ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

computer graphics lab

Prof. M. Gross
Remo Ziegler, Christian Vögeli, Daniel Cotting

Visual Computing
Trackball mittels Quaternionen

Hinweise zur Lösung

Im Folgenden sind einige spezielle Probleme beschrieben, die beim Lösen der Teilaufgaben auftauchen können. Sie sind als zusätzliche Hinweise für ein tieferes Verständnis gedacht.

1) Verständnis des vorgegebenen Codes

Wie man schnell einsehen kann, wird durch ein Registrieren resp. Deregistrieren eines Idle-Callbacks die Animation gestartet und gestoppt. Um zu entscheiden, ob die Animation gestoppt werden soll, wird ein *time stamp* Mechanismus eingesetzt. Time stamps werden sowohl im Motion- als auch im Mouse-Callback gespeichert und beim Loslassen des mit der Rotation assoziierten Mouse-Buttons auf Gleichheit getestet. Wichtig zu erwähen ist, dass die für das Setzen der time stamps verwendete Funktion `glutGet(GLUT_ELAPSED_TIME)` nur im Millisekunden-Bereich auflöst. Würde diese Funktion feiner auflösen, würde dieser Vergleich auf Gleichheit von time stamps, welche in verschiedenen Funktionen gesetzt werden, kaum funktionieren.

2) Implementierung

Anstelle bei der Projektion der aktuellen Mausposition sicherzustellen, dass die berechneten Koordinaten im Intervall [-1.0, 1.0] liegen, kann alternativ der Entry-Callback der GLUT Library verwendet werden, um zu detektieren, ob die Maus das Viewing-Window verlassen hat.

- Entry-Callback: `void glutEntryFunc(void (*func)(int state))`

3) Herleitung der Rotationsmatrix

Die Lösung kann mit `Maple` wie folgt aussehen:

```
> restart;
  with(linalg):
> qpq := proc(q,p)
    local c,u,v,tmp,res;
    res := vector(4,0);
    u := vector(3, [q[1],q[2],q[3]]);
    c := q[4];
    v := vector(3, [p[1],p[2],p[3]]);
    tmp := evalm(scalarmul(crossprod(u,v),2*c);
    tmp := evalm(tmp + scalarmul(u,2*dotprod(v,u,'orthogonal')));
    tmp := evalm(tmp + scalarmul(v,(c^2-dotprod(u,u,'orthogonal'))));
    res[1] := tmp[1]; res[2] := tmp[2];
    res[3] := tmp[3]; res[4] := 0;
    RETURN(res);
  end:
> #vector to be transformed:
  p := vector(4, [x,y,z,0]);
```

```
> #quaternion:
  q := vector(4);
> #do calculation:
  temp := evalm(qpq(q,p));
```

Nach dieser Berechnung liegt ein Ausdruck mit vier Komponenten vor, welcher den rotierten Punkt beschreibt. Darin kommen sowohl die Komponenten des Punktes x,y,z als auch diejenigen des Rotationsquaternions `q1,q2,q3,q4` vor. Gesucht ist nun die Matrix, welche auf den Vektor $\underline{p}$ angewandt genau die berechneten Terme in `temp` erzeugt. Um die einzelnen Einträge, zum Beispiel die erste Zeile, zu bestimmen, muss man den Einfluss der ersten Komponente aus `temp` auf die x-, die y- und die z-Komponente bestimmen. Dies geht am einfachsten, indem man in die erste Komponente von `temp` die speziellen Punkte (x, 0, 0), (0, y, 0) und (0, 0, z) einsetzt. Analog kann man für die anderen Einträge verfahren.
In Maple kann man dafür zum Beispiel die folgenden Anweisungen verwenden (gezeigt für die erste Komponente):

```
> x:='x': y:=0: z:=0: collect(temp[1],x);
> x:=0: y:='y': z:=0: collect(temp[1],y);
> x:=0: y:=0: z:='z': collect(temp[1],z);
```