Einführung in die C++-Programmierung

Progammierkurs an der Uni Konstanz, SS2014

Übungsblatt 2

Alle Aufgaben sind mit den bisher gelernten Konstrukten lösbar. Natürlich dürft ihr euch zusammensetzen, und eure Lösungsansätze diskutieren. Trotzdem solltet ihr immer noch das Programm am Ende selber schreiben und einzeln abgeben. Bitte sendet mir die Quelltexte in einzelnen Dateien spätestens bis **16.5.2014** *jeweils angegebenenen Dateinamen per E-Mail zu. Bitte vermeidet Leer- und Sonderzeichen in den Dateinamen, mit Ausnahme von Unterstrichen.*

Übungen

(Stichworte: If-Branches, Zahlensysteme);

Lösung Kugelaufgabe

Kugelgleichung:

$$\left| \begin{pmatrix} x_x \\ x_y \\ x_z \end{pmatrix} - \begin{pmatrix} m_x \\ m_y \\ m_z \end{pmatrix} \right|^2 = r^2$$

Geradengleichung:

$$\begin{pmatrix} x_x \\ x_y \\ x_z \end{pmatrix} = \mu \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} + \begin{pmatrix} o_x \\ o_y \\ o_z \end{pmatrix}$$

Einsetzen:

$$\left| \mu \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} + \begin{pmatrix} o_x \\ o_y \\ o_z \end{pmatrix} - \begin{pmatrix} m_x \\ m_y \\ m_z \end{pmatrix} \right|^2 = r^2$$

Substitution: $\vec{p} := \vec{o} - \vec{m}$, in Programm dann "Vorausberechnung":

```
double p_x = o_x - m_x;
double p_y = o_y - m_y;
double p_z = o_z - m_z;
```

Dann den Betrag von

$$\left| \mu \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \right|^2 = r^2$$

symbolisch berechnen, dies ergibt quadratische Gleichung mit einer Unbekannten (μ):

$$\left(\sqrt{(\mu d_x + p_x)^2 + (\mu d_y + p_y)^2 + (\mu d_z + p_z)^2} \right)^2 = r^2$$

$$\Leftrightarrow (\mu d_x + p_x)^2 + (\mu d_y + p_y)^2 + (\mu d_z + p_z)^2 = r^2$$

$$\Leftrightarrow \mu^2 d_x^2 + 2\mu d_x p_x + p_x^2 + \mu^2 d_y^2 + 2\mu d_y p_y + p_y^2 + \mu^2 d_z^2 + 2\mu d_z p_z + p_z^2 = r^2$$

$$\Leftrightarrow \mu^2 \cdot (d_x^2 + d_y^2 + d_z^2) + \mu \cdot \big(2(d_x p_x + d_y p_y + d_z p_z)\big) + p_x^2 + p_y^2 + p_z^2 - r^2 = 0$$

Substitutionen (Werden im Programm wieder "Vorberechnungen"):

$$a := d_x^2 + d_y^2 + d_z^2$$

$$b \coloneqq 2(d_x p_x + d_y p_y + d_z p_z)$$
$$c \coloneqq p_x^2 + p_y^2 + p_z^2 - r^2$$

Berechnen der Diskrimanten D der Mitternachtsformel:

$$D = b^2 - 4ac$$

- a

Einsetzen von μ, bzw. μ_1 und μ_2 in die Geradengleichung liefert dann die Koordinate der Schnittpunkte.
Trotz der langen Herleitung ist der Quelltext der tatsächlichen Berechnung recht kurz:

```
#include <iostream>
#include <cmath>      // Enable the calculation of square roots
using namespace std;

int main(){
  double r(10.0);
  double m_x(5.0), m_y(6.0), m_z(7.0);
  double d_x(8.0), d_y(9.0), d_z(10.0);
  double o_x(13.0), o_y(12.0), o_z(11.0);

  // First substitution
  double p_x = o_x - m_x;
  double p_y = o_y - m_y;
  double p_z = o_z - m_z;

  // Second substitution
  double a = d_x * d_x + d_y * d_y + d_z * d_z;
  double b = 2 * ( d_x * p_x + d_y * p_y + d_z * p_z);
  double c = p_x * p_x + p_y * p_y + p_z * p_z - r * r;

  // Calculate Discriminant
  double D = b * b - 4 * a * c;

  // Calculate intersections (if they exist)
  if(D < 0.0){
     cout << "There are no points of intersection." << endl;
  }
  else if(D == 0.0){
    cout << "There is one point of intersection." << endl;
    double mu = b / (2 * a);
    double isec_x = mu * d_x + o_x;
```

```
    double isec_y = mu * d_y + o_y;
    double isec_z = mu * d_z + o_z;
    cout << "Its coordinates are (" << isec_x << ", "
         << isec_y << ", " << isec_z << ")"<< endl;
  }
  else {
    cout << "There are two point of intersection" << endl;
    double sqD = sqrt(D);
    double mu1 = (b - sqD) / (2 * a);
    double isec1_x = mu1 * d_x + o_x;
    double isec1_y = mu1 * d_y + o_y;
    double isec1_z = mu1 * d_z + o_z;
    double mu2 = (b + sqD) / (2 * a);
    double isec2_x = mu2 * d_x + o_x;
    double isec2_y = mu2 * d_y + o_y;
    double isec2_z = mu2 * d_z + o_z;
    cout << "Their coordinates are (" << isec1_x << ", "
         << isec1_y << ", " << isec1_z << ") and ("
         << isec2_x << ", " << isec2_y << ", "
         << isec2_z << ")" << endl;
}
```

Anmerkung: Die Aufgabe kann mit diesem Ansatz auf einem Rechner nicht "mathematisch exakt" gelöst werden, da `double`, wie schon mehrfach erwähnt, nur eine begrenzte Anzahl von Zahlen darstellen kann, und somit D nur näherungsweise berechnet werden kann, d.h. das tatsächliche Ergebnis wird auch, wenn eigentlich $D = 0$ herauskommen sollte, leicht um 0.000 herum streuen.
Eine praktibable Möglichkeit ist in solchen Fällen, ein "Toleranzintervall" zu festzusetzen, in dem sich D befinden darf, damit das Programm den Programmpfad (2) beschreitet, auch wenn die Schnittpunkte sehr dicht beieinanderliegen:

```
  // Calculate intersections (if they exist)
  double tolerance = 1e-5;
  if(D < (-tolerance))
  {
    //...
  } else if ( ((-toleranc) <= D) && (D <= tolerance))
  {
    //...
```

```
}
else // if(D > tolerance)
{
  //...
}
```

Auch von Hand ist diese Problem natürlich nur exakt berechenbar, wenn alle Parameter aus $\mathbb{Q}$ stammen. Für diesen Fall ist es möglich, mit zwei `int` eine rationale Zahl darzustellen und damit dann "exakt" zu rechnen. Die dazugehörigen Rechenregeln für die Grundrechenarten müssen dann als eigene Funktionen implementiert werden (idealerweise, indem die benötigten Rechenoperatoren überladen werden). Fertige Implementierungen dafür gibt es zum Beispiel in der Boost-Library.