Übungsaufgabe mit Abgabe bis 22. Dez 2021, 18:00 als upload in ILIAS

Hinweis zur Abgabe:

Erstellen Sie ein Dokument mit den Ergebnissen der Aufgaben. Upload der Ergbnisdarstellung als pdf Dokument und Quellcode als tgz *oder* zip Archiv.

- Kopfbereich des Dokuments: auf jeder Seite Ihren Namen und Ihre Immatrikulationsnummer angeben, um eine eindeutige Zuordnung zu ermöglichen.
- Geben Sie auf der ersten Seite des Dokuments Ihren Namen, Immatrikulationsnummer und Studiengang an.
- Aufgabe kann in Gruppen mit bis zu vier Studierenden bearbeitet werden: In diesem Fall sind die Namen, Immatrikulationsnummer und email (KIT email) aller Teilnehmer auf der ersten Seite tabellarisch aufzulisten.
- Upload der beiden Dateien bis 22.12.2021 18Uhr als upload in ILIAS.

Die Aufgaben bauen aufeinander auf. Lösen Sie diese in der vorgesehenen Reihenfolge. Generell gilt, dass Sie sich an den Übungen und deren Lösung orientieren können. Vieles ist gleich aufgebaut.

Hintergrund zur Aufgabe:

In vielen Anwendungen muss die Rotation/Drehung von Objekten berechnet werden. In den höheren Mathematikvorlesungen / technische Mechanik wurden hierzu Rotationsmatrizen und Eulerwinkel verwendet. Diese Darstellung der Rotation hat seine Schwächen, so sind manche Orientierungen nicht eindeutig bestimmt (Pole) und Sonderfälle treten auf.
Es werden daher in der Robotik, Computergraphik, Materialwissenschaften etc. alternative und eindeutige Darstellungen der Rotation mit Hilfe von Rodrigues Vektoren oder Quaternionen verwendet.
Sie werden sich in den Aufgaben einige Eigenschaften/Rechenvorschriften der Quaternionen implementieren und anwenden. Alle notwendigen Informationen zu Quaternionen sind in der Aufgabenstellung gegeben. (es gibt verschiedene Definitionen, falls Sie in der Literatur danach suchen).

Tipps:

- Erstellen einer neuen Klasse: wenn Sie eclipse verwenden ➔ Sie können eine neue Klasse direkt erstellen über das Menü FILE->NEW->CLASS
  danach ist die Eingabe des Klassennamens notwendig und Sie müssen auswählen, welche Standardmethoden Sie implementieren wollen.
- Überlegen Sie sich, welche Teilaufgaben sinnvollerweise von Funktion erledigt werden können! Wenn Sie anfangen Codeteile zu kopieren, ist dies schon mal ein guter Hinweis über eine geeignete Funktion nachzudenken.....

Aufgabe 1: Quaternion

Erstellen Sie eine Klasse mit dem Name MyQuaternion, die folgende Grundrechenarten durchführen kann. Implementieren Sie hierzu die unten aufgeführten Methoden und Operatoren. Spalten Sie die Klasse in Kopf- und Quelldatei auf. Verwenden Sie zum Übersetzen ein Makefile. Sie können ein Makefile aus der Übung übernehmen und anpassen.

Definition Quaternion:

Es handelt sich um einen Vektor mit 4 Komponenten

$$\boldsymbol{Q} = (e_1, e_2, e_3, e_4)$$

Quaternion: $\boldsymbol{Q}_A = \left(\vec{t}_A, e_4^A\right) = (e_1^A, e_2^A, e_3^A, e_4^A)$, wobei der Vektor $\vec{t}_A \in R^3$ und e_4 ein Skalar (Datentype double) ist.

Interpretation der Komponenten:

$\vec{t}_A$: Vektor

e_4: enthält Rotationswinkelinformation

TODO: Zu implementierende Methoden:

- Konstruktor: MyQuaternion()
- Kopierkonstruktor: MyQuaternion(Q)
- Konstruktor mit 4 Argumenten **MyQuaternion(**e_1, e_2, e_3, e_4)

- Addition: operator+(Q_B)
  $\boldsymbol{Q} = \boldsymbol{Q}_A + \boldsymbol{Q}_B = \left(e_1^A + e_1^B, e_2^A + e_2^B, e_3^A + e_3^B, e_4^A + e_4^B\right)$
- Subtraktion: operator-(Q_B):
  $\boldsymbol{Q} = \boldsymbol{Q}_A - \boldsymbol{Q}_B = \left(e_1^A - e_1^B, e_2^A - e_2^B, e_3^A - e_3^B, e_4^A - e_4^B\right)$
- Zugriff auf Komponenten (Setter/Getter): operator[]
- Zugriff lesend (Getter): operator[] const
- Kopieroperator: operator=(Q_rhs)
- Zuweisungsoperator mit Skalar als Argument (operator=(skalar))
- conj() const; => gibt konjugierte Quaternion $\overline{\boldsymbol{Q}}_A$ zurück
  $$\boldsymbol{Q}_A.\mathbf{conj}(\quad) = \overline{\boldsymbol{Q}}_A = \left(-\vec{t}_A, e_4^A\right) = (-e_1^A, -e_2^A, -e_3^A, +e_4^A)$$
- Berechnung des Produkts zweier Quaternionen: operator*(Q_B):
  ($\boldsymbol{Q} = \boldsymbol{Q}_A * \boldsymbol{Q}_B$)
  $$\begin{aligned} e_1 &= e_1^A e_4^B + e_4^A e_1^B - e_2^A e_3^B + e_3^A e_2^B \\ e_2 &= e_2^A e_4^B + e_4^A e_2^B - e_3^A e_1^B + e_1^A e_3^B \\ e_3 &= e_3^A e_4^B + e_4^A e_3^B - e_1^A e_2^B + e_2^A e_1^B \\ e_4 &= e_4^A e_4^B - e_1^A e_1^B - e_2^A e_2^B - e_3^A e_3^B \end{aligned}$$
  Mit $\boldsymbol{Q}_A = (e_1^A, e_2^A, e_3^A, e_4^A)$ und $\boldsymbol{Q}_B = (e_1^B, e_2^B, e_3^B e_4^B)$

- Skalierung aller Komponenten einer Quaternion mit einem Skalar: operator*(Skalar)

$$\boldsymbol{Q}_S = \boldsymbol{Q} * f$$

Wobei f ein Fließzahl (double) ist.

TODO: Verwenden Sie das Testprogramm main_aufgabe1.cpp, um die Funktionen zu testen.

Ergebnisdarstellung:

A) Quellcode der Klasse, Makefile
B) Füge Sie die Ausgabe des Testprogramms main_aufgabe1.cpp in Ihr Dokument zur Abgabe ein.

Aufgabe 2: Rotation von Ortsvektoren um den Ursprung

Information: Rotationsberechnung

Eine Rotation um die Achse $\widehat{\boldsymbol{\rho}} = (\rho_1, \rho_2, \rho_3)$ **(normiert auf 1)** um den Winkel θ lässt sich durch eine Einheitsquaternion (Norm=1) wie folgt ausdrücken:

$$\boldsymbol{Q} = (\widehat{\boldsymbol{\rho}} \sin\theta/2\,, \cos\theta/2) = \left(\rho_1 \sin\frac{\theta}{2}, \rho_2 \sin\frac{\theta}{2}, \rho_3 \sin\frac{\theta}{2}, \cos\frac{\theta}{2}\right) \qquad (2.1)$$

und es gilt Norm = $e_1^2 + e_2^2 + e_3^2 + e_4^2 = 1$. Die Inverse zu $\boldsymbol{Q}$ lautet $\overline{\boldsymbol{Q}}$ (Winkel oder Achse hat anderes Vorzeichen).

Info zur Berechnen der Rotation eines Vektors $\boldsymbol{x} \in \boldsymbol{R^3}$ in drei Dimensionen
Dazu wird eine sogenannte reine Quaternion $\boldsymbol{X} = (\boldsymbol{x}, 0)$ gebildet.
Rotation sein durch $\boldsymbol{Q}$ gegeben: die rotierte Quaternion $\boldsymbol{X}'$ ergibt sich aus der Vorschrift

$$\boldsymbol{X}' = \boldsymbol{Q}\boldsymbol{X}\overline{\boldsymbol{Q}} \qquad (2.2)$$

Den rotierten Ortsvektor $\boldsymbol{x}'$ erhält man aus den ersten drei Komponenten von $\boldsymbol{Q}'$**:**

$$\boldsymbol{X}' = (\boldsymbol{x}', \boldsymbol{e_4})$$

Kontrollieren Sie Ihr Ergebnis mit nachfolgendem Beispiel!

Beispiel: Vektor (2,0,0) soll um y-Achse um 180° gedreht werden (Gleichung (2.1))
$\boldsymbol{X} = (\boldsymbol{x}, 0) = (2.,0.,0.,0.)$; Drehung mit $\boldsymbol{Q} = (0.,1.,0.,0.)$
$\boldsymbol{X}' = \boldsymbol{Q}\boldsymbol{X}\overline{\boldsymbol{Q}} = (-2,0.,0.,0.)$ **=>** gedrehter Ortsvektor ist (-2.,0.,0.)

TODO: main_aufgabe2.cpp ; ergänzen Sie:

- Ergänzen Sie die Funktion **quaternion_rotation**, die aus einem Vektor und einem Winkel nach Gleichung (2.1) eine Einheitsquaternion zurückgibt.
- Ergänzen Sie die Funktion **rotateX**, die eine Quaternion rotiert (Gleichung (2.2)) Schnittstelle :
  `MyQuaternion rotateX(MyQuaternion const &X,MyQuaternion const &Q);`

Ergebnisdarstellung:

- ergänzter Quellcode: (Funktionen in eigene Quellcodedatei ausgelagert: MyFunctions.cpp/h)
- führen Sie die Testrechnungen mit den drei Startdateien aufgabe2_input[1-3].dat durch und fügen Sie die Ausgabe in Ihr Dokument ein.

Aufgabe 3: Zufallsverteilung von Rotationsachsen in Winkeln

Teil 3.1: Misorientierungsberechnung

Info: Betrachten Sie zwei Ortsvektoren $\boldsymbol{x}_A$ und $\boldsymbol{x}_B$, die im Ausgangszustand deckungsgleich sind. Gegenstand A (an $\boldsymbol{x}_A$) und B (an $\boldsymbol{x}_B$) werden rotiert (Siehe Aufgabe 2). Die entsprechenden Rotationen sind durch die Quaternionen $\boldsymbol{Q}_A$ und $\boldsymbol{Q}_B$ gegeben.
Um nun den rotierten Gegenstand A direkt in B zu überführen, muss A zunächst in das Ausgangssystem mit der inversen Rotation $\overline{\boldsymbol{Q}}_A$ zurückgedreht und anschließend mit $\boldsymbol{Q}_B$ rotiert werden. Die Gesamtrotation lautet somit

$$\boldsymbol{Q} = \boldsymbol{Q}_B \overline{\boldsymbol{Q}}_A \tag{3.1}$$

TODO:
Berechnen Sie den Rotationswinkel θ von $\boldsymbol{Q}$ und die Rotationsachse $\widehat{\boldsymbol{\rho}}$ aus Gleichung (2.1).
Funktionsdeklaration:
`double get_theta(const MyQuaternion &Q);`

Teil 3.2: statistische Verteilung von Misorientierungen

Info: Die mitgelieferte Funktion random_vec4d gibt ein Vektor (Quaternion) mit 4 Komponenten zurück (siehe Code), die einer zufälligen Rotation in drei Dimensionen entspricht. Die Verteilung ist gleichverteilt.

TODO: Legen Sie ein Feld mit std::vector<MyQuaternion> QL(nmax) an und setzen Sie zunächst z.B. nmax=100. Initialisieren Sie alle Elemente mit Zufallsrotationen mit Hilfe der Funktion `random_vec4d(Vec4d &Q)` .

Teil 3.2.A) Misorientierungsverteilung zwischen Zufallsrotationen:

Berechnen Sie die paarweise Misorientierung $\theta(AB)$ zwischen allen Quaternionen des obigen Feldes und geben Sie diese in die Datei mit Name ergebnis_random.dat aus.

$\boldsymbol{Q} = \boldsymbol{Q_B}\overline{\boldsymbol{Q}}_A$ => Misorientierung $\theta(AB)$

Datenformat:

- eine Zahl pro Zeile.
- die Ausgabe der Misorientierung muss in Grad erfolgen

Erstellen Sie ein normiertes Histogramm und speichern Sie dieses in hist_random.dat ab.
Datenformat: pro Zeile Mittelpunkt_ des_Kästchens Häufigkeit**_ des_ Auftretens**
Stellen Sie das Ergebnis z.B. mit gnuplot dar. (Histogramm: Wertebereich 0,...,360 Grad; 180 Kästchen/bins bzw. wählen Sie eine geeignete Breite eines Kastens; Wählen Sie nmax so, dass die Verteilung einigermaßen „glatt“ wird (nmax>1000))

Teil 3.2.B) Misorientierungsverteilung unter Berücksichtigung von Symmetrien

Berechnen Sie die paarweise Misorientierung $\theta(AB)$ zwischen allen Quaternionen des obigen Feldes *unter Berücksichtigung einer Symmetrieeigenschaft des „Gegenstands“* und geben Sie diese in die Datei mit Name ergebnis_cube.dat aus.

Info: Als Beispiel dient ein kristallines Material: Aluminium oder Kupfer. Deren Kristallstruktur ist ein kubisch flächenzentrierten Gitter und viele (genauer 24 + Inversion) Raumrichtungen sind kristallographisch äquivalent. Für kristallographisch äquivalente Raumrichtung ergibt sich eine Misorientierung von Null, da physikalisch kein Unterschied besteht. Beispiel aus Werkstoffkunde 1 (Bachelor): die [100],[010] und [001] Richtung sind äquivalent; mit <100> bezeichnet man alle kristallographisch äquivalenten Richtungen. Für kubische Symmetrie können wir die Komponenten des Vektors <u,v,w> beliebig tauschen und Vorzeichen ändern, da alle äquivalent sind.

TODO:
Zu implementierende Vorschrift: Wenden Sie folgende Vorschrift auf die Quaternionen $\boldsymbol{Q}$ an, um die Symmetrie zu berücksichtigen (Herleitung im **Info**rmationsteil angedeutet)

- Nehmen sie den Betrag der Komponenten von $\boldsymbol{Q} = (q_1, q_2, q_3, q_4)$ und ändern sie die Reihenfolge der Komponenten (nun mit e_i bezeichnet), so dass gilt $0 \leq e_1 \leq e_2 \leq e_3 \leq e_4$.
  (Beispiel: aus $(-4.,3,1,5)/(\sqrt{51})$ wird $\boldsymbol{Q_1} = (1,3,4,5)/(\sqrt{51})$)
  Betrachten Sie dann folgende drei Quaternionen

$$\boldsymbol{Q_1} = (e_1, e_2, e_3, e_4)$$
$$\boldsymbol{Q_2} = (e_1 - e_2, e_1 + e_2, e_3 - e_4, e_3 + e_4) \,/\, \sqrt{2}$$
$$\boldsymbol{Q_3} = (e_1 - e_2 - e_4 + e_3, e_2 - e_4 - e_3 + e_1, e_3 - e_4 - e_1 + e_2, e_4 + e_1 + e_2 + e_3)/2$$

- Berechnen Sie den kleinsten Rotationswinkel aus den Quaternionen $\boldsymbol{Q_1}, \boldsymbol{Q_2}, \boldsymbol{Q_3}$ und schreiben Sie diesen in die Datei ergebnis_cube.dat (Format wie in Teil 1).
  Implementieren Sie dies als Funktion:
  `double get_theta_cubic(const MyQuaternion &Q)`

Erstellen Sie ein normiertes Histogramm und speichern Sie dieses in der Datei hist_cube.dat ab. Stellen Sie das Ergebnis z.B. mit gnuplot dar.
Passen Sie den Wertebereich und Kastenbreite und nmax geeignet an.

Ergebnisdarstellung in allen Teilaufgaben
- Code
- Programmausausgabe
- Histogramme
  Kommentieren Sie kurz die Ergebnisse: Breite, Maxima,..

— viel Erfolg — good luck — bonne chance — Qapla' — merde — veel geluk —