3D Simulationen von Bewegung entlang geodätischer Linien auf der Erdkugel

Marvin Veltz, Lena Kirrmann
Mathematik und Simulation, MIB2
U. Hahne, SoSe 2023

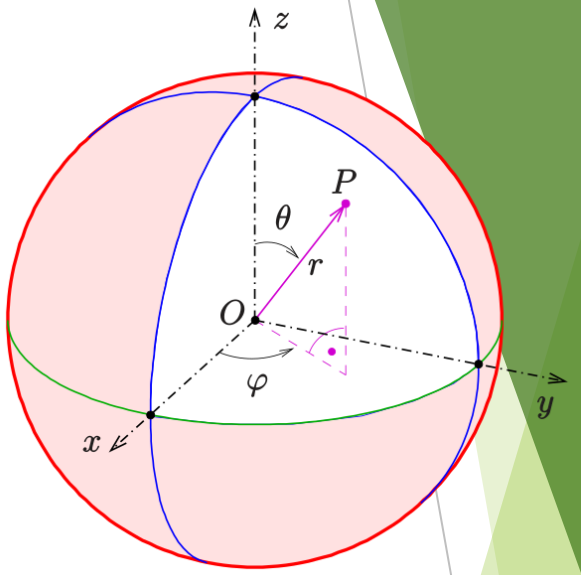
Aufgabenstellung

- Gegeben sind die Koordinaten zweier Orte P und Q auf der Erdkugel.
- Gesucht wird die kürzeste Verbindung zwischen diesen zwei Punkten auf der Erdoberfläche.
- Die Verbindung soll in einer JAVA-Applikation dargestellt werden.
- Dabei soll die Auswahl der zwei Orte als Längen- und Breitengrade ermöglicht werden.
- Die Auswahl der axonometrischen Angaben s1 und α ermöglicht.
- Die Bewegung eines Punktes vom Start- zum Zielort entlang des entsprechenden Großkreises auf dem Drahtgittermodell visualisiert werden.

Geographische Kugelkoordinaten

$$(\theta, \varphi) \rightarrow \begin{pmatrix} r & \cos(\theta) & \cos(\varphi) \\ r & \cos(\theta) & sin(\varphi) \\ r & \sin(\theta) & \end{pmatrix}$$

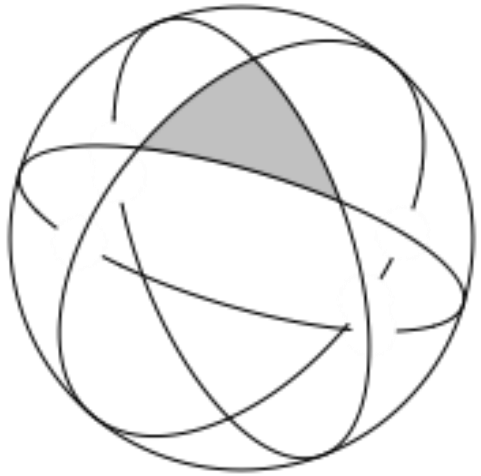
θ = Breitengrad(winkel), φ = Längengrad(winkel), r = Kugelradius



- Kugelkoordinaten werden verwendet, um Punkte auf einer Kugeloberfläche zu lokalisieren und zu beschreiben.

Großkreise

- Großkreise haben den größtmöglichen Umfang im Vergleich zu anderen Kreisen auf der Kugeloberfläche. Ihr Radius entspricht dem Radius der Kugel.
- Ein Großkreis bietet die kürzeste Verbindung zwischen zwei Punkten auf der Kugeloberfläche. Jeder Teil eines Großkreises ist gleichzeitig die kürzeste Verbindung zwischen den Punkten, die er verbindet.
- Sind P und Q “antipodale” Punkte, also wenn sie auf derselben Gerade mit dem Erdmittelpunkt sind, gibt es unendlich viele Großkreise, die P und Q verbinden.



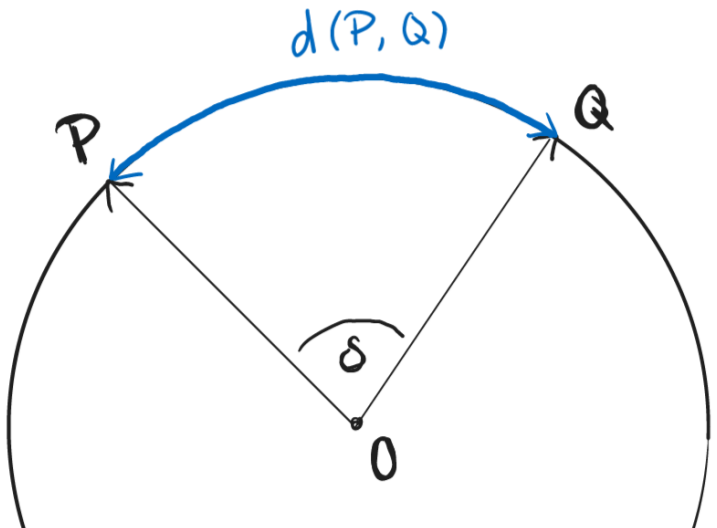
Verbindung P nach Q

- Wir bestimmen den Winkel δ zwischen den Vektoren $\vec{p} = \overrightarrow{OP}$ und $\vec{q} = \overrightarrow{OQ}$:

$$\cos(\delta) = \frac{\vec{p} \cdot \vec{q}}{\|\vec{p}\| \cdot \|\vec{q}\|}$$

- Dann bestimmen wir die Entfernung d(P , Q) der beiden Punkte mithilfe des Radius r und des Mittelpunktwinkels δ:

$$d(P, Q) = r \cdot \delta \qquad \rightarrow \boldsymbol{\delta} \textbf{ muss im Bogenmaß angegeben werden!}$$



- Wir parametrisieren eine Kurve auf dem Äquator, die beim Nullmeridian $\varphi = 0$ startet und beim Längengrad $\varphi = \delta$ endet.

$$g_0(\mathrm{t}) = \begin{pmatrix} r & \cos(t) \\ r & \sin(t) \\ 0 & \end{pmatrix}, \qquad \mathrm{t} \in [0, \delta]$$

- Der Kurve wird eine Drehung D unterworfen, damit

  $g_0(t) = (r, 0, 0)$ auf dem Punkt P liegt und

  $g_0(\delta) = (\mathrm{r}\cos(\delta), \mathrm{r}\sin(\delta), 0)$ auf dem Punkt Q liegt.

- Wir erhalten die Parametrisierung g unserer geodätischen Kurve:

  $g(\mathrm{t}) = \mathrm{D}(g_0(t)), \quad \mathrm{t} \in [0, \delta]$

Drehung D

- Einheitsverktor $\hat{x}$ = (1, 0, 0) wird auf den Einheitsvektor $\hat{p}$ abgebildet:

$$\hat{p} = \frac{\vec{p}}{\|\vec{p}\|}$$

- Einheitsvektor $\hat{z}$ = (0, 0, 1) wird auf den Einheitsvektor $\hat{n}$ abgebildet:

$$\hat{n} = \frac{\vec{p} \times \vec{q}}{\|\vec{p} \times \vec{q}\|}$$

- Einheitsvektor $\hat{y}$ = (0, 1, 0) wird auf den Einheitsvektor $\hat{u}$ abgebildet:

$$\hat{u} = \hat{z} \times \hat{x} \qquad \hat{u} = \frac{\hat{n} \times \hat{p}}{\|\hat{n} \times \hat{p}\|}$$

Die Matrix [D] hat die Form: $= [(\hat{\boldsymbol{p}})\ (\hat{\boldsymbol{u}})\ (\hat{\boldsymbol{n}})]$

Parametrisierung der Kurve

- Kombinieren wir die erhaltenen Ergebnisse, ergibt sich die Parametrisierung

$$g(\mathrm{t}) = \mathrm{D}(g_0(t)) = [D] \cdot \begin{pmatrix} r & \cos(t) \\ r & \sin(t) \\ 0 & \end{pmatrix} = [(\hat{p})\ (\hat{u})\ (\hat{n})] \cdot \begin{pmatrix} r & \cos(t) \\ r & \sin(t) \\ 0 & \end{pmatrix}$$

$$= r\ \cos(\mathrm{t})\ \hat{p}\ +\ r\ \sin(\mathrm{t})\ \hat{u}$$

Punkte auf dem Bildschirm

- Kugelkoordinaten werden in homogener Koordinatenform dargestellt. → $\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix}$
- Um einen Punkt auf dem Bildschirm zu erhalten, wird die Projektionsmatrix $M(s_1, \alpha)$ verwendet.
- Die Projektionsmatrix hat die Form: $M(s_1, \alpha) = \begin{bmatrix} -s_1 \cdot \sin(\alpha) & 1 & 0 & \frac{w}{2} \\ -s_1 \cdot \cos(\alpha) & 0 & -1 & \frac{h}{2} \end{bmatrix}$

w = Bildschirmbreite, h = Bildschirmhöhe

- Durch Multiplikation der Kugelkoordinaten mit der Projektionsmatrix M wird der Punkt auf dem Bildschirm berechnet.

Mathematische Formeln in JAVA

3D-Punkte zu Bildschirmpunkten:

$$\begin{bmatrix} -s_1 \cdot \sin(\alpha) & 1 & 0 & \frac{w}{2} \\ -s_1 \cdot \cos(\alpha) & 0 & -1 & \frac{h}{2} \end{bmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix}$$

```java
public static double[] multMatVec(double[][] m, double[] v) {
    double[] ResultMatrix = new double[m.length];
    for (int i = 0; i < m.length; i++) {
        for (int j = 0; j < m[0].length; j++) {
            ResultMatrix[i] += m[i][j] * v[j];
        }
    }
    return ResultMatrix;
}
```

```java
double[][] projectionMatrix = {
        {-s1ScaleFactor * Math.sin(alpha), 1.0, 0.0, w},
        {-s1ScaleFactor * Math.cos(alpha), 0.0, -1.0, h}
};
```

```java
double [] e1InHomogen = {e1WithScale[0],e1WithScale[1],e1WithScale[2],1};
double [] e2InHomogen = {e2WithScale[0],e2WithScale[1],e2WithScale[2],1};
double [] e3InHomogen = {e3WithScale[0],e3WithScale[1],e3WithScale[2],1};

double[] xAxisEndPoint = Maths.multMatVec(projectionMatrix, e1InHomogen);
double[] yAxisEndPoint = Maths.multMatVec(projectionMatrix, e2InHomogen);
double[] zAxisEndPoint = Maths.multMatVec(projectionMatrix, e3InHomogen);
```

Mathematische Formeln in JAVA

- Die Methode "cosineOfAngleBetweenVectors" berechnet den Kosinus des Winkels zwischen zwei Vektoren.
- Es werden die Eingabeparameter "vector1" und "vector2" erwartet, die als Arrays von Gleitkommazahlen (double) dargestellt werden.
- Die Methode verwendet eine Schleife, um über die Elemente der Vektoren zu iterieren.
- Während der Iteration werden das Skalarprodukt der beiden Vektoren sowie die Quadrate der Magnituden der Vektoren berechnet.
- Die Methode verwendet die Math.sqrt-Funktion, um die Wurzeln der Quadrate zu berechnen und die Magnituden der Vektoren zu erhalten.
- Schließlich wird der Kosinus des Winkels berechnet, indem das Skalarprodukt durch das Produkt der Magnituden dividiert wird.
- Die Methode gibt den berechneten Kosinus des Winkels zurück.

```java
public static double cosineOfAngleBetweenVectors(double[] vector1, double[] vector2) {
    double dotProduct = 0;
    double magnitudeP = 0;
    double magnitudeQ = 0;
    for (int i = 0; i <= vector1.length-1; i++) {
        dotProduct += vector1[i] * vector2[i];
        magnitudeP += vector1[i] * vector1[i];
        magnitudeQ += vector2[i] * vector2[i];
    }
    magnitudeP = Math.sqrt(magnitudeP);
    magnitudeQ = Math.sqrt(magnitudeQ);
    return dotProduct / (magnitudeP * magnitudeQ);

}
```

$$\cos(\delta) = \frac{\vec{p} \cdot \vec{q}}{\|\vec{p}\| \cdot \|\vec{q}\|}$$

Mathematische Formeln in JAVA

- Die Methode "normalizeVector" normalisiert einen gegebenen Vektor.
- Der Eingabeparameter "vector" wird erwartet und repräsentiert ein Array von Gleitkommazahlen (double).
- Die Methode verwendet eine Schleife, um über die Elemente des Vektors zu iterieren.
- Während der Iteration wird die Magnitude des Vektors berechnet, indem das Quadrat jedes Elements summiert wird.
- Die Methode verwendet die Math.sqrt-Funktion, um die Wurzel der Magnitude zu berechnen.
- Es wird ein neues Array "normalized" mit der gleichen Länge wie der ursprüngliche Vektor erstellt.
- Eine weitere Schleife wird verwendet, um über die Elemente des Vektors zu iterieren.
- Während der Iteration wird jeder Wert des Vektors durch die Magnitude geteilt und im "normalized" Array gespeichert.
- Das "normalized" Array wird zurückgegeben.

```java
public static double[] normalizeVector(double[] vector) {
    double magnitude = 0;
    for (int i = 0; i <= vector.length-1; i++) {
        magnitude += vector[i] * vector[i];
    }
    magnitude = Math.sqrt(magnitude);

    double[] normalized = new double[vector.length];
    for (int i = 0; i <= vector.length-1; i++) {
        normalized[i] = vector[i] / magnitude;
    }
    return normalized;
}
```

$$\hat{p} := \frac{\vec{p}}{\|\vec{p}\|}$$

Mathematische Formeln in JAVA

- Die Methode "crossProduct" berechnet das Kreuzprodukt von zwei gegebenen Vektoren.
- Die Eingabeparameter "vector1" und "vector2" werden erwartet und repräsentieren Arrays von Gleitkommazahlen (double).
- Ein neues Array "result" mit der gleichen Länge wie die Vektoren wird erstellt, um das Ergebnis zu speichern.
- Das Kreuzprodukt wird berechnet, indem bestimmte Berechnungen für jede Komponente des Ergebnisvektors durchgeführt werden.
- Für den x-Komponente von "result" wird die Subtraktion von Produkt der y-Komponente von vector1 und z-Komponente von vector2 und Produkt der z-Komponente von vector1 und y-Komponente von vector2 durchgeführt.
- Ähnliche Berechnungen werden für die y- und z-Komponenten des Ergebnisvektors durchgeführt.
- Das "result" Array, das das Ergebnis des Kreuzprodukts enthält, wird zurückgegeben.

```java
public static double[] crossProduct(double[] vector1, double[] vector2) {

    double[] result = new double[vector1.length];
    result[0] = vector1[1] * vector2[2] - vector1[2] * vector2[1];
    result[1] = vector1[2] * vector2[0] - vector1[0] * vector2[2];
    result[2] = vector1[0] * vector2[1] - vector1[1] * vector2[0];
    return result;
}
```

Mathematische Formeln in JAVA

- Die Methode "crossProductMagnitude" berechnet die Magnitude des Kreuzprodukts von zwei gegebenen Vektoren.
- Die Eingabeparameter "vector1" und "vector2" werden erwartet und repräsentieren Arrays von Gleitkommazahlen (double).
- Ein neues Array "result" mit der gleichen Länge wie die Vektoren wird erstellt, um das Ergebnis des Kreuzprodukts zu speichern.
- Das Kreuzprodukt wird berechnet, indem bestimmte Berechnungen für jede Komponente des Ergebnisvektors durchgeführt werden.
- Für den x-Komponente von "result" wird die Subtraktion des Produkts der y-Komponente von vector1 und z-Komponente von vector2 und des Produkts der z-Komponente von vector1 und y-Komponente von vector2 durchgeführt.
- Ähnliche Berechnungen werden für die y- und z-Komponenten des Ergebnisvektors durchgeführt.
- Die Magnitude des Kreuzprodukts wird berechnet, indem die Wurzel der Summe der Quadrate der Komponenten des Ergebnisvektors gebildet wird.
- Das berechnete Magnitude wird zurückgegeben.

```java
public static double crossProductMagnitude(double[] vector1, double[] vector2) {
    double[] result = new double[vector1.length];
    result[0] = vector1[1] * vector2[2] - vector1[2] * vector2[1];
    result[1] = vector1[2] * vector2[0] - vector1[0] * vector2[2];
    result[2] = vector1[0] * vector2[1] - vector1[1] * vector2[0];
    return Math.sqrt(result[0] * result[0] + result[1] * result[1] + result[2] * result[2]);
}
```

Mathematische Formeln in JAVA

- Die Methode "divVecWithNumber" teilt jeden Wert eines gegebenen Vektors durch eine gegebene Zahl.
- Der Eingabeparameter "vector" repräsentiert ein Array von Gleitkommazahlen (double), während "number" die Zahl ist, durch die dividiert wird.
- Ein neues Array "result" mit der gleichen Länge wie der ursprüngliche Vektor wird erstellt, um das Ergebnis zu speichern.
- Eine Schleife wird verwendet, um über die Elemente des Vektors zu iterieren.
- Während der Iteration wird jeder Wert des Vektors durch die gegebene Zahl dividiert und im "result" Array gespeichert.
- Das "result" Array, das den Vektor mit den dividierten Werten enthält, wird zurückgegeben.

```java
public static double[] divVecWithNumber(double[] vector, double number){
    double[] result = new double[vector.length];
    for(int i = 0; i <= vector.length-1; i++){
        result[i] = vector[i]/number;
    }
    return result;
}
```

Mathematische Formeln in JAVA Anwendung

$$\left[(\hat{p}) \quad (\hat{u}) \quad (\hat{n})\right] \cdot \begin{pmatrix} r\cos(t) \\ r\sin(t) \\ 0 \end{pmatrix}$$

```
tDelta += 0.01;
double[] geodeticCurve =  {
        scaleFactor * Math.cos(tDelta),
        scaleFactor * Math.sin(tDelta),
        0
};
```

$$\hat{p} := \frac{\vec{p}}{\|\vec{p}\|} \qquad \hat{u} = \frac{\hat{n} \times \hat{p}}{\|\hat{n} \times \hat{p}\|} \qquad \hat{n} = \frac{\vec{p} \times \vec{q}}{\|\vec{p} \times \vec{q}\|}$$

```
double[] unitVectorP = Maths.normalizeVector(vector0_Pos1);
double[] unitVectorN = Maths.divVecWithNumber(Maths.crossProduct(vector0_Pos1, vector0_Pos2),Maths.crossProductMagnitude(vector0_Pos1,vector0_Pos2));
double[] unitVectorU = Maths.divVecWithNumber(Maths.crossProduct(unitVectorN, unitVectorP),Maths.crossProductMagnitude(unitVectorN,unitVectorP));
```

```
double[] turnedCurveVector = Maths.multMatVec(transMatrixD,geodeticCurve);
double[] homogenTCVector = {turnedCurveVector[0],turnedCurveVector[1],turnedCurveVector[2],1};
double [] twoDHTCVector = Maths.multMatVec(projectionMatrix,homogenTCVector);
g2d.setColor(Color.BLUE);
g2d.fillOval((int) twoDHTCVector[0] , (int) twoDHTCVector[1] ,  width: 10,  height: 10);
```

```
double[][] transformMatrixD = {
        {unitVectorP[0],unitVectorU[0],unitVectorN[0]},
        {unitVectorP[1],unitVectorU[1],unitVectorN[1]},
        {unitVectorP[2],unitVectorU[2],unitVectorN[2]}
};
```