Softwareprojekt über Anwendung effizienter Algorithmen
Dozent: Prof. Dr. Günter Rote
Freie Universität Berlin

Wintersemester 2013/14

Abschlussbericht zum Softwareprojekt „Visualisierung der 3– und 4–dimensionalen Punktgruppen“

Marcel Ehrhardt Nadja Scharf Alexander Steen
Simon Tippenhauer Oliver Wiese Maximilian Wisniewski

9. April 2014

Inhaltsverzeichnis

1 Einleitung

1.1 Aufgabenstellung

- Die endlichen Gruppen von 3-dimensionalen orthogonalen Transformationen, die den Ursprung fest lassen, stellen die Symmetrien von 3-dimensionalen Polyedern dar. Solche Polyeder sollen modelliert und graphisch dargestellt werden. Durch Verformungen soll dabei deutlich werden, welche Ecken einander zugeordnet sind.
- Die endlichen Gruppen von 4-dimensionalen orthogonalen Transformationen, die den Ursprung fest lassen, stellen die Symmetrien von 4-dimensionalen Polyedern dar. Solche Polyeder lassen sich als dreidimensionale Raumteilungen durch sogenannte Schlegeldiagramme darstellen.
- Visualisierung als Schlegeldiagramme;
- Anzeigen des Fundamentalbereichs.

Ziel dieses Softwareprojekts ist die Visualisierung der drei– und vierdimensionalen Punktgruppen. Punktgruppen aus dem Bereich der euklidischen Geometrie sind endliche Gruppen von orthogonalen Transformationen, die mindestens einen Punkt fest lassen, und die Symmetrien eines drei– bzw. vierdimensionalen Polyeders darstellen. Diese Polyeder sollen modelliert und grafisch dargestellt werden, sodass durch Interaktion weitere Erkenntnisse darüber gewonnen werden können, welche Ecken einander zugeordnet sind.

Dazu wird anhand einer ausgewählten Punktgruppe ein frei wählbarer Punkte aus dem zugehörigen Fundamentalbereich 3.5 unter den zugehörigen Symmetrien abgebildet. Die Abbildungen werden mittels Quaternionen 3.1, die zur Darstellung der Rotationen und Spiegelungen verwendet werden, durchgeführt. Anschließend wird die konvexe Hülle der Abbildungen gebildet und visualisiert. Im dreidimensionalen kann das Polyeder der konvexen Hülle direkt grafisch dargestellt werden. Für den vierdimensionalen Fall lassen sich die dreidimensionalen Raumteilungen durch sogenannte Schlegeldiagramme darstellen.

Um dies zu realisieren, muss sowohl der Fundamentalbereich als auch das Schlegeldiagramm berechnet werden. Als Softwarehilfsmittel kommt dazu polymake [**?**] zum Einsatz, dass eine Reihe von Funktionen im Zusammenhang mit konvexen Polyedern bereitstellt. Für die abschließende grafische Darstellung wird die auf dreidimensionale und mathematische Visualisierung spezialisierte Java Bibliothek jReality [**?**] verwendet.

2 Projektorganisation

- spline Pad für Aufgabenverteilung, Statusübersicht
- Wöchentliche Meetings
- GitHub zur Versionskontrolle, Wiki, Doku
- Mailinglist zur Kommunikation abseits der wöchentlichen Treffen

Bei einer eher geringen Gruppengröße von 6 Personen, haben wir uns dafür entschieden keine aufwändige Projektstruktur anzulegen. Kern unserer Organisation war ein wöchentliches Treffen, für das wir einen Raum im Keller des Instituts reserviert hatten, sodass wir ungestört arbeiten konnten. Dabei stellte jeder seine Ergebnisse der letzten Woche vor. Dazu gehörten auch eventuelle Schwierigkeiten, die anschließend in der Gruppe diskutiert wurden. Oftmals entstanden dadruch neue Lösungsansätze.
Die Ergebnisse dieser Besprechungen und die Aufgaben für die nächste Woche wurden in einem Spline–Pad festgehalten (`http://pad.spline.de`). So konnte jederzeit der aktuelle Stand und anstehende Aufgaben nachgelesen werden. Auch Designentscheidungen lassen sich anhand dieser Protokolle nachvollziehen.
Um parallel am selben Quellcode arbeiten zu können, haben wir ein Projekt auf GitHub (`https://github.com`) erstellt. Hier wurde unser Projekt gehostet und mit Git verwaltet. Wir haben die Entwicklung in unterschiedlichen Branches durchgeführt, sodass jede neue Entwicklung zuerst in einem eigenen Branch publiziert und getestet wurde. Anschließend kamen ausreichend gestetete Neuerungen in den Master–Branch, der immer die neueste stabile Version des Projekts enthielt. Das heißt der Master–Branch enthielt immer eine lauffähige oder zumindest kompilierende Version des Projekts und war somit der Ausgangspunkt für alle Weiterentwicklungen.
Außerdem bietet GitHub die Möglichkeit ein Wiki anzulegen. Dies haben wir hauptsächlich dafür genutzt, Berechnungen zu erläutern oder den Aufbau des Systems zu dokumentieren. Auch Quellen, aus denen wir die Berechnungen abgeleitet haben, finden sich hier. Zusätzlich zu dieser Form der Dokumentation war jeder angehalten, seinen Quelltext ausreichend zu kommentieren — hauptsächlich mit Javadoc —, sodass er auch von anderen Gruppenmitgliedern verstanden und erweitert werden konnte.
Des Weiteren haben wir uns eine Mailingliste bei Spline eingerichtet (`http://lists.spline.de/`), die zur Kommunikation abseits der wöchentlichen Treffen diente. Hierüber konnten Fragen, die zwischen den wöchtendlichen Treffen aufkamen und deren Klärung für ein Weiterarbeiten wichtig war, gestellt und diskutiert werden. Außerdem erhielt dadurch jeder einen Eindruck, welche Probleme die Anderen gerade hatten und Termine für das nächste Treffen (falls abweichend) konnten abgestimmt werden.

Irgendwie ist das alles sehr nichtssagend. Aber viel mehr kann man dazu auch nicht schreiben, oder?

3 Mathematische Grundlagen

3.1 Quaternionen

Zur einheitlichen Darstellung von drei- und vierdimensionalen Rotationen und Spiegelungen werden sog. **Quaternionen** benutzt. Die Menge der Quaternionen $\mathbb{H}$ bilden einen Erweiterungskörper der komplexen Zahlen $\mathbb{C}$ und besitzen überraschend praktische Eigenschaften für den Einsatz bei geometrischen Anwendungen. Die Algebra der Quaternionen wurde 1843 von Hamilton entwickelt [**?**].

Defintion 3.1 (Quaternion) Es seien $a, b, c, d \in \mathbb{R}$ und i, j, k imaginäre Einheiten für die gilt:

$$i^2 = j^2 = k^2 = ijk = -1$$

.... Dann heißt die Zahl $p = a + bi + cj + dk$ Quaternion. Die Menge der Quaternionen wird mit $\mathbb{H}$ bezeichnet und ist äquivalent zu rellen Vektorraum $\mathbb{R}^4$. ⌟

....

Defintion 3.2 (Kojugiertes Quaternion, Norm, Reziprokwert) ... $q^* := a - bi - cj - dk$ konjugiert $\|q\| := \sqrt{qq^*}$ norm

⌟

....

Defintion 3.3 (Einheitsquaternion) Ein Quaternion q heißt Einheitsquaternion falls $\|q\| = 1$. ⌟

3.2 Quaternionen und dreidimensionale Symmetrien

Verwendung von Einheitsquaternionen für Rotationen Einheitsquaternionen praktisch.

Eulers Rotationstheorem: Jede Drehung oder Folge von Drehungen um einen festen Punkt kann für eine einfache Drehung um einen Winkel θ und eine Achse (sog. Eulerachse) durch diesen festen Punkt beschrieben werden. Dafür identizifiert man einen Drehwinkel θ und die Drehachse $\overline{u} = (u_1, u_2, u_3)$. Dann beschreibt das Quaternion $q = e^{\frac{1}{2}\theta(u_1 i + u_2 j + u_3 k)}$ diese Drehung. Die Rotationsfunktion wird dann von der Abbildung $[q] : p \mapsto qpq^{-1}$ beschrieben (entspricht der Konjugation von p durch q). Dadurch, dass bei Einheitsquaternionen gilt $q^{-1} = q^*$, wird auch Rechenaufwand gespart.

3.3 Quaternionen und vierdimensionale Symmetrien

....

3.4 Punktgruppen

...gibts hier was interessantes?

3.5 Fundamentalbereiche

Wie bereits erwähnt wurde, ist eine Punktegruppe die Gruppenwirkung einer Symmetriegruppe auf ein gegebenes Element im zu grundeliegendem Raum der Symmetrieoperationen. Für eine Gruppenwirkung können wir zunächst die Verallgemeinerung der Punktegruppe definiere.

Defintion 3.4 (Gruppenwirkung)
Eine (Links-)Wirkung einer Gruppe $(G, \star)$ auf eine Menge X is eine Funktion

$$\triangleright : G \times X \longrightarrow X$$

mit den Eigenschaften

- $(g \star h) \rhd x = g \rhd (h \rhd x)$ für alle $g, h \in G$ und $x \in X$.
- $e \rhd x = x$ für alle $x \in X$ und e neutrales Element von G.

⌟

Defintion 3.5 (Orbit)
Für eine Gruppe $(G, \star)$ mit Gruppenwirkung $\rhd$ auf X ist für ein Element $x \in X$ die *Bahn von x bezüglich G* definiert als

$$G \rhd x := \{g \rhd x \,|\, g \in G\}$$

⌟

Über die Orbite können wir eine Äquivalenzrelation beschreiben, mit $x_1 \sim x_2$ genau dann wenn $x_2 \in G \rhd x_1$ und $G \setminus X$ ist die Menge der Representanten.

Nun ist $G \setminus X$ schon annährend ein Fundamentalbereich falls G eine Symmetriegruppe ist. Da Symmetrien auf $\mathbb{R}^n$ nun aber Isometrien sind, wissen wir, dass die Länge unserer Vektoren immer erhalten bleibt. Daher interessieren uns bei der Betrachtung nur Vektoren gleicher Länge. Wir definieren daher den Fundamentalbereich wie folgt.

Defintion 3.6 (Fundamentalbereich)
Sei $G \subseteq O(n)$ eine Symmetriegruppe auf $\mathbb{R}^n$.

Dann ist ein Fundamentalbereich $G \setminus S^{n-1}$ – ein Representatensystem für die Wirkung auf die $n-1$ - Sphäre. ⌟

Für die Darstellung der Fundamentalbereiche, wählen wir nur spezielle zusammenhängende Representanten. Wir wählen die folgenden Representanten.

Defintion 3.7 (Voronoi - Fundamentalbereich)
Sei $O \subset 2^{S^{n-1}}$ die Menge der Orbits der Gruppenwirkung von G auf S^{n-1} und $x \in S^{n-1}$ ein ausgezeichneter Punkt.

Dann ist der Voronoi - Fundamentalbereich die Menge

$$VF(x) := \left\{ \operatorname*{argmin}_{y \in o} d(x, y) \,|o \in O \right\}.$$

Damit ist die Menge der $VF(y)$ mit $y \in G \rhd x$ eine Voronoidiagram, da wir jeweils die Punkte mit Minimalem Abstand genommen haben. ⌟

4 Umsetzung

4.1 Verwendete Softwarehilfsmittel

- jReality
- polymake
- maven

4.2 Softwarearchitektur

- Eventsystem (Eventstruktur)
- Pipeline (Symmetriegruppe wählen, Fundamentalbereich berechnen und anzeigen, Punkt im Fundamentalbereich auswählen, Punkt unter Symmetrien abbilden, konvexe Hülle Berechnen, im 3–dimensionalen konvexe Hülle anzeigen/im 4–dimensionalen Schlegeldiagram berechnen, anzeigen)
- GUI

4.2.1 Eventssystem

Events

Events sind konkrete Reaktionen auf bestimmte Ereignisse und enthalten entsprechende Kontextinformationen. Alle Typen von Events erben von der abstrakten Klasse Event<H extends EventHandler>. Klassen, die auf bestimmte Event-Typen reagieren wollen, implementieren das zum Event zugeordnete Interface EventHandler.

Jede Klasse, die Events abfeuern möchte, muss einen Verweis auf den EventDispatcher besitzen. Ein Singleton von diesem Dispatcher kann mittels EventDispatcher.get() eingeholt werden. Soll ein konkretes Event e gefeuert werden, wird dies via dispatcher.fireEvent(e) erledigt. Alle Klassen, die sich vor dem Feuern eines Events via dispatcher.addHandler(eventType, handler) registriert haben, werden die Events vom Typ eventType erhalten. Der Event-Typ sollte, via Konvention, durch KonkretesEvent.TYPE gegeben sein (wobei KonkretesEvent eine Implementieren von Event ist).

Eventimplementierung

Event-Typen werden durch Anlegen einer Event-Klasse und einer EventHandler-Klasse hinzugefügt.

Das EventHandler-Template ist

```
1                   import pointGroups.gui.event.EventHandler;
2
3                   public interface ConcreteHandler
4                       extends EventHandler
5                   {
6                       public void onConcreteEvent(final ConcreteEvent
                            event);
7                   }
```

Das Event-Typ-Template ist

```
1                   import pointGroups.gui.event.Event;
2
3                   public class ConcreteEvent
4                       extends Event<ConcreteHandler>
5                   {
6                       public final static Class<ConcreteHandler> TYPE =
7                           ConcreteHandler.class;
8
9                       @Override
10                      public final Class<ConcreteHandler> getType() {
11                          return TYPE;
12                      }
13
14                      @Override
15                      protected void dispatch(final ConcreteHandler
                            handler) {
16                          handler.onConcreteEvent(this);
17                      }
18                  }
```

wobei Concrete durch konkrete Namen ersetzt werden kann.

Implementierte Events

Eventuelle Auflistung aller oder nur einzelner.

4.3 Fancy name

- Symmetriegruppen (Berechnung mittels Generatoren, hart kodiert, Darstellung/Repräsentation)
- Fundamentalbereich (Berechnung, Darstellung/Repräsentation)

4.3.1 Symmetriegruppen

4.3.2 Fundamentalbereich

In Abschnitt 3.5 wurde in Definition 3.7 der Voronoi-Fundamentalbereich beschrieben. Wir wollen eine dieser Zellen nehmen und Visualisieren. Da wir einen Abschnitt der S^3 nicht leicht visualsieren können, benötigen wir zunächst eine Projektion das Vornoi-Fundamentalbereiches auf eine $\mathbb{R}^3$ Ebene.

Idee

Wir lassen zunächst die Gruppe auf ein Ausgezeichnetes Element x wirken.

Als erstes Berechnen wir die Voronoi - Zelle für x bezüglich des Orbits von x. Alle Punkte die dort drinnen liegen sind Elemente von S^3. Da die Gruppe symmet risch ist, können wir nicht zwei Elemente aus dem selben Orbit haben, da der zweite Punkt näher an einem anderen Element aus $G \triangleright x$ liegen müsste.

Das einzig wichtige an diesem Punkt ist, dass x nicht auf einer Rotationasachse oder Spiegelebene befindet, da sonst die Voronoizellen zu groß ausfallen.

Haben wir diese Zelle, nehmen wir eine beliebige Ebene, die tangential auf dem Kugelsegment ist und projezieren darauf. Die spezielle Ebene, die wir wählen ist die Ebene $h : [t-x]\cdot x = 0$ in Normalendarstellung, da wir wissen, dass x auf der Kugeloberfläche im Kreissegment liegt und x auch ein Normalenvektor ist.

Umsetzung

Eine Voronoizelle von x für eine Menge von Sites S ist definiert als

$$VC(x) = \bigcap_{s \in S \setminus \{x\}} h^+_{x,s}$$

Wobei $h^+_{x,s}$ der Halbraum ist, der rechts von der Hyperebene liegt, die zu x und y in jedem Punkt den selben Abstand hat.

Polymake hat die Möglichkeit ein Polytope, das die Voronoizelle ja ist, gerade über den Schnitt von Halbräumen zu definieren. Dies geht über

```
1              new Polytope(INEQUALITIES=>\$hyperplanes)
```

wobei $hyperplanes eine Menge von Hyperebenen ist. Falls ein affines Polytope vorliegt, wie in unserem Fall in der Ebene h, die wie im letzten Abschnitt definiert ist, können wir auch dies mit in die Erzeugung eingeben mittels

```
1              new Polytope(INEQUALITIES=>\$hyperplanes, EQUATIONS=>{h})
```

und haben so $VC(x)$ berechnet. Da die konkrete Berechnung über den Schnitt von allen Hyperebenen zu lange dauert, filtern wir die in frage kommenden Hyperebenen vor über die Nachbern des Voronoi Diagrams. Dummerweise konnten wir nicht ermitteln, wie man in Polymake sich direkt eine Voronoizelle ausgeben lassen kann und haben daher diesen Umweg gewählt.

Haben wir nun das Polytope in der Ebene, müssen wir zur Darstellung in $n-1$ Dimensionen noch eine geeignete Basis berechnen, so dass wir die Eckpunkte des Polytopes durch $n-1$ Koordinaten darstellen können. Wir wissen schon, dass x senkrecht auf der Ebene steht, also für unsere Darstellung unerheblich ist. Wir berechnen eine Orthonormalbasis für die Ebene mit x als erstem Basisvektor. Nun können wir leicht eine Basiswechselmatrix angeben, da wir ja die Bilder der neuen Basis in der alten kennen.

Darüber hinaus ist diese Matrix orthogonal – da die Spalten ja eine orthogonal Basis waren – und kann daher durch transponieren invertiert werden.

Wir haben nun also zwei Matrizen um beide Darstellungen in einander umzurechnen. In der neuen Basis können wir nun die erste Komponente weg lassen, da diese nach Projektion nur genullt wird und haben so eine Darstellung in $n-1$ Dimensionen.

Beim zurückrechnen müssen wir uns nur erinnern, dass wir uns in einer affinen Ebene mit Stützvektor x befinden.

Berechnung

Die Hyperebenen Definition in Polymake hat die Darstellung

$$h : [a_0, ..., a_n] \rightsquigarrow a_0 + a_1x_1 + \cdots + a_nx_n = 0$$

und Halbräume ebenso mit ≥ 0.

Die Halbräume für unser Polytope genügen der Gleichung

$$t \cdot (x - s) \geq 0 \Leftrightarrow 0 + (x_1 - s_1) \cdot t_1 + \cdots + (x_n - s_n)t_n \geq 0$$

wobei x der gewählte Vektor war und $s \not= x$ aus dem Orbit von x stammt.

Damit können wir alle Halbräume aus dem Orbit berechnen. Die affine Ebene genügt der Gleichung

$$(t - x) \cdot x = 0 \Leftrightarrow -\left(\sum_{i=1}^{n} x_i^2\right) + t_1x_1 + \cdots t_nx_n = 0$$

und kann auch leicht erstellt werden. Um an unser Polytope heran zu kommen, benutzen wir das folgende Polymake-Script

```
1  my \$poly = new Polytope(INEQUALITIES=>\$hyperplanes,
       EQUATIONS=>\$affine);
2  print \$poly->VERTICES;
3  print \$poly->EDGES;
4  print \$poly->FACETS;
```

Mit diesen Ergebnissen können wir nun die Basiswechselmatrizen berechnen und so die Vertices umrechnen. Das Object Fundamental ist somit eine Sammlung dieser Eigenschaften.

```
1  public interface Fundamental {
2     public double[][] getVertices();
3
4     public Edge<Integer,Integer>[] getEdges();
5
6     public double[] revertPoint(double[] point);
7
8     public boolean inFundamental(double[] point);
9  }
```

Die ersten beiden Funktionen sind selbst erklärend. Die dritte methode *revertPoint* nimmt einen Punkt in $\mathbb{R}^{n-1}$ und liftet ihn auf die Oberfläche der S^{n-1} zurück. Die letzte

Methode überprüft, ob ein angegebener Punkt überhaupt in der Voronoizelle lag. Diese Methode wird bei der anzeige beötigt um bei der Punktauswahl keine Punkte außerhalb das Polytops zuzulassen.

Das ganze ist ein interface, da wir als Fallback-case, falls die Berechnung das Fundamentalbereichs einmal fehl schlägt immer noch den Bereich S^{n-2} benutzen können. Dies ist kein Fundamentalbereich, da wir alle Orbite mehrfach treffen, allerdings werden wir außer im Falle der Identität wirklich jeden Orbit mindestens einmal treffen.

5 Fazit

- Ziele erreicht?
- Welche Schwierigkeiten?
- Welche alternativen Lösungen wurden verworfen? (MVC vs. Eventbasiert, jReality vs. JavaView, ...)
- Sonstiges?