Strukturen im n-dimensionalen Raum

Wolfhard Hövel

Abstract

Im n-dimensionalen euklidischen Raum werden Vektoren über das Skalarprodukt derart miteinander verknüpft, dass sich für jede Dimension n > 1 spontan Strukturen ausbilden können. Paarweise zusammengefasste Ortsvektoren werden iterativ durch Einheitsvektoren antiparallel gegeneinander verschoben. Dadurch bleibt der Schwerpunkt der Struktur im Raum fixiert.
Das System ist durch maximal vorgegebene Beträge der Abstandsvektoren beschränkt. An der Begrenzung erfolgt eine Drehung der Einheitsvektoren, sodass das Skalarprodukt gebildet aus Abstands- und Verschiebungsvektor positiv wird und der Abstand der Parallelen, auf denen die Verschiebungsvektoren liegen, einen vorgegebenen Betrag annimmt. Damit wird das Drehmoment der antiparallelen Verschiebungsvektoren konstant gehalten.
Die Attraktoren zeigen zum Teil ein kompliziertes dynamisches Verhalten. Zunächst erzeugt der Algorithmus eine chaotische Punktwolke. Durch Selbstorganisation formen sich jedoch oft nach einer gewissen Zeit spontan Attraktoren. Die Muster können nach einer bestimmten Lebensdauer in andere stabilere Formen umspringen.
Das ausführbare Java-Programm „Attractor.jar“ veranschaulicht die Ergebnisse und kann unter
http://www.ohm-hochschule.de/bib/ohmdok/hoevel/Attractor.jar

heruntergeladen werden.

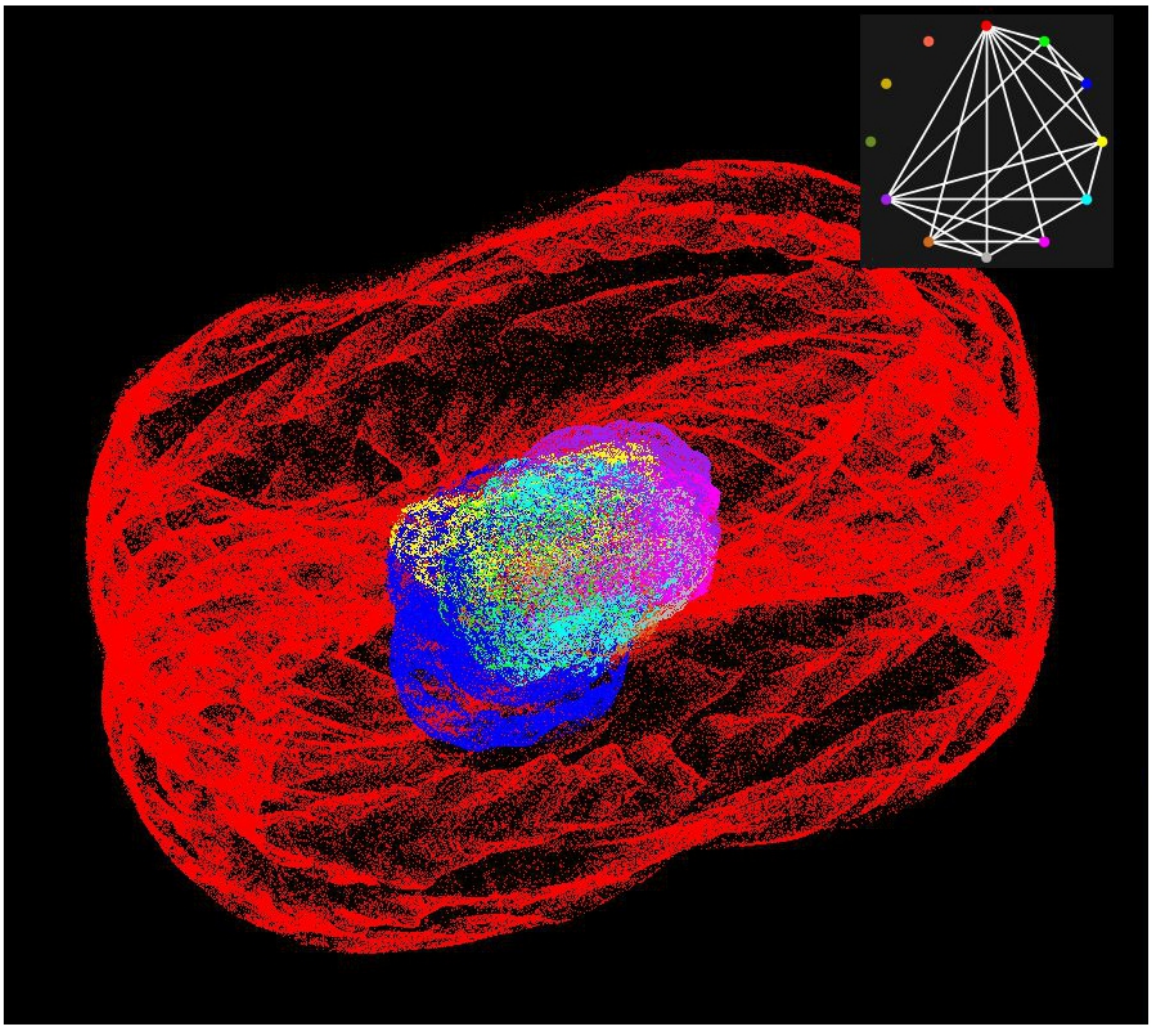

Bild 1. Beispiel für einen Attraktor mit 9 Ortsvektoren, Dimension n = 6.

Algorithmus

Als Grundstruktur betrachte man zwei Ortsvektoren r1 und r2. Diese werden durch den Abstandsvektor r12 = r2 – r1 miteinander verknüpft. Ein zunächst beliebiger Einheitsvektor dr12 wird zu r1 addiert und von r2 subtrahiert. Durch Iteration dieses Vorgangs bewegen sich die beiden Ortsvektoren antiparallel im n-dimensionalen Raum. Damit bleibt der Schwerpunkt der Anordnung erhalten und liegt in der Mitte des Abstandsvektors.

Sobald eine vorgegebene Schranke e überschritten wird, wird dr12 gedreht, sodass für das Skalarprodukt r12*dr12 > 0 gilt und der Abstand der Parallelen, auf denen dr12 und -dr12 liegen, den konstanten Wert s annimmt. Dadurch bleibt das Drehmoment der beiden antiparallelen Verschiebungsvektoren konstant.

Weitere Ortsvektoren lassen sich analog durch beliebige Paarbildung hinzufügen. Es ist jeder mögliche Graph zulässig, dabei entsprechen die Ortsvektoren den Knoten und die Abstandsvektoren den Kanten des Graphs. Zuletzt werden die einzelnen Verschiebungsvektoren für jeden Ortsvektor entsprechend überlagert. Damit sind alle Ortsvektoren voneinander abhängig.

Die Ortsvektoren bewegen sich nun durch den n-dimensionalen Raum entweder chaotisch oder in Form von vielfältigen Attraktoren. Diese Attraktoren können quasi stabil sein oder nur eine begrenzte Lebensdauer aufweisen. Unter der oben angegebenen Internetadresse kann das Java-Programm „Attractor.jar“ heruntergeladen werden, mit dem weitere spezielle Untersuchungen der Attraktoren ausführbar sind.

Mit der beschriebenen Rechenmethode kann praktisch eine beliebige Anzahl von Attraktoren visualisiert werden. Durch geeignete Wahl der Anfangsbedingungen, der Parameter s und e, der Dimension und der Struktur des Graphs lässt sich der Ablauf der Iteration vielfältig beeinflussen.

Im Anhang sind einige Java-Quellprogramme aufgeführt. In der Klasse „Vector“ werden die benötigten Vektorrechenoperationen zusammengefasst. Die Methode „reflect“ dreht den Verschiebungsvektor in der beschriebenen Art und Weise. Es wird bei jeder Iteration zunächst die Grenzbedingung geprüft und bei Erfüllung eine Drehung der Verschiebungsvektoren durch „reflect“ veranlasst. Zum Schluss gewinnt man die Lageänderung der Ortsvektoren durch Überlagerung der zugehörigen Verschiebungsvektoren.

Um die Einfachheit des Rechenverfahrens zu zeigen, ist mit einem möglichst leicht verständlichen Programm „AttracSimple“ die Wirkungsweise aufgelistet worden.

Anhang

Einige einfache Programmbeispiele sollen anhand von Java-Quelltexten die Rechenmethode erläutern. In der Klasse „Vector“ sind die Vektorrechenoperationen, die benötigt werden, zusammengeführt. Die Methode „randVect“ erzeugt zufällige Vektoren im n-dimensionalen Raum, die die Anfangsbedingungen für die Orts- und Verschiebungsvektoren bestimmen. Von besonderer Bedeutung ist die Methode „reflect“. Diese dreht den Verschiebungsvektor dre in der Ebene, die durch den Einheitsvektor dre und den Abstandsvektor r aufspannt wird in der Art und Weise, dass das Skalarprodukt positiv ist und die Bedingung für s eingehalten wird.

Die Klasse „AttracDemo“ beinhaltet die eigentliche Vektorrechnung zwischen den Markierungen
//==========

//==========.
Zunächst werden die Abstandsvektoren berechnet. In diesem Beispiel wurden drei Ortsvektoren und drei Abstandsvektoren gewählt. Dann wird geprüft, ob die Summe aller Abstände größer als eine vorgegebene Schranke e ist. Falls dies zutrifft, werden die „weglaufenden“ Verschiebungsvektoren zurückgedreht. Danach werden die entsprechenden Verschiebungsvektoren mit jedem zugehörigen Ortsvektor überlagert.

Das weitere lineare Programm „AttracSimple“ soll durch ein möglichst einfaches Beispiel die Wirkungsweise des Rechenverfahrens verdeutlichen. Es sind 3 Ortsvektoren und 2 Abstandsvektoren gewählt worden. Die Vektoren wurden hier in Komponentenschreibweise dargestellt.

```
package attracdemo;

public class Vector {

   public  double[] vect;
   public  static int dim = 4; //dim > 1
   public  static int s = 1;     //s > 0
   public  static int e = 40;  //e > 0

   //Constructor
   public Vector( double[] vect) {
      this.vect = vect;
   }


   //Returns a random vector (initial conditions).
   public Vector randVect(double c){
      double[] temp = new double[dim];

      for (int i = 0; i < dim; i++) {
         temp[i] = c*(Math.random()-0.5);
      }
      return new Vector(temp);
   }

   // vector times scalar
   public Vector mult(double b){
      double[] temp = new double[dim];
      Vector a = this;
      for (int i = 0; i < dim; i++)
         temp[i] = b * a.vect[i];
      Vector c = new Vector(temp);
      return c;
   }

   //Sum of two vectors
   public Vector plus(Vector b){
      double[] temp = new double[dim];
      Vector a = this;
      for (int i = 0; i < dim; i++)
         temp[i] = a.vect[i] + b.vect[i];
      Vector c = new Vector(temp);
      return c;
   }
  //
```

```
    //Difference of two vectors
    public Vector minus(Vector b){
        double[] temp = new double[dim];
        Vector a = this;
        for (int i = 0; i < dim; i++)
            temp[i] = a.vect[i] - b.vect[i];
        Vector c = new Vector(temp);
        return c;
    }

    // Dot product of two vectors
    public double dot(Vector b){
        Vector a = this;
        double c = 0.0;
        for (int i = 0; i < dim; i++)
            c = c + a.vect[i] * b.vect[i];
        return c;
    }

    //Returns the magnitude of this vector.
    public double mag() {
        return Math.sqrt(this.dot(this));
    }

    //Returns the unit vector of this vector.
    public Vector unit() {
        double[] tmp = new double[dim];
        double scalar = this.mag();

        if (scalar > 0.0) {
            scalar = 1.0 / scalar;
        }

        for (int i = 0; i < dim; i++) {
            tmp[i] = scalar * this.vect[i];
        }

        return new Vector(tmp);
    }

    //Reflection at the boundary
    public Vector reflect(final Vector dre) {
      double r, rq, scalar, radi, x;
```

```
        rq = this.dot(this);
        if (rq < 1.0E-15)  rq = 1.0E-15;

        r = Math.sqrt(rq);
        scalar = this.dot(dre);
        radi = (rq / (s*s) - 1.0)
             * (1.0 - scalar*scalar / rq);
        if (radi < 0) radi = 0;
        x = -scalar / r + Math.sqrt(radi);

        Vector dra = dre.plus(this.mult(x / r)).unit();
        return dra;
    }
}

//-----------------------------------------------------------------------------------------------

package attracdemo;


import javax.swing.SwingUtilities;
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;

public class AttracDemo {

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
        @Override  public void run() {
                createAndShowGUI();
            }
        });
    }
```

```
    private static void createAndShowGUI() {
        JFrame f = new JFrame("AttracDemo");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(new MyPanel());
        f.pack(); //Liefert großes Window
        f.setVisible(true);
    }
}


class MyPanel extends JPanel {

    public static Vector r0 = new Vector(null);
    public static Vector r1 = new Vector(null);
    public static Vector r2 = new Vector(null);

    public static Vector r01 = new Vector(null);
    public static Vector r02 = new Vector(null);
    public static Vector r12 = new Vector(null);

    public static Vector dr01 = new Vector(null);
    public static Vector dr02 = new Vector(null);
    public static Vector dr12 = new Vector(null);

    Color color0 = new Color( 255, 0, 0 );
    Color color1 = new Color( 0, 255, 0 );
    Color color2 = new Color( 0, 0, 255 );

    public MyPanel() {
        setBackground(Color.BLACK);
    }

    @Override
    public Dimension getPreferredSize() {
        return new Dimension(800,800);
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
```

```
        //Paint specifications
        double zoom = 100.0;
        int x0 = 400;
        int y0 = 400;
        int n = 300000;  //Number of pixels

        //Initial conditions
        r0 = r0.randVect(0.1);
        r1 = r1.randVect(0.1);
        r2 = r2.randVect(0.1);

        dr01 = dr01.randVect(1.0);
        dr02 = dr02.randVect(1.0);
        dr12 = dr12.randVect(1.0);

        double eist;
//=============================================================
//Algorithm with vectors

       for( int i = 0; i < n; ++i){

          r01 = r1.minus(r0);
          r02 = r2.minus(r0);
          r12 = r2.minus(r1);

          // Boundary value
          eist = r01.mag() + r02.mag() + r12.mag();

          //Reflection?
          if (eist > Vector.e && r01.mag() > Vector.s) dr01 = r01.reflect(dr01);
          if (eist > Vector.e && r02.mag() > Vector.s) dr02 = r02.reflect(dr02);
          if (eist > Vector.e && r12.mag() > Vector.s) dr12 = r12.reflect(dr12);

          // Superposition
          // node 0  (red)
          r0 = r0.plus(dr01);
          r0 = r0.plus(dr02);
          // node 1  (green)
          r1 = r1.minus(dr01);
          r1 = r1.plus(dr12);
          // node 2  (blue)
          r2 = r2.minus(dr02);
          r2 = r2.minus(dr12);
//=============================================================
//
```

```
            //Draw Pixels
            g.setColor(color0);
            g.drawLine(x0 + (int)(zoom*(r0.vect[0])), y0 - (int)(zoom*(r0.vect[1])),
                    x0 + (int)(zoom*(r0.vect[0])), y0 - (int)(zoom*(r0.vect[1])));
            g.setColor(color1);
            g.drawLine(x0 + (int)(zoom*(r1.vect[0])), y0 - (int)(zoom*(r1.vect[1])),
                    x0 + (int)(zoom*(r1.vect[0])), y0 - (int)(zoom*(r1.vect[1])));
            g.setColor(color2);
            g.drawLine(x0 + (int)(zoom*(r2.vect[0])), y0 - (int)(zoom*(r2.vect[1])),
                    x0 + (int)(zoom*(r2.vect[0])), y0 - (int)(zoom*(r2.vect[1])));

        }
    }
}
```

```
//-------------------------------------------------------------------------------------------------
```

```
package attracsimple;


public class AttracSimple {
 public static void main(String[] args) {

        /* Example program
         * Three - node - graph  1, 2, 3
         * Two edges 1 -> 2, 1 -> 3
         * Output: position (x,y) of nodes 1,2,3 */

        //*********************************************************//
        //  This parameters can be changed:                        //
                double e = 2.0,  s = 1.0;   // s > 0               //
        //*********************************************************//

        double radi, dr, x;

        //Initial conditions (x, y, z)
        //Node1
        double x1 = 0.01,  y1 = -0.02, z1 = 0.04;
        //Node2
        double x2 = -0.04, y2 = -0.05, z2 = 0.01;
```

```
    //Node3
    double x3 = 0.05,  y3 = -0.04, z3 = -0.01;
    //Edge12
    double rq12, r12, scalar12, dx12 = 1.0, dy12 = -1.0, dz12 = 1.0;
    //Edge13
    double rq13, r13, scalar13, dx13 = -1.0, dy13 = 1.0, dz13 = -1.0;

    double eist = 1.0;

    int  N = 15000;  //steps

  //Iteration loop-------------------------------------------------------------
  for(int i = 0; i < N; i++) {

  //Edge12---------------------------------------------------------------------
        rq12 = (x2-x1)*(x2-x1)+(y2-y1)*(y2-y1)+(z2-z1)*(z2-z1);

           r12 = Math.sqrt(rq12);
           scalar12 = dx12*(x2-x1)+dy12*(y2-y1)+dz12*(z2-z1);

           if(eist > e && r12 > s) {
              radi = (rq12/(s*s)-1.0)*(1.0-(scalar12*scalar12)/rq12);
              if(radi < 0.0)  radi= 0.0;

              x = -scalar12/r12+Math.sqrt(radi);

              dx12 = dx12-x*(x1-x2)/r12;
              dy12 = dy12-x*(y1-y2)/r12;
              dz12 = dz12-x*(z1-z2)/r12;

              dr = Math.sqrt(dx12*dx12+dy12*dy12+dz12*dz12);

              dx12 = dx12/dr;
              dy12 = dy12/dr;
              dz12 = dz12/dr;
            }

  //Edge13---------------------------------------------------------------------
           rq13 = (x3-x1)*(x3-x1)+(y3-y1)*(y3-y1)+(z3-z1)*(z3-z1);

           r13 = Math.sqrt(rq13);
           scalar13 = dx13*(x3-x1)+dy13*(y3-y1)+dz13*(z3-z1);
```

```
        if(eist > e && r13 > s) {
           radi = (rq13/(s*s)-1.0)*(1.0-(scalar13*scalar13)/rq13);
           if(radi < 0.0)  radi = 0.0;

           x = -scalar13/r13+Math.sqrt(radi);

           dx13 = dx13-x*(x1-x3)/r13;
           dy13 = dy13-x*(y1-y3)/r13;
           dz13 = dz13-x*(z1-z3)/r13;

           dr = Math.sqrt(dx13*dx13+dy13*dy13+dz13*dz13);

           dx13 = dx13/dr;
           dy13 = dy13/dr;
           dz13 = dz13/dr;
         }
    //---------------------------------------------------------------------

     //Boundary value
      eist = r12+r13;

     //Superposition
      x1 = x1+dx12+dx13;
      y1 = y1+dy12+dy13;
      z1 = z1+dz12+dz13;

      x2 = x2-dx12;
      y2 = y2-dy12;
      z2 = z2-dz12;

      x3 = x3-dx13;
      y3 = y3-dy13;
      z3 = z3-dz13;
```

```
        //Position of node1, node2 and node3
          System.out.format("%10f %10f %10f %10f %10f %10f %n",
                                  x1, y1, x2, y2, x3, y3);

          /* Plot (x1, y1)
           *       (x2, y2)
           *       (x3, y3)
           */

        }
    }
}
```

Einen Ansatz, den Algorithmus hauptsächlich mit ganzen Zahlen zu formulieren, findet man unter http://www.ohm-hochschule.de/bib/ohmdok/hoevel/Graph.zip .