Programação e Interacção

Avaliação por trabalho individual

Enunciado - 2003/04

Introdução

O trabalho que a seguir se apresenta constitui a avaliação da disciplina H2: Tópicos de Programação e Interacção do Mestrado em Computação Gráfica e Ambientes Virtuais. O objectivo principal do trabalho é consolidar os conhecimentos adquiridos nas aulas e que visaram fornecer mecanismos de homogeneização nas linguagens de programação que vão ser necessárias no decurso do mestrado.

Com vista ao reforço das noções de programação em C++, o presente trabalho opta por uma problemática retirada da área da computação gráfica, por forma a que esforços efectuados no âmbito da resolução do trabalho possam eventualmente ser reutilizados noutras disciplinas.

O tempo para resolução do trabalho é de 2 semanas, estando a sua apresentação oral marcada para o dia 28 de Novembro de 2003. Os alunos devem enviar email para `anr@di.uminho.pt` por forma a sugerir alternativas no dia 27 ou 29.

Durante a apresentação os alunos devem entregar um pequeno relatório com a descrição do trabalho realizado, principais decisões tomadas e anexar o código fonte.

Objectivos

É objectivo do trabalho que se possam efectuar as seguintes operações:

1. Criar uma classe que representa pontos no espaço 3D (numa perspectiva vectorial). Estabelecer para esses pontos um conjunto de operações que permitam somar, multiplicar, efectuar deslocamentos sobre vectores 3D, que estão fornecidas no header file (.h) que é disponibilizado.

2. Criar uma classe que permita representar um conjunto de vectores 3D (uma nuvem de pontos). Esta classe deve poder guardar os vectores 3D e permitir acrescentar pontos ao conjunto, calcular a bounding box do conjunto e a bounding sphere do mesmo.

3. Criar uma classe que represente um plano no espaço 3D. Esta classe deve conseguir determinar a distância de um vector 3D (um ponto 3D) ao plano.

4. Criar uma classe que implemente um View Frustum e determine se a bounding sphere de um conjunto de vectores 3D se encontra dentro do view frustum.

Descrição

Vectores 3D

Considere-se que o seguinte header file representa um vector 3D.

```
class Vect3D
{

public:
```

```
        float x,y,z;


        Vect3D::Vect3D(float x, float y, float z);
        Vect3D::Vect3D(const Vect3D &v);
        Vect3D::Vect3D();
        Vect3D::~Vect3D();

        Vect3D Vect3D::operator +(Vect3D v);
        Vect3D Vect3D::operator -(Vect3D v);
        Vect3D Vect3D::operator *(Vect3D v);
        Vect3D Vect3D::operator *(float t);
        Vect3D Vect3D::operator /(float t);
        Vect3D Vect3D::operator -(void);
        float length();
        void Vect3D::normalize();
        float Vect3D::innerProduct(Vect3D  v);
        void copy(Vect3D v);
        void set(float x,float y, float z);
        Vect3D scalarMult(float a);

        void Vect3D::print();

};
```

No ficheiro `vect3D.cpp`,que também se anexa, estão já definidas grande parte das operações necessárias para trabalhar com vectores 3D.

Sugestão: para determinar se o comportamento pretendido para esta classe está bem implementado crie um programa de teste (com uma função `main()`).

Conjunto de Vectores 3D

Após termos capacidade de termos vectores 3D é necessário ter agora a capacidade de guardarmos uma núvem de pontos 3D (um conjunto de pontos). Ao termos uma classe com esta capacidade podemos efectuar algumas operações que envolvam os pontos 3D que temos.

Podemos calcular a bounding box que contém todos os pontos, ou a bounding sphere que os circunscreve.

A classe que descreve o comportamento de um conjunto de pontos 3D tem a seguinte header file (incompleto):

```
#include <vector>
#include "Vect3D.h"

using namespace std;

class Object3D {

private:
  vector <Vect3D> points;
  Vect3D mins,maxs; // estes dois triplos definem a bounding box do objecto
  //mais definições que precise de colocar
  //...
```

```
public:
  Object3D::Object3D();
  Object3D::~Object3D();
  Object3D::Object3D(vector<Vect3D>*p);
  void print();
  void addPoint(Vect3D *v);
  //mais funções que sejam necessárias
  //...

};
```

A classe `Object3D` deve possuir métodos que permitam acrescentar pontos 3D aos já existentes, forçando a recalcular a *bounding box* e a *bounding sphere*.

Bounding Box e *Bounding Sphere*

1. A *bounding box* de um conjunto de pontos é o paralelipipedo mínimo que os contém a todos. Bastam dois pontos para a definir, pois as diferenças de valores nas coordenadas x, y e z estabelecem o valor dos lados da bounding box;

2. A *bounding sphere* é a esfera mínima que contém todos os pontos. Um algoritmo para a calcular consiste em determinar o ponto médio das coordenadas de todos os vectores 3D, fazendo a média das coordenadas em x, y e z, seguindo-se o cálculo da distância vectorial (euclidiana) até ao ponto mais afastado. Esse valor representa o raio da esfera com centro no ponto médio.

Plano

Pretende-se que construa uma classe Plano que permita representar um plano no espaço. Um plano é definido pela fórmula $Ax + By + Cz + D = 0$ e estabelece valores de intersecção com os eixos x, y e z.

A classe que modela um plano é relativamente simples e além dos habituais métodos construtores, selectores e modificadores (se tal se justificar) deve conter um método:

`float Plano::distanceToPlan(const Vect3D& point);`

que permita saber qual a distância vectorial de um ponto 3D ao plano.

View Frustum

A classe que modela um *View Frustum* deve permitir agregar 6 planos (near, far, top, bottom, left e right) e além dos métodos necessários deve ter um método que permita determinar se uma bounding sphere está completamente dentro do *View Frustum*.

Para tal sugere-se que implemente a função:

`bool ViewFrustum::insideFrustum(const Object3D& conj);`

que aceita como parâmetro um objecto `Object3D` que contém a informação necessária sobre a bounding sphere: o centro e o raio. Depois é apenas necessário calcular a distância do centro a cada um dos seis planos e verificar se a distância é sempre igual ou superior ao raio da esfera. Se tal não acontecer a esfera não está dentro do view frustum.

Documentação sobre o view frustum é fornecida em anexo e nela encontra como calcular os planos. Como medida de simplificação considere os seguintes valores para os planos:

- Sendo $e = 1$, $f = 10$ e $a = 1$, temos:

near: $A = 0$, $B = 0$, $C = -1$,$D = -1$

far: $A = 0$, $B = 0$, $C = 1$,$D = 10$

left: $A = 1/sqrt(2)$, $B = 0$, $C = -1/sqrt(2)$,$D = 0$

right: $A = -1/sqrt(2)$, $B = 0$, $C = -1/sqrt(2)$,$D = 0$

bottom: $A = 0$, $B = 1/sqrt(2)$, $C = -1/sqrt(2)$,$D = 0$

top: $A = 0$, $B = -1/sqrt(2)$, $C = -1/sqrt(2)$,$D = 0$

Requisitos comuns a todas as classes

Por forma a conseguir mostrar o comportamento do seu programa, sugere-se que as as classes que desenvolva devem ter uma função que permita inspecionar o estado das classes e obter uma string. Depois apenas será necessário mandar imprimir essa string.

Entrega do trabalho

Por forma a entregar o trabalho deve criar um programa (um ficheiro.cpp) que contenha um método `main()` no qual declara pontos 3D, cria conjuntos de pontos, planos e frustum. Na entrega podemos alterar o programa para acrescentar pontos, redefinir planos por forma a vermos o comportamento que exibe.

Não se preocupe como é que são lidos os valores para o programa. Se conseguir ler valores de ficheiros pode adoptar essa via, senão pode criar no programa principal objectos "com valores definidos à mão".

Avaliação

O objectivo da disciplina de Tópicos de Programação e Interacção é conseguir que os alunos do mestrado relembrem, e consolidem, conhecimento sobre linguagens de programação. Tendo em consideração que existem níveis diferentes de conhecimento, a avaliação será feita em função do que apresentarem, sendo que estão definidos 3 patamares de classificação em função das entregas abrangerem:

1. classes Vect3D e Object3D a funcionarem correctamente (com cálculo de bounding box e bounding sphere);
2. classe que modela um plano;
3. classe que implementa um view frustum e permite determinar se uma bounding sphere está nele contida.

Anexos

Encontra em anexo, no website da disciplina a classe `Vect3D` quase completamente implementada e um esqueleto da classe `Object3D`. Encontra também documentação sobre o que é um *View Frustum*.