

TRABALHO PRÁTICO DE COMPUTAÇÃO GRÁFICA

SISTEMA SOLAR

Guilherme Brandão Biber Sampaio

Curso de Ciência da Computação – Uni-BH (www.unibh.br)

Resumo – O presente trabalho apresenta um simulador de sistema solar baseado em OpenGL e GLUT (freeglut), com escalas proporcionais, uso de texturas, animação de explosão, detecção de colisão, análise de desempenho assim como especificado pela descrição do trabalho proposto pelo professor Moisés Ramos.

Palavras-chave – simulação, sistema solar, OpenGL, GLUT, colisão.

1 Introdução

Como praticamente todo software de computação gráfica, o trabalho foi baseado em diversas bibliotecas e outros softwares, como OpenGL e o GLUT. A estrutura geral foi baseada nos programas estudados em sala. Para alguns recursos mais “avançados”, foram adaptados outros softwares amplamente disponibilizados pela Internet. Alguns desses recursos não foram abordados com maiores detalhes na disciplina de Computação Gráfica do curso de Ciência da Computação do UNI-BH. Por isso, este trabalho foi uma ótima oportunidade para adquirir novos conhecimentos e colocar em prática conhecimentos de várias disciplinas, em especial Álgebra Linear. Por exemplo, para a análise de colisão, foram usadas funções comuns de álgebra linear e princípios gerais do uso das matrizes de “projeção” e “visualização” do OpenGL. A técnica utilizada, “Pick Ray”, é amplamente tratada em livros, artigos e sites especializados (principalmente games). As referências conceituais e de software, assim como outros materiais usados (ex. imagens das texturas), estão dispostas em comentários dos programas.

O software foi elaborado no Eclipse, utilizando-se o compilador GCC e a biblioteca “freeglut”. Sua compatibilidade foi testada no Windows 7 (MinGW), OpenSuse 12.3 e Ubuntu 11.04. Para a compilação e “linkedição” no Linux, existe o programa “make.sh” (baseado na saída de console construção do projeto do Eclipse). Obviamente, as bibliotecas do OpenGL e o “freeglut” devem estar instalados para a criação do executável. Para o Windows, o executável foi “linkado” de forma estática e deve funcionar mesmo sem o “runtime” do “freeglut.dll”. Para a carga das texturas, que é opcional, o programa deve ser executado partindo-se de um diretório que contenha o subdiretório “**texturas**” com os bitmaps usados.

2 Comandos para utilização do software

Teclas:

- l: Liga/Desliga Luz
- p: Pausa/Continua a movimentação (passagem do tempo)
- q: Aumenta escala de raio dos planetas
- a: Diminui escala de raio dos planetas
- w: Aumenta distância dos planetas
- s: Diminui distância dos planetas
- z: Volta para escala padrão
- x: Tamanho "ampliado"
- o: Liga/Desliga órbitas
- r: Volta os planetas explodidos
- e: Ativa asteróide
- d: Dispara/termina asteróide
- c: Termina asteróide
- t: Liga/Desliga textura
- j: Finaliza programa

Mouse:

Botão:

- Direito: Mover câmera
- Esquerdo: Atirar asteróide e também mover câmera
- Meio: Surpresinha ;)

Rodar botão do meio:

- Para frente: Aumenta zoom
- Para trás: Diminui zoom

Setas:

- Cima: Aumenta zoom
- Baixo: Diminui zoom
- Direita: Diminui tempo
- Esquerda: Aumenta tempo

Focar:

- 1: Sol
- 2: Mercúrio
- 3: Vênus
- 4: Terra
- !: Lua
- 5: Marte
- 6: Júpiter
- 7: Saturno
- @: Titan
- #: Febe
- \$: Hiperion
- 8: Urano
- 9: Netuno
- 0: Plutão
- : Asteroide

3 Alguns detalhes e exemplos

Como descrito na especificação do trabalho, o software usa uma estrutura de dados aninhada, na qual um “astro principal” contém os seus “secundários”. O astro principal é a referência (ponto $\{0,0,0\}$) para o desenho dos seus secundários que orbitam ao seu redor. Os dados de rotação, translação, distância espacial e raio planetário foram obtidos em sites de astronomia. O programa faz a conversão de escalas. Como as distâncias entre os planetas é muito maior que o tamanho deles, o programa usa duas escalas: uma para a distância espacial e outra para as dimensões dos corpos celestes (raio). A escala de raio é bem maior que a da distância espacial, para destacar os planetas. O software permite a mudança dessas escalas, mas com limitação e controle para evitar deformação visual ou excesso de “renderização” em escalas muito pequenas e distantes (problema observado na prática). Sempre que possível, o programa emite uma mensagem explicativa das limitações ou bloqueio de comandos.

A câmera foca qualquer “corpo celeste”, que pode ser selecionado pelo teclado. O movimento de câmera é esférico, em torno de seu foco, com zoom de velocidade adaptativa pela distância (afasta mais rápido com o aumento da distância). No Windows, o botão de rolamento (roda) do mouse pode ser usado para zoom.

Foi elaborada uma animação de objeto, no caso um “asteróide”, com detecção de colisão com os demais “elementos visuais”. Ele faz um movimento linear, partindo da posição da câmera em direção ao ponto projetado pela posição do mouse na tela (2D projetado em 3D). O segmento de reta da trajetória do “asteróide” é desenhado, caso a opção de desenho de órbita esteja ligada. Para fazer a detecção de colisão, o software necessita conhecer a posição no espaço “3D” de cada objeto. Como os objetos são circulares, basta conhecer o ponto central das esferas (ou o círculo do anel de saturno). Para determinar de forma correta as posições, executa-se uma passagem pela sequência de “renderização” (`glPushMatrix`, `glTranslated`, `glRotated`, `glPopMatrix`, etc), mas sem fazer o desenho, apenas coletando-se os dados de projeção (`gluUnProject`), baseados nas matrizes de “modelo” e “visualização” do OpenGL. Depois de calculados os pontos executam-se as verificações de colisão. Esses cálculos ocorrem na “thread” do evento de espera do GLUT (`glutIdleFunc`). A “renderização” efetiva, ocorre na “thread” do evento de desenho (`glutDisplayFunc`).

O programa adotou um controle de dia e passagem de tempo variável, assim como os exemplos estudados em práticas de laboratório da disciplina de CG. Por isso, não foi necessário o uso da temporização (`glutTimerFunc`), pois a “velocidade” percebida dos movimento pode ser definida pelo usuário. A captura de comandos do usuário (interatividade) utilizou as “callbacks” padrão do GLUT, com a exceção da rolagem do mouse que é específica do “freeglut”.

A detecção e resposta às colisões seguiu a especificação do trabalho. Porém, foram adotadas duas regras específicas: (1) no caso de colisão com o sol, o asteróide deve ser simplesmente eliminado e (2) a colisão com o anel de saturno não destrói o asteróide, para possibilitar um efeito visual “melhor”. A colisão com os “planetas” calcula a distância entre os centros das “esferas”. Caso a distância do centro do asteróide até o centro de qualquer planeta seja menor que os raios somados, existe uma colisão. Essa análise de colisão foi relativamente simples. O problema identificado nos testes é que, ao acelerar muito o tempo, entre um quadro e outro o asteróide poderia (como ocorreu) percorrer uma distância maior que a do planeta, sem possibilidade de detectar a colisão. Uma alternativa seria fixar a aceleração do tempo máxima.

Porém, o movimento de translação Júpiter é tão lento (mais de 200 anos) que só é notável em alta “velocidade” de passagem de tempo. Por isso, calcula-se a distância percorrida pelo asteroide entre um quadro e outro. Se ela for maior que o diâmetro do asteroide, o programa reduz a aceleração do tempo ou impede o acionamento do asteroide.

No caso de colisão, inicia-se uma animação de explosão. A animação usa dois elementos básicos: partículas e detritos. Os elementos são posicionados ao longo da superfície da esfera definida pelo planeta que “explode”. Elas se dispersam em forma esférica. Cada elemento tem um vetor de deslocamento com direção radial e “intensidade” (módulo ou velocidade) sorteados aleatoriamente (função `rand()`). Na animação, esses vetores são multiplicados fazendo o movimento. As partículas são pontos. Os detritos são triângulos. Os detritos giram também com ângulos sorteados. Todo o sorteio é feito no início da explosão. Depois, ao longo da animação, os vetores são apenas deslocados (multiplicação por escalar). O tamanho dos detritos e o tempo de duração da explosão (combustível) variam com o raio dos planetas; quanto maior o planeta, mais tempo de explosão e maior são os detritos. Por fim, durante o deslocamento, a cor de cada partícula é alterada, forçando uma cor gradualmente mais vermelha. Esse efeito é simples: basta, a cada movimento, subtrair os valores da cor RGB, reduzindo um valor “k” na cor vermelha, “4*k” na cor verde e “8*k” na cor azul. Ao longo do tempo, apenas a cor vermelha ficará com intensidade luminosa. O software adotou um modelo de referência amplamente citado na Internet, mas fez grandes mudanças como, por exemplo, o posicionamento inicial ao longo da esfera do planeta, que antes era em ponto central. Além disso, adicionou-se o controle de tempo, assim as explosões acompanham a aceleração de tempo do sistema solar. Foram feitas várias outras modificações e acréscimos, a maioria não trivial.

Saturno foi, de longe, o astro com mais requisitos de software e complicações. A lua Hiperion tem movimento de rotação, raios da elipse e translação aleatórios. Esse foi um controle difícil por causa da rotação de Saturno e por causa do fator “dia” no posicionamento relativo ao tempo. Note-se, na figura 1, que a órbita de Hiperion (em verde) passa de praticamente circular para elíptica (variação do raio maior e menor da elipse). Febe também tem órbita elíptica. Optou-se por usar uma órbita com inclinação em relação ao plano Y, apenas para variação visual. A lua Titan é simples, pois o sistema “renderização” padrão faz seu tratamento normal, como os demais planetas de órbita circular.

O anel de Saturno usa o recurso de desenho em pontos, “Stipple”, do OpenGL. Ao ser ativado (`glPolygonStipple`), a “renderização” filtra os pontos que são projetados na tela. O filtro é uma sequência de bits: os “bits” “0” (zero) anulam a projeção do ponto e os “1”(um) permitem a projeção do ponto. Isso faz um efeito de “transparência”. Esse mesmo tipo de efeito é usado para desenhar as linhas das órbitas, que ficam parecendo pontilhadas. Além disso, para complementar a transparência utiliza-se a opção “`glBlendFunc`” que permite e controla o uso de cores com índice “alfa” de transparência. A transparência com índice “alfa” é um efeito resultante da “fusão” da cor de um ponto anteriormente projetado no mesmo “voxel” por objeto mais distante. Por isso, a ordem de “renderização” é muito importante. Veja na figura 2 o efeito da transparência no anel de Saturno.

A colisão com o anel é diferente, pois ele tem forma plana. A solução foi verificar se o asteroide cruza o plano definido pelo anel, no caso o plano $\{0,Y,0\}$. A cada movimento, se o asteroide intercepta esse plano, verifica-se a distância do seu ponto central até o centro do anel (o centro de Saturno) para determinar se ocorre colisão.

Quando ocorre a colisão, a animação do “colapso” dos anéis faz com que eles se tornem vermelho, expandido seu raio ao longo do tempo, em direção “infinita”, reduzindo a cor até sumir. A animação tenta simular um “escape” dos “detritos” do anel da ação da força gravitacional de Saturno.

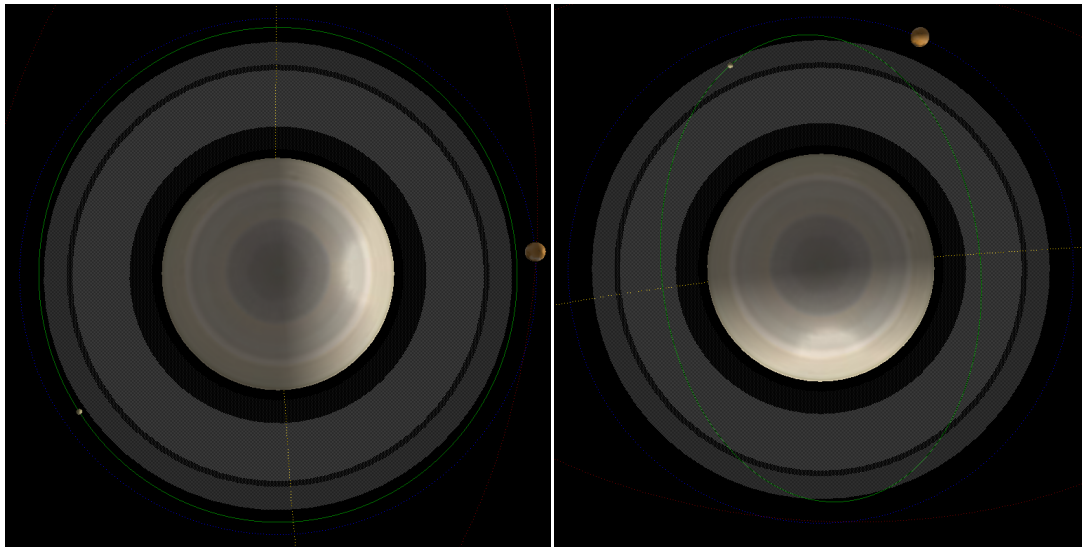


Figura 1 - Variação aleatória de Hiperion

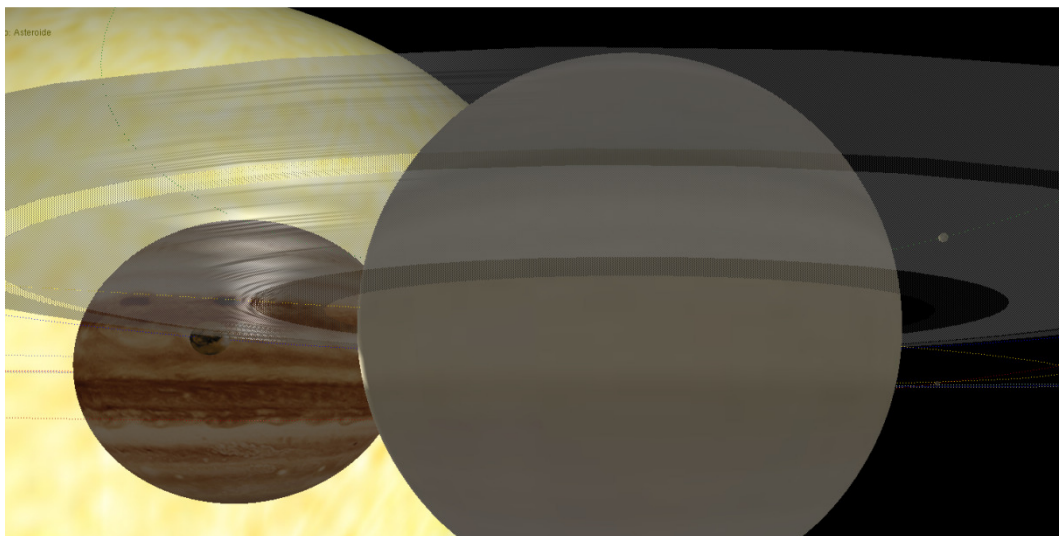


Figura 2- Transparência no anel de saturno

Por fim, como o sistema de detecção de colisão forneceu as posições dos objetos no espaço, foi elaborado um sistema de seleção de objetos pelo ponteiro do mouse. Esse foi o controle mais complexo e trabalhoso e elaborado. Ele é baseado na técnica “Pick Ray”, que calcula a proximidade do objeto com um segmento de reta formado pela posição da câmera e a posição projetada do mouse na tela (2D) no espaço 3D . O segmento de reta é composto pelos pontos que interceptam os planos “near” e “far” do volume de visualização. Veja ilustração do segmento entre os pontos p_0 - p_1 na figura 3. Esse segmento é similar à reta da orbita do

asteróide. Porém, uma reta pode interceptar objetos atrás da câmera, e esse foi um dos problemas encontrados nos testes.

Com o segmento de reta de seleção, basta calcular a distância entre os pontos centrais dos planetas com esse segmento de reta: se for menor que o raio do planeta, a reta o intercepta. Para todos os “planetas” que interceptarem o segmento, escolhe-se o que tiver menor distância com o ponto da câmera (distância entre dois pontos no espaço).

Para o anel de saturno, a seleção é diferente. Basta descobrir se existe e qual é o ponto de interseção do segmento de seleção (p_0 - p_1) com o plano definido pela superfície do anel que, no caso, é o plano Y. Com esse ponto de interseção, basta verificar se a distância entre ele e o centro de Saturno é menor que o raio do anel. Porém, a seleção do anel de saturno ainda não foi elaborada.

O teste de seleção de objetos pelo mouse é relativamente complexo, principalmente se comparado com a detecção de colisão. Por isso, mesmo sem ter encontrado erro na versão final, esse sistema não foi amplamente testado como a detecção de colisão, pois não é um fator de avaliação do presente trabalho.

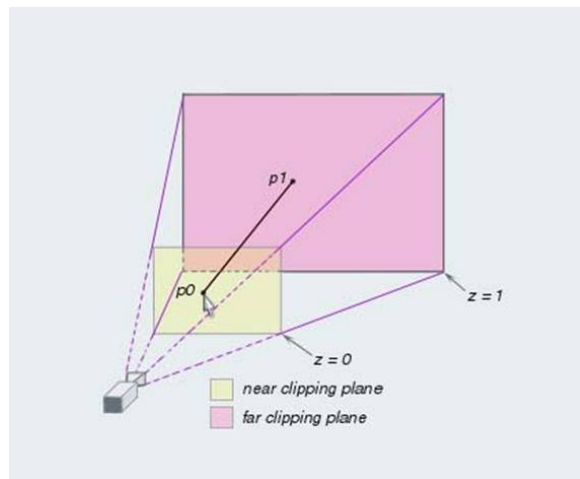


Figura 1 - Volume de visualização e segmento de reta de seleção (p_0 - p_1)

4 Conclusão

O trabalho foi realmente penoso, mas os resultados foram compensadores.

Agradeço ao professor Moisés pela oportunidade e adiamento do prazo, sem o qual não seria possível completar o trabalho.

Como o trabalho foi feito individualmente, e não em grupo, devo agradecer ao meu Tio, e colega de faculdade, Frederico Sampaio, pela ajuda e horas de ensino, esclarecimentos, exemplos e revisão do trabalho.