Computação Gráfica

Projecto

João Miguel Moreira de Carvalho Brás Simões, Nº 2011150045
João Ricardo Maximiano Leitão Ribeiro Lourenço, Nº 2011151194
Joaquim Pedro Bento Gonçalves Pratas Leitão, Nº 2011150072

25 de Junho de 2014

Relatório

1 Introdução

Com o presente trabalho pretende-se aprofundar e aplicar os conhecimentos abordados *Computação Gráfica*, trabalhando outros aspetos não lecionados na disciplina.

Os principais requisitos e componentes do projeto resumem-se à criação de uma aplicação gráfica, que recorre à tecnologia de *OpenGL*, e que contém o modelo de um robot, elemento chave da aplicação.

2 O jogo – A ideia

O jogo é jogado numa sala rectangular, na qual estão várias caixas, algumas das quais transparentes. Um robô está colocado nesta sala, emitindo um laser. O objectivo do jogo consiste em utilizar as propriedades reflectoras e refractivas dos objectos para direccionar o laser para um alvo colocado aleatoriamente no mapa. Quando se atinge o alvo, este é reposicionado. O jogador pode mover e rodar os objectos.

Neste sentido, não se trata propriamente de um "jogo". Apesar de ser um conceito divertido por um bocado, ele serve como uma plataforma para ilustrar diversas potencialidades da computação gráfica e a maneira como as implementámos.

3 Conceitos de Computação Gráfica aplicados

Como já referido, procurámos evidenciar as potencialidades de vários conceitos, uns mais sofisticados do que outros. Assim:

- Utlização apenas do pipeline programável. Ao optarmos por utilizar o pipeline programável tivemos de implementar por nós o sistema de iluminação, tendo optado por usar o *Modelo de Phong*, que faz a interpolação das normais, isto é, funciona por fragmento e não por vértice[1].

- Implementação de transparências (e a conjugação de objectos transparentes com objectos opacos). Para tal recorremos às potencialidades do pipeline programável para definir transparências por objecto de jogo (em vez de por vértice). Além disso, gerimos separadamente os objectos opacos e os transparentes, para evitar problemas com o "z-buffer".

[1]Para nos auxiliar na implementação do pipeline programável, baseamo-nos em diversas fontes, algo que se reflecte na nossa abordagem e no nosso código. São elas: `http://www.codeincodeblock.com/2013/07/modern-opengl-32-tutorial-series-with.html`, `http://open.gl/`, `http://tomdalling.com/blog/modern-opengl/` e `http://en.wikibooks.org/wiki/GLSL_Programming/`

- Movimentação da câmara como num jogo do estilo "FPS". Para atingir este objectivo utilizámos as capacidades da biblioteca *GLEW*, sabendo a cada instante o estado do rato e do teclado.

- Carregamento de modelos no formato .obj. para carregar os vários modelos (feitos por nós, com excepção do robot, em *blender*) desenvolvemos o nosso parser de ficheiros .obj.

- Associação entre modelos e texturas, bem como outras propriedades – conceito de objecto. O nosso jogo possui dentro de si um primitivo motor de jogo que poderia ser usado para conceitos completamente distintos. Parte deste motor de jogo assenta fortemente na ideia de existir objectos, que são uma ssociação entre modelos e texturas, com outras propriedades (propriedades de transparência, *Bounding Boxes*, etc)

- Detecção de colisões. Uma parte significativa do tempo dispendido no desenvolvimento do projecto focou-se na correcta detecção de colisões. Assumimos um modelo simplificado do mundo, em que apenas possuímos rotações em torno do eixo vertical (y), recorrendo a *Bounding Boxes* com o aspecto de prismas rectangulares.

- Gestão de colisões. Não menos importante do que a detecção de colisões, é fulcral saber como reagir face à detecção de uma colisão entre objectos que movem, objectos parados, objectos e o jogador, etc.

- O laser – uma forma de ray-tracing. Podemos de algum modo afirmar que o nosso laser é uma forma simplificada de ray-tracing. Efectivamente, temos de ver o caminho que o laser percorre, quais os objectos em que incide, reflecti-lo, ou refractá-lo (consoante as propriedades do objecto), suportando até índices de refracção diferente.

- Transformações. É evidente que o nosso projecto exige diversas transformações geométricas, não só a nível gráfico (objectos no ecrã), como a nível das estruturas internas (*Bounding Boxes*, etc)

4 Breve explicação mais detalhada de alguns mecanismos

4.1 Phong Shading

O *Phong Shading* foi implementado a nível do shader de fragmentos, com os coeficientes de especularidade R,G,B para cada objecto, bem como o seu brilho (*Shininess*) passados ao shader e variando de objecto para objecto. O cálculo da iluminação em cada fragmento é, pois, feito de acordo com o

modelo teórico abordado nas aulas, simplesmente movido do cálculo por vértice para um cálculo por fragmento

4.2 Laser

Graficamente, o laser consiste numa linha. Esta linha é desenhada com recurso a um shader específico (o shader do laser), para o separar do shader de iluminação. Para além disso, em cada objecto em que o laser incide, perto do ponto de incidência, é colocada uma pequena luz que apenas incide sobre esse objecto (algo que é conseguido através do shader de iluminação), para produzir um efeito de mais "ficção científica", semelhante ao visto em vários jogos.

Interiormente, o caminho do laser é calculado de forma recursiva (até uma profundidade máxima configurável), detectando as colisões dos lasers com todos os objectos[2] e escolhendo qual a que está mais próxima do ponto de origem. Se for uma superfície opaca, o laser é reflectido e o processo reiniciado (até que atinja o alvo ou uma parede do mapa, ou até que atinja a profundidade máxima). Caso o objecto seja transparente, o laser é propagado por dentro dele, com o coeficiente de refracção do material, seguindo-se outra refracção "à saída" deste objecto. Implementámos, pois, uma forma primitiva de ray-tracing.

4.3 Detecção de colisões

A detecção de colisões é feita com recurso a *Bounding Boxes* definidas pelo utilizador do nosso motor de jogo. A intersecção das bounding boxes é feita a 2D, pois não permitimos rotações nos eixos x e z, possibilitando esta simplificação. Para detectar colisões, o método que implementámos intersecta as rectas das bounding boxes e, caso não se intersectem, verifica ainda se um objecto está dentro ou fora do outro. O processo de detecção de colisões é feito até que não haja mais objectos a colidir (em conjunto com o de gestão de colisões).

4.4 Gestão de colisões

Para gerir as colisões, os objectos são afastados de acordo com o eixo que os une, sendo que o objecto que se afasta é aquele que sabemos não estar originalmente a ser empurrado. Internamente, utilizamos também o conceito de velocidade para levar várias objectos a deslocarem-se. De todos os modelos que testámos, este pareceu-nos o mais simples e eficaz.

[2]Na prática, apesar de possuirmos um motor tridimensional, podemos considerar que temos um motor bidimensional com alturas, pois a maior parte do nosso código funciona de forma bidimensional, fazendo apenas uma verificação por consistência das alturas.

4.5 Detecção do objecto apontado pelo utilizador

O utilizador apenas pode fazer rodar o objecto que está directamente à sua frente no plano do chão. Esta simplificação permite-nos partilhar código comum entre o laser (que tem de traçar um raio que passa pelos objectos) e a detecção do objecto à frente do utilizador (traçando um raio imaginário que vai do utilizador e chega aos objectos). Poderíamos ter utilizado técnicas mais sofisticadas, mas dado que limitámos o jogo a objectos que não se empilham[3], pareceu-nos uma solução adequada.

[3] Apesar de em vários aspectos tal ser suportado – por exemplo, o conceito de gravidade está presente no mesmo.