas. Considerando-se a velocidade inicial de uma determinada partícula em coordenadas esféricas $\vec{v_0} = (v_0, \theta, \phi)$, o resultado da força de arraste de uma determinada força eólica $\vec{f} = (f_x, f_y, f_z)$ é dado pela vibração da direção inicial de acordo com:

$$\frac{d^2\theta}{dt^2} + \gamma_\theta \frac{d\theta}{dt} + \omega_\theta^2 \theta = \frac{f_\theta}{m} ,$$

$$\frac{d^2\phi}{dt^2} + \gamma_\phi \frac{d\phi}{dt} + \omega_\phi^2 \phi = \frac{f_\phi}{m} ,$$

$$f_\theta = -f_x \, sen\theta + f_y \, cos\theta \text{ e}$$

$$f_\phi = f_x \, cos\theta + f_y \, sen\theta$$

onde m, ω_θ, ω_ϕ, γ_θ e γ_ϕ são constantes.

2.8 Comparação com Técnicas Convencionais

Relacionando-se a representação através de sistemas de partículas e a representação utilizando técnicas convencionais de modelagem, pode-se apresentar três diferenças básicas:

- Diferentemente das técnicas convencionais que definem um objeto através de polígonos e superfícies, definindo a sua fronteira, a modelagem através de sistemas de partículas representa um objeto através de um conjunto de partículas que definem o volume do objeto;

- Sistemas de partículas são intrinsecamente dinâmicos, uma vez que à medida que o parâmetro temporal é modificado, a sua forma também é modificada. Além disso, em um sistema de partículas, as partículas são constantemente geradas e eliminadas. Tal efeito não é observado em técnicas convencionais, onde a forma dos objetos é definida separadamente do processo de animação destes objetos. Ou seja, o parâmetro temporal não está relacionado com a forma do objeto, mas sim ao processo de animação que irá aplicar um conjunto de operações sobre este objeto;

- A forma de um sistema de partículas em um determinado instante de tempo não é totalmente especificada, uma vez que processos estocásticos são aplicados sobre o movimento das partículas proporcionando uma representação não determinística dos objetos. Deste modo, pode-se obter objetos levemente diferentes através da simples modificação do valor inicial que controla as perturbações estocásticas do movimento do sistema de partículas.

Ainda, relacionado-se a modelagem através de sistemas de partículas com as técnicas convencionais de modelagem, a modelagem através de sistemas de partículas apresenta algumas vantagens e desvantagens. Dentre as vantagens, citam-se:

- Considerando-se que as partículas são primitivas muito mais simples que as primitivas utilizadas pelas técnicas convencionais, no caso polígonos ou superfícies definidas matematicamente, no mesmo período de tempo pode-se processar muito mais partículas e criar objetos muito mais complexos. Ainda, devido à simplicidade das partículas pode-se obter efeitos de borrão de movimento (*motion blur*) com relativa facilidade e com um custo computacional baixo, diferentemente das técnicas convencionais de modelagem onde tal procedimento exige um custo computacional muito maior;

- Ainda, a modelagem através de sistemas de partículas é baseada no conceito de amplificação de dados (*database amplification*), ou seja, um conjunto básico de informações é fornecido pelo usuário ao sistema que, por sua vez, utilizando processos estocásticos, cria um conjunto maior de partículas. Esta vantagem possibilita a criação de objetos com alto detalhamento a partir da especificação de poucos parâmetros;

- Outra vantagem é que sistemas de partículas são definidos proceduralmente e controlados por processos estocásticos, de modo que pode-se obter objetos com níveis de detalhamento variável. Tal propriedade possui grande utilidade quando se está projetando um sistema de partículas e deseja-se obter uma previsão do seu desenvolvimento ao longo da animação;

- Considerando-se que sistemas de partículas possuem um nível de detalhamento ajustável, pode-se aproximar o ponto de observação do objeto modelado e ainda obter um conjunto de detalhes satisfatório, tal como ocorre na representação através de fractais;

- Outra grande vantagem da modelagem através de sistemas de partículas é que estes são especialmente adequados para a criação de animações, uma vez que possuem um parâmetro temporal implícito. Deste modo, sistemas de partículas são extremamente vantajosos na modelagem de objetos cujas modificações da forma e da posição ao longo do tempo são complexas;

- Considerando-se que as partículas são primitivas simples e de fácil controle, os objetos modelados através de sistemas de partículas possuem fácil controle sobre as suas deformações, especialmente no caso de colisões com outros objetos.

Porém, sistemas de partículas possuem algumas desvantagens que têm limitado o seu uso, especialmente em ambientes com recursos computacionais limitados. São elas as principais:

- Difícil obtenção de efeitos realísticos de sombra e de reflexão, uma vez que a sua utilização em ambientes de *rendering* como *Ray Tracing* é praticamente impossível, se não for utilizada nenhuma técnica de aceleração [FOL 90]. Isso é devido ao fato que um sistema de partículas com um grau de detalhamento considerável pode ter centenas de milhares de partículas, o que inviabilizaria o seu uso em um sistema de *Ray Tracing*;

- Outra desvantagem difícil de ser remediada é o fato que a dinâmica de um sistema de partículas é bastante específica para cada efeito ou fenômeno que ser quer modelar. Ou seja, apesar da maioria dos efeitos possuir um conjunto de regras comuns, cada efeito possui um conjunto de regras específicas que são de difícil integração;

- Uma última desvantagem é o alto consumo de memória. Esta desvantagem, entretanto, pode ser solucionada através do aumento dos recursos computacionais, o que vem ocorrendo com grande rapidez. Porém, para o uso de sistemas de partículas em ambientes limitados, tal aspecto deve ser considerado.

2.9 Conclusões

Este capítulo apresentou as principais características que devem ser consideradas em um sistema de partículas, bem como as suas principais aplicações, principalmente na modelagem de fenômenos naturais, tais como fogo, nuvens, água, fumaça, etc.

Dentre as principais vantagens para o uso de sistemas de partículas ao invés de técnicas convencionais de modelagem encontram-se o fato de serem constituídas por primitivas simples, utilizarem o conceito de amplificação de dados, serem definidas proceduralmente e serem controladas por processos estocásticos, possuírem um nível de detalhamento ajustável, serem adequados para uso em animações e possuírem um grande controle sobre as suas deformações.

Porém, os sistemas de partículas possuem algumas limitações e restrições que causaram o pouco desenvolvimento de algoritmos específicos nesta área. Dentre estas

limitações, as principais são a dificuldade de obtenção de efeitos realísticos de sombra e reflexão, o alto consumo de memória e o fato dos sistemas de partículas possuírem um processo de animação específico para cada efeito que se quer modelar.

Finalmente, sistemas de partículas têm sido cada vez mais utilizados na modelagem de determinados fenômenos. Porém, as suas limitações, deficiências e restrições têm dificultado um emprego ainda maior em aplicações que necessitem um uso massivo de partículas juntamente com efeitos realísticos de iluminação e sombreamento.

3 *Ray Tracing*

3.1 Introdução

O mundo real é constituído por um conjunto infindável de fenômenos extremamente complexos, dos quais os cientistas não entendem completamente o seu funcionamento e as suas propriedades. Tendo isto em vista, os processos de síntese de imagens procuram produzir imagens que refletem uma aproximação do mundo real, representando o nível atual de compreensão dos fenômenos físicos envolvidos, normalmente utilizando uma variedade de técnicas foto-realísticas. Dentre as técnicas mais utilizadas, encontram-se as técnicas de *Ray Tracing* e de *radiosidade*. A técnica de *Ray Tracing* é uma das técnicas mais utilizadas devido ao fato de ser bastante simples, de fácil compreensão e baseada em princípios óticos.

Este capítulo tem por principal objetivo descrever resumidamente o processo de geração de imagens através da técnica de *Ray Tracing*, assim como as suas vantagens, desvantagens e restrições. Este capítulo apresenta também alguns dos métodos mais utilizados para a aceleração de *Ray Tracing*.

3.2 A Técnica de *Ray Tracing*

A técnica de *Ray Tracing*, como já foi dito, tenta simular o efeito produzido pelos raios de luz emitidos, refletidos e transmitidos dentro de um determinado ambiente. Inicialmente, alguns aspectos devem ser levados em consideração antes de ser efetuada uma explanação mais detalhada. Dentre estes detalhes, os mais importantes são:

1. O ambiente é descrito como uma coleção de objetos primitivos (como, por exemplo, esferas, polígonos, caixas) e de fontes de luzes;

2. As fontes de luz, normalmente, são pontuais e emitem luz uniformemente em todas as direções;

3. Por convenção, objetos não emitem luz e fontes de luz não são diretamente observáveis. Esta separação entre fontes de luz e objetos é uma conveniência computacional, sendo, entretanto, passível de mudança caso se queira implementar sistemas mais complexos constituídos por objetos emissores de luz que possam ser observados diretamente.

Tendo tais restrições em vista, define-se um *ponto de observação* no espaço tridimensional do qual o ambiente é observado através de um janela retangular denominada *janela de observação*. A imagem da janela de observação é, então, mapeada na tela de

vídeo através de uma associação direta entre os pontos da tela de vídeo e os pontos na janela de observação. Tal procedimento pode ser observado mais claramente na figura 3.1.

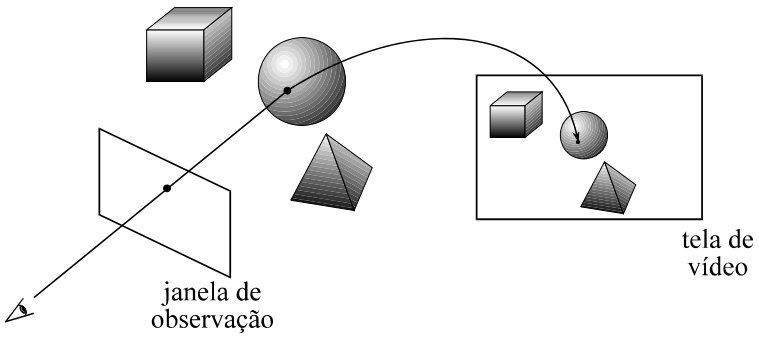

Figura 3.1 - Ambiente de *Ray Tracing* e associação entre
a janela de observação e a tela de vídeo.

Provavelmente, o método mais direto e intuitivo para se obter uma imagem do ambiente é seguir os raios de luz das fontes de luz até os objetos e trabalhar com os que interseccionam a janela de observação. Esta técnica de *Ray Tracing* é denominada *Forward Ray Tracing*. Entretanto, esta alternativa não é praticável, uma vez que a maioria dos raios não irão necessariamente interseccionar a janela de observação. Deste modo, o uso do *Forward Ray Tracing* como uma técnica para a síntese de imagens torna-se proibitivo devido à enorme quantidade de raios necessários para se obter uma imagem com boa qualidade. Também, podem surgir espaços não preenchidos entre pontos na tela de vídeo e que devem ser tratados, sendo tal procedimento bastante complicado.

Uma maneira mais eficiente de realizar tal procedimento é utilizar o processo reverso, ou seja, gerar raios de luz partindo do ponto de observação através da janela de observação e determinar os objetos que contribuem para o cálculo da cor deste ponto. Tal técnica de *Ray Tracing* é denominada *Reverse Ray Tracing*, uma vez que os raios de luz são percorridos do ponto de observação para as fontes de luz. Devido ao fato da técnica de *Reverse Ray Tracing* ser a técnica utilizada na maioria dos casos de *Ray Tracing*, ela é também referenciada simplesmente como *Ray Tracing*.

Para ilustrar melhor a diferença básica entre estas duas alternativas, considere as figura 3.2 e 3.3. A figura 3.2 mostra o processo em um *Forward Ray Tracing*, onde os raios partem da fonte de luz L_1, atingem os objetos da cena e, finalmente, atingem a janela de observação. Porém, pode-se observar que vários raios não contribuem para a formação da imagem final e sequer são considerados, como por exemplos os raios R_4 e R_7.

Devido ao fato de a maioria dos raios não apresentarem contribuições significativas para a imagem, a técnica de *Reverse Ray Tracing* considera os raios partindo do ponto de observação e atravessando a janela de observação, como na figura 3.3. Nesta abordagem, para cada ponto na tela de vídeo é disparado um raio, que por sua vez percorre o ambiente e verifica a contribuição de cada objeto particular para este ponto. Como pode-

se observar, esta abordagem mostra-se viável computacionalmente, uma vez que todos os raios disparados são utilizados.

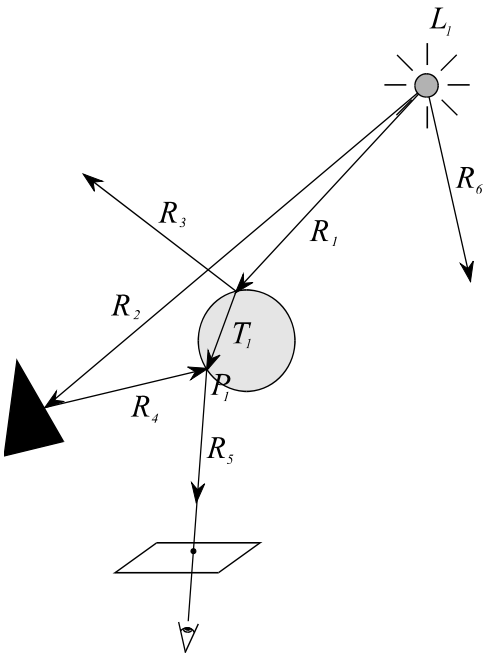

Figura 3.2 - Percurso dos raios em um sistema de *Forward Ray Tracing*, que modela o fenômeno ótico tal como ele ocorre na natureza.

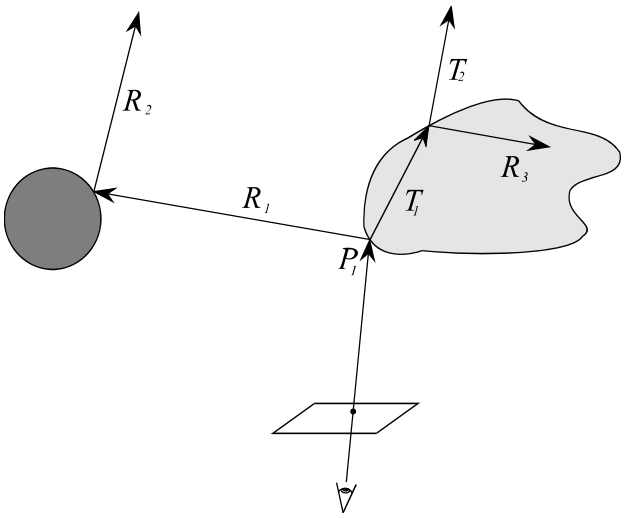

Figura 3.3 - Percurso dos raios em um sistema de *Ray Tracing*, no sentido inverso, para otimização computacional.

Uma vez determinado o objeto que é visível para um determinado ponto da janela de observação, é necessário determinar a cor que este ponto possuirá. Considerando-se que o raio partindo do ponto de observação através da janela de observação intercepta o objeto em um ponto *P*, pode-se afirmar que a coloração deste ponto é totalmente determinada pelas propriedades físicas do objeto, assim como pela luz que atinge o objeto neste ponto (considerando-se que o objeto não emite luz). Esta luz que atinge o objeto é denominada *luz incidente* e pode ser subdividida em *luz refletida* e *luz transmitida*. A luz refletida é a luz que, de acordo com as propriedades físicas e óticas do objeto é refletida de volta ao ambiente, enquanto que a luz transmitida é a luz que passa através do objeto devido ao fato da superfície ser transparente ou translúcida.

A luz incidente, por sua vez, é repassada ao ambiente (através de reflexão e/ou transmissão) através de duas maneiras: *propagação especular* e *propagação difusa*. A propagação especular é resultante de uma reflexão ou transmissão perfeitas. Já a propagação difusa propaga a luz uniformemente em todas as direções. Isso é devido ao fato que uma superfície pode ser bastante rugosa, sendo o raio incidente dispersado em várias direções. Por isso, a propagação difusa é algumas vezes denominada também *dispersão difusa*. O cálculo das reflexões e transmissões difusas, entretanto, é um processo bastante complexo e que apresenta um custo computacional elevado (normalmente tratado por técnicas como *radiosidade*, *Ray Tracing* distribuído, etc.). Um sistema de *Ray Tracing* tradicional utiliza uma componente, denominada *iluminação ambiente*, para efetuar a simulação das inter-reflexões entre as superfícies difusas dos objetos do ambiente.

Deste modo, um ambiente de *Ray Tracing* deve especificar as propriedades *especular*, *difusa* e *ambiente* para cada objeto de modo que seja possível calcular a contribuição de cada objeto para o ponto na janela de observação. O processo de cálculo da cor de um raio de luz partindo de um determinado ponto é classicamente denominado sombreamento (*shading*).

Esta é uma descrição bastante breve da técnica de *Ray Tracing*. Como não é objetivo deste trabalho apresentar detalhadamente esta técnica, mas sim apresentar o processo de integração da modelagem por sistemas de partículas com a técnica de *Ray Tracing*, o leitor pode obter informações mais detalhadas em [GLA 89].

3.3 Sombreamento

Existem vários modelos de iluminação elaborados para efetuar o sombreamento em um sistema de *Ray Tracing*. Dentre eles, os modelos mais conhecidos são Phong [BUI 75], Whitted [WHI 80], Cook-Torrance [COO 82] e Hall [HAL 89]. Porém, o modelo mais utilizado em sistemas de *Ray Tracing* é o modelo *Phong*, basicamente devido à sua facilidade de implementação e à sua facilidade de controle (devido ao número reduzido de parâmetros). Este modelo apresenta um controle suficientemente simples sobre os seus parâmetros.

Considere a figura 3.4, onde $\vec{n}$ é o vetor normal à superfície no ponto *P*, $\vec{l}$ é o vetor que aponta para a fonte de luz, $\vec{v}$ é o vetor que aponta para o observador e $\vec{r}$ é o vetor refletido do vetor $\vec{v}$. Segundo Foley [FOL 90], o modelo de iluminação *Phong* modela o efeito de sombreamento através da seguinte expressão:

$$I_\lambda = I_a k_a O_{d\lambda} + \sum_{i=0}^{m} f_{att_i} I_{p\lambda_i} \left[k_d O_{d\lambda} \left(\vec{n} \cdot \vec{l}_i \right) + k_s O_{s\lambda} \left(\vec{r} \cdot \vec{l}_i \right)^n \right] + k_s I_{r\lambda} + k_t I_{t\lambda}$$

onde I_λ é a intensidade do sombreamento no ponto *P* em questão, *m* é o número de fontes de luz da cena, $I_{p\lambda_i}$ é a intensidade da *i*-ésima fonte de luz, f_{att_i} é o fator de atenuação para a *i*-ésima fonte de luz, $I_{r\lambda}$ é a intensidade da luz refletida pelo objeto e $I_{t\lambda}$ é a intensidade transmitida pelo objeto. Ainda, quanto aos parâmetros específicos do objeto, $I_a k_a$ é uma constante que modela a componente ambiente da superfície, k_d é a coeficiente de reflexão difusa (valor entre 0 e 1), k_s é o coeficiente de reflexão especular, k_t é o coeficiente de transparência do objeto, $O_{d\lambda}$ é a cor difusa do objeto e $O_{s\lambda}$ é a cor especular do objeto. O símbolo λ em alguns coeficientes e intensidades significa que estes são dependentes de um determinado comprimento de onda, como por exemplo, vermelho, verde ou azul.

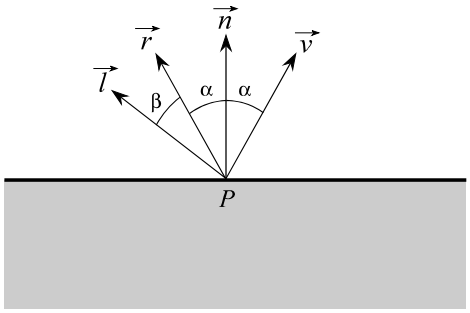

Figura 3.4 - Raios de reflexão, iluminação (detecção de sombra), observação e vetor normal em uma determinada superfície.

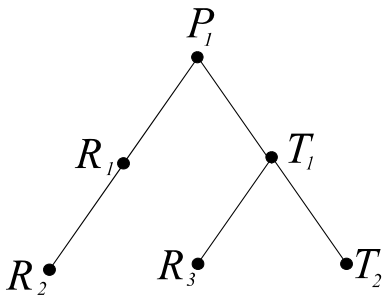

Figura 3.5 - Árvore de raios emitidos da situação apresentada na figura 3.3.

Esta expressão modela a recursividade existente no algoritmo básico de *Ray Tracing*, uma vez que para determinar a intensidade em um determinado ponto, é necessário levar em consideração a intensidade do raio refletido, assim como a intensidade do raio transmitido. Deste modo, na figura 3.3 pode-se constatar que para obter a intensidade no ponto P_1 deve-se obter as intensidades dos raios R_1 e T_1, que por sua vez, necessitam obter as intensidades de R_2, R_3 e T_2, e assim por diante. Considerando-se a situação da figura 3.3, pode-se construir uma árvore de raios emitidos como mostra a figura 3.5.

Ainda, considerando-se a importância da reflexão e da transmissão em um sistema de *Ray Tracing*, duas outras equações devem ser abordadas de maneira especial.

A primeira é a equação para a obtenção da direção do raio refletido, dada por:

$$\vec{r} = \vec{v} + 2\vec{n}\left(\vec{v}\cdot\vec{n}\right)$$

onde $\vec{r}$ tem a mesma direção que a do raio refletido, $\vec{v}$ tem a mesma direção que a do raio incidente e $\vec{n}$ é o vetor normal à superfície no ponto em questão [GLA 89].

A segunda é a equação para a obtenção da direção do raio transmitido, com refração, que por sua vez, é definida por:

$$\vec{t} = \eta_{it}\,\vec{v} + \left(\eta_{it}C_i - \sqrt{1+\eta_{it}^{\,2}\left(C_i^{\,2}-1\right)}\right)\vec{n}$$

onde $\vec{t}$ tem a mesma direção que a do raio transmitido com refração, $\vec{v}$ tem a mesma direção que a do raio incidente, $\vec{n}$ é o vetor normal à superfície no ponto em questão, $C_i = -\vec{v}\cdot\vec{n}$ e η_{it} é a razão entre o índice de refração do meio incidente (meio de onde o raio incidente provém) e o índice de refração do meio transmitido (meio para o qual o raio incidente é transmitido) [GLA 89].

3.4 Métodos para a Aceleração de *Ray Tracing*

Um dos maiores problemas de um sistema de *Ray Tracing* é o alto consumo computacional que ele demanda. Devido a este problema, diversos métodos para acelerar o tempo de processamento exigido para a geração de uma imagem em um sistema de *Ray Tracing* foram desenvolvidos através de várias abordagens, como por exemplo, estruturas de dados, métodos numéricos, geometria computacional, métodos estatísticos, computação distribuída, entre outras.

Apesar da grande dificuldade de se definir uma taxinomia geral quanto aos métodos para a aceleração de *Ray Tracing*, Andrew Glassner [GLA 89], apresentou uma

taxinomia bastante efetiva que engloba a maior parte das técnicas para a aceleração de *Ray Tracing*. De acordo com a classificação apresentada por Glassner, os métodos para aceleração de *Ray Tracing* podem ser divididos em três categorias principais: *redução do custo médio da intersecção de um raio com a cena*, *redução do número total de raios que interceptam a cena* e *substituição dos raios individuais por uma entidade mais genérica*. Ainda, a primeira categoria é subdividida em duas: *utilizando intersecções mais rápidas entre raio e objeto* e *efetuando menos intersecções entre raio e objeto*. Esta subdivisão de categorias pode ser vista de uma maneira mais estruturada na figura 3.6.

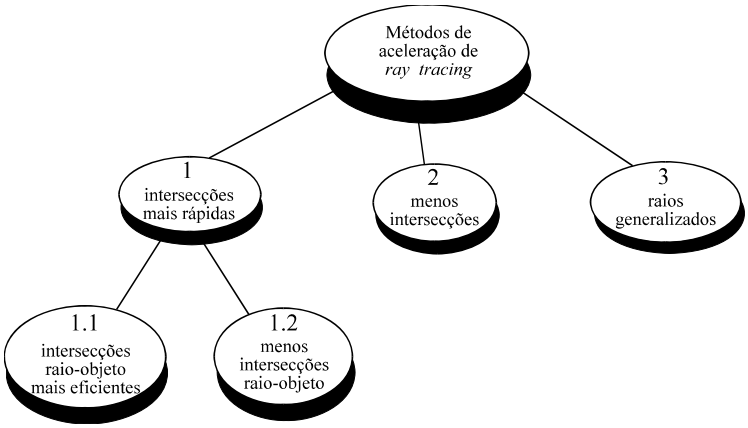

Figura 3.6 - Classificação dos métodos para aceleração de *Ray Tracing*.

Citando alguns exemplos para localizar melhor os métodos, na categoria 1.1 encontram-se o uso de *Bounding Volumes* ou envelopes [WHI 80] e o uso de intersecções eficientes para superfícies parametricamente definidas, superfícies fractais, entre outras. Já na categoria 1.2., encontram-se os métodos que utilizam hierarquias de envelopes [RUB 80] [WEG 84], subdivisões espaciais [GLA 84] [KAP 85] [FUJ 86] e técnicas direcionais [ARV 87]. Na categoria 2, estão os métodos com controle adaptativo da profundidade máxima da árvore de raios emitidos [HAL 83], assim como os métodos com otimizações estatísticas [COO 86]. Finalmente, na categoria 3, encontram-se os métodos que utilizam, por exemplo, *Beam Tracing* [HEC 84], *Cone Tracing* [AMA 84] e *Pencil Tracing* [SHI 87].

Dentre os métodos citados, os que apresentam resultados mais significativos e que são mais utilizados são os métodos que empregam envelopes ou hierarquias de envelopes, subdivisões espaciais e técnicas direcionais. Isto é devido ao fato que a maior parte do tempo de processamento parar a geração de uma imagem é gasto no cálculo das intersecções entre os raios e os objetos da cena. Ainda, considerando-se que o tempo de processamento para a geração de uma imagem de uma determinada cena é proporcional ao quadrado do número de objetos primitivos que esta cena possui, quanto maior a quantidade de objetos na cena, maior é o tempo de processamento. Deste modo, quando a quantidade de objetos em uma cena é grande, existe a necessidade de diminuir a quantidade total de intersecções necessárias para sintetizar a imagem. Os métodos que utilizam subdivisão espacial são normalmente subdivididos em duas classes: através de subdivisão espacial

uniforme e subdivisão espacial não-uniforme. Também dentre os métodos que utilizam técnicas direcionais, o método através da classificação de raios possui destaque especial. Este três métodos serão analisados em maior detalhe a seguir.

3.4.1 Subdivisão Espacial Não Uniforme

Os métodos de subdivisão espacial não uniforme dividem o espaço tridimensional em regiões com um tamanho variável e dependente da características da cena. Deste modo, pode-se efetuar um número maior de subdivisões nas áreas que apresentam uma quantidade maior de objetos, deixando as regiões com uma quantidade menor de objetos ou sem objetos com poucas subdivisões.

Glassner [GLA 84] introduziu um método que utiliza *octrees* para efetuar a subdivisão do espaço. Neste método cada célula é subdividida em oito novas células até que se obtenha uma lista de objetos associados à célula menor que um determinado limiar. Deste modo se obtém uma quantidade aproximadamente uniforme em cada célula elementar da *octree*.

Também, Kaplan [KAP 85] *apud* [GLA 89] introduziu um método similar baseado em *BSP-tree* (*binary space partitioning trees*). Nesta abordagem, o espaço é subdividido em duas partes através de um plano divisor e assim por diante recursivamente.

Estes métodos apresentam a vantagem de subdividir o espaço tridimensional uniformente, porém apresentam a desvantagem que os algoritmos de percurso da *octree* e da *BSP-tree* são relativamente complexos e demorados.

3.4.2 Subdivisão Espacial Uniforme

Esta abordagem, inicialmente introduzida por Fujimoto *et al*. [FUJ 86], apresenta o espaço discretizado em uma grade retangular tridimensional, sendo associado a cada *voxel* os objetos cuja superfície o intercepta.

Esta abordagem apresenta a desvantagem de ser independente da estrutura da cena, sendo portanto suscetível ao efeito "*teapot in a statium*", onde existe um objeto complexo ocupando poucos *voxeis* em uma cena simples e grande. A grande vantagem, porém, desta abordagem, é o fato que os algoritmos de percurso da grade tridimensional são simples, rápidos e podem ser efetuados incrementalmente, como feito por Amanatides [AMA87].

3.4.3 Classificação de Raios

O algoritmo de classificação de raios apresentado por Arvo e Kirk [ARV 87] é baseado na observação que os raios em um espaço tridimensional possuem cinco graus de liberdade (três para posição e dois para direção). Deste modo, o algoritmo efetua o particionamento do espaço pentadimensional dos raios emitidos para cena em pequenos hipercubos pentadimensionais, associando uma lista de objetos candidatos a cada um. Cada hipercubo pentadimensional representa um conjunto de raios com mesma origem e direções similares, e, na sua lista de candidatos, contém os objetos que podem ser atingidos por quaisquer desses raios.

3.5 Conclusões

Este capítulo descreve os principais aspectos que devem ser considerados no processo de geração de imagens através de *Ray Tracing*, tais como, sombreamento, iluminação, recursividade, intersecções e métodos para a aceleração.

Dentre tais aspectos, deve-se observar que um dos principais problemas da técnica de *Ray Tracing* é o alto consumo computacional demandado, uma vez que o tempo de processamento em um sistema comum de *Ray Tracing* é proporcional ao quadrado do número de objetos na cena. Deste modo, quando a quantidade de objetos existentes na cena é grande, é indispensável o uso de um ou mais métodos para aceleração de *Ray Tracing*.

4 Proposta para Integração de Sistemas de Partículas em Ambientes de *Ray Tracing*

Como já foi exposto no segundo capítulo, um dos principais problemas apresentados pela modelagem de objetos através de sistemas de partículas é a obtenção de efeitos de reflexão, transparência e sombreamento. Até o momento, com base na bibliografia pesquisada, nenhum trabalho foi realizado com a intenção de avaliar especificamente o problema da integração de sistemas de partículas e ambientes de *Ray Tracing*. Alguns trabalhos, tal como o apresentado por *Kajiya* [KAJ 84], efetuam a síntese de imagens de fenômenos naturais através de um sistema de *Ray Tracing*, porém a técnica de modelagem utilizada não é sistemas de partículas.

Este capítulo tem, então, o principal objetivo de abordar a técnica desenvolvida para a integração de sistemas de partículas em um sistema de *Ray Tracing*, de modo eficiente e expansível a modificações. Este problema merece particular atenção devido ao fato de estar sendo proposta uma técnica que seja eficiente para a síntese de imagens de sistemas de partículas com uma quantidade expressiva de partículas, como por exemplo, centenas de milhares ou milhões de partículas.

Ainda, outro problema que merece atenção é a quantidade de memória exigida para o armazenamento do sistema de partículas. Apesar dos sistemas de computação pessoal atualmente apresentarem grandes capacidades de memória, quando se trabalha com uma quantidade muito grande de partículas, esta capacidade pode se esgotar. Deste modo, é necessário tratar este problema de modo que o sistema de partículas exija poucos requisitos de memória, as suas aproximações não afetem o objeto que se quer modelar e não seja computacionalmente custosa.

Assim sendo, este capítulo também introduz os procedimentos utilizados para a obtenção de um uso eficiente de memória para a armazenagem de um sistema de partículas, através de aproximações, mas de baixo custo computacional. Também, são expostos alguns métodos alternativos, com um custo computacional mais elevado, porém com algumas outras vantagens.

4.1 Algoritmo para a Integração de Sistemas de Partículas e Ambientes de *Ray Tracing*

Considerando-se que, em um sistema de *Ray Tracing*, o tempo de processamento para a síntese de uma imagem é proporcional ao quadrado da quantidade de objetos na cena, o fato de uma determinada cena possuir um sistema de partículas que, por sua vez, possui um conjunto enorme de primitivas, torna o tempo de processamento para a geração da imagem desta cena extremamente grande. Devido a este fato, técnicas especiais

para a aceleração de sistemas de *Ray Tracing* devem ser utilizadas de modo a tornar viável o uso de sistemas de partículas.

O método propõe o uso da subdivisão espacial uniforme do espaço tridimensional onde o sistema de partículas se encontra como uma técnica eficiente para tornar viável o uso de sistemas de partículas em sistemas de *Ray Tracing*.

A técnica consiste em subdividir o espaço tridimensional onde o sistema de partículas se encontra em uma grande quantidade de células com dimensões uniformes, associando-se a cada célula um conjunto de partículas. Esta associação é efetuada com base na posição que uma determinada partícula possui. Assim sendo, pode-se considerar o sistema de partículas como sendo uma grade tridimensional, onde cada célula possui uma lista de partículas que estão contidas nesta célula.

A associação de uma lista de partículas a cada célula da grade tridimensional acelera enormemente o cálculo de intersecção entre o sistema de partículas e um raio, uma vez que somente as partículas contidas nas células atravessadas são consideradas. Assim, sendo, durante a execução do sistema de *Ray Tracing*, sempre que for necessário efetuar o cálculo da menor intersecção entre um raio e o sistema de partículas, ao invés de se calcular a intersecção do raio com todas as partículas do sistema, somente as partículas contidas nas células atravessadas pelo raio são consideradas. Este procedimento reduz consideravelmente o tempo de processamento na síntese de uma cena com um sistema de partículas uma vez que o número de primitivas envolvidas é consideravelmente menor.

Para ilustrar melhor a técnica, considere o caso bidimensional exposto na figura 4.1. Neste exemplo, o espaço bidimensional ocupado pelo sistema de partículas é dividido em uma grade com dimensões 8 x 8, ou seja, contendo 64 células. Especificamente, o sistema de partículas contém 128 partículas uniformemente distribuídas na grade bidimensional, com uma média de 2 partículas por célula. Deste modo, em um sistema tradicional sem nenhum método de aceleração, para cada raio que é disparado na cena é necessário efetuar 128 cálculos de intersecção entre um raio e uma partícula. Para o raio apresentado na figura, entretanto, é necessário efetuar apenas 22 cálculos de intersecção entre um raio e uma partícula. Isto incorre uma redução de aproximadamente 83% do tempo de processamento para a situação deste raio.

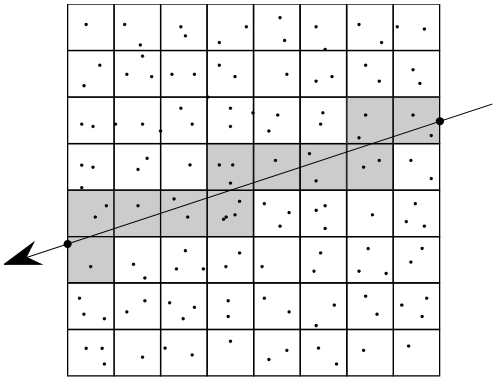

Figura 4.1 - Raio interseccionando uma grade bidimensional contendo partículas.

Porém este fator de aceleração pode aumentar ou diminuir, uma vez que o raio da situação anterior atravessou apenas 11 células, ao passo que um raio pode atravessar até 15 células (dada a grade com dimensões de 8 x 8), como na figura 4.2, ou então apenas uma ou mesmo nenhuma célula.

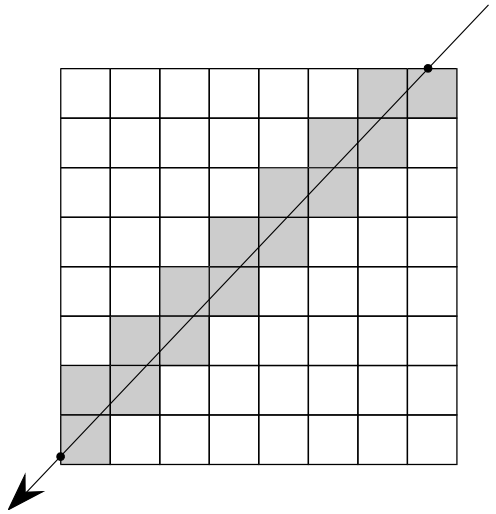

Figura 4.2 - Raio atravessando 15 células.

A introdução deste método de aceleração conduz a três alterações no algoritmo básico de *Ray Tracing*. A primeira alteração deve ser efetuada no procedimento que calcula a intersecção entre um raio e o sistema de partícula. A segunda diz respeito ao processo de mudança da cor do raio à medida que este atravessa o sistema de partículas. Por fim, uma terceira alteração é necessária no algoritmo de iluminação que efetua a determinação da intensidade de atenuação devido ao fato de um objeto estar sob influência de sombras.

4.2 Processo de Intersecção Entre Um Raio e Um Sistema de Partículas

Na primeira alteração, onde deve-se calcular a menor intersecção entre um sistema de partículas e o raio proveniente do ponto de observação, os seguintes procedimentos devem ser realizados:

❶ Inicialmente deve-se calcular as intersecções inicial e final com a caixa ou ortoedro que define os limites da grade tridimensional que contém o sistema de partículas;

❷ Se o ponto de observação estiver contido na grade tridimensional, à intersecção inicial deve ser atribuído o valor zero. Considerando-se que quando o ponto de observação está dentro da grade tridimensional, existem uma intersecção com valor positivo e outra com valor negativo, a intersecção com valor negativo deve ser reposicionada no início do percurso, como pode ser examinado na figura 4.3. Se tal procedimento não for efetuado o percurso da grade iria começar em t_0 e iria até t_2, o que é incorreto. O correto é que o percurso comece em t_1 e termine em t_2;

❸ Após determinadas as intersecções inicial e final, deve-se então percorrer as células da grade tridimensional até que seja encontrada uma célula que possua uma ou mais partículas. Este procedimento pode ser efetuado de maneira incremental, como feito por Amanatides [AMA 87], ou através da especificação de um incremento fixo, como feito por Sakas [SAK 90]. A implementação realizada utiliza um incremento fixo devido ao fato de ser mais facilmente implementado e pelo fato de ser mais eficiente para a criação de previsões de cenas e/ou animações;

❹ Finalmente, deve-se determinar a menor distância do ponto de observação às partículas contidas na célula. Esta distância mínima é então processada com uma candidata à distância mínima. Caso nenhuma célula possua alguma partícula, de modo que a distância t_2 é alcançada, o raio não intercepta o sistema de partículas e, assim sendo, nenhuma intersecção necessita ser processada.

Estes passos podem ser estruturados de forma algorítmica e mais detalhada da seguinte maneira:

```
ParticleSystemIntersect(OBJECT, POS, RAY)
Início
  Calcular a intersecção T1 e T2 entre o raio definido por POS e
  RAY e a caixa especificada em OBJECT

  Se houve intersecções então
    Se o ponto de observação é interno à grade então
      Transformar a intersecção negativa para zero
    FimSe
```

```
  FimSe

  Definir um incremento STEP_T

  Para T = T1 até T2 com incremento STEP_T faça
    Atualizar a posição P = POS + T * RAY
    Se o ponto P está fora da grade então
      Sair do laço
    FimSe

    Achar a posição X, Y e Z na grade para o ponto P

    Se a célula definida por X, Y e Z não foi visitada então
      Para cada partícula da célula faça
        Se a partícula é interceptada pelo  raio então
          Determinar a distância DIST entre o ponto de
          observação e a partícula
        FimSe
      FimPara
    FimSe
  FimPara
  Se houve intersecção então
    Processar intersecção DIST
  FimSe
Fim
```

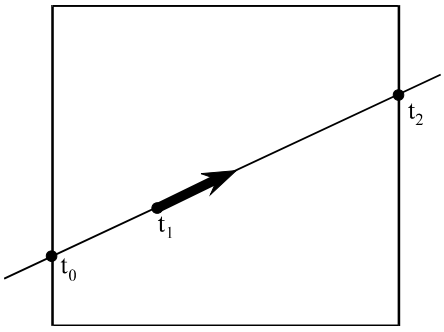

Figura 4.3 - Situação com o ponto de observação dentro da grade.

4.3 Alteração da Cor do Raio

No segundo procedimento que deve ser modificado, onde a cor do raio é modificada à medida que este percorre a grade tridimensional e é afetado pelas partículas, os seguintes procedimentos devem ser avaliados:

❶ Inicialmente, deve-se obter a segunda menor intersecção com os objetos da cena. Este cálculo é necessário, pois serão processadas as partículas contidas nas células entre a menor distância e a segunda menor distância;

❷ Efetuar o percurso das células entre a menor distância e a segunda menor distância obtidas anteriormente;

❸ Para cada célula atravessada pelo raio, calcular a contribuição de cada partícula para a mudança da cor do respectivo raio. Também, deve ser avaliada a alteração da opacidade acumulada do sistema de partículas;

❹ Se a opacidade acumulada ultrapassar o limiar mínimo, deve-se interromper o percurso da grade tridimensional;

❺ Se alguma das componentes R, G ou B do raio ultrapassar o valor máximo, deve-se parar o percurso da grade tridimensional. Tal procedimento é efetuado para evitar a saturação das componentes R, G e B.

❻ Após calcular a contribuição para a cor do raio, deve-se continuar o procedimento recursivo do algoritmo de *Ray Tracing*, efetuando-se o cálculo de sombreamento do objeto seguinte.

Igualmente, tais procedimentos podem ser estruturados de forma algorítmica e mais detalhada da seguinte maneira:

```
FindColor(COLOR, POS, RAY)
Início
  Se o objeto mais próximo não for um Sistema de Partículas então
    Calcular o ponto de intersecção HIT
    Achar o vetor normal em HIT
    Efetuar o procedimento de sombreamento de acordo com as
    propriedades do objeto sólido
  Senão
    Atribuir a menor intersecção já calculada à S1
    Calcular a segunda menor intersecção S2
    Definir um incremento STEP_T
    Para cada T = S1 até S2 com incremento STEP_T faça
      Atualizar a posição P = POS + T * RAY
      Se o ponto P está fora da grade então
        Sair do laço
      FimSe

      Achar a posição X, Y e Z na grade para o ponto P

      Se a célula definida por X, Y e Z não foi visitada então
        Para cada partícula da célula faça
          Calcular a contribuição da partícula para mudança da
          cor do raio
          Se a opacidade atingiu o limiar mínimo então
            Sair do laço
          FimSe
          Se alguma das componentes R, G ou B saturou então
            Sair do laço
          FimSe
          Acumular a contribuição da partícula em COLOR
```

```
        FimPara
      FimSe
    FimPara
    Calcular a próxima intersecção
    Se algum objeto foi interceptado então
      FindColor(NEW_COLOR, POS, RAY)
      COLOR = COLOR + NEW_COLOR
    Senão
      COLOR = COLOR + BACKGROUND_COLOR
    FimSe
  FimSe
Fim
```

4.3.1 Cálculo da Contribuição das Partículas

Um tópico que merece destaque além do processo de cálculo de intersecção entre uma partícula e um raio é o processo de cálculo da contribuição de uma partícula individual para a composição da cor do raio.

Um procedimento normal para estes aspectos seria calcular a menor distância de uma partícula ao raio e verificar se esta distância é menor que uma determinada distância mínima que especifica o tamanho da partícula. Deste modo pode-se utilizar as seguintes equações:

$$C_r = C_r + C_p . T_p \qquad \text{Se } d < r_p$$

onde C_r é a cor do raio, C_p é a cor da partícula, T_p é a transparência da partícula e r_p é o parâmetro que define o tamanho da partícula.

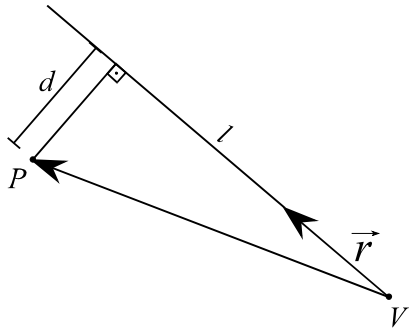

Figura 4.4 - Distância de uma partícula a um raio.

Considerando-se a figura 4.4, a distância *d* pode ser calculada utilizando-se a fórmula de Pitágoras, de modo que:

$$\left|\overrightarrow{VP}\right|^2 = d^2 + l^2$$

.

Ainda, considerando-se que $\vec{r}$ é um vetor normalizado, a distância *l* pode ser obtida através do produto escalar entre os vetores $\vec{VP}$ e $\vec{r}$, de modo que

$$d^2 = \left|\vec{VP}\right|^2 - \left(\vec{VP} \cdot \vec{r}\right)^2 \quad \text{,onde}$$

$$\vec{VP} = P - V \cdot$$

Porém, tal procedimento pode produzir partículas como se fossem círculos com bordas bem definidas, gerando um efeito conhecido como *aliasing*. Assim sendo, este trabalho propõe o uso de uma função auxiliar de modulação de transparência de modo a produzir partículas que apresentem bordas mais suaves. Deste modo, o cálculo final da contribuição de uma partícula para a mudança da cor do raio é dada por

$$C_r = C_r + C_p . T_p . Tr(d)$$

onde a função para modulação de transparência $Tr(d)$ é dada por

$$Tr(d) = At^{\left(\frac{d}{d_{At}}\right)^2} .$$

Esta função foi desenvolvida pelo autor de modo a solucionar o efeito de *aliasing* que comumente ocorre na síntese de imagens de sistemas de partículas, especialmente quando as partículas ocupam uma área da tela de vídeo proporcionalmente grande, comparada à área total da tela de vídeo. Deste modo, foi proposto o uso de uma função gaussiana com a distância da partícula ao raio como uma variável independente para se obter o efeito de atenuação desejado.

Esta função apresenta a propriedade de ser semelhante à uma função gaussiana, sendo que $Tr(d)$ retorna um valor entre 0 e 1. Ainda, os parâmetros *At* e d_{At} servem para especificar a forma da curva gaussiana que define a função de modulação, conforme é mostrado na figura 4.5. Estes parâmetros possuem o seguinte significado: *At* é a atenuação efetuada pela função de modulação quando a distância for d_{At}. Estes parâmetros são necessários para definir indiretamente o tamanho da partícula, uma vez que são necessários dois pontos para a definição da curva gaussiana em questão. O primeiro ponto é implícito e especifica que quando a distância for 0 não ocorre atenuação pela função de modulação, ou seja, $Tr(d) = 1$.

Esta função de modulação, quando comparada com o uso simples de uma distância mínima, eliminou os efeitos de *aliasing* na síntese de sistemas de partículas através de um sistema de *Ray Tracing*, apresentando uma transição suave da cor da partícula para a cor de fundo.

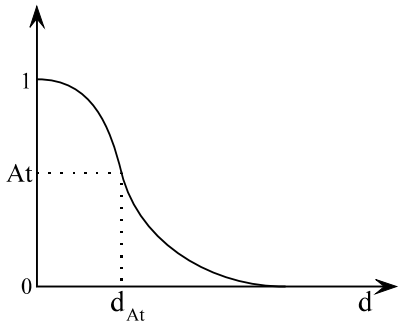

Figura 4.5 - Gráfico da variação da função de modulação
em função da distância da partícula ao raio.

4.4 Iluminação e Detecção de Sombras

O terceiro procedimento que necessita ser alterado para efetuar a completa integração dos sistemas de partículas em um ambiente de *Ray Tracing* diz respeito ao algoritmo de iluminação que determina a intensidade de atenuação da intensidade luminosa à medida que o raio atravessa o sistema de partículas. Este processo passa então a apresentar os seguintes procedimentos:

❶ Obter a menor e a segunda menor intersecção do sistema de partículas com o raio, partindo do ponto onde se quer verificar a iluminação na direção da fonte de luz;

❷ Deve-se então percorrer as células da grade tridimensional e acumular a opacidade das partículas encontradas nestas células;

❸ Após obtida a opacidade acumulada do sistema de partículas entre a menor e a segunda menor intersecção, deve-se atenuar a fonte de luz de acordo com este fator de opacidade;

❹ Se uma das intersecções for de um objeto sólido, deve-se considerar o caso de sombra total.

Novamente, estes procedimentos são mais detalhados de acordo com o seguinte algoritmo:

```
Brightness(LIGHT, POS, RAY)
Início
  Obter a menor intersecção S1
  Se S1 for menor que a distância à fonte de luz então
    Se o objeto não for um sistema de partículas então
      Ponto apresenta sombra total
    Senão
      Obter a segunda menor intersecção S2 ignorado o sistema de
```

```
        partículas
        Se S2 for menor que a distância à fonte de luz então
          Se interceptou algum objeto então
            Ponto apresenta sombra total
          FimSe
        FimSe

        Definir um incremento STEP_T

        Para cada T = S1 até S2 com incremento STEP_T faça
          Atualizar a posição P = POS + T * RAY
          Se o ponto P está fora da grade então
            Sair do laço
          FimSe

          Achar a posição X, Y e Z na grade para o ponto P

          Se a célula definida por X, Y e Z não foi visitada então
            Para cada partícula da célula faça
              Acumular a opacidade da partícula
              Se a opacidade atingiu o limiar mínimo então
                Sair do laço
              FimSe
            FimPara
          FimSe
        FimPara
      FimSe
    FimSe

    Se S1 for menor que a distância à fonte de luz então
      Iluminação no ponto é LIGHT.Bright*opacidade
    Senão
      Iluminação no ponto é LIGHT.Bright
    FimSe
  Fim
```

Pode-se observar as diferentes situações apresentadas na figura 4.6, onde os pontos P_1 e P_2 apresentam uma sombra total (intensidade da fonte de luz igual a zero). Também, o ponto P_3 apresenta uma sombra parcial (intensidade da fonte de luz atenuada pela opacidade acumulada do sistema de partículas) e o ponto P_4 que não apresenta sombra (intensidade da fonte de luz original).

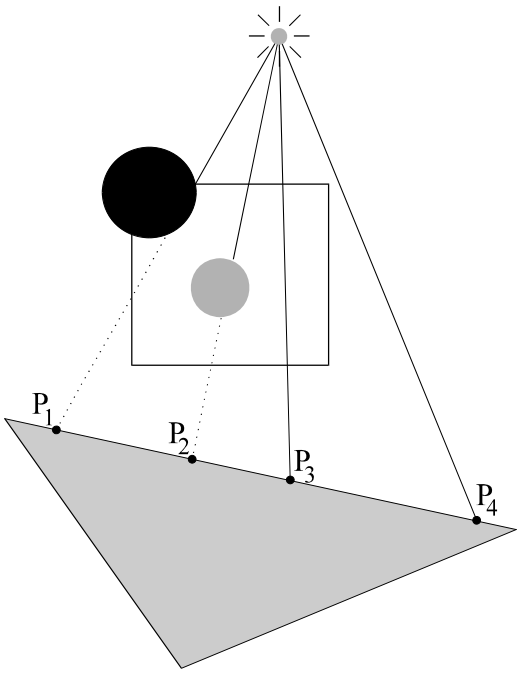

Figura 4.6 - Situações de iluminação afetadas por um sistema de partículas.

4.5 Redução do Consumo de Memória

Um último problema que deve ser considerado no processo de integração de sistemas de partículas e ambientes de *Ray Tracing* é o consumo de memória demandado. Neste processo deve levar em consideração que todas as partículas estão armazenadas em um arquivo e que elas devem ser carregadas para a memória.

Para se fazer uma breve verificação, pode-se considerar um sistema de partículas com 500 mil partículas, onde cada partícula necessita os atributos posição, cor e transparência para ser sintetizada. Nesta situação, normalmente, a posição é representada em 24 bytes, a transparência por 8 bytes e a cor por 3 bytes, totalizando 35 bytes por partícula. Isto representa um total de aproximadamente 17 megabytes para armazenar todo o sistema de partículas.

Este trabalho propõe o uso de representações inteiras para armazenar os valores de pontos flutuantes. Esta técnica é bastante utilizada na programação de sistemas que exigem manipulação eficiente e em tempo real.

Deste modo, através de representações inteiras de valores de ponto flutuantes, pode-se reduzir a representação da posição para 6 bytes, da transparência para 1 byte, permanecendo a cor representada por 3 bytes, totalizando 10 bytes. Com isso, pode-se armazenar o mesmo sistema de partículas com 500 mil partículas com menos 5 megabytes. Uma comparação completa do tamanho de todos os possíveis atributos de uma partícula,

bem como do tamanho dos atributos na sua representação inteira, é mostrada na tabela da figura 4.7. Observe que são apresentados todos os atributos utilizados no processo de animação e não somente os atributos utilizados no sistema de *Ray Tracing*.

	Tamanho normal	Tamanho reduzido
Posição	**24**	**6**
Velocidade	**24**	**6**
Cor	**3**	**3**
Transparência	**8**	**1**
Tempo de Vida	**8**	**2**
Próxima Partícula	**4**	**3**
Total	**71**	**21**

Figura 4.7 - Comparação do tamanho dos atributos na representação normal e inteira.

O método consiste simplesmente de utilizar 1 byte para a representação da parte inteira do número e 1 byte para a representação da parte decimal. Para isto, deve-se inicialmente especificar um domínio onde os valores são válidos e utilizar funções simples de divisão e resto de divisão para a obtenção da parte inteira e fracionária.

Como será apresentado em maiores detalhes no capítulo sobre a análise dos resultados, o uso de representações inteiras não causa nenhum problema significativo no processo de síntese dos sistemas de partículas. Isto também é devido ao fato de sistemas de partículas apresentarem perturbações estocásticas na sua forma.

Uma metodologia alternativa para uma representação mais compacta é o uso de coordenadas esféricas para a representação dos pontos e vetores tridimensionais. Neste método, os pontos tridimensionais são convertidos para as respectivas representações em coordenadas esféricas que por sua vez são convertidas para valores inteiros.

Deste modo, considerando-se que um ponto tridimensional é agora representado por uma distância e dois ângulos, e que os ângulos possuem um domínio fixo entre 0 e 360 graus, pode-se reduzir a quantidade de informação necessária para armazenar o ponto tridimensional.

Deste modo, utilizando-se um ponto tridimensional representado em coordenadas esféricas, pode-se utilizar somente 5 bytes para representá-lo, usando dois bytes para a distância e os três bytes restantes para os dois ângulos. Isto implica o uso de 12 bits para cada ângulo, ou seja, uma amostragem a cada 0°5'16".

Um problema para tal representação é que ela requer que os valores seja constantemente convertidos da notação esférica para a notação cartesiana, o que aumenta bastante o custo computacional para a síntese da imagem. Ainda, devido ao fato da representação esférica não ser uniforme em todo o espaço discretizado, tal como ocorre na no uso da representação inteira simples, ocorre um fator de erro máximo menor na representação esférica do que na representação inteira, apesar do fator de erro médio ser menor na representação inteira.

4.6 Conclusões

Neste capítulo foi visto como pode ser realizada a integração de sistemas de partículas em ambientes de *Ray Tracing* de maneira eficiente, levando em consideração a solução dos problemas de obtenção de efeitos de reflexão, transparência e sombreamento. Também foi exposto como pode ser solucionado o problema do alto consumo de memória para a armazenagem dos sistemas de partículas.

Este capítulo apresentou como a técnica de subdivisão espacial tridimensional pode ser utilizada de modo eficiente para a redução significativa do número de cálculos de intersecção entre raio e partícula. Também, foram apresentados os principais módulos de um sistema de *Ray Tracing* que devem ser alterados de modo a incorporar as alterações para a integração de sistemas de partículas.

Foi visto como uma função simples de modulação de transparência pode afetar a qualidade das imagens geradas, sem que com isto afete significativamente o tempo de processamento para a síntese da imagem.

Finalmente, o capítulo apresentou como o consumo de memória pode ser significativamente reduzido através de uma técnica simples e conhecida na área de produção de jogos, no caso a técnica de representação inteira e redução do domínio de valores em ponto flutuante. Mostrou também que o consumo pode ser ainda mais reduzido através do uso da representação dos vetores e pontos tridimensionais em coordenadas esféricas.

5 Proposta para a Detecção de Colisões Entre Sistemas de Partículas e Objetos Sólidos

Como foi apresentado no segundo capítulo, o processo de animação e desenvolvimento de sistemas de partículas ao longo de um determinado período, normalmente, não considera a interferência dos demais objetos que se encontram no ambiente. Porém, quando se tenta considerar tais interferências, o processo de detecção de colisões de todos os elementos do sistema de partículas com todos os objetos da cena torna-se computacionalmente custoso. Tal problema é ainda mais explícito quando o número de partículas de um sistema é consideravelmente alto.

Novamente, como na situação do capítulo anterior, poucos trabalhos foram realizados com a intenção de tratar de forma mais eficiente o processo de interação entre sistemas de partículas e objetos sólidos. Yoshiaki Takai *et al.* [TAK 95] utilizam um processo de subdivisão espacial uniforme, onde cada célula possui um estado de transição e, com base neste estado e nos estados das células vizinhas, um processo estatístico detecta a colisão das partículas da célula com as partículas das suas células vizinhas. Esta técnica possui várias desvantagens, como ser viável apenas para sistemas bidimensionais e ser extremamente custosa para sistemas tridimensionais devido à quantidade de atributos que devem ser associados a cada célula. Ainda, o processo de detecção de colisões é altamente estatístico, apesar de apresentar efeitos visuais suficientes. Porém, não é adequada para o uso em sistemas de simulação.

Este trabalho apresenta uma técnica para a detecção eficiente de colisão entre um sistema de partículas e os objetos sólidos da cena. Dentro da bibliografia pesquisada, nenhuma técnica semelhante foi encontrada. Esta técnica é eficiente mesmo para um número expressivo de partículas, como por exemplo, centenas de milhares de partículas. Ainda, considerando-se que o processo de animação de sistemas de partículas é normalmente específico para cada fenômeno que se quer modelar, o algoritmo de detecção de colisão necessita ser de fácil adaptação a cada algoritmo de animação de sistemas de partículas sem que seja preciso alterá-lo. O algoritmo apresentado é suficientemente genérico de modo a ser necessária somente a alteração no processo de animação sem serem necessárias quaisquer alterações no algoritmo de detecção de colisão.

5.1 O Método para Detecção de Colisões

Considerando-se a obtenção de um determinado quadro da animação, é simples observar que o tempo de processamento para o tratamento de colisões é proporcional ao número de objetos multiplicado pelo número de partículas. Ainda, considerando-se que o número de partículas pode estar na faixa de centenas de milhares de partículas, com apenas uma dezena de objetos na cena, o número de cálculos de intersecção fica na faixa de milhões de intersecções entre uma partícula e um objeto por quadro da animação. Isto é

relativamente custoso, especialmente se a animação apresentar um número expressivo de quadros.

Uma boa maneira de se acelerar este processo é reduzir o número de partículas que devem ser consideradas no cálculo de intersecção entre partícula e objeto sólido. O método proposto neste trabalho utiliza uma grade pentadimensional de modo a efetuar uma seleção prévia das partículas que possuem alguma possibilidade de interceptar algum objeto da cena.

O método é baseado na observação que, na maior parte de uma animação com sistemas de partículas, as partículas não estão colidindo com os objetos da cena. Também, na maioria dos fenômenos modelados as partículas se movem em uma mesma direção base, com algumas perturbações na direção. Algumas exceções são os fogos de artifício e explosões em geral. Porém, sistemas de partículas são comumente utilizados para a modelagem de fenômenos naturais como fogo, fumaça, neblina, entre outros, sendo que nestas situações as premissas apresentadas são verdadeiras.

Basicamente, o método utiliza uma grade pentadimensional para acelerar a detecção de intersecção de um raio definido pelo movimento de uma partícula e os demais objetos da cena. O uso de grades pentadimensionais como uma técnica de aceleração foi inicialmente introduzido por Arvo e Kirk [ARV 87]. Arvo e Kirk utilizaram grades pentadimensionais em um sistema de *Ray Tracing* para efetuar a aceleração do processo de cálculo de intersecção entre os raios emitidos para a cena e os demais objetos da cena. No método apresentado neste capítulo, porém, tal metodologia é especialmente adaptada para o uso em animações de sistemas de partículas onde é necessária a detecção de colisões entre as partículas e os demais objetos da cena.

5.1.1 Discretização do Movimento das Partículas

As abordagens em que se apoia o método são bastante simples de serem entendidas. Inicialmente, deve-se considerar que a direção do movimento de uma partícula possui apenas cinco graus de liberdade. Estes cinco graus de liberdade são definidos pela posição inicial do movimento, com três graus de liberdade, e por uma direção, constituída por dois graus de liberdade, como pode ser observado na figura 5.1. Observando-se esta figura, pode-se verificar que, através do uso de coordenadas esféricas, é possível separar a direção do movimento e a distância a ser movimentada. Esta propriedade é bastante útil, como será visto a seguir.

Pode-se observar que a direção do movimento de uma partícula pode ser expressa através de cinco parâmetros, sendo que a partir deles cria-se uma tabela com cinco valores independentes.

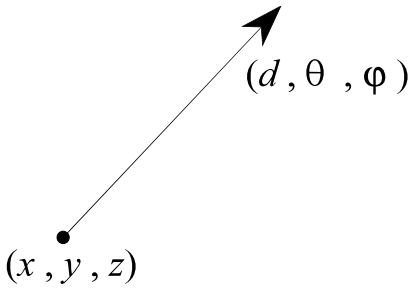

Figura 5.1 - Representação dos cinco graus de liberdade do movimento de uma partícula.

5.1.2 Obtenção do Cone de Direção das Partículas

A próxima abordagem que deve ser considerada considera como é realizada a detecção da colisão da partícula em movimento com o objeto a partir da grade pentadimensional. O procedimento baseia-se simplesmente em obter uma primitiva geométrica a partir dos valores independentes de uma determinada entrada da grade pentadimensional. No caso da implementação realizada, um cone é definido com base nos cinco parâmetros, onde os três parâmetros que definem a posição tridimensional do centro da célula são utilizados como vértice do cone, e os dois parâmetros que definem a direção do movimento da partícula são utilizados para se obter o eixo de direção do cone. Como nesta tabela com cinco parâmetros trabalha-se com intervalos, os intervalos dos dois parâmetros que definem a direção são utilizados para definir os ângulos de abertura do cone, obtendo-se desta forma uma primitiva geométrica completa. A obtenção desta primitiva pode ser vista com maiores detalhes na figura 5.2.

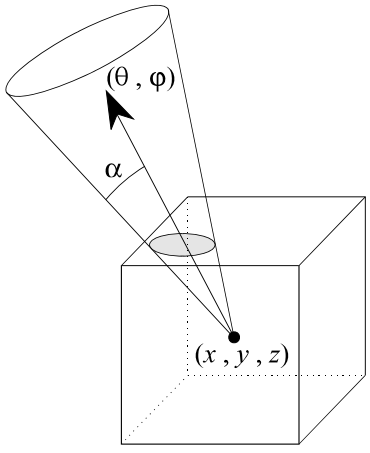

Figura 5.2 - Obtenção do cone que define o conjunto de movimentos das partículas de uma célula da grade pentadimensional.

Apesar da primitiva geométrica utilizada ter sido o cone, outras primitivas são passíveis de utilização, como por exemplo, pirâmides com base retangular. Porém, no caso de ser utilizada tal primitiva, deve-se atentar para o ângulo de rotação da primitiva em torno do eixo.

Um problema que surge, quando do uso da posição central da célula como vértice do cone, é que algumas partículas podem estar fora do cone e, portanto, podem ser obtidos resultados errôneos. Este problema é solucionado movimentando-se o cone na direção oposta à do movimento base das partículas (eixo do cone) de modo que o cone encapsule a célula que define o intervalo das posições tridimensionais. Este processo pode ser visto na figura 5.3.

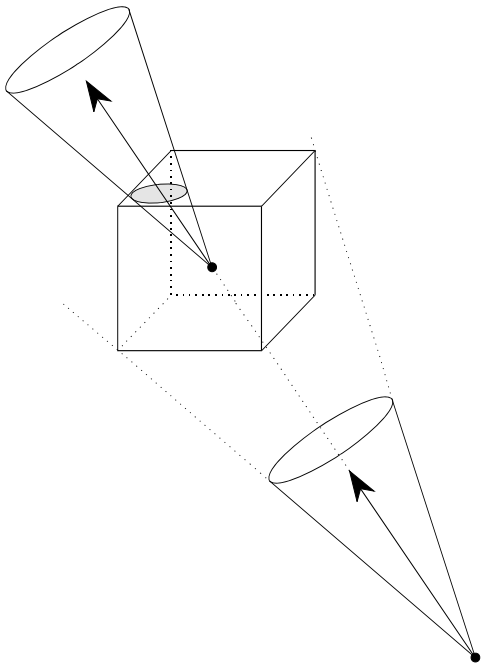

Figura 5.3 - Obtenção do cone que encapsula
o ortoedro que define as posições.

5.1.3 Obtenção das Partículas Candidatas à Intersecção

Uma vez obtido o cone que define o conjunto de movimentos das partículas de uma célula, deve-se determinar se este cone possui pontos de intersecção com os demais objetos da cena. Neste ponto um problema bastante preocupante ocorre, uma vez que o procedimento de intersecção entre um cone e um objeto genérico é demasiadamente custoso e particularmente de difícil computação. Ainda, é necessário um procedimento de cálculo específico para cada primitiva e, se forem considerados objetos CSG, a obtenção das equações de intersecção de modo analítico não é possível. Se não bastasse este

problema é necessário ainda obter a distância mínima do objeto ao centro da célula. Este cálculo é também bastante custoso computacionalmente e, em determinados casos, não é possível a sua obtenção analiticamente.

Este problema é solucionado através da utilização de procedimentos aproximativos. De outra maneira, o método simplesmente efetua uma amostragem de raios emitidos com posição inicial no centro da célula e com direções dentro do cone que define o conjunto de direções das partículas. Uma vez gerados estes raios, simplesmente, calculam-se as intersecções deles com os objetos da cena. Caso algum raio intercepte o objeto, significa que existe pelo menos uma área de colisão entre o objeto e o cone de direções das partículas. Para determinar a distância mínima utiliza-se o mesmo processo, selecionando-se a menor das distâncias de intersecção entre os raios e o objeto. Deve-se observar que somente os raios que efetivamente interceptaram o objeto devem ser considerados no processo. Deste modo, obtém-se uma distância mínima aproximada entre o objeto e o cone de direções das partículas. Tal processo é mostrado na figura 5.4.

O processo de utilizar uma amostragem de raios ao invés de uma abordagem analítica apresenta vantagens e desvantagens. Dentre as vantagens, encontra-se a possibilidade de utilização do mesmo algoritmo para os diversos objetos sólidos, mesmo objetos CSG. Outra vantagem é que esta abordagem apresenta um custo computacional menor que em uma abordagem analítica, exceto se forem utilizados um grande número de raios para a amostragem, o que não é o caso. A principal desvantagem da utilização de um processo por amostragem é que ele retornará somente uma aproximação do valor ou situação real. O fato da distância mínima ser somente uma aproximação pode ser parcialmente compensado através do uso de um fator de erro.

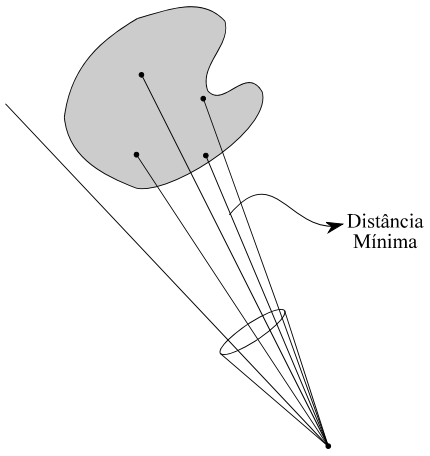

Figura 5.4 - Processo de amostragem do cone de direções das partículas.

Finalmente, se o cone de direções das partículas interceptou com sucesso o objeto sólido da cena, deve-se verificar se as partículas apresentam alguma possibilidade real de intersecção com o objeto. Considerando-se que uma aproximação da distância

mínima das partículas ao objeto é conhecida e que a distância do movimento de cada partícula também é conhecido (uma vez que a direção do movimento das partículas encontra-se em coordenadas esféricas), pode-se simplesmente efetuar uma comparação entre estes dois valores para determinar a real possibilidade de intersecção entre cada partícula e os objetos da cena. Deste forma, se a distância que a partícula percorrerá (definida pelo módulo do vetor velocidade da partícula) é maior que a distância mínima da partícula ao objeto (e considerando-se que o cone de direções das partículas intercepta o objeto) existe uma grande possibilidade que a partícula colida com o objeto em questão. Deste modo, é necessária então calcular a intersecção entre o raio definido pela posição e direção da partícula e o objeto, efetuando a reflexão apropriada se necessário.

5.1.4 Características do Algoritmo

Assim sendo, o método para a detecção e tratamento de colisões entre um sistema de partícula e os demais objetos da cena apresenta os seguinte procedimentos:

❶ Obter o cone de direções das partículas de uma determinada célula a partir do centro da célula, da direção base das partículas e da variação das direções das partículas;

❷ A partir do cone de direções das partículas, são obtidos diversos raios de amostragem;

❸ Detectar a intersecção entre os raios amostrados e cada objeto da cena, assim como obter a menor intersecção entre os raios e o objeto. Estes valores representam uma aproximação do cálculo de intersecção entre o cone de direções das partículas e o objeto, assim como a distância mínima entre as partículas e o objeto;

❹ Testar se existe a possibilidade real de colisão entre alguma partícula e o objeto em questão, comparando-se a distância mínima das partículas e o objeto com o módulo do vetor velocidade de cada partícula;

❺ Caso exista a possibilidade real de colisão deve ser testada analiticamente a intersecção da partícula com o objeto e refleti-la adequadamente se for necessário.

Estes procedimentos podem ser observados em maiores detalhes no seguinte algoritmo. Deve-se atentar para o fato que alguns passos do algoritmo, como por exemplo o passo "Aplicar forças externas sobre a partícula", fogem do escopo deste trabalho. Porém, tal passo permite que o leitor saiba onde devem ser realizadas as perturbações devidas às forças externas sobre uma determinada partícula, como por exemplo, gravidade, vento, vórtices, etc. Outros passos, como por exemplo, "Calcular a intersecção entre o raio definido pela posição e direção da partícula com o objeto OBJ", são procedimentos

tradicionais, integrantes dos sistemas de *Ray Tracing*, não sendo necessário abordá-los neste trabalho.

```
EvolveParticles()
Início
  Para cada célula da grade pentadimensional faça
    Pegar o número de partículas da célula
    Definir o número de raios amostrados N_RAY_SAMPLES
    Se o número de partículas for maior que N_RAY_SAMPLES então
      Calcular o cone de direções das partículas
      Amostrar N_RAY_SAMPLES raios a partir do cone de direções
      Para cada objeto OBJ na cena faça
        Para cada raio amostrado faça
          Calcular a intersecção entre o raio e o objeto OBJ
          Se o raio interceptou o objeto OBJ então
            Computar a menor intersecção T_MIN
            Incrementar o número de raios interceptados
          FimSe
        FimPara
        Se o número de intersecções for diferente de zero então
          Para cada partícula da célula faça
            Aplicar forças externas sobre a partícula
            Obter a velocidade da partícula em coordenadas
            esféricas (D,θ,φ)
            Perturbar as direções da velocidade da partícula
            Definir um fator de erro F_ERR
            Se D > T_MIN*F_ERR então
              Calcular a intersecção entre o raio definido
              pela posição e direção da partícula e o objeto
              OBJ
              Se a partícula interceptou o objeto OBJ então
                refletir a partícula no objeto OBJ e
                calcular a sua nova posição
              Senão
                Calcular a nova posição da partícula
              FimSe
            Senão
                Calcular a nova posição da partícula
            FimSe
          FimPara
        Senão
          Para cada partícula da célula faça
            Aplicar forças externas sobre a partícula
            Obter a velocidade da partícula em coordenadas
            esféricas (D,θ,φ)
            Perturbar as direções da velocidade da partícula
            Calcular a nova posição da partícula
          FimPara
        FimSe
      FimPara
    Senão
      Para cada objeto OBJ da cena faça
        Para cada partícula faça
          Aplicar forças externas sobre a partícula
          Obter a velocidade da partícula em coordenadas
```

```
            esféricas (D,θ,φ)
            Perturbar as direções da velocidade da partícula
            Calcular a intersecção entre o raio definido pela
            posição e direção da partícula e o objeto OBJ
            Se o raio interceptou o objeto OBJ então
              Refletir a partícula no objeto OBJ e calcular
              a sua nova posição
            Senão
              Calcular a nova posição da partícula
            FimSe
          FimPara
        FimPara
      FimSe
    FimPara
  Fim
```

Após a observação do algoritmo apresentado e dos procedimentos básicos do método, pode-se constatar as vantagens do algoritmo. Através do algoritmo pode-se verificar que somente são calculadas as intersecções entre as partículas e os demais objetos da cena quando a partícula for potencialmente candidata a uma intersecção. Também, o método é suficientemente genérico, de modo que pode ser facilmente incorporado a diversas simulações de fenômenos naturais que exigem tratamento especial para a animação dos sistemas de partículas.

Uma desvantagem aparente é a necessidade de conversão de coordenadas cartesianas para coordenadas esféricas e vice-versa. Porém, a perturbação aleatória que é realizada sobre o vetor velocidade das partículas à medida que estas são movimentadas é feita sobre os ângulos de direção do movimento da partícula. A perturbação é realizada sobre os ângulos de direção do movimento porque se for realizada em coordenadas cartesianas podem ocorrer fenômenos errôneos.

Um destes fenômenos errôneos é a mudança total da direção da partícula ao invés de uma simples perturbação. Por exemplo, considerando-se que os parâmetros de perturbação, normalmente, são valores pequenos, se for especificada uma perturbação em coordenadas cartesianas é possível a ocorrência da mudança total da direção da partícula, o que não ocorre se for especificada uma perturbação em coordenadas esféricas.

Por exemplo, especificando-se que a perturbação das partículas de um sistema é de $\pm 0{,}5$ nas coordenadas X, Y e Z, no caso das componentes do vetor velocidade serem maiores que 0,5, a direção do movimento sofrerá somente uma perturbação. Porém se uma das componentes for menor que 0,5, a direção do movimento sofrerá uma grande perturbação, apesar dos valores pequenos que estão sendo utilizados. Exemplificando, se o vetor velocidade de uma determinada partícula é $(0{,}3 \quad 0{,}2 \quad 0{,}1)$ e a perturbação a ser efetuada é $(-0{,}4 \quad -0{,}2 \quad -0{,}3)$, o vetor velocidade resultante após esta perturbação será $(-0{,}1 \quad 0 \quad -0{,}2)$. Isto indica uma mudança total da direção do movimento, o que nem sempre é desejável.

Através do uso de coordenadas esféricas, por sua vez, este problema não ocorre, uma vez que a perturbação é realizada diretamente sobre a direção do movimento. Por exemplo, se o vetor velocidade de uma determinada partícula for $\left(0{,}2 \quad \frac{\pi}{6} \quad \frac{\pi}{4}\right)$ e for realizada uma perturbação de $-\frac{\pi}{36}$ e $-\frac{\pi}{24}$, respectivamente, sobre os ângulos de direção, o vetor resultante será $\left(0{,}2 \quad \frac{5\pi}{36} \quad \frac{5\pi}{24}\right)$. Com isto, pode-se observar que a direção do movimento é mantida.

Outra razão para a utilização de perturbações sobre os ângulos de direção das partículas é que, segundo as equações apresentadas por Shinya e Fournier [SHI 92], os efeitos de forças externas como ventos, podem ser melhor modelados através de perturbações nos ângulos de direções utilizando equações diferenciais.

A maior desvantagem deste método é que o processo de detecção de colisões entre partículas e objetos sólidos da cena é realizado de forma aproximativa, porém suficiente para a maioria dos casos. Outra desvantagem é a quantidade de memória exigida para a alocação da grade pentadimenisonal. Alguns casos de exceção serão abordados no capítulo seguinte.

5.2 Conclusões

Neste capítulo foi apresentado um novo método com o principal objetivo de reduzir o tempo de processamento para a detecção de colisões entre sistemas de partículas e os demais objetos sólidos da cena. O método se baseia na utilização de grades pentadimensionais para classificar o movimento das partículas e, deste modo, acelerar o processo de intersecção entre partículas e objetos através de um processo seletivo.

O método, através da classificação das partículas, seleciona as partículas com probabilidade de intersecção com os objetos e somente efetua o cálculo de intersecção se estas estiverem potencialmente próximas ao objeto (o fator proximidade é determinado através da comparação com o módulo do vetor velocidade) e em direção a este objeto.

Este método se baseia principalmente na observação de que, na maior parte de uma animação envolvendo sistemas de partículas, as partículas apresentam baixa probabilidade de intersecção com os objetos. Assim sendo, o método mostra-se bastante válido e com potencial uso principalmente nos quadros iniciais das animações, onde geralmente os sistemas de partículas encontram-se distantes dos demais objetos da cena.

Dentre as principais vantagens do método, fora o fato de acelerar o processo de detecção das colisões com os objetos da cena, encontram-se a generalidade do método para uso em animações de sistemas de partículas e a possibilidade de uso do método mesmo para objetos complexos, como objetos CSG, superfícies Bèzier e Splines.

Dentre as desvantagem do método, por sua vez, encontram-se o fato do método ser aproximativo e a grande quantidade de memória exigida para a alocação da grade pentadimensional.

6 Resultados Obtidos e Análise dos Resultados

Este capítulo tem por finalidade apresentar os principais resultados obtidos neste trabalho, bem como realizar uma análise crítica destes resultados, de modo a comprovar ou não as propriedades esperadas dos métodos e algoritmos apresentados nos capítulos anteriores. Serão apresentados os resultados obtidos a partir da integração de determinados sistemas de partículas em um sistema de *Ray Tracing* e após os resultados obtidos do processo de animação de um sistema de partículas efetuando a detecção de colisões com os demais objetos da cena.

Inicialmente, no processo de integração de sistemas de partículas, o principal objetivo é avaliar a aceleração obtida através do uso de grades tridimensionais. Para tal, foram gerados sistemas de partículas com expressivas quantidades de partículas e observadas suas propriedades.

Assim sendo, foram gerados diversos sistemas de partículas com a mesma forma básica, no caso uma esfera, de modo a ocupar praticamente o mesmo volume. Para cada imagem gerada foi computado o tempo de processamento e composto um gráfico. O tempo de processamento para a geração de imagens com um sistema de partículas em um sistema de *Ray Tracing* sem nenhuma técnica de aceleração não foi computado. Isto é devido ao fato de que, com os recursos computacionais atuais, o tempo de processamento seria exageradamente alto. Para um número de partículas na ordem de milhares ou dezenas de milhares, o tempo de processamento estimado alcança horas ou dias. Isto é devido ao fato que, durante o percurso do sistema de partículas, para cada partícula atingida, deve ser novamente testada a intersecção com as demais partículas do sistema, além de ser necessário o cálculo do sombreamento. Isto torna o tempo de processamento proporcional ao quadrado do número de partículas do sistema.

Deste modo, no que tange ao processo de integração de sistemas de partículas com ambientes de *Ray Tracing*, somente serão avaliados os tempos de processamento para o sistema com o método de aceleração.

No processo de geração foi utilizado uma imagem simples com um plano, uma esfera e uma fonte de luz. O sistema de partículas é um conjunto de partículas aleatoriamente geradas dentro de uma esfera, de modo que ocupem praticamente o mesmo volume. Algumas das imagens mais significativas podem ser observadas na figura 6.1.

O tempo de processamento para a geração destas imagens, juntamente com o número de partículas do sistema para cada imagem é mostrado na tabela da figura 6.2. Também, um gráfico relacionando o número de partículas com o tempo de processamento é mostrado na figura 6.3.

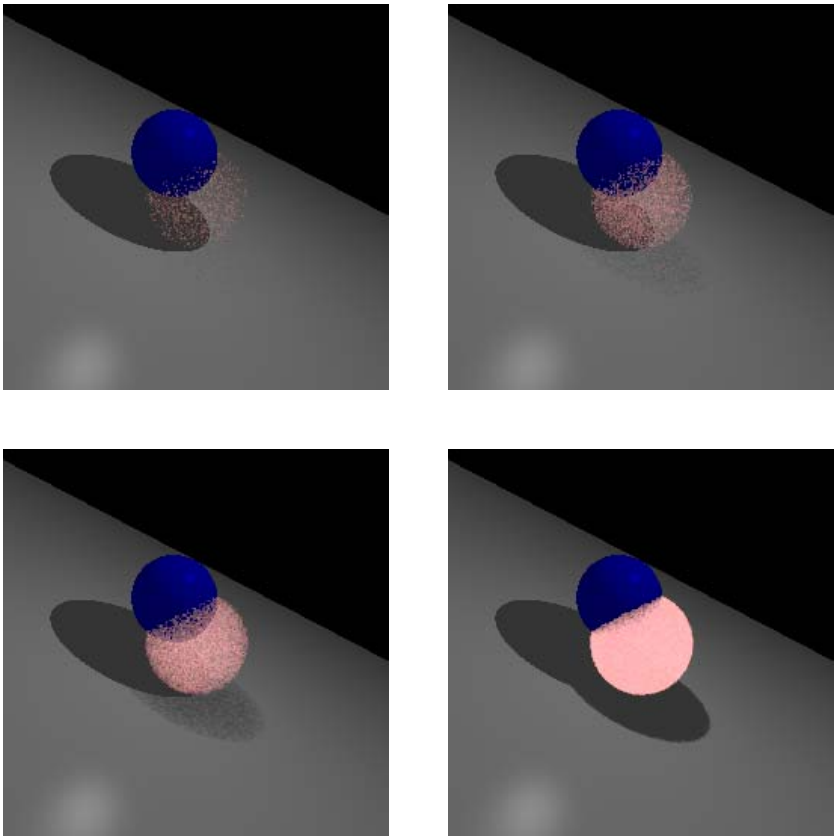

Figura 6.1 - Imagens de sistemas de partículas com aproximadamente 5 mil, 22 mil, 118 mil e 1,2 milhões de partículas.

Para a mensuração do tempo de processamento, foram utilizadas funções próprias do sistema operacional, executadas no início e no fim da execução do programa de *Ray Tracing*. Considerando-se que o relógio do processador possui uma resolução na ordem de centésimos de segundo e que a quantidade de instruções necessárias para a obtenção do tempo é pequena, pode-se garantir a precisão das medidas até a ordem de décimos de segundo.

Após esta apresentação de resultados obtidos da integração de sistemas de partículas em um ambiente de *Ray Tracing*, alguns detalhes podem ser destacados:

❶ A partir da análise do algoritmo, pode-se observar que, no percurso de uma grade tridimensional cúbica com n células em cada dimensão e com uma média de μ partículas por célula, o número máximo de partículas a serem consideradas para o cálculo da alteração da cor do raio é de $\mu.(3n-2)$ (como é demonstrado a seguir). Porém, deve-se considerar que há um tempo de processamento mínimo que é decorrente do processo de percurso da grade tridimensional pelo raio. Deste modo, pode-se dizer que o tempo para o cálculo das contribuições das partículas para os raios no sistema de *Ray*

Tracing é linear. Deve-se observar que para casos onde ocorre efeitos de sombreamento de um objeto por um sistema de partículas, o tempo de processamento para obtenção do fator de atenuação da sombra também é linear. Também, o tempo de processamento para a geração de uma imagem é altamente dependente da geometria da cena, porém o tempo de percurso da grade tridimensional é sempre linear.

❷ A partir dos valores mostrados na tabela da figura 6.2, observa-se que o tempo de processamento para um sistema de partículas com até aproximadamente 25 mil partículas é constante. Isto é devido ao fato que a quantidade de partículas atingidas no percurso das células não ocasionou consumo computacional suficiente para alterar o tempo de processamento;

❸ A partir da figura 6.1, pode-se observar as propriedades de conjunto do sistema de partículas. Observe que, à medida que o sistema de partículas possui mais partículas, tanto o objeto formado pelo sistema de partículas torna-se mais sólido como a sombra projetada pelo sistema de partícula torna-se mais definida e opaca. Isto demonstra um efeito de compactação das partículas, ou seja, quanto maior a densidade das partículas, mais o sistema de partículas se comporta como se fosse um objeto sólido.

Nº Partículas	*Tempo (s)*
2.557	**105**
5.061	**105**
9.896	**105**
22.105	**105**
46.205	**109**
82.319	**119**
118.341	**131**
155.534	**144**
203.213	**163**
242.913	**179**
539.812	**271**
1.145.083	**530**

Figura 6.2 - Tabela comparativa do tempo de processamento para geração das imagens com os sistemas de partículas.

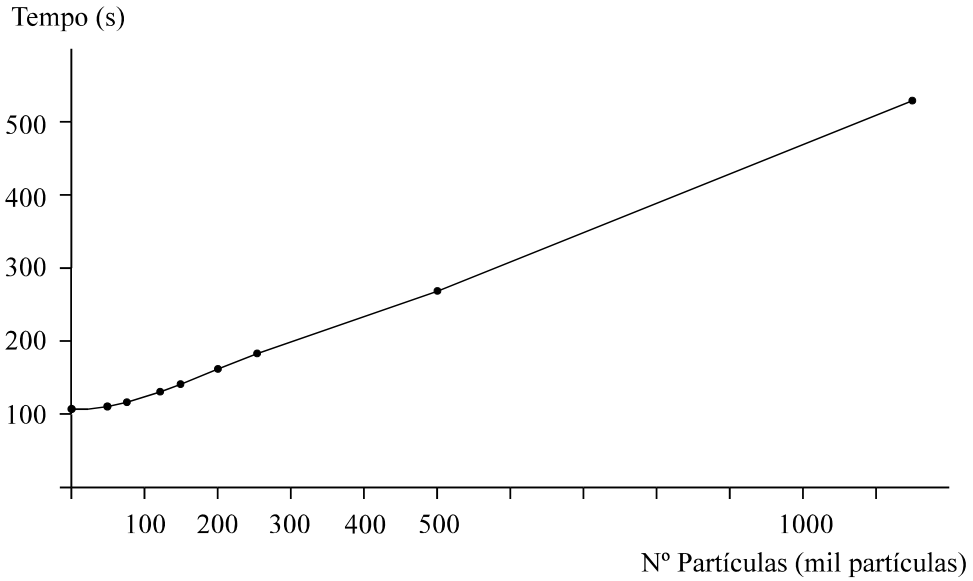

Figura 6.3 - Gráfico do tempo de processamento pelo número de partículas do sistema.

Ainda, considerando-se o fator de aceleração obtido pela utilização de grades tridimensionais para a integração com o ambiente de *Ray Tracing*, pode-se estimar o fator de redução do tempo de processamento para a geração da imagem.

De acordo com a figura 6.4, em uma grade bidimensional quadrada, pode-se observar que um dos percursos que envolve a travessia do maior número de células utiliza exatamente $2n-1$ células, onde n é o número de células de cada lado do quadrado. Esta equação é obtida a partir da observação que, em uma grade bidimensional, um raio que se encontra em uma determinada célula somente pode atravessar no máximo duas células vizinhas. Deste modo, como a dimensão de cada lado da grade bidimensional é de n células, o raio deveria percorrer $2n$ células. Porém, a primeira célula (ou a última) é considerada somente uma vez, já que é o ponto de entrada (ou saída) do raio. Assim sendo, o número máximo de células percorridas por um raio em uma grade bidimensional é $2n-1$.

Seguindo o mesmo procedimento, em uma grade tridimensional, o raio em uma determinada célula só pode atravessar três células vizinhas, como exceção da primeira (ou última) célula, que é considerada somente uma vez. Deste modo, a quantidade de células de uma grade tridimensional consideradas no percurso de um raio é de $3n-2$.

No caso da figura, utiliza-se uma grade de 8 x 8 células e o número máximo de células percorridas é 15. Deste modo, considerando-se que o número médio de partículas por células é μ, a quantidade total de partículas processadas para o cálculo das contribuições das partículas para a alteração da cor do raio é

$$n_{ac} = \mu.(2n-1) \ .$$

Entretanto, o número de partículas processadas em um sistema de *Ray Tracing* sem nenhum tipo de aceleração é extremamente mais custoso. Supondo que, no percurso do sistema de partículas, o raio atravessará em média uma partícula por célula, significa que serão processadas $2n-1$ partículas. Porém, para a determinação de qual partícula será processada, é necessário calcular a intersecção do raio com todas as demais partículas, ou seja, com todas as $\mu.n^2$ partículas da grade. Assim sendo, tem-se um total de $\mu.n^2(2n-1)$ partículas processadas.

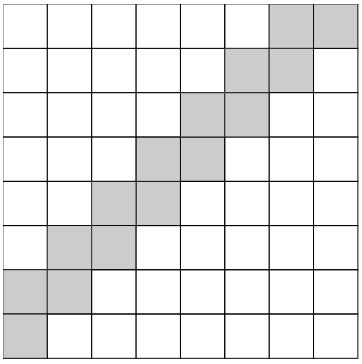

Figura 6.4 - Número máximo de células visitadas no percurso de um raio através de uma grade.

Para obter o fator de aceleração de um sistema com a utilização de grade bidimensional sobre um sistema sem nenhum tipo de aceleração, basta efetuar a razão entre as quantidades de partículas processadas, ou seja,

$$s = \frac{\mu.n^2(2n-1)}{\mu(2n-1)} .$$

Simplificando esta expressão tem-se que o tempo de processamento é acelerado de acordo com um fator proporcional a n^2, o que torna o método bastante praticável. Este raciocínio pode ser estendido para um modelo tridimensional, de modo que o fator de aceleração é então proporcional n^3.

Obviamente, alguns fatores devem ser observados, dentre eles o fato que o tempo de travessia das células não foi considerado, o que pode afetar o fator de aceleração quando o número de células for muito grade. Porém, mesmo assim, o fator de aceleração continua muito grande. Nos resultados apresentados neste trabalho, são utilizadas grades tridimensionais de 64 x 64 x 64 células.

Outro fato que deve ser analisado é a redução do consumo de memória. Considerando-se o que foi apresentado no capítulo 4.5, pode-se constatar que foi obtida uma redução de aproximadamente 70,4% do consumo de memória de um sistema de partículas no sistema de animação (que envolve todos os atributos) e de 71,4% no sistema de *Ray Tracing* (que envolve somente os atributos posição, cor e transparência). Estes

valores percentuais são obtidos comparando-se a quantidade de memória exigida em uma representação normal (sem redução de memória) e a quantidade de memória exigida em uma representação reduzida através de representações inteiras.

Para se obter o fator de erro ocasionado pela técnica de redução de memória foi efetuada a geração de 100 milhões de pontos tridimensionais aleatoriamente. Estas partículas foram então convertidas para o modo reduzido e reconvertidas para o valor em ponto flutuante e a diferença calculada, de modo a obter a perda de precisão com o uso da representação inteira. O fator de erro foi obtido a partir da razão da diferença calculada pelo módulo do vetor original, representando, deste modo, a perda de precisão proporcionalmente ao módulo do vetor utilizado, como mostra a seguinte equação:

$$\varepsilon = \frac{\left| \vec{v}' - \vec{v} \right|}{\left| \vec{v} \right|}$$

onde $\vec{v}$ é o vetor posição do ponto tridimensional em ponto flutuante e $\vec{v}'$ é o vetor posição do ponto tridimensional convertido para a representação inteira e reconvertido para ponto flutuante. No final do processo, um valor médio foi calculado, assim como o maior erro. Este mesmo procedimento é realizado para o caso de utilização de coordenadas esféricas. Os resultados obtidos são mostrados na tabela da figura 6.5.

	Erro Médio	Erro Máximo
Coordenadas Cartesianas	0,000044	0,015395
Coordenadas Esféricas	0,000776	0,009621

Figura 6.5 - Tabela com os fatores de erro para os métodos de redução do consumo de memória.

Para garantir a precisão das medidas efetuadas, as variáveis internas utilizadas para os procedimentos de cálculo do fator de erro possuem 64 bits de precisão, sendo 1 bit para o sinal, 11 para o expoente e 52 para a mantissa. Deste modo, possuindo-se 52 bits para a representação da mantissa do número, pode-se garantir uma precisão de até 15 algarismos significativos. Isto, por certo, garante a validade da precisão dos valores apresentados.

Pode-se observar que, de acordo com os valores obtidos, o método que utiliza a representação inteira dos valores em ponto flutuante apresenta um fator de erro médio menor do que a que utiliza coordenadas esféricas. Porém, devido ao fato das coordenadas esféricas não utilizarem um espaçamento constante para a representação inteira, o seu fator de erro máximo é menor. Isto é decorrente do fato que, quando o módulo do vetor é

consideravelmente pequeno, o fator de erro quando se utiliza coordenadas cartesianas é bastante grande, o que não ocorre em uma representação utilizando coordenadas esféricas.

Aqui, pode-se constatar que os fatores de erro representam pequenas variações no vetor original, de modo que podem ser utilizados sem afetar o efeito visual a ser obtido, uma vez que, em sistemas de partículas, perturbações são constantemente introduzidas no processo de animação.

Já no processo de detecção de colisões entre sistemas de partículas e os demais objetos sólidos da cena, o objetivo principal é avaliar a validade do uso da grade pentadimensional como um meio para determinar a proximidade entre as partículas de uma determinada célula e o objeto sólido. Para isto foi realizada uma animação simples de um sistema de partículas, comparando-se os tempos de processamento com e sem a utilização de uma grade pentadimensional. A cena na qual o sistema de partículas evolui é composta de uma esfera e de um plano para simplificar a avaliação. Nesta cena, o sistema de partículas se comporta como uma nuvem de fumaça subindo e eventualmente colidindo com a esfera. O plano serve para melhor localizar o sistema de partículas e a esfera, assim como para avaliar a projeção das sombras. A evolução consiste de quarenta quadros básicos, sendo que quatros deles são exibidos na figura 6.6.

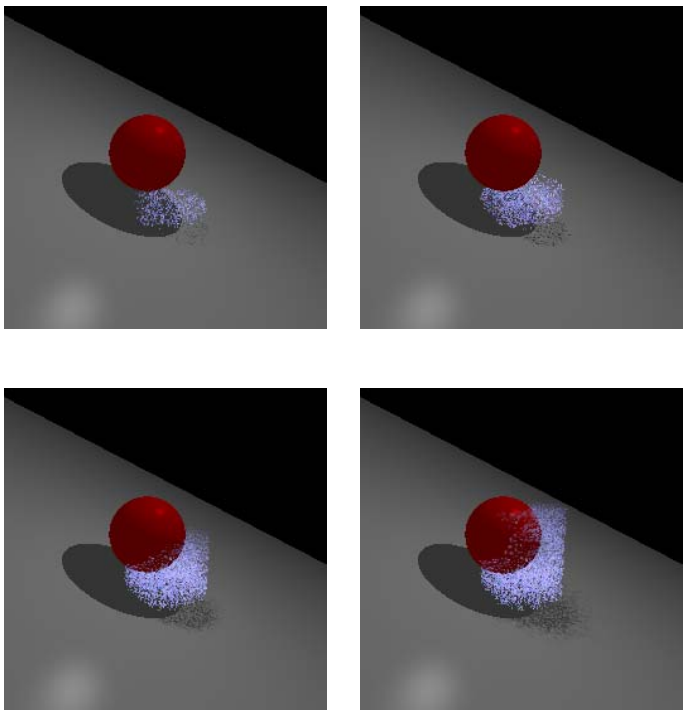

Figura 6.6 - Quadros da animação do sistema de partículas nas posições 10, 20, 30 e 40.

Quadros	Número de partículas	Número de partículas testadas	Número de partículas refletidas	Tempo com aceleração 5-D (segundos)	Tempo sem aceleração 5-D (segundos)
1	4.990	0	0	3,9	4,3
2	9.787	0	0	4,0	4,7
3	14.902	0	0	4,1	5,2
4	20.039	0	0	4,2	5,7
5	24.971	0	0	4,3	6,1
6	30.090	0	0	4,4	6,5
7	34.964	0	0	4,4	7,0
8	39.871	0	0	4,5	7,4
9	44.881	539	19	4,6	7,9
10	49.949	3.024	1.146	5,0	8,4
11	54.745	4.014	3.695	5,4	9,0
12	59.711	5.697	3.771	5,6	9,5
13	64.720	6.025	4.194	5,7	10,0
14	69.863	7.408	4.275	5,9	10,5
15	74.942	7.597	4.317	6,0	10,9
16	80.080	7.592	4.238	6,1	11,4
17	84.907	7.438	4.260	6,1	11,8
18	90.075	7.692	4.364	6,2	12,3
19	95.304	7.709	4.237	6,3	12,7
20	100.224	7.546	4.329	6,4	13,2
21	105.378	7.695	4.396	6,5	13,7
22	110.213	7.494	4.293	6,5	14,1
23	115.445	7.526	4.283	6,6	14,5
24	120.334	7.608	4.222	6,7	15,0
25	125.395	7.655	4.346	6,8	15,4
26	130.346	7.813	4.338	6,9	15,9
27	135.382	7.911	4.360	7,0	16,3
28	140.308	7.969	4.464	7,1	16,8
29	145.432	7.727	4.337	7,1	17,2
30	150.618	7.945	4.429	7,2	17,7
31	155.707	7.757	4.496	7,3	18,2
32	160.891	7.633	4.393	7,4	18,6
33	165.827	7.820	4.322	7,4	19,0
34	170.695	7.889	4.446	7,5	19,5
35	175.605	7.977	4.411	7,6	20,0
36	180.578	8.021	4.438	7,7	20,4
37	185.686	8.228	4.460	7,8	20,8
38	190.811	7.975	4.564	7,9	21,3
39	195.746	8.159	4.437	7,9	21,7
40	200.728	8.292	4.529	8,0	22,2

Figura 6.7 - Tabela como os dados e tempos de processamento obtidos.

Ainda, uma tabela comparativa dos tempos com outros parâmetros coletados é exibida na figura 6.7.

Nesta tabela é apresentada a quantidade de partículas existentes em cada quadro, o número de partículas que necessitaram ser testadas as intersecções com os objetos sólidos da cena, o número de partículas que foram efetivamente refletidas, assim como os tempos de processamento com e sem a utilização da grade pentadimensional.

A partir desta tabela, pode-se constatar que o tempo de processamento com o uso da grade pentadimensional é pouco dependente do número total de partículas do sistema, sendo mais dependente do número de partículas testadas e refletidas. Pode-se observar que, comparativamente com o tempo de processamento sem o uso da grade pentadimensional, o tempo de processamento é significativamente menor à medida que a quantidade de partículas aumenta. Isto é devido ao fato que o tempo de processamento sem o uso de grades pentadimensionais é mais dependente da quantidade total de partículas no sistema.

No caso, a resolução do relógio do processador do computador utilizado para gerar as animações é de centésimos de segundos. Deste modo, como os tempos de processamento são apresentados com precisão de décimos de segundo, pode-se garantir a precisão das medidas.

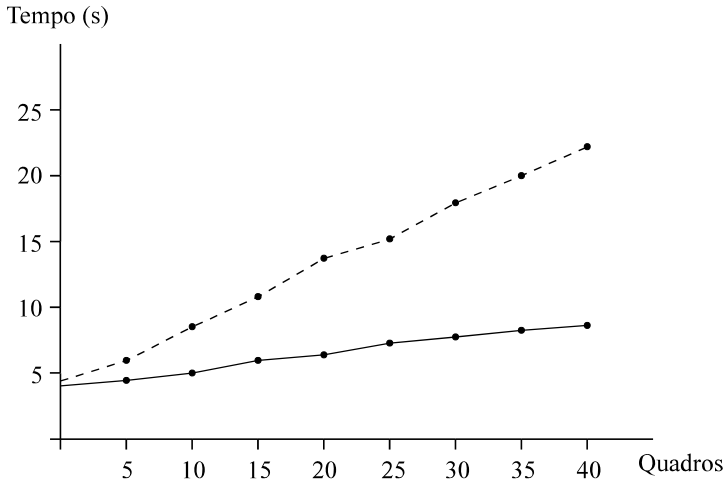

Figura 6.8 - Gráfico comparativo dos tempos de processamento
com e sem a utilização da grade pentadimensional.

Também, na figura 6.8 é mostrado o gráfico comparativo dos tempos de processamento.

A partir deste gráfico pode-se observar que ambos os tempos de processamento são lineares. Porém, como o tempo de processamento com o uso da grade pentadimensional é proporcional ao número de partículas testadas e não ao número total de partículas, a inclinação da reta é menor, comparativamente com a reta produzida pelo tempo de processamento sem o uso da grade pentadimensional.

Tendo em vista tais resultados do uso da grade pentadimensional, alguns aspectos devem ser destacados:

❶ O tempo de processamento é alterado quando as partículas, num número significativo, devem ser testadas e/ou refletidas. A varredura de todas as células da grade não apresenta um impacto significativo sobre o tempo de processamento;

❷ As partículas somente são testadas se estiverem significativamente próximas aos objetos da cena e em direção de colisão com eles. Caso as partículas estejam distantes ou em direção de movimento oposta ao objeto o tempo de detecção de colisão não há impacto significativo sobre o tempo de processamento;

❸ O método apresenta uma diminuição significativa do tempo de processamento, além de ser pouco dependente do número de partículas no sistema, sendo mais dependente número de partículas potencialmente próximas aos objetos da cena.

Deve-se observar que o fator de aceleração é bastante dependente da geometria da cena e do tipo de sistema de partícula que se está modelando. Por exemplo, no caso de se modelar uma explosão ou um gás se movendo dentro de uma pequena caixa transparente, onde as partículas se movimentam em todas as direções e estão na maior parte da animação colidindo com a caixa, a fator de aceleração é mínimo, se houver algum. Isto é devido basicamente ao fato de que o método se baseia na premissa que o sistema de partículas apresentará poucas colisões com os objetos da cena, o que não ocorre nesta exceção.

7 Conclusões

Este trabalho possui como principais objetivos o desenvolvimento de um método para a integração de sistemas de partículas em ambientes de *Ray Tracing* que apresente um custo computacional compatível, assim como apresentar um modelo para a redução do consumo de memória exigido para a armazenagem de um sistema de partículas. Ainda, um dos objetivos deste trabalho foi apresentar um método para acelerar o processo de detecção de colisões entre um sistema de partículas e os demais objetos sólidos da cena.

Os métodos e algoritmos apresentados neste trabalho representam avanços na área de síntese de sistemas de partículas, assim como avanços na área de detecção de colisões entre sistemas de partículas e objetos sólidos. Pouco, se algum desenvolvimento, tem sido realizado ultimamente levando em consideração tais aspectos.

Quanto à técnica proposta para a integração de sistemas de partículas e um ambiente de *Ray Tracing*, em comparação com as demais técnicas tradicionais para síntese de sistemas de partículas em uma cena composta, pode-se observar os seguintes aspectos:

❶ É possível a obtenção de efeitos altamente realísticos através do uso da técnica de *Ray Tracing*, como por exemplo, efeitos de reflexão, transparência, sombreamento, etc.;

❷ Através de uma função de modulação para o uso no cálculo da transparência e no cálculo da cor de uma determinada partícula é possível obter efeitos de *anti-aliasing*. Esta função de modulação proporciona a obtenção de efeitos de *anti-aliasing* sem a necessidade de gerar raios extras. Também, através do uso desta função de modulação é possível controlar o tamanho médio das partículas;

❸ O consumo de memória exigida para o armazenamento de partículas pode ser consideravelmente reduzido através da utilização de uma representação compacta de diversos atributos do sistema de partículas;

❹ Apesar de apresentar um consumo computacional mais elevado para a geração da imagem final, a integração de sistemas de partículas e ambientes de *Ray Tracing* apresenta uma qualidade maior das imagens geradas, com efeitos difíceis de serem obtidos.

Já no que tange ao método para a aceleração da detecção de colisão entre sistemas de partículas e os demais objetos sólidos da cena, pode-se observar as seguintes características quando comparado com o método tradicional de cálculo de colisões:

❶ Houve uma aceleração significativa no processo de detecção de colisões entre o sistema de partículas e os objetos da cena, uma vez que relativamente poucas partículas são testadas contra os objetos da cena;

❷ Considerando-se que o método baseia-se na detecção da proximidade e direcionamento das partículas com relação ao objeto, a grade pentadimensional apresenta um bom uso. Este uso favorável é devido ao fato que ela discretiza tanto a posição quanto a direção das partículas para comparação da distância;

❸ O tempo de processamento para a detecção das colisões entre as partículas e os objetos é dependente principalmente do número de partículas potencialmente próximas aos objetos, ao invés do número total de partículas do sistema. Este fato é de grande importância, uma vez que a detecção das partículas potencialmente próximas é realizada apenas com consultas a tabelas e com comparações simples, não envolvendo cálculos complicados ou custosos computacionalmente. O único processo computacionalmente custoso é a conversão de coordenadas cartesianas para coordenadas esféricas, porém este processo deve ser realizado independentemente para o processo de perturbação do vetor velocidade das partículas, sendo seus resultados reutilizados sem custo no algoritmo de detecção de colisão.

Quanto às dificuldades encontradas no desenvolvimento do trabalho, pode-se salientar a grande escassez de material bibliográfico sobre sistemas de partículas, assim como dos métodos utilizados para a solução dos problemas. Deve-se salientar também que pouquíssimos trabalhos foram publicados com vistas a solucionar os problemas da técnica de sistemas de partículas, sendo a quase totalidade das publicações voltadas para a simulação de fenômenos ou modelagem de objetos. Nestes trabalhos, o processo de visualização dos sistemas de partículas ou não é abordado, ou simplesmente se utiliza de processos extremamente simplificados. Quando o trabalho integra objetos sólidos na cena, o processo de integração ou não ocorre ou não é abordado.

7.1 Trabalhos Futuros

Existem ainda alguns problemas que devem ser solucionados, assim como extensões que podem ser realizadas nos métodos propostos. Estas alterações podem ser o foco de pesquisas e projetos futuros. Dentre as principais melhorias sugeridas pode-se destacar as seguintes:

❶ A modificação da função de modulação no processo de integração de sistemas de partículas em um ambiente de *Ray Tracing* de modo a incorporar o efeito de borrão de movimento (*motion blur*). Uma equação foi elaborada, porém não foi implementada para verificar a sua viabilidade e os seus efeitos visuais;

❷ A realização de um estudo mais elaborado sobre as potencialidades do uso de coordenadas esféricas para a redução do consumo de memória exigida por um sistema de partículas, assim como o seu uso de modo mais eficiente

no processo de aceleração do cálculo de intersecção das partículas com o raio;

❸ No processo de detecção de colisões entre as partículas de um sistema e os objetos sólidos da cena, seria de grande utilidade se o método utilizasse outros procedimentos para a obtenção da direção da partícula, de modo que o método pudesse ser utilizado em outros tipos de animação e não somente com sistemas de partículas. Está sendo estudado o uso de árvores BSP pentadimensionais, porém o estudo encontra-se apenas em fase inicial;

❹ Está sendo realizado um estudo para a integração de sistemas de partículas em ambiente de radiosidade, utilizando-se para isto procedimentos semelhantes aos apresentados por Christian Hofsetz [HOF 97] para o cálculo do fator de forma e detecção de oclusão por outros objetos e outras partículas. Deve-se observar que apesar do consumo computacional exigido, através do uso da técnica de radiosidade pode-se obter efeitos bastante realísticos para a simulação de partículas emissoras de luz, como por exemplo, no processo de simulação de fogo;

❺ Quanto ao processo de tratamento de colisões, está sendo estudado o processo de tratamento de colisões entre dois sistemas de partículas ou dentro do mesmo sistema de partículas. Tal procedimento pode ser realizado através da utilização da densidade de partículas em uma determinada célula tridimensional.

Este trabalho mostrou ser possível a utilização de sistemas de partículas em ambientes de *Ray Tracing* de maneira relativamente rápida e com um custo computacional reduzido. Também, foi demostrado que o uso de grade pentadimensional pode ser de grande valia no processo de animação de sistemas de partículas quando se leva em consideração a detecção de colisões entre partículas e objetos sólidos da cena.

7.2 Aplicabilidade dos Métodos

Pelo que já foi apresentado, pode-se observar várias aplicações para os métodos apresentados. A integração de sistemas de partículas e o ambiente de *Ray Tracing* possui grande utilidade para aplicações que pretendem obter efeitos de fumaça, nuvens, fluidos, dentre outros objetos difusos não emissores de luz, com alto grau de realismo.

Também, o método para a detecção de colisões entre partículas e objetos sólidos pode ser utilizado em processos de visualização científica de fluidos. Considerando-se que o processo de visualização da trajetória de fluidos em um ambiente pode ser realizado através de partículas e que tais procedimentos podem envolver uma quantidade considerável de partículas, o uso de grades pentadimensionais pode acelerar consideravelmente tal processo de visualização.

Por fim, apesar das dificuldades encontradas, acredita-se que os objetivos de desenvolver métodos para aumentar a qualidade de geração de imagens criadas com sistemas de partículas, assim como do desenvolvimento de um método para a aceleração da detecção de colisões entre sistemas de partículas e objetos sólidos em uma cena foram alcançados. Também, acredita-se que, uma vez que sistemas de partículas começam a ser utilizados em diversas outras áreas do conhecimento, tais procedimentos podem integrar várias áreas do conhecimento científico.

Anexo A

Código em linguagem C para as funções ***FindColor*** e ***ShadeParticleSystem***.

A função ***FindColor*** possui como finalidade encontrar a cor do sombreamento para o raio definido por ***pos*** e ***ray*** (posição e direção do raio), retornando o resultado em ***color***.

```
void FindColor(color, pos, ray)
P3d *pos, *ray;
Color *color;
{
   P3d          hit, normal;
   SurfaceNode *surface;
   int          surftype;
   double       s1, s2, step_t, t, transp;
   ObjectNode  *o1, *o2;
   ParticleSystemObj *ps1;
   int          ant_grid_x, ant_grid_y, ant_grid_z,
                grid_x, grid_y, grid_z, f;
   Color        newcolor;


   if (ClosestObject->ObjectType != PARTICLE_SYSTEM_TYPE) {
      hit.x = pos->x + ClosestDistance*ray->x;
      hit.y = pos->y + ClosestDistance*ray->y;
      hit.z = pos->z + ClosestDistance*ray->z;

      (*ObjectNormal[ClosestObject->ObjectType])(ClosestObject,
                                        hit, &normal);
      if (DotP(Ray, &normal)>0) {
          normal.x = -normal.x;
          normal.y = -normal.y;
          normal.z = -normal.z;
      }
      surface = ClosestObject->Surface;
      Shade[surface->type](ClosestObject, &hit, ray, &normal,color);
   } else {
      color->r = 0;
      color->g = 0;
      color->b = 0;

      s1 = ClosestDistance;
      o1 = ClosestObject;
      ps1 = (ParticleSystemObj *) o1->Descr;

      ClosestDistance = FAR_AWAY;
      ClosestObject   = NULL;
      CurrentObject   = FirstObject;

      while (CurrentObject != NULL) {
```

```
            if (CurrentObject->ObjectType != PARTICLE_SYSTEM_TYPE) {
              (*ObjectIntersect[CurrentObject->ObjectType])
                          (CurrentObject, *pos, *ray);
            }
            CurrentObject = CurrentObject->Next;
          }
          s2 = ClosestDistance;
          o2 = ClosestObject;

          step_t = ps1->Step;
          ant_grid_x = -1;
          ant_grid_y = -1;
          ant_grid_z = -1;
          transp = 1.0;

          for (t=s1 ; t <= s2 ; t += step_t) {
            hit.x = pos->x + t*ray->x;
           hit.y = pos->y + t*ray->y;
            hit.z = pos->z + t*ray->z;

            if (fabs(hit.x-ps1->Center.x)-ps1->Size.x > ROUND_ERROR ||
                fabs(hit.y-ps1->Center.y)-ps1->Size.y > ROUND_ERROR ||
                fabs(hit.z-ps1->Center.z)-ps1->Size.z > ROUND_ERROR) {
              break;
            } else {
              FindGridPosition(hit, &grid_x, &grid_y, &grid_z, ps1);

              if (grid_x != ant_grid_x || grid_y != ant_grid_y ||
                grid_z != ant_grid_z) {
                ShadeParticleSystem(ps1, color, &transp, grid_x,
                          grid_y, grid_z, &f, ray, pos);
                if (transp < 0.05 || f) {
                        break;
                }
                ant_grid_x = grid_x;
                ant_grid_y = grid_y;
                ant_grid_z = grid_z;
              }
            }
          }

          if (o2 != NULL) {
            FindColor(&newcolor, pos, ray);
            color->r += newcolor.r;
            color->g += newcolor.g;
            color->b += newcolor.b;
          } else {
            color->r += Background.r;
            color->g += Background.g;
            color->b += Background.b;
          }
        }
      }
```

A função ***ShadeParticleSystem*** determina a cor do sombreamento do raio definido por ***position*** e ***ray*** (posição e direção do raio), ao percorrer a célula do sistema de partículas ***ps1*** definida por ***grid_x***, ***grid_y*** e ***grid_z***. A cor é retornada em ***color*** e o fator de

atenuação devido à transparência das partículas é retornado em ***transp***. Se ocorrer saturação de uma das componentes R, G ou B da cor, ***f*** retorna TRUE, senão retorna FALSE.

```
void ShadeParticleSystem(ps1, color, transp, grid_x, grid_y, grid_z,
                         f, ray, position)
ParticleSystemObj *ps1;
Color *color;
double *transp;
int grid_x, grid_y, grid_z, *f;
P3d *ray, *position;
{
  int line, offset, line1, offset1, line2, offset2;
  double transp1, tr, d1, dd;
  long int pos, pos1, pos2, i, part;
  unsigned char p1, p2, p3;
  P3d pa, pb;
  unsigned int x1, y1, z1;

  *f = 0;
  pos = 3*(grid_x+64L*grid_y+4096L*grid_z);

  if (pos == 0) {
    pos1 = 0;
    pos2 = 0;
  } else {
    fseek(FHeader,(long)(pos-3),SEEK_SET);
    fscanf(FHeader, "%c%c%c", &p1, &p2, &p3);
    pos1 = p1*65536L + p2*256L + p3;

    fscanf(FHeader, "%c%c%c", &p1, &p2, &p3);
    pos2 = p1*65536L + p2*256L + p3;
  }

  for(i=10*pos1;i<10*pos2;i+=10) {

    fseek(FParticles,(long)(i),SEEK_SET);

    ReadInt(FParticles, &x1);
    ReadInt(FParticles, &y1);
    ReadInt(FParticles, &z1);

    pa.x = x1*256.0/65536.0-128.0 - position->x;
    pa.y = y1*256.0/65536.0-128.0 - position->y;
    pa.z = z1*256.0/65536.0-128.0 - position->z;

    d1 = DotP(&pa,ray);
    dd = pa.x*pa.x + pa.y*pa.y + pa.z*pa.z - d1*d1;
    tr = pow(0.5, dd/(ps1->PartSize * ps1->PartSize));

    fscanf(FParticles, "%c", &p1);
    transp1 = tr*p1/255.0;
    *transp *= (1.0-transp1);

    fscanf(FParticles, "%c%c%c", &p1, &p2, &p3);

    color->r += p1*transp1;
```

```
        color->g += p2*transp1;
        color->b += p3*transp1;

        if (color->r > 255.0) {
            color->r = 255.0;
            *f = 1;
        }
        if (color->g > 255.0) {
            color->g = 255.0;
            *f = 1;
        }
        if (color->b > 255.0) {
            color->b = 255.0;
            *f = 1;
        }
      }
    }
```

Anexo B

Código em linguagem C para a função ***Brightness***.

A função ***Brightness*** calcula o termo de iluminação utilizando as equações de sombreamento. A iluminação é calculada para a fonte de luz ***light***, utilizando o raio definido por ***pos*** (posição de intersecção do raio incidente com o objeto mais próximo) e ***ray*** (direção do raio apontando para a fonte de luz).

```
double Brightness(light, pos, ray)
LightNode  *light;
P3d        *pos, *ray;
{
  ObjectNode    *tmpcurrentobject2, *tmpclosestobject2,
                *tmpcurrentobject, *tmpclosestobject;
  int           tmpobjectlistlevel, tmpobjectlistlevel2;
  double        tmpclosestdistance, tmpclosestdistance2, distance,
                L_dot_Ll;
  P3d           NL, Focus, hit, pa;
  double        s1, s2, step_t, t, transp, d1, dd, tr;
  ObjectNode    *o1, *o2;
  ParticleSystemObj *ps1;
  int           ant_grid_x, ant_grid_y, ant_grid_z,
                grid_x, grid_y, grid_z, f;
  unsigned long int pos0, pos1, pos2, i;
  unsigned char p1, p2, p3;
  unsigned int  x1, y1, z1;

  tmpcurrentobject = CurrentObject;
  tmpclosestobject = ClosestObject;
  tmpclosestdistance = ClosestDistance;
  tmpobjectlistlevel = ObjectListLevel;

  CurrentObject = FirstObject;
  ClosestDistance = FAR_AWAY;
  ClosestObject = NULL;
  ObjectListLevel = 1;

  while ((CurrentObject != NULL) && (ClosestDistance >=
                          (LightDistance-ROUND_ERROR))) {
    (*ObjectIntersect[CurrentObject->ObjectType])(CurrentObject,
                                    *pos, *ray);
    CurrentObject = CurrentObject->Next;
  }

  s1 = FAR_AWAY;

  if (ClosestDistance <= LightDistance) {
    if (ClosestObject->ObjectType != PARTICLE_SYSTEM_TYPE) {
        CurrentObject   = tmpcurrentobject;
        ClosestObject   = tmpclosestobject;
        ClosestDistance = tmpclosestdistance;
```

```
    ObjectListLevel = tmpobjectlistlevel;
    return 0.0;
} else {
    s1 = ClosestDistance;
    ps1 = (ParticleSystemObj *) ClosestObject->Descr;

    ClosestDistance = FAR_AWAY;
    ClosestObject   = NULL;
    CurrentObject   = FirstObject;

    while (CurrentObject != NULL) {
      if (CurrentObject->ObjectType != PARTICLE_SYSTEM_TYPE) {
        (*ObjectIntersect[CurrentObject->ObjectType])
                    (CurrentObject, *pos, *ray);
      }
      CurrentObject = CurrentObject->Next;
    }

    if (ClosestDistance <= LightDistance) {
      CurrentObject   = tmpcurrentobject;
      ClosestObject   = tmpclosestobject;
      ClosestDistance = tmpclosestdistance;
      ObjectListLevel = tmpobjectlistlevel;
      return 0.0;
    }

    s2 = ClosestDistance;

    step_t = ps1->Step;
    ant_grid_x = -1;
    ant_grid_y = -1;
    ant_grid_z = -1;
    transp = 1.0;

    for (t=s1 ; t <= s2 ; t += step_t) {
      hit.x = pos->x + t*ray->x;
      hit.y = pos->y + t*ray->y;
      hit.z = pos->z + t*ray->z;

      if (fabs(hit.x-ps1->Center.x)-ps1->Size.x>ROUND_ERROR ||
              fabs(hit.y-ps1->Center.y)-ps1->Size.y>ROUND_ERROR ||
              fabs(hit.z-ps1->Center.z)-ps1->Size.z>ROUND_ERROR) {
        break;
     } else {
        FindGridPosition(hit, &grid_x, &grid_y, &grid_z,
                    ps1);

        if (grid_x != ant_grid_x || grid_y != ant_grid_y ||
          grid_z != ant_grid_z) {
                pos0 = 3*(grid_x+64L*grid_y+4096L*grid_z);

                if (pos0 == 0) {
                  pos1 = 0;
                  pos2 = 0;
                } else {
                  fseek(FHeader,(long)(pos0-3),SEEK_SET);
```

```
                    fscanf(FHeader, "%c%c%c", &p1, &p2, &p3);
                    pos1 = p1*65536L + p2*256L + p3;

                    fscanf(FHeader, "%c%c%c", &p1, &p2, &p3);
                    pos2 = p1*65536L + p2*256L + p3;
                  }

                  for(i=10*pos1;i<10*pos2;i+=10) {
                    fseek(FParticles,(long)(i),SEEK_SET);

                    ReadInt(FParticles, &x1);
                    ReadInt(FParticles, &y1);
                    ReadInt(FParticles, &z1);

                    pa.x = x1*256.0/65536.0-128.0 - pos->x;
                    pa.y = y1*256.0/65536.0-128.0 - pos->y;
                    pa.z = z1*256.0/65536.0-128.0 - pos->z;

                    d1 = DotP(&pa,ray);
                    dd = pa.x*pa.x + pa.y*pa.y + pa.z*pa.z - d1*d1;
                    tr = pow(0.5,dd/(ps1->PartSize*ps1->PartSize));

                    fscanf(FParticles, "%c", &p1);
                    transp *= (1.0-tr*p1/255.0);
                  }

                  ant_grid_x = grid_x;
                  ant_grid_y = grid_y;
                  ant_grid_z = grid_z;
              }
            }
          }
        }
      }

    CurrentObject   = tmpcurrentobject;
    ClosestObject   = tmpclosestobject;
    ClosestDistance = tmpclosestdistance;
    ObjectListLevel = tmpobjectlistlevel;

    if (s1 <= LightDistance) {
      return light->Bright * transp;
    } else {
      return (light->Bright);
    }
  }
```

Anexo C

Código em linguagem C para a função ***ParticleSystemIntersect***.

A função ***ParticleSystemIntersect*** testa a intersecção entre o raio definido por ***pos*** e ***ray*** (posição e direção do raio) e o sistema de partícula definido no objeto ***object***.

```
void ParticleSystemIntersect(object, pos, ray)
ObjectNode *object;
P3d        pos, ray;
{
  double            s, xhit, yhit, zhit, intersection[2];
  ParticleSystemObj *boundary;
  int p;
  double aux, t, step_t, dist_sp, d1, dd;
  int ant_grid_x, ant_grid_y, ant_grid_z, intersec,
     grid_x, grid_y, grid_z;
  P3d hit, pa;
  unsigned long int pos0, pos1, pos2;
  unsigned char p1, p2, p3;
  unsigned int x1, y1, z1;
  unsigned long int i;

  boundary = (ParticleSystemObj *) (object->Descr);

  pos.x = pos.x - boundary->Center.x;
  pos.y = pos.y - boundary->Center.y;
  pos.z = pos.z - boundary->Center.z;

  p = 0;

  if ( ray.x != 0 ) {
    s = (boundary->Size.x - pos.x) / ray.x;
    yhit = fabs(pos.y + s * ray.y);
    zhit = fabs(pos.z + s * ray.z);
    if ((yhit <= boundary->Size.y) && (zhit <= boundary->Size.z)){
        intersection[p] = s;
        p++;
    }
    s = (-boundary->Size.x - pos.x) / ray.x;
    yhit = fabs(pos.y + s * ray.y);
    zhit = fabs(pos.z + s * ray.z);
    if ((yhit <= boundary->Size.y) && (zhit <= boundary->Size.z)){
        intersection[p] = s;
        p++;
    }
  }

  if ( ray.y != 0 ) {
    s = (boundary->Size.y - pos.y) / ray.y;
    xhit = fabs(pos.x + s * ray.x);
    zhit = fabs(pos.z + s * ray.z);
    if ((xhit <= boundary->Size.x) && (zhit <= boundary->Size.z)){
```

```
        intersection[p] = s;
        p++;
    }
    s = (-boundary->Size.y - pos.y) / ray.y;
    xhit = fabs(pos.x + s * ray.x);
    zhit = fabs(pos.z + s * ray.z);
    if ((xhit <= boundary->Size.x) && (zhit <= boundary->Size.z)){
        intersection[p] = s;
        p++;
    }
}

if ( ray.z != 0 ) {
    s = (boundary->Size.z - pos.z) / ray.z;
    xhit = fabs(pos.x + s * ray.x);
    yhit = fabs(pos.y + s * ray.y);
    if ((xhit <= boundary->Size.x) && (yhit <= boundary->Size.y)){
        intersection[p] = s;
        p++;
    }

    s = (-boundary->Size.z - pos.z) / ray.z;
    xhit = fabs(pos.x + s * ray.x);
    yhit = fabs(pos.y + s * ray.y);
    if ((xhit <= boundary->Size.x) && (yhit <= boundary->Size.y)){
      intersection[p] = s;
        p++;
    }
}

if (p) {
    if (intersection[0]<ROUND_ERROR&&intersection[1]<ROUND_ERROR){
        return;
    } else {
        if (intersection[0] < ROUND_ERROR) {
          intersection[0] = 0.001;
        }
        if (intersection[1] < ROUND_ERROR) {
          intersection[1] = 0.001;
        }
    }

    if (intersection[0] > intersection[1]) {
        aux = intersection[0];
        intersection[0] = intersection[1];
        intersection[1] = aux;
    }

    step_t = boundary->Step;
    ant_grid_x = -1;
    ant_grid_y = -1;
    ant_grid_z = -1;

    intersec = 0;

    pos.x = pos.x + boundary->Center.x;
```

```
pos.y = pos.y + boundary->Center.y;
pos.z = pos.z + boundary->Center.z;

for(t=intersection[0] ; t<=intersection[1] ; t+=step_t) {
    hit.x = pos.x + t*ray.x;
    hit.y = pos.y + t*ray.y;
    hit.z = pos.z + t*ray.z;

    if (fabs(hit.x - boundary->Center.x) -
      boundary->Size.x > ROUND_ERROR ||
       fabs(hit.y - boundary->Center.y) -
      boundary->Size.y > ROUND_ERROR ||
       fabs(hit.z - boundary->Center.z) -
      boundary->Size.z > ROUND_ERROR) {
       break;
    } else {
       FindGridPosition(hit, &grid_x, &grid_y, &grid_z,
                  boundary);
       if (grid_x != ant_grid_x || grid_y != ant_grid_y ||
         grid_z != ant_grid_z) {
          pos0 = 3*(grid_x+64L*grid_y+4096L*grid_z);

          if (pos0 == 0) {
                  pos1 = 0;
                  fseek(FHeader,0L,SEEK_SET);
                  fscanf(FHeader, "%c%c%c", &p1, &p2, &p3);
                  pos2 = p1*65536L + p2*256L + p3;
          } else {
                  fseek(FHeader,(long)(pos0-3),SEEK_SET);
                  fscanf(FHeader, "%c%c%c", &p1, &p2, &p3);
                  pos1 = p1*65536L + p2*256L + p3;
                  fscanf(FHeader, "%c%c%c", &p1, &p2, &p3);
                  pos2 = p1*65536L + p2*256L + p3;
          }

          if ((pos2 - pos1) > 0) {
                  for(i=10*pos1;i<10*pos2;i+=10) {
                    fseek(FParticles,(long)(i),SEEK_SET);
                    ReadInt(FParticles, &x1);
                    ReadInt(FParticles, &y1);
                    ReadInt(FParticles, &z1);

                    pa.x = x1*256.0/65536.0-128.0 - pos.x;
                    pa.y = y1*256.0/65536.0-128.0 - pos.y;
                    pa.z = z1*256.0/65536.0-128.0 - pos.z;
                    d1 = DotP(&pa,&ray);
                    dd = pa.x*pa.x + pa.y*pa.y + pa.z*pa.z - d1*d1;

                    if (dd>boundary->PartSize*boundary->PartSize) {
                            continue;
                    } else {
                            dist_sp = t;
                            intersec = 1;
                            goto end1;
                    }
                  }
```

```
            }
          }
          ant_grid_x = grid_x;
          ant_grid_y = grid_y;
          ant_grid_z = grid_z;
        }
    }
end1:
    if (intersec) {
        ProcessIntersect(dist_sp, object);
    }
  }
}
```

Anexo D

Código em linguagem C para a função ***EvolveParticles***.

A função ***EvolveParticles*** efetua a animação das partículas do sistema utilizando o método de aceleração com a grade pentadimensional.

```
void EvolveParticles(void)
{
  int i, j, k, t, f;
  unsigned long int p, np, np1, np2, n;
  double t_min, f_err= 0.9, x, y,  z, d, u, v;
  unsigned int vx, vy, vz, px, py, pz;
  P3d pos[N_RAY_SAMPLES], ray[N_RAY_SAMPLES];

  for(i=0;i<8;i++) {
    for(j=0;j<8;j++) {
      for(k=0;k<8;k++) {
        for(t=0;t<16;t++) {
          for(f=0;f<32;f++) {
            p = (8192*f+512*t+64*i+8*j+k)*3;
            fseek(FHeader, p+3, SEEK_SET);
            Read3Bytes(FHeader, &np1);
            fseek(FHeader, p, SEEK_SET);
            Read3Bytes(FHeader, &np2);
            np = np2-np1;
            if (np > N_RAY_SAMPLES) {
              GetSampleRays(pos, ray, i, j, k, t, f);
              for(n=0;n<N_RAY_SAMPLES;n++) {
                Intersect(&pos[n], &ray[n]);
              }
              if (ClosestObject != NULL) {
                t_min = ClosestDistance;
                for(n=0;n<np;n++) {
                  fseek(FParticles, np1+n*21+6,SEEK_SET);
                  ReadInt(FParticles, &vx);
                  ReadInt(FParticles, &vy);
                  ReadInt(FParticles, &vz);
                  x = vx/256.0-128.0;
                  y = vy/256.0-128.0;
                  z = vz/256.0-128.0;
                  Conv2duv(x, y, z, &d, &u, &v);
                  if (d > t_min*f_err) {
                    fseek(FParticles, np1+21*n,SEEK_SET);
                    ReadInt(FParticles, &px);
                    ReadInt(FParticles, &py);
                    ReadInt(FParticles, &pz);
                    GetWorldCoordinates(px,py,pz,&pos[0]);
                    ray[0].x = x;
                    ray[0].y = y;
                    ray[0].z = z;
                    intersect(&pos[0], &ray[0]);
                    if (ClosestObject != NULL) {
```

```
                    Reflect(&pos[0], d);
                  } else {
                    pos[0].x += ray[0].x;
                    pos[0].y += ray[0].y;
                    pos[0].z += ray[0].z;
                  }
                  EvolveParticleAttributes(np1);
                  SaveParticle(pos[0], ray[0], np1);
                } else {
                  fseek(FParticles, np1+21*n,SEEK_SET);
                  ReadInt(FParticles, &px);
                  ReadInt(FParticles, &py);
                  ReadInt(FParticles, &pz);
                  GetWorldCoordinates(px,py,pz,&pos[0]);
                  ray[0].x = x;
                  ray[0].y = y;
                  ray[0].z = z;
                  pos[0].x += ray[0].x;
                  pos[0].y += ray[0].y;
                  pos[0].z += ray[0].z;
                  EvolveParticleAttributes(np1);
                  SaveParticle(pos[0], ray[0], np1);
                }
              }
            } else {
              for(n=0;n<np;n++) {
                fseek(FParticles, np1+n*21+6,SEEK_SET);
                ReadInt(FParticles, &vx);
                ReadInt(FParticles, &vy);
                ReadInt(FParticles, &vz);
                x = vx/256.0-128.0;
                y = vy/256.0-128.0;
                z = vz/256.0-128.0;
                Conv2duv(x, y, z, &d, &u, &v);
                fseek(FParticles, np1+n*21,SEEK_SET);
                ReadInt(FParticles, &px);
                ReadInt(FParticles, &py);
                ReadInt(FParticles, &pz);
                GetWorldCoordinates(px, py, pz, &pos[0]);
                ray[0].x = x;
                ray[0].y = y;
                ray[0].z = z;
                pos[0].x += x;
                pos[0].y += y;
                pos[0].z += z;
                EvolveParticleAttributes(np1);
                SaveParticle(pos[0], ray[0], np1);
              }
            }
          } else {
            for(n=0;n<np;n++) {
              fseek(FParticles, np1+n*21+6,SEEK_SET);
              ReadInt(FParticles, &vx);
              ReadInt(FParticles, &vy);
              ReadInt(FParticles, &vz);
              x = vx/256.0-128.0;
```

```
                                y = vy/256.0-128.0;
                                z = vz/256.0-128.0;
                                Conv2duv(x, y, z, &d, &u, &v);
                                fseek(FParticles, np1+n*21,SEEK_SET);
                                ReadInt(FParticles, &px);
                                ReadInt(FParticles, &py);
                                ReadInt(FParticles, &pz);
                                GetWorldCoordinates(px, py, pz, &pos[0]);
                                ray[0].x = x;
                                ray[0].y = y;
                                ray[0].z = z;
                                intersect(&pos[0], &ray[0]);
                                if (ClosestObject != NULL) {
                                  Reflect(&pos[0], d);
                                } else {
                                  pos[0].x += ray[0].x;
                                  pos[0].y += ray[0].y;
                                  pos[0].z += ray[0].z;
                                }
                                EvolveParticleAttributes(np1);
                                SaveParticle(pos[0], ray[0], np1);
                              }
                            }
                          }
                        }
                      }
                    }
                  }
                }
```

Bibliografia

[AMA 84] AMANATIDES, J. Ray Tracing with Cones. **Computer Graphics**, New York, v.18, n.3, p.129-135, July 1984. Trabalho apresentado na Annual Conference on Computer Graphics and Interactive Techniques, 11, 1984, Minneapolis, US.

[AMA 87] AMANATIDES, J.;WOO, A. A Fast Voxel Transversal Algorithm for Ray Tracing. In: EUROGRAPHICS, 1987. **Proceedings...** [S.l.:s.n.], 1987. p.3-10.

[AMB 93] AMBONI, R. **Síntese de Imagens Aplicando Sistemas de Partículas Refletoras de Luz**. Porto Alegre: CPGCC da UFRGS, 1993. 111 p. Dissertação de Mestrado.

[ARV 87] ARVO, J.;KIRK, D. Fast Ray Tracing by Ray Classification. **Computer Graphics**, New York, v.21, n.4, p.55-64, July 1987. Trabalho apresentado na Annual Conference on Computer Graphics and Interactive Techniques, 14., 1987.

[BUI 75] BUI-TUONG, P. Illumination for Computer Generated Images. **Communications of the ACM**, New York, v.18, n.6, p.311-317, 1975.

[COO 82] COOK, R. L.;TORRANCE, K. E. A Reflectance Model for Computer Graphics. **ACM Transaction on Graphics**, New York, v.1, n.1, p.7-24, 1982.

[COO 86] COOK, R. L. Stochastic Sampling in Computer Graphics. **ACM Transaction on Graphics**, New York, v.5, n.1, p.51-72, 1986.

[CUN 95] MCCUNE, J. A. **Interdependent Particle Systems**. Disponível em http://www.tc.cornell.edu/Visualization/Education/cs718/fall1995/mccune/final/report.html, 1995. (nov. 1995).

[EBE 94] EBERT, D. S.;CARLSON, W. E.;PARENT, R. E. Solid Spaces and Inverse Particle Systems for Controlling the Animation of Gases and Fluids. **The Visual Computer**, [S.l.], n.10, p. 179-190, 1994.

[FOL 90] FOLEY, J. D. et al. **Computer Graphics**: Principles and Practice. New York: Addison-Wesley, 1990. 1174 p.

[FUJ 86] FUJIMOTO, A.;TANAKA, T.;IWATA, K. ARTS: Accelerated Ray Tracing System. **IEEE Computer Graphics and Applications**, New York, v.6, n.4, p.16-26, 1986.

[GLA 84] GLASSNER, A. S. Space Subdivision for Fast Ray Tracing. **IEEE Computer Graphics and Applications**, New York, v.4, n.10, p. 15-22, 1975

[GLA 89] GLASSNER, A. S. **An Introduction to Ray Tracing**. San Diego: Academic Press Inc., 1989. 329 p.

[HAL 83] HALL, R. A.;GREENBERG, D. P. A Testbed for Realistic Image Synthesis. **IEEE Computer Graphics and Applications**, New York, v.3, n.8, p.10-20, 1983.

[HAL 89] HALL, R. A. **Illumination and Color in Computer Generated Imagery**. New York: Springer-Verlag, 1989.

[HEC 84] HECKBERT, P.;HANRAHAN, P. Beam Tracing Polygonal Objects. **Computer Graphics**, New York, v.18, n.3, p. 119-127, July 1984. Trabalho apresentado na Annual Conference on Computer Graphics and Interactive Techniques, 11., 1984, Minneapolis, US.

[HER 95] HERMAN, G.;REDKEY, D. **A Particle System Representation of Candle Wax**. Disponível em http://www-graphics.standford.edu/courses/cs348c-95-fall/projects/ redkey-herman/writeup.html, 1995. (nov. 1995).

[HOF 97] HOFSETZ, C. **Radiosidade Volumétrica**. Porto Alegre: CPGCC da UFRGS, 1997. Dissertação de Mestrado.

[KAJ 84] KAJIYA, J. T.;HERZEN, B. P. V. Ray Tracing Volume Densities. **Computer Graphics**, New York, v.18, n.3, p. 165-174, July 1984. Trabalho apresentado na Annual Conference on Computer Graphics and Interactive Techniques, 11., 1984, Minneapolis, US.

[KAP 85] KAPLAN, M. R. Space Tracing: A Constant Time Ray Tracer. **Computer Graphics**, New York, v.19, n.3, July 1985. Trabalho apresentado na Annual Conference on Computer Graphics and Interactive Techniques: Course Notes, 12., 1985.

[LOK 92] LOKE, T. S. et al. Rendering Fireworks Displays. **IEEE Computer Graphics and applications**, New York, v.12, n.3, p.33-43, 1992.

[NAG 94] NAGASAWA, M.;KUWAHARA, K. Smoothed Particle Rendering for Fluid Visualization in Astrophysics. In: **Scientific Visualization of Physical Phenomena**. New York: Springer-Verlag, 1994. p. 589-605.

[PAR 82] PARAMOUNT. **Star Trek II**: The Wrath of Khan. [S.l.]: Paramount, 1982. Filme.

[PER 95] PEREIRA, A. E.;HSU, T. **Modeling Lava Flows with a Particle System**. Disponível em http://www-graphics.standford.edu/courses/cs348c-95-fall/projects/ pereira-hsu/lava.html, 1995. (nov. 1995).

[REE 83] REEVES, W. T. Particle Systems: A Technique for Modeling a Class of Fuzzy Objects. **Computer Graphics**, New York, v.17, n.3, p.359-376, July 1983. Trabalho apresentado na Annual Conference on Computer Graphics and Interactive Techniques, 10., 1983.

[REE 85] REEVES, W. T.;BLAU, R. Approximative and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems. **Computer Graphics**, New York, v.19, n.3, p.313-322, July 1985. Trabalho apresentado na Annual Conference on Computer Graphics and Interactive Techniques, 12., 1985.

[REY 87] REYNOLDS, C. W. Flocks, Herds, and Schools: A Distributed Behavioral Model. **Computer Graphics**, New York, v.21, n.4, p.25-34, July 1987. Trabalho apresentado na Annual Conference on Computer Graphics and Interactive Techniques, 14., 1987.

[RUB 80] RUBIN, S. M.;WHITTED, T. A Three Dimensional Representation for Fast Rendering of Complex Scenes. **Computer Graphics**, New York, v.14, n.3, p.110-116, July 1980. Trabalho apresentado na Annual Conference on Computer Graphics and Interactive Techniques, 7., 1980.

[SAK 90] SAKAS, G. Fast Rendering of Arbitrary Distributed Volume Densities. In: EUROGRAPHIS, 1990. **Proceedings...** [S.l:s.n.], 1990. p.519-530.

[SHI 87] SHINYA, M.;TAKAHASHI, T.;NAITO, S. Principles and Applications of Pencil Tracing. **Computer Graphics**, New York, v.21, n.4, p.45-54, July 1987. Trabalho apresentado na Annual Conference on Computer Graphics and Interactive Techniques, 14., 1987.

[SHI 92] SHINYA, M.;FOURNIER, A. Stochastic Motion: Motion Under the Influence of Wind. In: EUROGRAPHICS, 1992. **Proceedings...** [S.l.:s.n.], 1992. p.119-128.

[SIM 90] SIMS, K. Particle Animation and Rendering Using Data Parallel Computation. **Computer Graphics**, New York, v.24, n.4, p.405-413, August 1990. Trabalho apresentado na Annual Conference on Computer Graphics and Interactive Techniques, 17., 1990.

[TAK 95] TAKAI, Y.;ECCHU, K.;TAKAI, N. K. A Cellular Automaton Model of Particle Motions and Its Applications. **The Visual Computer**, New York, n.11, p.240-252, 1995.

[TAY 96] TAYLOR, B. **Creating Realistic Fireworks Displays**. Disponível em http://www.cs.uml.edu/~btaylor/ fireworks.html, 1996. (mar. 1996).

[THA 86] THALMANN, D. et al. The Integration of Particle and Polygon Rendering Using an A-Buffer algorithm. In: EUROGRAPHICS, 1986. **Proceedings...** [S.l.:s.n.], 1986. p.161-169.

[TUL 95] TULL, A.;TSOI, H. **Controlable Rain, Rainbow and Oil Films**. Disponível em http://www-graphics.standford.edu/courses/cs348c-95-fall/projects/tull-tsoi/ writeup.html, 1995. (nov. 1995).

[WEG 84] WEGHORST, H.;HOOPER, G.;GREENBERG, D. P. Improved Computational Methods for Ray Tracing. **ACM Transaction on Graphics**, New York, v.3, n.1, p.52-69, 1984.

[WHI 80] WHITTED, T. An Improved Illumination Model for Shaded Display. **Comunications of ACM**, New York, v.23, n.6, p.343-349, 1980.