

THEMA 3: 3D SIMULATION VON BEWEGUNGEN ENTLANG GEODÄTISCHER LINIEN

AUFGABENSTELLUNG

- Das Erstellen einer Java-Applikation, die
 - die Auswahl von Start- oder Zielorten ermöglicht,
 - die Auswahl der axonometrischen Angaben s_1 und α ermöglicht,
 - ein Drahtgittermodell der Erdkugel erzeugt,
 - und den Flug visualisiert und markiert

DREIBEIN

- Mitte des Fensters ermittelt -> Als Vektor gemerkt
- Projektion Matrix in homogenen Koordinaten erstellt
- Die Endpunkte von jeder Achse bestimmt
- Linie von Mittelpunkt zu jedem Endpunkt gezogen mithilfe von Matrix

DREIBEIN CODE

```
double scaleFactor = 200.0;
double s1 = scaleFactor * (s1ScaleFactor / Math.sqrt(2.0));
double s2 = scaleFactor * 1.0;
double s3 = scaleFactor * 1.0;

double[][] projectionMatrix = {
    {-s1 * Math.sin(alpha), s2, 0, originX},
    {-s1 * Math.cos(alpha), 0, -s3, originY}
};

double[] e1H = {1.0, 0.0, 0.0, 1.0};
double[] e2H = {0.0, 1.0, 0.0, 1.0};
double[] e3H = {0.0, 0.0, 1.0, 1.0};
double[] originH = {0.0, 0.0, 0.0, 1.0};

//method multiplication of matrix-vector -> now we can project the Axes corre
double[] xAxisEndPoint = multMatVec(projectionMatrix, e1H);
double[] yAxisEndPoint = multMatVec(projectionMatrix, e2H);
double[] zAxisEndPoint = multMatVec(projectionMatrix, e3H);
```

```
//Draw Background
g.setColor(Color.BLACK);
g.fillRect(x: 0, y: 0, this.getWidth(), this.getHeight());

//set Line Width
g2d.setStroke(new BasicStroke(width: 3.0f));
//draw E'2
g.setColor(Color.GREEN);
g.drawLine(originX, originY, (int) yAxisEndPoint[0], (int) yAxisEndPoint[1]);
//draw E'3
g.setColor(Color.BLUE);
g.drawLine(originX, originY, (int) zAxisEndPoint[0], (int) zAxisEndPoint[1]);
//draw E'1
g.setColor(Color.RED);
g.drawLine(originX, originY, (int) xAxisEndPoint[0], (int) xAxisEndPoint[1]);
```

KUGELFÖRMIGES DRAHTGITTERMODELL

- Definieren Theta und Phi für Longitudinale und Latitudinale Linien.
- Äußerer Schleife für den Feinheitsgrad der Linien
- Innere Schleife für die Anzahl der Linien
- Mit jedem Durchlaufen der Schleife wird ein Punkt des Gitters ausgerechnet und gezeichnet

DRAHTGITTERMODELL CODE

```
// Draw the longitudinal lines
g2d.setStroke(new BasicStroke( width: 1.0f));
g.setColor(Color.lightGray);
for (double i = 0; i <= 1; i += 0.001) {
    //i =: amount of dots per line
    double theta = i * 2 * Math.PI;
    for (double j = 0; j <= 1; j += 0.1) {
        //j=: amount of lines
        double phi = j * Math.PI;

        double[] sphere = {
            radius * Math.cos(theta) * Math.cos(phi),
            radius * Math.cos(theta) * Math.sin(phi),
            radius * Math.sin(theta),
            1
        };

        double[] longitude = multMatVec(projectionMatrix, sphere);
        g2d.drawOval((int) longitude[0] , (int) longitude[1] , width: 1, height: 1);
    }
}
```

```
// Draw the latitudinal lines
for (double i = 0; i <= 1; i += 0.01) {
    //i =: amount of dots per line
    double phi = i * 2 * Math.PI;
    for (double j = 0; j <= 1; j += 0.1) {
        //j=: amount of lines

        double theta = j * 2 * Math.PI;

        double[] sphere = {radius * Math.cos(theta) * Math.cos(phi),
            radius * Math.cos(theta) * Math.sin(phi),
            radius * Math.sin(theta),
            1
        };

        double[] latitude = multMatVec(projectionMatrix, sphere);
        g2d.drawOval((int) latitude[0] , (int) latitude[1] , width: 1, height: 1);
    }
}
```

BESTIMMUNG VON PUNKTEN AUF DEM GITTER

- Längen- und Breitengrade werden eingegeben
- Aus diesen Werten werden die 3D Werte berechnet
- Danach die Länge des Vektors
- Anschließend werden sie in Cartesian-Koordinaten umgerechnet
- Diese werden mit der Projektionsmatrix in 2D Koordinaten transformiert
- Dieser Punkt wird dann gezeichnet

CODE

```
//Make a point on a desired position of the sphere
g2d.setStroke(new BasicStroke( width: 3.0f));
g2d.setColor(Color.GREEN);

double pos1Horizontal_angle = Pos1X; //Pos1X is defined by the user through the InputWindow
double pos1Vertical_angle = Pos1Y; //Pos1Y is defined by the user through the InputWindow

//Calculate the XYZ coords of Pos1 by transforming the Lat and Lon into 3d coords
double xPos1 =Math.cos(pos1Vertical_angle) * Math.cos(pos1Horizontal_angle);
double yPos1 =Math.cos(pos1Vertical_angle) * Math.sin(pos1Horizontal_angle);
double zPos1 = Math.sin(pos1Vertical_angle);

//calculate the distance from the center of the sphere to know where to draw the point
double rPos1= Math.sqrt(Math.pow(xPos1,2)+Math.pow(yPos1,2) + Math.pow(zPos1,2));

//transform the XYZ coords into homogeneous cartesian coords
double[] pos1Cartesian = {
    rPos1*xPos1,
    rPos1*yPos1,
    rPos1*zPos1,
    1
};

//transform the homogenous cartesian coords into 2d coords
double[] pos1 = multMatVec(projectionMatrix,pos1Cartesian);

g2d.drawOval((int) pos1[0] , (int) pos1[1] , width: 5, height: 5);

//Do the same for pos2
```


DREHMATRIX BERECHNEN

- Vektoren von Ursprung zu eingegebenen Punkt bestimmt
- Winkel zwischen beiden Vektoren mit Formel bestimmt: $\cos(\delta) = \frac{\vec{p} \cdot \vec{q}}{\|\vec{p}\| \cdot \|\vec{q}\|}$
- Einheitsvektor von P berechnen: $\hat{p} := \frac{\vec{p}}{\|\vec{p}\|}$
- Einheitsvektor N berechnen: $\hat{n} := \frac{\vec{p} \times \vec{q}}{\|\vec{p} \times \vec{q}\|}$
- Einheitsvektor U berechnen: $\hat{u} := \frac{\hat{n} \times \hat{p}}{\|\hat{n} \times \hat{p}\|}$
- Drehmatrix erstellen: $[D] = \begin{bmatrix} (\hat{p}) & (\hat{u}) & (\hat{n}) \end{bmatrix}$

CODE

```
//calculate the vectors from middle point to our desired positions
double[] vector0_Pos1 = subtractVectors(pos1Cartesian,originH);
double[] vector0_Pos2 = subtractVectors(pos2Cartesian,originH);
double delta = Math.acos(cosineOfAngleBetweenVectors(vector0_Pos1,vector0_Pos2)) ; //calculate angle between both vectors

int earthRadius = 6371; //km
distance = Math.acos(Math.sin(Pos1X)*Math.sin(Pos2X)+Math.cos(Pos1X)*Math.cos(Pos2X)*Math.cos(Pos2Y-Pos1Y))*earthRadius; //calculate real distance
AnimationWindowPanel.distanceP1P2.setText(String.valueOf((int) distance) + "Km");//overwrite distance in window

g2d.setStroke(new BasicStroke( width: 1.0f));

//calculate unit vectors
double[] unitVectorP = normalizeVector(vector0_Pos1);
double[] unitVectorN = divVecWithNumber(crossProduct(vector0_Pos1, vector0_Pos2),crossProductMagnitude(vector0_Pos1,vector0_Pos2));
double[] unitVectorU = divVecWithNumber(crossProduct(unitVectorN, unitVectorP),crossProductMagnitude(unitVectorN,unitVectorP));

//define transformation Matrix using the unit vectors
double[][] transMatrixD = {
    {unitVectorP[0],unitVectorU[0],unitVectorN[0]},
    {unitVectorP[1],unitVectorU[1],unitVectorN[1]},
    {unitVectorP[2],unitVectorU[2],unitVectorN[2]}
};
```

GEODÄTISCHE LINIE UND ANIMATION

- Geodätische Kurve von 0 bis Delta deklariert
- Wird mit Drehmatrix gedreht
- Wird in 2 Dimensionale-Koordinaten umgewandelt
- Anschließend wird auf diese Koordinate gezeichnet
- Mit jedem durchlaufen von "Paint Component" wird so der Flug animiert
- Wenn "tdelta" den Wert von Delta erreicht hat, wird die Animation gestoppt und die Kurve gezeichnet

CODE

```
//animate the flight path by using the fact Paint Component is getting called constantly
tDelta += 0.01;
double[] geodeticCurve = {
    radius * Math.cos(tDelta),
    radius * Math.sin(tDelta),
    0
};

//turn the geodetic curve to be positioned as desired
double[] turnedCurveVector = multMatVec(transMatrixD, geodeticCurve);
double[] homogenTCVector = {turnedCurveVector[0], turnedCurveVector[1], turnedCurveVector[2], 1};
double [] twoDHTCVector = multMatVec(projectionMatrix, homogenTCVector); //turn the homogeneous vector into a 2d vector
g2d.setColor(Color.magenta);
g2d.fillOval((int) twoDHTCVector[0] , (int) twoDHTCVector[1] , width: 10, height: 10); //draw an oval on the desired position
if(tDelta >= delta) {
    //after tDelta reaches the value of delta, draw the line and stop tDelta from moving any further
    tDelta = delta;
    for(double i = 0; i <= delta; i += 0.1) {
        geodeticCurve = new double[]{
            radius * Math.cos(i),
            radius * Math.sin(i),
            0
        };
        turnedCurveVector = multMatVec(transMatrixD, geodeticCurve);
        homogenTCVector = new double[]{turnedCurveVector[0], turnedCurveVector[1], turnedCurveVector[2], 1};
        twoDHTCVector = multMatVec(projectionMatrix, homogenTCVector);
        g2d.setColor(Color.magenta);
        g2d.fillOval((int) twoDHTCVector[0], (int) twoDHTCVector[1], width: 10, height: 10);
    }
}
```




Vielen Dank für Ihre Aufmerksamkeit!