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

1. 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.

2 Projektorganisation

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

3 Mathematische Grundlagen

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

Definition

Der Fundamentalbereich einer Punktgruppe ist ein Repräsentantensystem der Wirkung der konkreten Gruppe auf eine Punktmenge auf der Einheitskugel.

Die Symmetrien der Gruppe partitioniere so die Oberfläche einer Kugel in einzelne Flächen, von denen keine zwei Punkte auf einander abgebildet werden.

Idee

Die Partitionierung kann berechnet werden indem man die Gruppe auf einen Punkt wirken lässt und den Schnitt der Kugeloberfläche mit dem Voronoidiagram dieser Punkte zu berechnen. Dabei muss man darauf achten, dass der Punkt, den man gewählt hat auf keiner der Symmetrieachsen liegt, da man sonst die Wirkung dieser Operation vernachlässigen würde.

Umsetzung

Haben wir eine Voronoizelle zum Punkt x_1 mit den Nachbarn $x_2, \ldots, x_n$ berechnet, betrachten wir das Ergebnis nun in einer Hyperbolischen Geometrie, d.h. wir betrachten die Kugeloberfläche als gerade. Insbesondere betrachten wir die Ebene, die durch den Kegel $(0, x_2, \ldots, x_n)$ mit der Kugeloberfläche gebildet wird.

Wir nehmen immer an, dass die Punkte sich in allgemeiner Lage befinden, also wird insbesondere der Schnitt immer ein Simplex ergeben. Das bedeutet ein Dreieck (3D) oder ein Tetraeder(4D).

Von diesem wissen wir, dass eine orthogonal Transformation existiert, so dass wir das ganze in einer Dimension niedriger Darstellen können.

Aber diese berechnen wir zur Zeit noch nicht. Sondern benutzen ein Einheitssimplex und projizieren in das berechnete hinein.

Konstruktion

Haben wir nun die Punkte $x_1, x_2, \ldots, x_n$ wählen wir je $d-1$ Punkte $y_1, \ldots, y_{d-1}$ aus $[2 \ldots n]$ die paarweise verschieden sind und berechnen den Schwerpunkt von $x_1, y_1, \ldots, y_{d-1}$. Dieser Punkt liegt auf einer Kante des Kegels $(0, x_2, \ldots, x_n)$.

Haben wir alle diese Punkte können wir daraus ein Simplex erstellen.

Hinweis: Wir haben $\binom{d}{d-1}$ viele Punkte erstellt aus denen wir in Allgemeiner Lage immer ein Simplex erstellen können, das n muss $d+1$ sein, da in allgemeiner Lage die Site x_1 genau d Nachbarn in d Dimensionen haben muss.

Haben wir nun die Punkte $y_1, \ldots, y_d$ müssen wir diese jetzt in $d-1$ Dimensionen darstellen. Dies passiert zur Zeit indem das ganze auf den Nullpunkt verschoben wird $y_i^* = y_i - y_1$ wobei nun $y_2^*, \ldots, y_d^*$ ein affiner Vektorraum mit y_1 als Translation sind um auf den Ursprünglichen zu kommen. Nun wollen wir das ganze auf ein anderes Simplex abbilden bestehend aus $e_1, \ldots, e_{d-1}$ Einheitsvektoren darstellen. Da es sich bei beiden um ein Simplex handelte sind die Ecken eine Basis des $\mathbb{R}^{d-1}$. Bilden wir also die lineare Abbildung $f(e_1) = y_{i+1}^*$ so wissen wir das nach Abbildung wiederum ein Simplex heraus kommt.

Die Matrix einer solchen Abbildung ist leicht zu bestimmen:

$A_f = (y_2^*, ..., y_d^*)$ Spaltenmatrix

Die Abbildung die wir erreicht haben ist nun nicht mehr orthogonal, da sie die Länge von Vektoren verändert, aber wir findet für jeden Punkt im Einheitssimplex einen eindeutigen Punkt im Ursprünglichen Simplex.

Berechnung

Bekommen wir nun einen Punkt x aus dem Simplex $e_1, ..., e_{d-1}$ gegeben, so können wir ihn in den Fundamentalbereich Abbilden mit der Berechnung

$x_f = normalize((A_f x) + y_1)$

Wir bilden also zunächst vom Einheitssimplex in das verschobene Simplex ab, dann verschieben wir den Punkt wieder zurück mit dem affinen Vektor y_1. Zum Schluss müssen

wir den Vektor noch normalisieren, da wir den Simplex ja mit einem Hyperebenen schnitt berechnet haben, der eigentliche Fundamentalbereich aber auf der Kugeloberfläche war.

5 Fazit

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