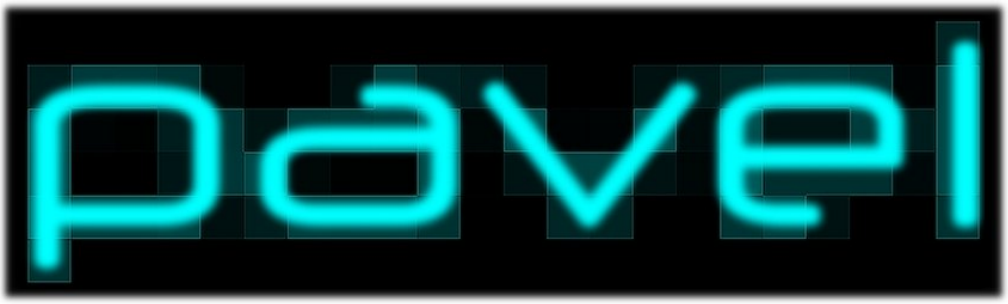
Universität Dortmund 2006/2007

Endbericht PG500

Intuitive Visualisierung und interaktive Auswertung von Paretomengen



Christoph Begau, Christoph Heuel, Raffael Joliet, Jan Kolanski, Mandy Kröller, Christian Moritz, Daniel Niggemann, Mathias Stöber, Timo Stönner, Jan Varwig, Dafan Zhai

4. Oktober 2007

Betreuer:
Prof. Dr. Heinrich Müller,
Dipl.-Inform. Petra Kersting,
Dipl.-Inform. Thomas Michelitsch

Inhaltsverzeichnis

1 Einleitung

Das Ziel dieser Projektgruppe bestand darin, ein Software-Werkzeug zur Auswertung von Simulationsergebnissen aus evolutionären Optimieralgorithmen sowie zur Verbesserung dieser Optimierungstechniken bereitzustellen. Um dieses Ziel zu erreichen, war es von großer Bedeutung, einen einfachen und intuitiven Weg im Umgang mit der großen Anzahl an evolutionär optimierten Simulationslösungen zu finden.

Die Projektgruppe setzte mit ihrer Arbeit direkt an der Entwicklung von Pareto-basierten evolutionären Algorithmen (EA) für Simulationen im Maschinenbau an, entwarf das Projekt im Kern jedoch unabhängig von dieser Anwendung. Dadurch ist es möglich, Lösungsmengen beliebiger mehrkriterieller Optimierprobleme zu analysieren, auszuwerten und mit den hierdurch gewonnenen Erkenntnissen die Optimierungstechnik selbst zu verbessern. Der Name des aus der Projektgruppe hervorgegangenen Werkzeugs ist „Pavel“ (**P**areto set **a**nalysis, **v**isualization and **e**va**l**uation). Mit Pavel wurden zwei Hauptziele erfüllt:

- **Analyse der Ergebnisse eines Simulationsmodells unter Verwendung verschiedener Parameter**
  Dem Anwender wird mit Pavel die Möglichkeit gegeben, Lösungsmengen einer Simulation sowohl im Parameter- als auch im Zielraum visualisieren zu können. Um Beziehungen zwischen Lösungen sichtbar zu machen, kann aus mehreren Visualisierungstechniken gewählt und diese zusätzlich an spezielle Bedürfnisse angepasst werden. Es ist beispielsweise möglich, neben den Parameter- und Zielräumen eigene Räume mit selbst definierten Kriterien und Gewichtungen anzulegen oder die Paretofront einer Menge direkt bestimmen zu lassen. Einzelne oder mehrere Lösungen können gesondert betrachtet werden, um ihre phänotypischen Erscheinungen darzustellen und sie unter gewählten Kriterien zu beurteilen und zu vergleichen.

- **Entwicklung und Verbesserung evolutionärer Algorithmen**
  Ein Entwickler evolutionärer Algorithmen bekommt durch die Visualisierung der Lösungsmengen direkte Rückmeldungen über die Ergebnisse

und Arbeitsweisen des Optimierprozesses. Dieses Feedback kann dazu genutzt werden, Stärken und Schwächen der Simulationssoftware oder des Optimierungsalgorithmus herauszufinden.

Umgang mit großen Datenmengen Da die Lösungsmengen im Parameter- und Zielraum sehr groß werden können, ist es unter Umständen unpraktisch, jeden einzelnen Punkt zu visualisieren. Aus diesem Grund enthält Pavel Algorithmen zur Datenreduktion sowie zum Clustering. Der Einsatz von Clusteralgorithmen ist dabei auf beliebigen Räumen möglich. Die Verfahren können damit sowohl auf Parameter als auch auf Zielfunktionswerte angewandt werden. Für die Datenreduktion stehen Techniken zur Filterung von Lösungen zur Verfügung, mit denen uninteressante Ergebnisse von der Verarbeitung ausgeschlossen werden können.

Visualisierung Die graphischen Anzeigemöglichkeiten umfassen Ansichten, welche mindestens vier Dimensionen gleichzeitig darstellen können, soweit sinnvoll auch mit Unterstützung von Stereoskopie. Dem Problem der Dimensionsreduktion von beliebig vielen auf drei oder vier Dimensionen wird durch verschiedene Ansätze Rechnung getragen. In erster Linie soll der Benutzer auswählen können, welche Kriterien für ihn wichtig sind. Es ist zusätzlich möglich, neue Kriterien zu definieren, in denen bereits existierende Kriterien mittels mathematischer Funktionen zusammengefasst oder modifiziert werden.

Um verschiedene Zusammenhänge zwischen den einzelnen Lösungen zu untersuchen, sind einzelne Visualisierungstypen unterschiedlich gut geeignet. Zum Beispiel ist die Darstellungsform der Scattermatrix darauf spezialisiert, Korrelationen zwischen Kriterien erkennbar zu machen. Genaue Funktionen und Eigenschaften der Visualisierungen werden im Abschnitt 5.3 detaillierter vorgestellt.

Anwendungsfälle und PlugIn-Struktur Für die Einbindung anwendungsfallspezifischer Visualisierungs- und Importmodule ist eine einfache PlugIn-Schnittstelle entworfen worden. Für zwei exemplarische Anwendungsfälle wurden bereits PlugIns implementiert. Es existieren daneben außerdem anwendungsfallunabhängige Import- und Darstellungsmodule. Diese generischen Varianten können für die Analyse und Verbesserung beliebiger Simulationen verwendet werden. Neben Visualisierungen für Lösungsmengen, Clusteringalgorithmen und Datenimportmodule können auch Visualisierungen für einzelne Lösungen eingebunden werden.

Diese Einzellösungsvisualisierungen dienen der Darstellung und dem Vergleich von einzelnen ausgewählten Lösungen unter Berücksichtigung der Besonderheiten des konkreten Anwendungsfalls. Ein Ingenieur kann beispielsweise eine spezielle Temperierbohrung als dreidimensionales, beliebig zoom- und rotierbares Modell visualisieren.

Pavel besitzt Bibliotheken, um die Entwicklung von PlugIns zu erleichtern. Beispielsweise können für die Darstellung von Lösungen Dreiecksnetze eingelesen und dreidimensional dargestellt werden.

Alle PlugIns wurden unabhängig von Pavels Hauptfunktionen entwickelt und können beliebig ergänzt werden.

1.1 Vorteile

Pavel kombiniert Entwicklung und Anwendung evolutionärer Algorithmen in einer Weise, dass beide Gebiete voneinander profitieren. Der Benutzer erhält neue Möglichkeiten, große Mengen von Simulationsergebnissen zu beurteilen, während der Entwickler die Qualität seiner Algorithmen beurteilen kann. Das Zusammenwirken dieser Vorteile trägt im Idealfall dazu bei, neue Anwendungsfelder evolutionärer Algorithmen schneller zu entwickeln, besser zu testen und die Qualität ihrer Ergebnisse somit zu steigern.

1.2 Der Endbericht

Während der Zwischenbericht in erster Linie die theoretischen Grundlagen des Projektes behandelte, liegt der Schwerpunkt dieses Endberichts auf der praktischen Umsetzung sowie der Bedienung und Erweiterung der Software. Da der Endbericht das Projekt in aller Vollständigkeit beschreiben soll, wird die zugrunde liegende Theorie nochmals aufgegriffen.

Zunächst wird die Entwicklung von Pavel im Rahmen der Projektgruppe vorgestellt. Dazu gehören die Ziele, eine Beschreibung von Organisation und Ablauf der Entwicklung, ein Fazit sowie ein Ausblick auf weitere Arbeiten am Projekt nach Ende der Projektgruppe. Im Anschluss daran wird eine Einführung in die Anwendungsgebiete im Maschinenbau gegeben. Zuletzt wird auf das von der PG erstellte Programm eingegangen; die Architektur wird beschrieben, Bedienungs- und Erweiterungsmöglichkeiten für Anwender und Entwickler erklärt.

2 Projektergebnis

In diesem Kapitel werden die Ziele der Projektgruppe vorgestellt, die in einem sogenannten Pflichtenheft zu Beginn des Projekts schriftlich festgehalten wurden. Des Weiteren wird die Erreichung dieser Ziele dokumentiert. Der darauf folgende Abschnitt behandelt die Organisation der Projektgruppe und deren Hilfsmittel, sowie ein Fazit, das aus diesem Projekt für die Teilnehmer gezogen werden kann. Zum Abschluss wird der Ort der Veröffentlichung bekannt gegeben.

2.1 Ziele

Die ursprünglichen im Pflichtenheft gesetzten Ziele wurden weitgehend erreicht. Einzelne anfänglich gesetzte Ziele wurden nur bedingt umgesetzt, da sie sich im Laufe der Entwicklung als nicht sinnvoll herausgestellt haben. Zusätzlich sind viele der Wunschkriterien erfüllt worden. Der Pflichtteil der Entwicklung wurde in zwei Kategorien **„Visualisierung“** und **„Auswahl und Analyse von Lösungen“** aufgeteilt, deren Ergebnisse kurz vorgestellt werden sollen.

Visualisierung:

- Intuitive Darstellung des Parameterraumes
- Intuitive Darstellung eines mindestens vier-dimensionalen Lösungsraumes
- Einsatz einer stereoskopischen Darstellung für Lösungs- und Parameterraum, vorausgesetzt, das gewählte Visualisierungsverfahren erlaubt diese Art der Darstellung
- Offene Schnittstelle für anwendungsfallspezifische Visualisierung des Parameterraumes; Möglichkeit, je nach Anwendungsfall, spezielle Lösungen bzw. Parametervektoren in den jeweiligen 3D-Modellen sowie in einer entsprechenden stereoskopischen Visualisierung, soweit diese sinnvoll ist, darstellen zu lassen

- Folgende Module zur anwendungsfallspezifischen Visualisierung sollen implementiert werden:
  - Flächenrekonstruktion
  - Fräsbahnoptimierung
  - Temperierbohrungen

Pavel bietet die Möglichkeit beliebige Räume zu visualisieren, wobei die Scatterplots auf vier Dimensionen beschränkt sind. Alle weiteren Visualisierungstechniken eignen sich auch zur Darstellung beliebig hoher Dimensionen.

Eine offene Schnittstelle zur Entwicklung zusätzlicher Darstellungstechniken wurde ebenfalls erstellt. Die technischen Details sowie die Benutzung werden im Entwicklerhandbuch (siehe Kapitel 9) detailliert vorgestellt.

Bei den anwendungsspezifischen Visualisierungen ist die Fräsbahnoptimierung sowie die Temperierbohrungen umgesetzt worden.

Stereoskopische Darstellung ist auf den Einsatz von Anaglyphen beschränkt, kann aber, soweit sinnvoll, in allen Visualisierungen genutzt werden.

Auswahl und Analyse von Lösungen:

- Navigation durch die Darstellung und Selektionsmöglichkeit einzelner Lösungen bzw. Parameter
- Anzeige detaillierter Informationen zwecks Analyse und Vergleich von Einzellösungen
- Möglichkeit, Lösungsmengen im Lösungsraum zu markieren und miteinander zu vergleichen
- Kenntlichmachung der Unterschiede beim Vergleich von Lösungen
- Clustering der Daten im Lösungs- und Parameterraum, auch manuell durch Auswahl von Teilmengen
- Manuelle Bestimmung von Repräsentanten für Cluster im Lösungsraum sowie im Parameterraum
- Filtern und Datenreduktion im Parameterraum

Für die Selektionen sind die Kriterien - wie zu Beginn der Entwicklung gefordert - vollständig umgesetzt worden. Auch die Darstellung von Lösungen und Möglichkeiten zum Vergleich sind gegeben.

Clustering auf Teilmengen und Teilräumen ist beliebig einzusetzen. Die manuelle Auswahl eines Clusterrepräsentanten ist nicht implementiert worden, da sich dieses Verfahren als nicht praktikabel herausstellte. Es ist jedoch möglich, zu einem automatisch generierten Clusterrepräsentanten dessen Ursprungspunkte darzustellen und daraus einen einzelnen Punkt auszuwählen.

Filtermöglichkeiten auf Basis von Grenzwerten einzelner Dimensionen sind beliebig erstellbar und können in jeder Visualisierung eingeschaltet werden. Dabei ist es sowohl im Parameter- als auch im Lösungsraum möglich, Filter zu setzen.

Wunschkriterien: Von den Wunschkriterien wurden folgende Punkte erfüllt:

- Darstellung von mehr als vier Dimensionen: Mit Ausnahme des Scatterplots ist dies in allen Visualisierungstechniken möglich.
- Auswahl aus mehreren Clusteringverfahren und Clusteringparametern: Implementiert sind zwei Verfahren, K-Means und HACM, die mit verschiedenen Parametern gesteuert werden können. Details hierzu sind dem Anwenderhandbuch (siehe Kapitel 8) zu entnehmen.
- Navigationshilfen im Parameterraum zum Finden von Lösungen mit gegebenen Parametern: Die Visualisierungs- und Selektionsmöglichkeiten sind nicht auf bestimmte Räume beschränkt und bieten damit auch Navigationsmöglichkeiten im Parameterraum.
- Multiple Vergleichsvarianten sollen auswählbar sein: Für den Vergleich ist es möglich, eine einzelne Lösung oder eine Kombination beliebig vieler Lösungen gleichzeitig darzustellen.
- Verschiedene Modi für Analyse einzelner Lösungen und Vergleich mehrerer Lösungen: Es können zum Beispiel Differenzbilder oder Maximalwerte zweier Lösungen überlagert angezeigt werden.

Teilweise umgesetzt sind:

- Interaktive Auswahl und Gewichtung der Fitnesswerte durch Definition von Hilfsfitnessfunktionen: Diese sind direkt im Programm erzeugbar. Allerdings ist eine interaktive Unterstützung nicht möglich, sondern diese müssen manuell vom Benutzer in Form einer mathematischen Formel definiert werden.

- Stereoskopische Darstellung auch mit Rot-Grün-Brillen: Dies stellt die einzige stereoskopische Variante dar. Andere, wie die Darstellung mit zwei Projektionen, sind derzeit nicht möglich.

Nicht umgesetzt sind:

- Darstellung des Optimierungsprozesses in Video-Form
- Unterstützung für mehrere Arten von Ein- und Ausgabegeräten
- Unterstützung verschiedener Sprachen (mindestens Deutsch und Englisch): Aus Zeitmangel wurde hierauf verzichtet. Pavel ist komplett in Englisch realisiert worden.

Bei der Performance wurde auf eine ausreichend hohe Arbeitsgeschwindigkeit auch bei großen Datenmengen geachtet. Als Referenz wurde ein Datensatz aus der Fräsbahnoptimierung herangezogen, der aus etwa 100000 Punkten mit jeweils 25 Dimensionen bestand. Für die Parallelplotdarstellung stellte sich die Füllrate der Grafikkarte als Flaschenhals heraus. Auf aktuellen Mittelklassegrafikkarten (Stand Mitte 2007) ist die Performance auch für diese Datenmengen ausreichend hoch um ein flüssiges Arbeiten zu ermöglichen. Auf leistungsschwacheren Onboardchips kommt es allerdings zu Pausen während des Renderns, die mehrere Sekunden dauern können.

2.2 Organisation

Eröffnet wurde die Projektgruppe (PG) durch eine zweitägige Seminarphase in Bommerholz, bei der sich die Teilnehmer durch je zwei Vorträge in die fachliche Thematik und die Organisation von Team- und Projektarbeiten einarbeiten konnten. Als Ergebnis dieser Phase wurde ein Zwischenbericht verfasst, der in Auszügen in dieses Dokument übernommen wurde. Im Laufe des Semesters wurden zweimal pro Woche PG-Sitzungen abgehalten, bei denen es einen Moderator und einen Protokollanten aus den eigenen Reihen gab. Zu Beginn der PG wurde die Planung des Projekts erstellt, wodurch die Sitzungen die vollen zwei Semesterwochenstunden ausfüllten. In der Implementierungsphase bestanden die Treffen aus Statusberichten zum Fortschritt des Projektes.

Als besonderes Highlight der PG wurde ein Kapitel für ein Fachbuch über evolutionäre Algorithmen geschrieben, welches vom Springer-Verlag unter dem Titel „Success in Evolutionary Computation“ veröffentlicht werden wird.

Der Abschluss der PG500 bildet dieser Endbericht und eine Abschlusspräsentation am 09.10.2007 Raum 219 GB IV.

2.2.1 Werkzeuge und Vorgehensweise

Pavel wurde mithilfe zahlreicher Entwickler-Tools implementiert. Diese hatten maßgeblichen Einfluss auf die Arbeitsweise.

Plattform Zu Beginn des Projektes galt es, die Implementierungssprache zu wählen, dabei wurden Java, C# sowie C++ in Erwägung gezogen. Das Hauptargument für Java war, dass die meisten Projektteilnehmer aus dem Software-Praktikum bereits Erfahrung mit Java hatten. Dem wurde entgegengehalten, dass eine Projektgruppe eine gute Gelegenheit sei, eine neue Sprache zu lernen. C++ erfreute sich keines großen Zuspruchs, sodass bei der Abstimmung letztendlich C# als Projektsprache gewählt wurde (6 Stimmen für C#, 2 für Java, 3 Enthaltungen). Mit der Wahl von C# standen dann weite Teile der Programmierumgebung fest. Entwickelt wurde ausschließlich auf und für die .NET-Plattform unter Windows, als IDE kam Visual Studio 2005 zum Einsatz.

Der gesamte Quellcode wurde mit Subversion[1] verwaltet. Als Kollaborations-Tool bot sich daher Trac[2] an, ein sehr populäres, in Python implementiertes Wiki, das eine Projektplanungsunterstützung, ein Ticket-System und einen Repository-Browser eng miteinander verzahnt. Die Verwendung von Subversion und Trac hat sich im Verlauf des Projektes aufgrund der sehr guten Zusammenarbeit beider Tools als enorm hilfreich für die Kommunikation zwischen den Teilnehmern erwiesen. Einfache Bedienung und die Möglichkeit, aus Sitzungsprotokollen im Wiki auf Bugs im Ticketsystem und von dort auf einzelne Zeilen von Quellcode einer bestimmten Revision verlinken zu können, führte dazu, dass das Ticketsystem auch intensiv benutzt wurde. Clientseitig wurde auf das Repository mit SmartSVN[3] zugegriffen, da Visual Studio keine vergleichbar gute Subversion-Integration bietet.

Dass wir für ein Projekt dieser Größe als Team ohne große Programmiererfahrung mit relativ wenig Bugs zu kämpfen hatten, ist neben der automatischen Speicherverwaltung von .NET/C# und dem konsequent eingesetzten Ticketsystem auch ein Resultat der routinemäßig durchgeführten Unit Tests für das Framework von Pavel. Diese wurden mittels NUnit[4] implementiert. Das Beheben in Tests entdeckter Bugs wurde durch das TestDriven.net[5] PlugIn stark vereinfacht, welches ermöglicht, Tests einfach aus Visual Studio heraus mit angehängtem Debugger laufen zu lassen.

[1] `http://subversion.tigris.org/`
[2] `http://trac.edgewall.org/`
[3] `http://www.syntevo.com/smartsvn/`
[4] `http://www.nunit.org/`
[5] `http://www.testdriven.net/`

Weitere Hilfsmittel Um die zwei- und dreidimensionalen Visualisierungen in Pavel realisieren zu können, wurde entschieden, die OpenGL Implementierung des Tao Frameworks[6] zu nutzen. Berichte wurden mit LaTeX angefertigt. Die Erzeugung von Klassenbeschreibungen aus Dokumentationskommentaren wurde mit Doxygen[7] durchgeführt.

Vorgehen Die Bearbeitung des Projektes wurde in vier Milestones aufgeteilt. Ziel des ersten Milestones war, das Framework, also die grundlegenden Daten- und Kontrollstrukturen und die Architektur der GUI fertigzustellen. Im zweiten Milestone wurden erste Visualisierungen und Clusteringalgorithmen sowie die Schnittstellen zur Erweiterung von Pavel implementiert. Der dritte Milestone brachte als neues Feature lediglich die Darstellung von einzelnen Lösungen sowie die Generierung individueller Achsen ein. Vor allem wurden Strukturen und Implementierungen, die sich zwischenzeitlich als ungeeignet oder schlecht strukturiert erwiesen hatten, aufgeräumt. Der vierte Milestone brachte weitere Features ein, die bisher nicht umgesetzt wurden: Stereoskopie, Vergleich von Einzellösungen und die Berechnung der Paretofront.

Nachdem in Vorbesprechungen die Ziele und die grobe Architektur des Programms festgehalten wurden, teilte sich die Gruppe für den ersten Milestone in GUI- und Framework-Teil um die Ziele umzusetzen. Im weiteren Verlauf des Projektes lief die Bildung von Arbeitsgruppen spontaner ab, auch wurden die Gruppen kleiner. Anstatt große Teile in großen Teams zu bearbeiten, kümmerten sich jeweils 1-3 Teilnehmer um kleinere Aspekte des Programms.

Insgesamt ist das Vorgehen bewusst sehr flexibel gehalten worden, obwohl im Rückblick ein wenig mehr Planung hilfreich gewesen wäre. Die iterative Arbeitsweise hätte ohne die Unterstützung durch ausgereifte, mächtige Werkzeuge (Subversion, Trac, .NET/C#, Unit Testing) leicht außer Kontrolle geraten können. Zwar wurden viele Entscheidungen ad hoc, mündlich und nur innerhalb kleiner Gruppe von Teilnehmern getroffen, das Festhalten und Verfolgen wichtiger Entscheidungen und Entwicklungen im Ticketsystem ermöglichte trotzdem allen, stets einen Überblick über den aktuellen Projektstatus zu erhalten und diesen in den wöchentlichen Sitzungen zu besprechen.

[6] `http://www.taoframework.com/`
[7] `http://www.stack.nl/~dimitri/doxygen/`

2.3 Veröffentlichung bei Sourceforge

Um möglichst vielen Menschen die Möglichkeit zu geben, Pavel zu nutzen, haben sich die Mitglieder der Projektgruppe entschieden, das Programm unter die GNU General Public License[8] Version 3 zu stellen und zu veröffentlichen. Dazu wurde auf dem OpenSource-Portal *SourceForge.net* ein Projekt unter der URL `http://pavel.sourceforge.net/` angelegt.

[8] `http://www.gnu.org/licenses/gpl.html`

3 Anwendungsfälle

3.1 Temperierbohrungen

Für die Qualität eines gegossenen Werkstücks spielt der Abkühlprozess der zuvor heißen Formmasse eine entscheidende Rolle. Um diesen möglichst schnell und präzise durchzuführen, werden Bohrungen in das Gießwerkzeug eingebracht, die sogenannten Temperierbohrungen. In dem folgenden Abschnitt werden zu Beginn die beiden Verfahren Druckguss und Spritzguss und die dazu benötigten Werkzeuge grundlegend erklärt. Anschließend wird anhand von Beispielen motiviert, warum es wichtig ist eine möglichst gute Auslegung von Temperierbohrungen zu beachten und auf welche Kriterien es dabei ankommt. Abschließend wird der Übergang zur Thematik der Projektgruppe angesprochen.

3.1.1 Gießverfahren

Druckguss

Druckgießen ist ein Verfahren NE[1]-Metalllegierungen wie Aluminium, Magnesium oder Zink in Dauerformen zu gießen. Dabei muss das zu verwendende Metall so erhitzt worden sein, dass es im flüssigen Zustand (Schmelze) zur Verfügung steht. Zu Beginn wird die Schmelze in die Gießkammer (siehe Abb. 3.1) eingefüllt und vom Kolben in Richtung des Formhohlraums, der auch als Kavität bezeichnet wird, gedrückt. Es muss dabei beachtet werden, dass für die Kavität die ausreichende Menge an Schmelze vorhanden ist. Dadurch, dass die Bewegung des Gießkolbens recht schnell mit mehreren m/s erfolgt, wird dieser Teil des Verfahrens meist als „Schuss“ bezeichnet.

Während der Kolben jetzt die Position am Anguss innehat und den Druck, der üblicherweise bei 2000-3000 bar[IS05] liegt, aufrechterhält, kann die Schmelze abkühlen. Ist die Masse gänzlich erstarrt und so fest, dass sie sich nicht mehr verformen kann, wird die bewegliche Hälfte des Werkzeuges von der festen getrennt (siehe Nummer 3 in Abbildung 3.1) und das Werkstück

[1] Nicht-Eisen

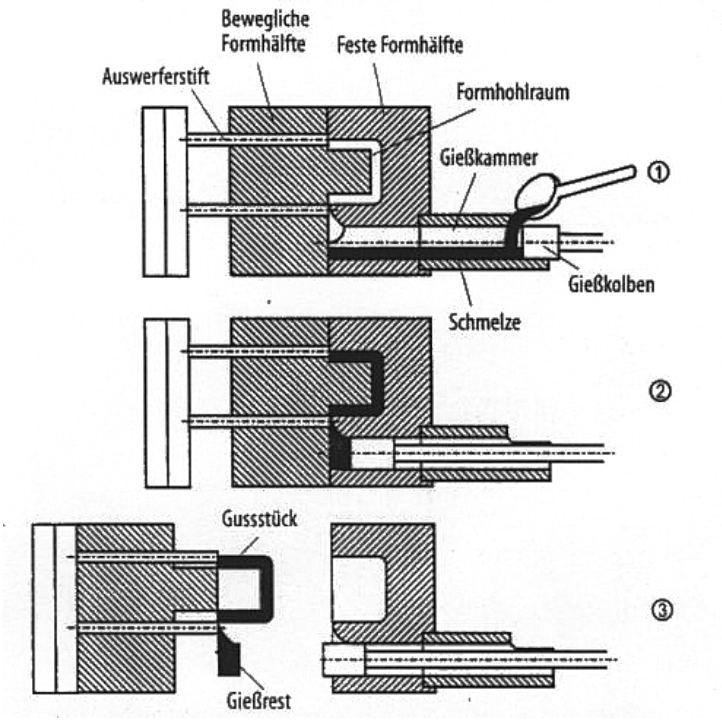


Abbildung 3.1: Ablauf des Druckguss-Verfahrens [IS05]

kann mithilfe der Schwerkraft oder z. B. Auswerferstiften aus dem Werkzeug entfernt, also ausgeworfen werden. Hierbei kann ein kleiner Gießrest am Werkstück übrig bleiben, der noch entfernt werden muss.

Vorteil dieses Verfahrens ist es, dass ausgehend vom Rohstoff direkt ohne Zwischenprodukte ein Fertigteil herstellbar ist, was es zu einem sehr schnellen Verfahren macht. Es ist möglich einfache sowie sehr komplexe Geometrien zu fertigen. Somit sind auch feinste Einzelheiten gießbar. Es entstehen Formen mit hoher Maßgenauigkeit und einer sauberen, glatten Oberfläche, sodass diese kaum nachbearbeitet werden muss. Es ist auch ein Verfahren, bei dem Teile mit sehr geringen Wandstärken herstellbar sind. Die wichtigste positive Eigenschaft ist, dass es eines der wirtschaftlichsten Gießverfahren für die Produktion bei Großserien ist.

Der größte Nachteil besteht darin, dass sich die Werkzeugherstellungskosten auf bis zu 500.000 €[IS05] belaufen können und sich erst ab einer bestimmten Menge zu fertigender Teile rentieren. Ein Grund für diese hohen Kosten ist, dass das Werkzeug hoher thermischer und mechanischer Belastung standhalten muss. Dadurch, dass zu Beginn des Füllvorgangs die Schmelze sehr heiß ist und bis zum Auswurf das Werkstück abkühlen muss, ist auch das

Werkzeug einem ständigen Temperaturwechsel ausgesetzt und sollte dabei keinen Schaden davon tragen. Die mechanische Belastung rührt daher, dass die bewegliche Formhälfte auf die feste gepresst werden muss, und zwar so stark, dass keine Schmelze in den Spalt dazwischen gelangen kann. Die sogenannte Schließkraft des Werkzeuges beträgt üblicherweise 45 [IS05]. Es müssen ebenfalls Maßnahmen zur Formentlüftung getroffen werden, d. h. die Luft in der Kavität muss vor bzw. während der Füllung entfernt werden, da sonst diese Luft in die Schmelze und somit in das fertige Werkstück eingeschlossen wird und zu einer geringeren Festigkeit des Werkstückes beiträgt. Alle diese Eigenschaften machen die Herstellung des Werkzeuges komplizierter und teurer.

Anwendung findet Druckguss bei Nicht-Eisen-Teilen bis 50 kg aus den verschiedensten Bereichen, wie z. B. dem Fahrzeugbau, der optischen und feinmechanischen Industrie, der Elektro- und Haushaltsgeräteherstellung, der Unterhaltungselektronik, der Computertechnik etc.

Spritzguss

Das *Spritzgießen* wird für thermoplastische[2] Kunststoffe eingesetzt, ähnelt aber dem Verfahren des Druckgießens. Hierbei wird der Kunststoff in Form von Granulat oder Pulver über einen Trichter in die Einspritzeinheit eingeführt. Die sich drehende Schnecke (vgl. Abbildung 3.2) befördert die Formmasse zur Schneckenspitze. Dabei erfolgt eine Erwärmung auf ca. $150 - 300\,℃$[IS05] durch die Kompression und zusätzlich durch die außen angebrachten Heizbänder, sodass eine plastifizierte Formmasse entsteht. Dies ist vergleichbar mit der Schmelze, die beim Druckguss eingesetzt wird. Die Schnecke muss bei diesem Vorgang nach und nach zurückweichen, sodass Platz für die steigende Menge an Formmasse frei wird. Nach dem Stopp der Schneckendrehung fährt die Einspritzeinheit gegen den Anguss und die Schnecke wird wie ein Kolben benutzt und verschiebt sich in axialer Richtung, sodass die Formmasse in die Kavität gedrückt wird. Es ist jedoch eine geringere Einspritzgeschwindigkeit als beim Druckguss notwendig. Das Werkstück muss nun auskühlen und kann nach dem Öffnen des Werkzeuges ausgeworfen werden.

Bei diesem Verfahren erreichen die Schließkräfte des Werkzeuges bis 100 MN[IS05]. Die Vorteile sind die gleichen wie beim Druckguss: komplexe Geometrien und geringe Wanddicken mit glatten Oberflächen formbar, niedrige Stückkosten bei großen Serien. Des Weiteren wird beim Spritzgießen eine

[2]Materialien, die sich in einem bestimmten Temperaturbereich verformen lassen.

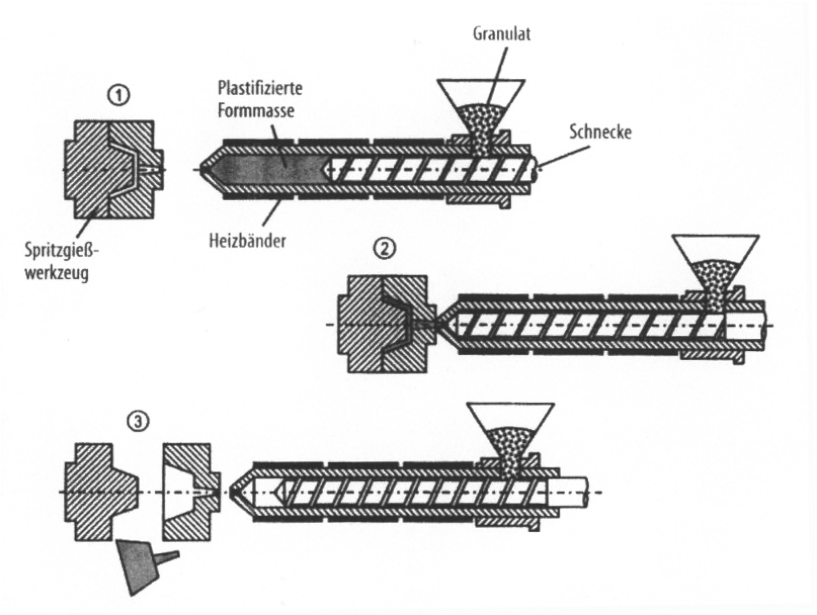


Abbildung 3.2: Ablauf des Spritzguss-Verfahrens [IS05]

niedrigere Verarbeitungstemperatur benötigt, da mit Kunststoffen gearbeitet wird. Somit sind die Werkzeuge nicht ganz so hohen Temperaturen ausgesetzt. Ein weiterer Vorteil besteht darin, dass, während des Aushärtens des Werkstückes, die Schnecke erneut befüllt, das Granulat erhitzt und zur Spitze befördert werden kann. Dies dient zur Vorbereitung des nächsten Zyklus, was eine Zeitersparnis bedeutet.

Hergestellt werden mit diesem Verfahren alle möglichen Teile aus Kunststoff, wie z. B. Kunststoffschüsseln, -spielzeug, Automobilteile aber auch größere Objekte wie Bootskörper und Müll-Großbehälter.

Aufbau der Gießwerkzeuge

Gießwerkzeuge bestehen meist aus Stahl und werden aus zwei Werkzeughälften aufgebaut: einer festen, die sich beim Anguss befindet, und einer beweglichen. Dazwischen befindet sich die Trennfläche. Es wird außerdem eine Aufspannplatte mit Führung und Zentrierung benötigt, sodass das Werkzeug korrekt geöffnet und geschlossen werden kann. Hinzu kommen das Anguss- und das Entformungssystem.

Damit die Abkühlzeit des Werkstückes drastisch reduziert werden kann, benötigt das Werkzeug eine Temperierung. Dadurch kann das Teil schneller entfernt werden, was zu einer höheren Wirtschaftlichkeit des Verfahrens führt.

3.1.2 Temperierbohrungen

Der Schmelze in der Kavität soll Wärme entzogen werden. Dies erfolgt über die Werkzeugwände, indem diese von einem Kanalsystem von Bohrungen durchzogen werden, durch das ein Kühlmedium gepumpt wird.

Da in Angussnähe die flüssige Formmasse am heißesten ist, wird dort auch die Werkzeugwand am heißesten (vgl. Abb. 3.3). Dort muss somit die größte Wärmemenge abgeführt werden. In Richtung des äußeren Randes nimmt die Temperatur der Schmelze ab, da sie bereits am Anguss etwas abgekühlt worden ist, und die Werkzeugwände werden an dieser Stelle die Temperatur, die am Anguss herrscht, nicht erreichen und müssen dem entsprechend weniger gekühlt werden.

Ein Kanalsystem kann aus vielen verschiedenen Bohrungen bestehen: langen, kurzen oder schrägen. In Abbildung 3.4 entspricht das Werkzeug dem durchscheinenden Quader. Die Bohrungen beginnen also immer an den Außenflächen des Werkzeuges.

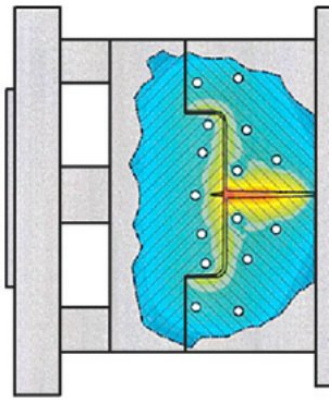


Abbildung 3.3: Wärmeverteilung im Werkzeug [Bay06]

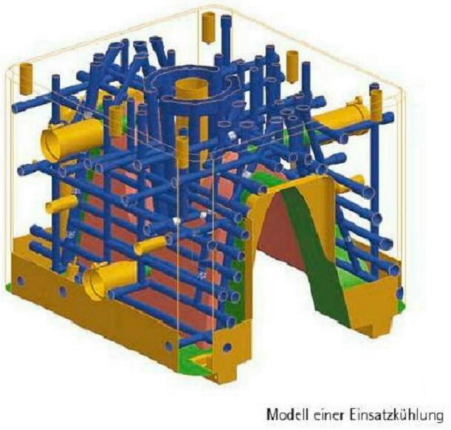


Abbildung 3.4: Beispiel eines Temperierbohrungssystems [VDW05]

Durch diese aus Bohrungen bestehenden Temperierkreisläufe wird ein Kühlmedium gepumpt, meist Wasser, Emulsion oder reines Öl. Wird Wasser benutzt, ist darauf zu achten, dass es möglichst kalkarm ist, da jede Ablagerung, sei sie durch Korrosion oder Kesselsteinbildung verursacht, den Durchfluss vermindert bis hin zur Verstopfung der Bohrung. Das Medium kann somit nicht im geplanten Umfang fließen und bestimmte Stellen nicht wie vorgesehen kühlen. Bei Verwendung von Öl als Temperiermedium ist zu

bedenken, dass die Wärmekapazität geringer und die Wärmeübergangszahl kleiner als bei Wasser ist, was dazu führt, dass der Bohrungskanaldurchmesser größer zu wählen ist, um die gleiche Menge an Wärme abzuführen.

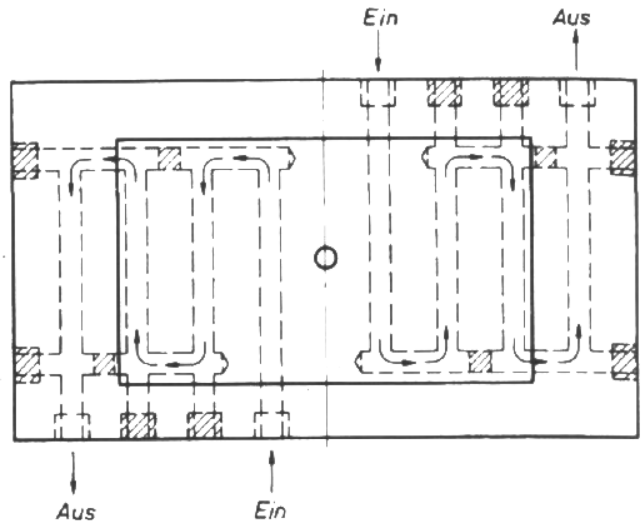


Abbildung 3.5: Temperierbohrungen für ein flächiges Werkstück [MMM99]

In Abbildung 3.5 ist ein recht einfaches Bohrungssystem für ein flächiges Werkstück dargestellt. Die Bohrungen bilden ein Gitternetz, das durch Stopfen zu einem Kreislauf geformt wird. Der Anguss befindet sich in der Mitte und wird als kleiner Kreis dargestellt. Die Eingänge der Temperierkreisläufe beginnen angussnah, da an dieser Stelle die meiste Wärme vom Temperiermedium aufgenommen werden muss, welches hier noch neu und somit am kühlsten im ganzen Kreislauf ist. Zum Rand des Werkstückes hin, muss die Schmelze einen längeren Weg passieren und wurde somit länger abgekühlt, was bedeutet, es muss auch weniger Wärme aus ihr abgeführt werden. Dieser Eigenschaft wird dadurch Rechnung getragen, dass die Temperatur des Mediums am Ende des Kreislaufes höher ist als zu Beginn und somit auch weniger Wärme abführbar ist.

Bei einem seitlichen Anguss wie in Abbildung 3.6 kann das Medium auch seitlich zugeführt werden und bis zur letzten Ecke verlaufen. Zur Befestigung der Stopfen wird meist ein Hilfsstift mit aufgeschobenen Stopfen benutzt (siehe Abbildung 3.7), der in eine Querbohrung eingeführt wird. Damit wird gewährleistet, dass kein Stopfen verrutscht und das Medium einen anderen als den geplanten Weg nehmen kann.

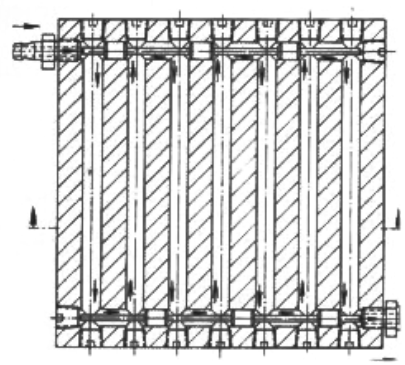


Abbildung 3.6: Temperierbohrungen bei seitlicher Anspritzung [MMM99]

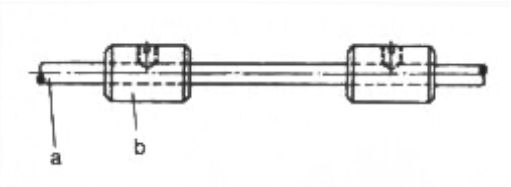


Abbildung 3.7: Hilfsstift (a) zur Befestigung des aufgeschobenen Stopfens (b) [MMM99]

Bei der Temperierung wird zwischen der Herstellung von Massenartikeln und Präzisionsteilen unterschieden. Für Massenartikel wird eine harte Temperierung bevorzugt, bei der die Bohrungen so nah an die Kavität angebracht werden, dass eine sehr niedrige Werkzeugwandtemperatur erreicht wird und die Werkstücke rapide abkühlen können. Das höchste Ziel bei solchen Großserien ist es, möglichst viele Teile in kürzester Zeit zu produzieren, weswegen evtl. größere Maßschwankungen bzw. größere Spannungen im Werkstück in Kauf genommen werden.

Bei den Präzisionsteilen wird jedoch eine hohe Qualität mit engen Maßtoleranzen angestrebt, d. h. beim Einspritzen soll eine hohe Werkzeugwandtemperatur herrschen, die dann langsam sinkt. Dies wird als weiche Temperierung bezeichnet, die eine lange Zykluszeit zur Folge hat. Da es schwieriger ist qualitativ hochwertige Produkte herzustellen, wird nachfolgend auf die Eigenschaften der weichen Temperierung eingegangen.

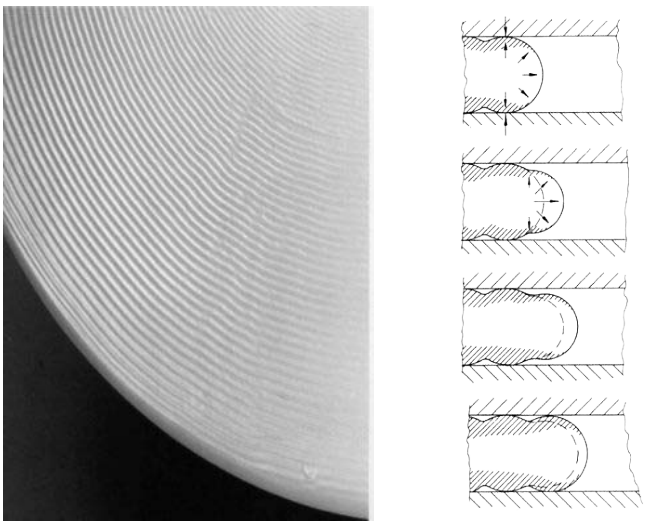
Falsche Temperierung

Warum ist es überhaupt wichtig eine *gute Temperierung* zu entwerfen? Dies sollen die nachfolgenden Beispiele illustrieren.

Schallplatteneffekt Hierbei ist beim Gießen eine rillenförmige Oberfläche entstanden, die an eine Schallplatte erinnert (siehe Abb. 3.8(a)). Zur Erklärung dieses Phänomens ist Abbildung 3.8(b) sehr gut geeignet. Die Fließfront der Schmelze bzw. der plastifizierten Formmasse breitet sich in alle Richtun-

gen gleichmäßig, also kugelförmig aus. Ist nun die Wandtemperatur sehr kalt, erstarrt die Oberfläche schnell und nur die heißere, innere Schmelze dringt weiter in Fließrichtung, kann jedoch die Form an dieser Stelle nicht mehr bis zum Rand ausfüllen, da sie nicht zurückfließen kann und selbst schnell erstarrt.

Als Gegenmaßnahme kann die Werkzeugwandtemperatur mittels Temperierung zu Beginn des Befüllens der Kavität höher gehalten werden, damit die Schmelze erst den gesamten Formhohlraum ausfüllen kann.

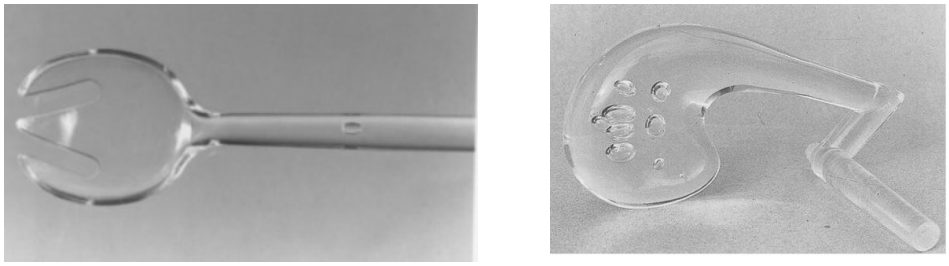


(a) (b)

Abbildung 3.8: Schallplatteneffekt: rillenförmige Oberfläche (a) und der Entstehungsprozess im zeitlichen Verlauf (b) [Bic06]

Lunker Lunker sind Hohlräume, die besonders bei durchsichtigen Teilen unschöne Stellen hinterlassen. Sie entstehen meist in Bereichen großer Wanddicken fern vom Anguss, da dort die Außenhaut, d. h. die äußere Schicht der Schmelze, schon abgekühlt ist, während der Kernbereich noch elastisch ist. Die Außenhaut nimmt die Schwindungsspannungen auf, die entstehen, wenn sich beim Abkühlen das Innere zusammenzieht. Die innere Masse wird durch das Erkalten nach außen gezogen und es entwickeln sich Lunker.

Dies kann u. a. vermieden werden, indem die Temperierung am Anguss nicht so stark ausfällt, sodass an dieser Stelle bis zuletzt flüssige Masse existiert, die zum Ausgleich der Schwindung an Bereichen großer Wandstärke nachgedrückt werden kann.



(a) (b)

Abbildung 3.9: Beispiele für Lunker: an einer Salatgabel (a), an dickwandigen Stellen (b) [Bic06]

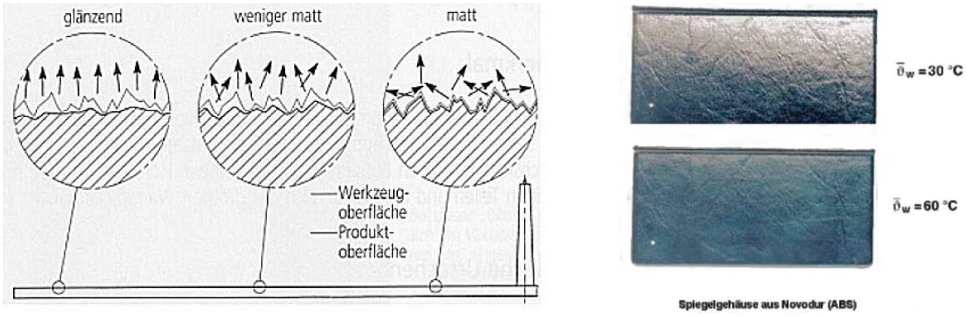
Freistrahlbildung Auch *Freistrahlbildung* tritt bei großen Hohlräumen auf. Hierbei kühlt der Schmelzstrahl ab bevor er auf eine Wand trifft und kann nicht mehr mit der nachfolgenden Masse verbunden werden, sodass er als Massestrang vor allem bei durchsichtigen Teilen sichtbar wird (Abb. 3.10). Abhilfe schafft eine höhere Werkzeugwandtemperatur im Bereich des Angusses.

Abbildung 3.10: Freistrahl [Bic06]

Glanz Ungewollter Glanz entsteht, wenn die Werkzeugwandtemperatur so niedrig ist, dass die Formmasse schon abgekühlt und damit fester geworden ist bevor sie die komplette raue Werkzeugoberfläche ausgefüllt hat, vgl. Abbildung 3.11(a). Die Produktoberfläche bildet hierbei eine glatte Ebene, sodass Lichtstrahlen in eine Richtung zurückgeworfen werden und das Werkstück als glänzend empfunden wird. Ist die Werkzeugtemperatur zu Beginn

des Einspritzens so hoch wie die der Schmelze, kann die Oberfläche sauber abgeformt werden und die Lichtstrahlen werden in verschiedene Richtungen reflektiert, was als matte Fläche sichtbar wird. Dargestellt sind diese beiden Effekte in Abbildung 3.11(b). Die obere Illustration zeigt Glanz, wenn eine Werkzeugtemperatur von $v = 30\,℃$ eingestellt ist, während auf der unteren Abbildung das gleiche Werkstück erkennbar ist, das jedoch bei einer Werkzeugtemperatur von $v = 60\,℃$ gegossen wurde und deswegen in mattem Zustand ist.

Das Problem bei diesem Phänomen liegt darin, dass die Oberfläche eines Werkstückes im Bereich des Angusses matt sein kann, da dort die Schmelze noch sehr warm und in der Lage ist die Kavität voll auszufüllen, während es angussfern zu glänzenden Effekten kommt (siehe Abbildung 3.11(a)). Um dies zu verhindern, muss beim Füllen der Kavität bis zu den Stellen fern des Angusses eine ausreichend hohe Temperatur des Werkzeuges vorliegen.



(a) (b)

Abbildung 3.11: Glanz (a) und Glanzunterschiede (b) [Bic06]

Unvollständige Formfüllung Bei der *unvollständigen Formfüllung* wird aufgrund einer zu niedrigen Werkzeugtemperatur das Formteil nicht vollständig ausgefüllt. Die Schmelze erstarrt bereits bevor sie bis zum angussfernen Bereich der Kavität gelangt. Abhilfe schafft eine höhere Temperatur, damit die Formmasse bis zur vollständigen Verteilung in der Kavität so flüssig bleibt, dass sie noch überallhin fließen kann.

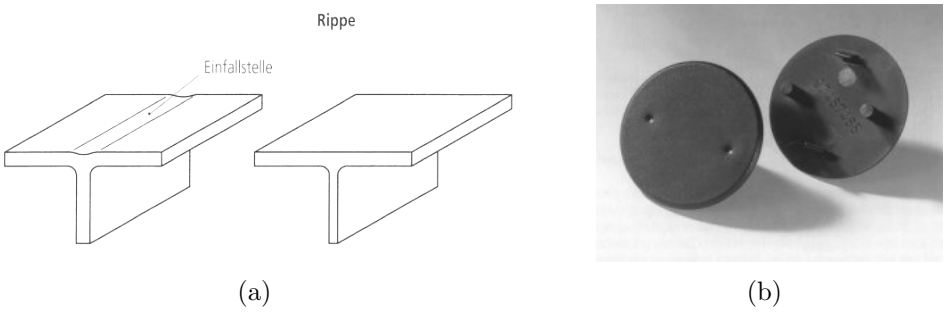
Verzug *Verzug*, also Deformation des Formteils, ist ein häufig auftretender Fehler bei Gussteilen. Dieser Effekt wird verursacht durch ungleichmäßige Abkühlung und somit auch ungleichmäßige Schwindung des Werkstückes,

wodurch innere Spannungen auftreten, die sich direkt oder nach einiger Zeit nach dem Entformen ausgleichen.

Dagegen hilft eine gleichmäßige Temperierung des Werkzeuges, sodass jede Stelle einheitlich abkühlt und sich dabei einheitlich zusammenzieht.

Einfallstellen Einfallstellen entstehen durch zu hohe Wandtemperaturen meist an dickwandigen Stellen, wie bei einer Rippe (siehe Abbildung 3.12(a)). Dabei ist die Randzone schon etwas erstarrt und es entstehen durch Schwindungskräfte Eigenspannungen in der Formteilmitte.

Werden diese zu groß und ist die Außenhaut noch nicht stabil genug, wird die äußere Schicht nach innen gezogen, was an Abb. 3.12(b) gut zu erkennen ist. Auch ist es möglich, dass dieser Fehler erst nach dem Entformen auftritt. Das Werkstück ist noch nicht genug abgekühlt, sodass die innere Wärme die Außenhaut aufwärmt und die durch Schwindung auftretenden Zugspannungen dadurch kompensiert werden, dass auch hierbei die Außenhaut nach innen gezogen wird. Verhindert werden kann dieses Phänomen, indem das Werkstück eine bessere, d. h. stärkeren und längeren, Kühlung erfährt.



(a) (b)

Abbildung 3.12: Einfallstelle schematisch an einer Rippe (a) und an einem gegossenen Bauteil (b) [Bic06]

Kosten

Einfluss auf die Kosten zur Herstellung eines Temperierbohrungssystems hat vor allem die Anzahl der Bohrungen, Stopfen und Dichtungen. Je mehr davon eingebracht werden, desto höher ist der Aufwand. Die Tiefe der Bohrungen spielt bei den Ausgaben ebenfalls eine wichtige Rolle, denn ab einer bestimmten Tiefe können herkömmliche Bohrer nicht mehr Bohrer benutzt werden, stattdessen muss ein sogenannter Tiefbohrer eingesetzt werden. Das bewirkt

einen sprunghaften Anstieg der Kosten, der in der Kalkulation berücksichtigt werden muss.

Die Herstellung des Systems wird u. a. umso teurer, je größer der Anstellwinkel zur Normalen für die Bohrung angesetzt werden muss. Die Bohrung senkrecht in das Werkzeug hinein, also mit einem Winkel von 0°, ist ohne weiteres möglich und somit recht billig. Soll jedoch eine Bohrung schräg angebracht werden, treten schon mehr Schwierigkeiten auf, wie z. B., dass der Bohrer leichter abrutschen kann und unmittelbar danach abbricht. In einem solchen Fall muss zuerst an der betroffenen Stelle eine Planfläche gefräst werden, damit sich der Bohrer auch bei sehr großen Winkeln richtig aufsetzen lässt.

Restriktionen

Bei der Erstellung des Systems von Bohrungen müssen einige Restriktionen eingehalten werden. Dazu zählt, dass die Bohrungen die Kavität weder berühren noch zu nah an dieser platziert sein dürfen, denn durch die Drücke, die während des Prozesses auf das Werkzeug wirken, kann es passieren, dass die Wand zwischen der Kavität und der Bohrung durchbricht bzw. bei einer Berührung dieser beiden gar nicht existiert und die Schmelze in die Bohrungskanäle laufen kann, sodass die gewünschte Werkstückform nicht mehr hergestellt werden kann. Des Weiteren muss gewährleistet sein, dass genügend Platz für die Bohrungen vorhanden ist und nicht durch Entformungssysteme oder Zentrierungsstifte die Positionierung verhindert wird. Die Bohrungen sollen sich schneiden und mithilfe von Stopfen einen Kreislauf bilden, damit das zugeführte Medium auch wieder abgeführt werden kann.

Innere Kreise, also Kreisläufe bei denen die Ein- und Ausgänge nicht am Rand des Werkzeuges liegen, und sich gegenseitig schneidende Kreisläufe, bei Benutzung mehrerer Kreisläufe, müssen verhindert werden. Ansonsten kann ein Kurzschluss entstehen, d. h. das Medium würde bestimmte Kanäle nicht mehr bzw. mit einer sehr geringen Strömungsgeschwindigkeit durchlaufen, also „Abkürzungen“ einschlagen. Dadurch wären diese Bereiche nicht mehr gekühlt und eine ungleichmäßige Abkühlung erfolgt, was zu Formteilfehlern führen kann.

Ziele

Das höchste Ziel eines Konstrukteurs von Temperierbohrungen ist, es die Wärme aus dem Werkstück abzuführen, um es zu kühlen und das Material

somit hart und fest werden zu lassen. Dabei wird eine gleichmäßige Temperaturverteilung während des gesamten Abkühlvorgangs angestrebt, sodass kein Bereich schneller abkühlt und das Formteil nicht durch die Schwindungsspannungen verzogen wird. Die Abkühlung sollte möglichst schnell geschehen, um die Kosten gering zu halten.

Um geringe Kosten zur Herstellung der Kühlkanäle zu erhalten, werden Lösungen mit wenigen Kreisläufen bevorzugt, die mit einer geringen Anzahl an Bohrungen konstruiert werden. Ein weiterer Grund für solche Lösungen ist, dass jede Bohrung eine Schwächung des Werkzeuges bedeutet und somit weniger Bohrungen stabiler sind. Ziel ist es auch Lösungen zu finden, die kleine Kreisläufe beinhalten, da die Temperaturdifferenz des Mediums zwischen Ein- und Ausgang so gering wie möglich gehalten werden soll. Ein längerer Weg, den das Medium durch die Kanäle einschlägt, bedeutet auch, dass das Medium mehr Wärme abführen muss, wobei es sich erwärmt. Je wärmer es ist, desto weniger kann es das umliegende Werkzeug und somit das Werkstück kühlen, was jedoch Sinn und Zweck seiner Errichtung gewesen war.

Zu den Zielen zählt auch, die Winkel zwischen sich schneidenden Bohrungen im Bereich zwischen 30° und 90° zu halten. Winkel unter 30° sind problematisch, da bei der Herstellung durch einen Bohrer Kräfte auf die Wände der Kanäle wirken und die Bohrwerkzeuge abbrechen könnten.

Schnittstellen

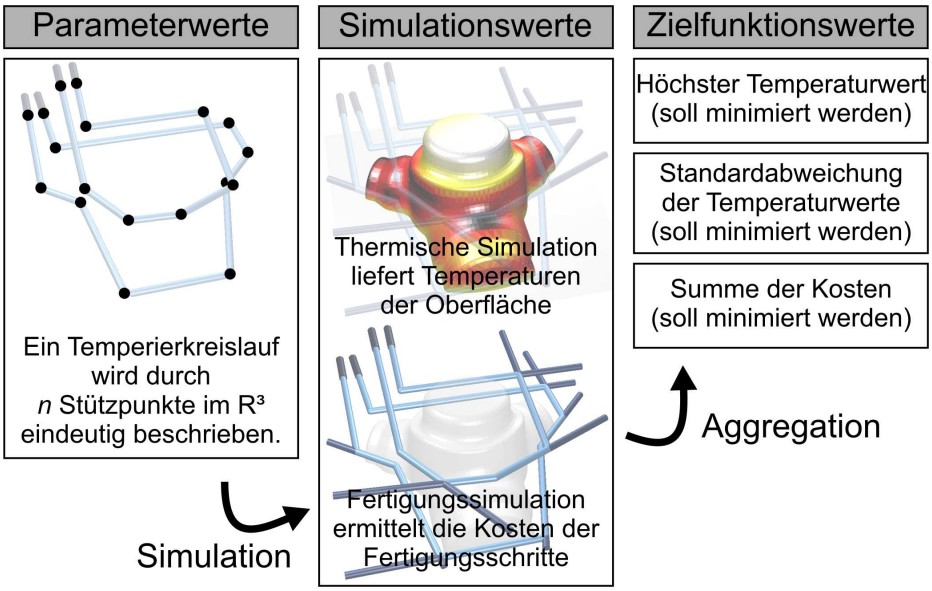


Abbildung 3.13: Gewinnung der Zielfunktionswerte aus den Simulationswerten [Mül06]

Ein Temperierkreislauf wird durch seine Eckpunkte in 3D-Raumkoordinaten beschrieben. Zusätzlich sind Informationen, in welche Richtung die Teilstrecken des Kreislaufes gebohrt werden sollen, zu berücksichtigen. Auf diesen Parameterwerten wird eine Simulation (Abb. 3.13) angewandt, die für jeden Punkt auf der Oberfläche des Werkstückmodells eine Temperatur liefert, die dort bei den gegebenen Bohrungen entstehen würden. Daraus kann der Ort der höchsten Temperatur ermittelt werden. Eine weitere Simulation berechnet die einzelnen Kosten für den gesamten Temperierkreislauf.

Aus diesen Werten werden die wenigen Zielfunktionswerte berechnet, die der evolutionäre Algorithmus optimiert, hier also minimiert. Zurzeit werden der höchste Temperaturwert auf der Werkstückoberfläche, die Standardabweichung der Temperaturwerte und die Summe der Kosten zur Herstellung der Bohrungen in einer Fitnessfunktion gewichtet zusammengefasst. Es gehen also nur sehr wenige Werte in diese einkriterielle Optimierung ein. Für diese Gewichtung wird schon vorher festgelegt, welchem Wert mehr Bedeutung beigemessen wird. In der Software, die die PG entwickelt hat, gelten die Werte bzw. weitere Werte wie z. B. Kosten der Bohrungen oder niedrigster Temperaturwert als Kriterien für die mehrkriterielle Optimierung und werden somit getrennt dargestellt.

Visualisierung

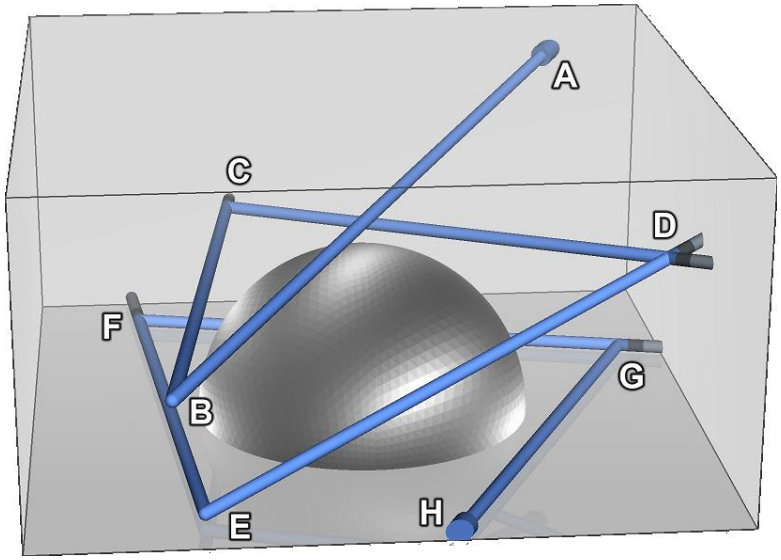


Abbildung 3.14: Ein Temperierkreislauf für ein halbkugelförmiges Werkstück [MM06]

Eine Lösung, die der evolutionäre Optimieralgorithmus liefert, könnte wie in Abbildung 3.14 veranschaulicht werden. Hierbei wird das CAD-Modell des Werkstückes dargestellt, hier eine einfache Halbkugel. Die Bohrungen sind als zylindrische Form visualisiert und die Enden, die mit Stopfen versehen werden, werden schwarz eingefärbt. Das Werkzeug wird quaderförmig angedeutet. Es muss durchsichtig sein, da sonst die wichtigeren Bohrungen und das Modell nicht zu sehen wären. Trotzdem muss es erahnbar sein, damit leicht erkennbar ist, dass die Bohrungen auch wirklich am Rand des Werkzeuges beginnen.

Die Simulationswerte der Temperaturen auf der Werkstückoberfläche können dort in Form von Farben oder Graustufen angezeigt werden. Dabei soll eine kalte Stelle als blau gezeichnet werden und je wärmer ein Ort ist, desto mehr geht die Farbe über das Gelbliche ins Rötliche. In diesem Graustufen-Bild wird eine Stelle umso heller gezeichnet, je kühler sie ist.

Die skalaren Werte, die aus der Optimierung resultieren (höchster Temperaturwert, Standardabweichung der Temperaturwerte, Summe der Kosten), können so nicht in die Graphik integriert und werden als Eigenschaftswerte einer Lösung separat in der Legende dargestellt (siehe 8.13 auf Seite 146).

Vergleich

Der evolutionäre Algorithmus gibt mehrere Kompromisslösungen aus, die sich durch die zu optimierenden Werte unterscheiden. Auch die grobe Form der Bohrungen spielt beim Vergleich eine Rolle, denn es gibt viele Lösungen, deren Grobstruktur ähnlich ist. In Abbildung 3.15 sind vier Beispiele aufgeführt, die sich aufgrund der Anordnung ihrer Bohrungen sehr unterscheiden. Beispiel (a) zeigt eine M-Form, während die Bohrungen in (b) nur einmal quer über das Werkstück verlaufen. Bei (c) und (d) bewegen sich die Bohrungen viereckig bzw. dreieckig um das Werkstück herum. Da sehr viele Lösungen vom evolutionären Algorithmus ausgegeben werden, ist es gut, diese in solche Gruppen einzuordnen. Das würde einem Clustern im Parameterraum (decision space) entsprechen. Sollen die Lösungen zusammengefasst werden, die z. B. aufgrund ihres höchsten Temperaturwertes ähnlich sind, so wird dies als Clustern im Zielfunktionsraum (objective space) bezeichnet.

Ein weiteres Mittel, damit die Lösungen besser zu vergleichen sind, ist die Temperatur als Farbe (hier: Graustufen) darzustellen. Es ist deswegen schnell zu erkennen, dass im Beispiel (b) der höchste Punkt der Halbkugel mehr gekühlt wird als der im Beispiel (c).
Abbildung 3.14 zeigt eine mögliche Lösung mit mehr als vier Bohrungen, während das System in Abbildung 3.16 ein anderes Werkstück kühlen muss, was hier mit zwei separaten Kreisläufen geschieht.

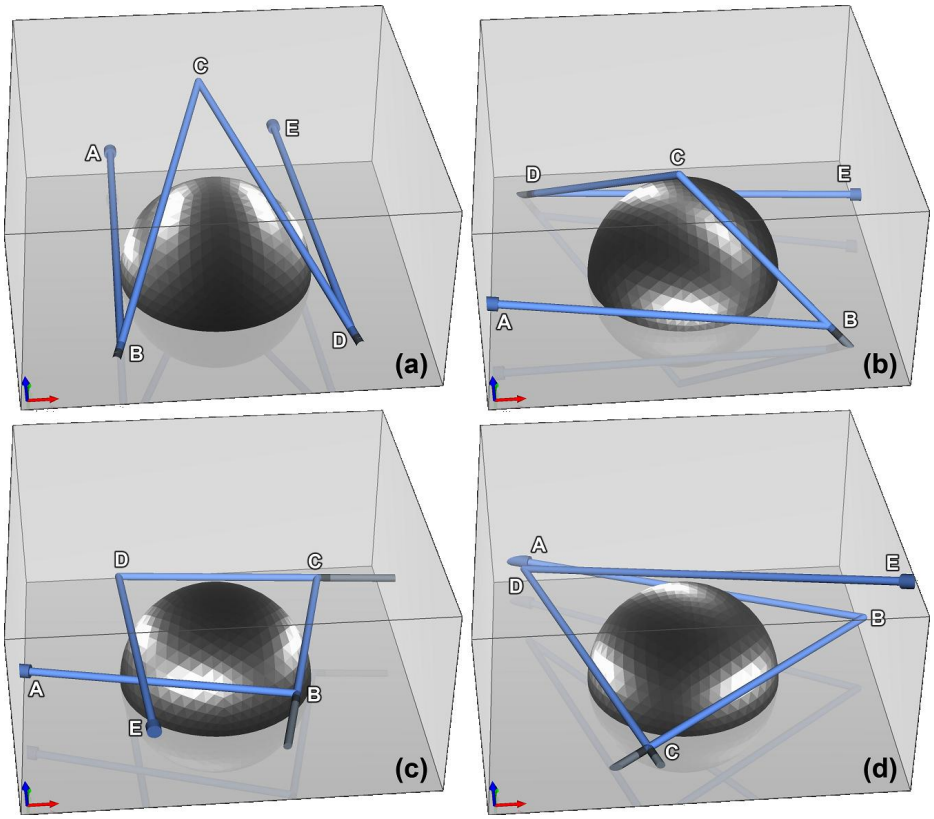


Abbildung 3.15: Visualisierung der Lösungen des EAs [MM06]

3.1.3 Fazit

Da die Optimierung eines Temperierbohrungssystem mehrkriteriell erfolgt und somit meist mehrere gute Kompromisslösungen existieren, kann der Computer nicht einfach die beste Lösung ausgeben. Deshalb muss der Mensch entscheiden, welches die beste Lösung für die entsprechende Situation ist, weil er noch einige Parameter mehr kennt, die in der Simulation bzw. bei der Bewertung nicht berücksichtigt worden sind bzw. werden konnten. Als Beispiel kann angegeben werden, dass der Ingenieur Kenntnis davon hat an welcher Stelle nicht gebohrt werden kann, da das Werkzeug dort an der Wand steht oder aus sonstigen Gründen Platzmangel herrscht. Auch weiß er um die Fehler der Simulation, die auftreten können. Liegt z. B. zusätzlich zwischen der Bohrung und dem Formteil ein Auswerfer und wurde dieser bei der Simulation nicht berücksichtigt, so fallen die realen Temperaturwerte an der entsprechenden Stelle höher aus als die der Simulation. Des Weiteren ist der Konstrukteur in der Lage zu beurteilen, ob die Stelle des höchsten

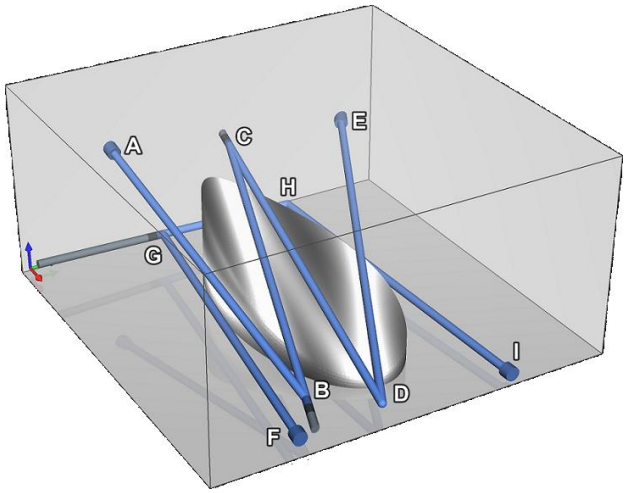


Abbildung 3.16: Mehrere Kreisläufe [MM06]

Temperaturwertes unkritisch ist für die Abkühlung des Werkzeuges oder ob dort aufgrund äußerer Gegebenheiten nicht intensiv gekühlt werden muss, da sich an dieser Seite ohnehin eine Außenwand befindet.

Aber selbst ein Mensch kann aufgrund der Vielzahl der möglichen Lösungen, die schon bei dem Beispiel aus Abbildung 3.15 mit nur vier Bohrungen entstehen, den für sein Problem besten Kompromiss nur schwer auswählen, wenn er nur die Darstellung der Lösungen nebeneinander anschauen kann.

Es war Ziel der von der PG zu entwickelnden Software, dass sie die beiden Räume der Lösungen einfach und übersichtlich darstellen kann und diese Lösungen zu vielsagenden Gruppierungen zusammenfasst, einerseits im Parameterraum, also aufgrund ihrer Anordnung der Bohrungen, und andererseits im Zielfunktionsraum, also wegen ihrer Werte bei den Kriterien. Des Weiteren sollten die Gruppen untereinander wie auch die Lösungen innerhalb der Gruppe miteinander so zu vergleichen sein, dass die Analyse dieser Vergleiche für den Betrachter einfacher, intuitiver und schneller erfolgen kann. Inwieweit diese Ziele erreicht wurden, wird in Abschnitt 2.1 erläutert.

3.2 Fräsbahnoptimierung

Für die Erstellung und Bearbeitung von Oberflächen und Formen im Werkzeug- und Formenbau ist der Einsatz von mehrachsigen Fräsmaschinen oft sinnvoll. In dem folgenden Abschnitt sollen sowohl die Grundlagen des Fräsprozesses, als auch die Schwierigkeiten, für ein gegebenes Werkstück eine optimale Fräsbahn zu finden, erläutert werden. Der Schwerpunkt wird auf die Erklärung der einzelnen Kriterien gelegt, mit denen die Güte der Fräsbahn bestimmt werden kann.

3.2.1 Grundlagen des Fräsens

Bevor auf die Optimierung von Fräsbahnen eingegangen wird, sollen zunächst die Eigenschaften und die Besonderheiten des Fräsprozesses beschrieben werden.

Einordnung des Fräsens als Fertigungsverfahren

Fräsen bezeichnet nach DIN 8580 ein Fertigungsverfahren, welches in der Untergruppe des „Trennens“ eingeordnet ist. Der Einsatzzweck ist demnach ein Materialabtrag von einer Form. Wie auch Bohren oder Drehen, ist Fräsen ein Verfahren des *Spanens mit geometrisch bestimmter Schneide*, beschrieben in DIN 8589. Der Materialabtrag und somit die Änderung der Werkstückform geschieht unter mechanischem Krafteinsatz in Form von Spänen, wobei die Schneiden des Werkzeuges in Form und Anzahl exakt bekannt sind. Im Gegensatz hierzu stehen *spanende Verfahren mit geometrisch unbestimmten Schneiden*, zu denen beispielsweise das Schleifen zählt. Hier sind die Schneiden unterschiedlich geformt; es sind nur statistische Größen über ihre Schneidgeometrie verfügbar.

Fräsen kann leicht durch zwei Eigenschaften von weiteren Verfahren des Spanens mit geometrisch bestimmten Schneiden unterschieden werden:

- Die Schnittbewegung erfolgt durch eine Rotation des Werkzeuges.
- Der Schnitt ist nicht kontinuierlich, sondern unterbrochen.

Einsatzgebiete des Fräsens

Übersicht der Fräsverfahren Es existieren zwei grundsätzliche Fräsverfahren, das *Umfangs-* sowie das *Stirnfräsen*. Der Unterschied ergibt sich aus der Lage der Rotationsachsen des Werkzeuges in Bezug auf die Vorschubrichtung.

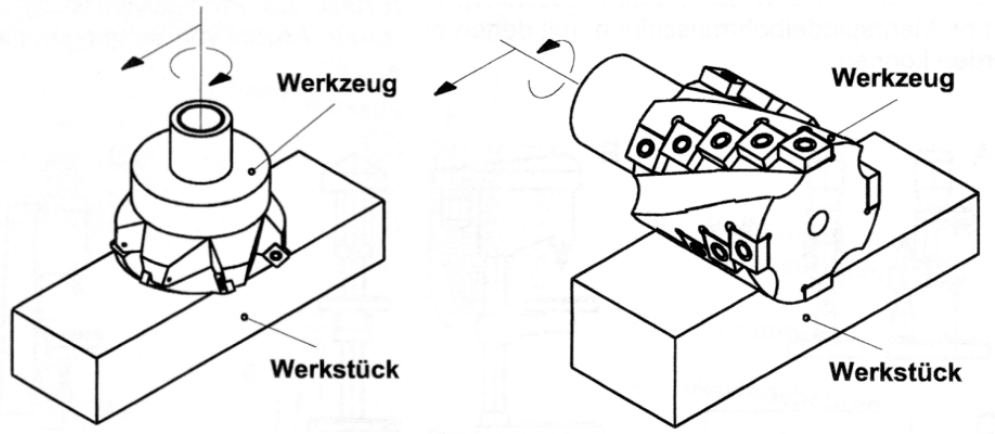


Abbildung 3.17: Darstellung von Stirn- und Umfangsfräsen. [Wei00]

Beim Stirnfräsen steht die Rotationsachse orthogonal zur Vorschubrichtung, beim Umfangsfräsen befinden sie sich in der gleichen Ebene. Kommen beide Verfahren gleichzeitig zum Einsatz, wird dies als Stirn-Umfangs-Fräsen bezeichnet.

Ein häufig eingesetztes Verfahren ist das *Formfräsen*, bei dem ein in mehreren Achsen frei beweglicher Fräser eine weitgehend beliebige Fläche auf einem Werkstück erzeugt. Dieses Verfahren wird im Folgenden detaillierter behandelt.

Typische Anwendungsgebiete Die Einsatzmöglichkeiten des Fräsens als Fertigungsverfahren sind sehr vielfältig. Ein Haupteinsatzgebiet liegt im Bereich des Werkzeug- und Formenbaus. Hierbei ist besonders das Formfräsen zu betonen, mit dem weitgehend beliebige räumliche Flächen hergestellt werden können. Durch die Flexibilität der Fräsmaschinen wird dieses Verfahren besonders für kleine Serien und Einzelfertigungen benutzt. Für eine Umstellung auf die Fertigung eines anderes Werkstück ist üblicherweise nur das NC-Programm der Fräsmaschine und gegebenenfalls das Werkzeugmagazin auszutauschen. Als Anwendung ergeben sich beispielsweise die Fertigung von Gesenken und Pressstempeln oder auch Turbinenschaufeln für den Flugzeugbau. Weitere Anwendungsgebiete, auch für mittlere bis große Serien, sind die Erzeugung von Freistichen und Nuten.

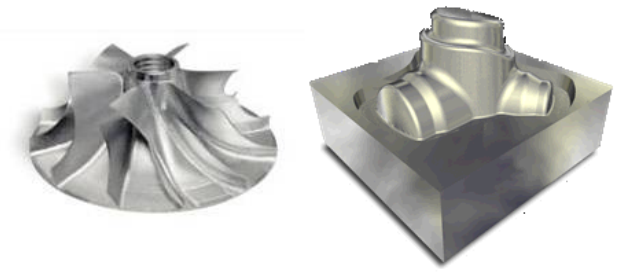


Abbildung 3.18: Beispiele: Impeller und Gesenk

Fräsen als Prozess

Für die Optimierung des Fräsprozesses ist ein Modell nach [TD04] (siehe Abbildung 3.19) sinnvoll, bei dem die verschiedenen Eingangs- und Ausgangsparameter in Verbindung gebracht werden.

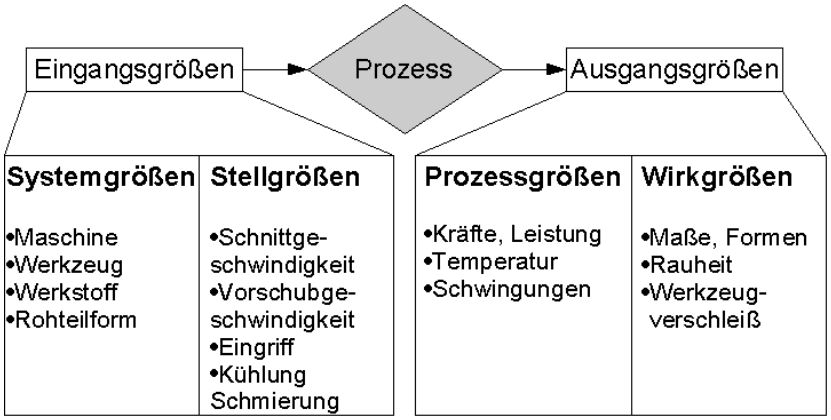


Abbildung 3.19: Zerspanprozess als System nach [TD04].

Die unter Systemgrößen zusammengefassten Parameter sind durch die vorhandenen Maschinen und das zu erstellende Werkstück vorgegeben und meist nicht veränderbar. Modifizierbare Parameter, welche für eine Optimierung herangezogen werden können, sind unter Stellgrößen zusammengefasst. Die Ausgangsgrößen des Fräsprozesses werden unterschieden in Prozessgrößen, die nur während der Bearbeitung auftreten, und Wirkgrößen, welche dauerhafte Veränderungen an Werkstück sowie Werkzeug beschreiben. Dieses Modell gilt allgemein für alle spanenden Verfahren; die besonderen Eigenschaften bezüglich des Fräsens sollen im Folgendem genauer behandelt werden.

Arbeitseingriff Ein großes Potenzial für eine Optimierung liegt in der Veränderung des Arbeitseingriffs der Schneiden. Hierbei werden beim Fräsen zwei Grundrichtungen unterscheiden: *Gleichlauffräsen* und *Gegenlauffräsen*. Die beiden Verfahren unterscheiden sich durch die Schnittrichtung des Werkzeugs und den Vorschub des Werkstücks.

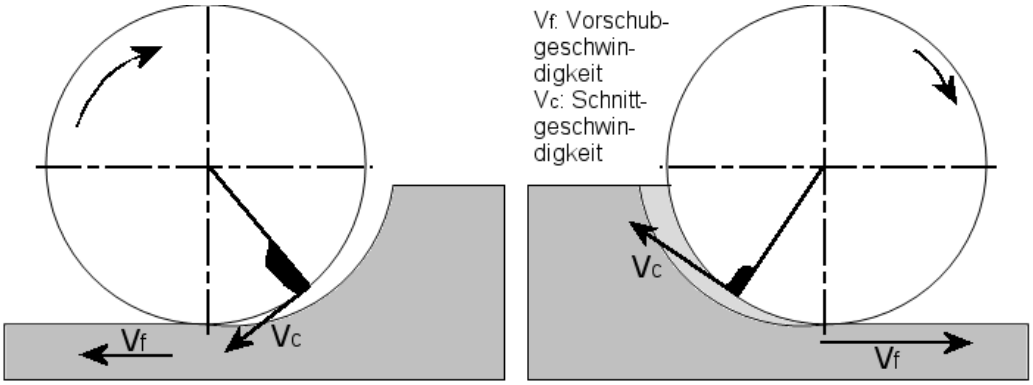


Abbildung 3.20: Darstellung des Gleich- und Gegenlauffräsens. [KK97]

Beim Gleichlauffräsen weisen Vorschub und Werkzeugdrehung in die gleiche Richtung. Die Schnittkraft ist im Moment des Schneideneingriffs am größten und nimmt mit fortlaufendem Schnitt langsam ab, bis beim Austritt der Schneide eine die Spandicke null wird. Der Eingriffswinkel zwischen Schneide und Werkstück ist dabei so klein, dass es zu keiner definierten Spanabnahme mehr kommt und einzig Reib- und Quetschvorgänge auftreten. Durch eine Steigerung des Vorschubs ist eine Erhöhung des Zeitspanvolumens zu erreichen, dies ist aber mit einem gleichzeitigen Anstieg der auftretenden Kräfte und meist auch einer Verschlechterung der Oberflächengüte verbunden. Grundvoraussetzung für das Gleichlauffräsen ist eine spielfreie Vorschubeinrichtung, um ein Einziehen des Werkstücks in das Werkzeug durch die hohen und abrupt auftretenden Anschnittskräfte zu vermeiden.

Beim Gegenlauffräsen wirkt die Schnittkraft der Vorschubbewegung entgegen. Die Schneiden treten mit Nullspanungsdicke in das Werkstück ein, die Spandicke nimmt mit Dauer des Schnittes zu, wodurch es zu einem Kraftaufbau kommt. Das Gegenlauffräsen besitzt zwei Nachteile. Durch den kleinen Winkel im Anschnitt kommt es unmittelbar vor dem Eindringen der Schneide zu Quetschvorgängen, welche die Werkstückoberfläche beschädigen kann. Zudem erzeugen die hohen Kräfte, die beim Schneidenaustritt vom Werkstück weggerichtet sind, einen starken Zug auf das Material, sodass vermehrt Spanausbrüche auftreten. Zusätzlich bedarf es bei labilen Materia-

lien, wie zum Beispiel Blechen, einer besonders festen Einspannung, die das gesamte Werkstück fixiert. Bedingt durch die von Werkstück weggerichtete Fräskraft, kann ein Blech von der Aufspannfläche angehoben werden oder zu Rattern beginnen. Aus diesem Grund wird, falls eine spielfreie Vorschubeinrichtung verfügbar ist, meist das Gleichlauffräsen bevorzugt. Dieses Ziel ist jedoch nicht immer erreichbar, da auf einer Fräsbahn meist Gleich- und Gegenlauffräsen gleichzeitig auftreten. Dennoch bleibt die Möglichkeit, den Eingriff zu verändern, einer der wichtigsten Faktoren in der Optimierung, da hierbei besonders die auftretenden Kräfte beeinflusst werden können.

Schnitt- und Vorschubgeschwindigkeit Eine Veränderung der Schnitt- und der Vorschubgeschwindigkeit ist eine weitere Möglichkeit, den Fräsprozess zu optimieren. Die Schnittgeschwindigkeit bezeichnet die Geschwindigkeit, mit der die Schneiden in Schnittrichtung durch das Material geführt wird. Sie berechnet sich aus dem Werkzeugumfang multipliziert mit der Drehzahl des Werkzeuges. Die Vorschubgeschwindigkeit ist, neben der Länge der Fräsbahn und der Schnittgeschwindigkeit, maßgeblich ausschlaggebend für die Fräsprozessdauer und damit besonders unter wirtschaftlichen Gesichtspunkten interessant.

Eine hohe Schnittgeschwindigkeit stellt große Anforderungen an die verwendete Maschine und deren Antriebe. Hohe Geschwindigkeiten haben einen günstigen Einfluss auf die Spanbildung (es bildet sich ein sehr glatter Schnitt), sie führen aber auch zu einem starken Anstieg der auftretenden Schnittkraft. Dieser Faktor ist jedoch stark materialabhängig. Bei hoher Schnittgeschwindigkeit steigen zudem die Temperaturen, welche das Schneidwerkzeug zusätzlich belasten.

Um die Vorschubgeschwindigkeit zu erhöhen, muss meist auch die Schnittgeschwindigkeit gesteigert werden. Hierbei stellen die Leistung der Maschinen und auch die Steifigkeit der Werkzeuge Grenzen dar, um eine einwandfreie Bearbeitung der Werkstückoberfläche zu erreichen. Eine dauerhafte Beschädigung des Werkzeuges durch zu hohe Kräfte ist auf jeden Fall zu vermeiden.

Oberflächengüte Die Qualität der Oberfläche nach dem Fräsen kann unter verschiedenen Gesichtspunkten betrachtet werden. Wichtig sind Maß-, Lage- und Formfehler, die aussagen, inwiefern die Geometrie des gefertigten Stückes dem gewünschten Modell entspricht. Liegen die Werte innerhalb der Toleranzen, kann das Stück als Gutteil verwendet werden. Falls die Abweichungen zu groß sind, muss das Teil nachbearbeitet, oder als Ausschuss aussortiert werden. Werden die Toleranzen nicht erfüllt, sollte der Fertigungsprozess

dahingehend überarbeitet werden, die Genauigkeit zu erhöhen.

Schwieriger als die Korrektur dieser geometrischen Fehler ist die Verbesserung der Oberfläche selbst, indem die Rauheit durch eine Bahnoptimierung verringert wird.

3.2.2 Mehrachsige Fräsbearbeitung

Nach den allgemeinen Grundlagen des Fräsens, sollen im folgenden Abschnitt detaillierter die Oberflächenbearbeitung durch mehrachsigen Fräsverfahren eingegangen werden.

Drei-Achs-Fräsen

Das Drei-Achs-Fräsen ist ein Standardverfahren zur Erzeugung und Bearbeitung von Werkstückoberflächen. Das Fräswerkzeug ist hierbei in den drei translatorischen Raumachsen innerhalb des Aktionsraumes der Fräsmaschine frei beweglich. Weiterhin ist die Drehzahl des Fräsers regelbar. Abhängig vom

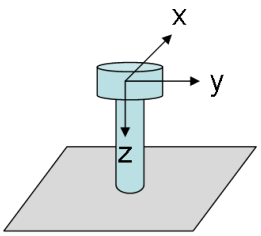


Abbildung 3.21: Bewegungsrichtungen des Drei-Achs-Fräsens.

Fräserdurchmesser kann allerdings nicht immer mit der maximalen Drehzahl gearbeitet werden, da besonders bei großen Fräsern die auftretenden Fräskräfte die Leistung der Spindelmotoren übersteigen. In diesem Fall müssen die Spindeldrehzahlen bedeutend reduziert werden.

Die gesamte Anlage besteht zumeist aus einer abgeschlossenen Zelle, in der sich die Fräsmaschine, Spannvorrichtungen und Werkzeugwechselvorrichtungen befinden, sowie einer Steuerungseinheit, welche den NC-Code in Steuersignale für die Schrittmotoren der Maschine übersetzt.

Ein CAM-Programm, üblicherweise auf einem Standard-PC, wird für die Erstellung der Fräsbahnen eingesetzt. Ausgehend von den erstellten Bahnen ist ein CAM-Programm in der Lage, mit einem geeigneten Post-Prozessor die CNC-Befehle für die jeweils eingesetzte Steuereinheit zu erzeugen.

Der Vorteil dieser Methode liegt darin, ein Programm auf verschiedenen Steuerungen laden zu können, indem nur das Post-Prozessor-Modul gewechselt wird.

Erzeugung einer Fräsbahn mittels CAM CAM-Software bietet heutzutage eine breite Palette von Werkzeugen zur Generierung von Fräsbahnen. In [Vog04] wird ein typischer Vorgang zur Fräsbahnerstellung beschrieben. Das zu fertigende Werkstück liegt dabei meist schon als CAD-Modell vor, welches direkt importiert werden kann. Weiterhin muss in der Software der verfügbare Werkzeugsatz eingerichtet werden, damit die Geometrien, maximaler Vorschub, Schnitttiefe und weitere Parameter der Fräsmaschine bei der Fräsbahngenerierung berücksichtigt werden können.

Besonders beim „Fräsen aus dem Vollen“, also der Erzeugung der Werkstückform aus einem massiven Stück Material, wird der Fräsprozess in einzelne Schritte unterteilt, bei denen mit verschiedenen Werkzeugen und Strategien gearbeitet wird. Üblich sind ein oder mehrere sogenannte Schruppgänge. Bei diesem ersten Schritt steht der Materialabtrag mittels eines großen Fräsers im Vordergrund. Um eine gleichmäßige Werkzeugbewegung sicherzustellen, wird der Fräser in Längs- oder Querbahnen geführt. Bei annähernd runden Formen kann auch eine ringförmige Werkzeugbahn sinnvoll sein. Um die Ergebnisse zu kontrollieren und auf Kollisionen zwischen Werkstück und Werkzeugschaft bzw. dem Werkzeughalter zu überprüfen, stehen Simulatoren zu Verfügung. Diese Kollisionen müssen auf jeden Fall verhindert werden, um dauerhafte und damit kostspielige Beschädigungen des Werkstückes oder des Werkzeuges zu verhindern.

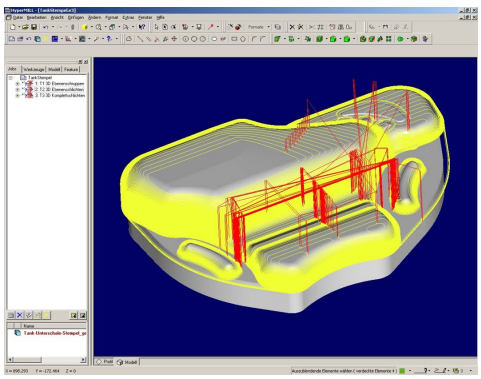


Abbildung 3.22: Ebenenschlichten am CAD-Modell in Hypermill. [A. Peuker]

Zur Erzeugung einer glatten Oberfläche müssen anschließend noch zusätzliche Schlichtgänge mit kleinen Fräsen vorgenommen werden. Zwar ist der Materialabtrag hierbei bedeutend geringer, aus der meist längeren Fräsbahn im Vergleich zum Schruppvorgang resultiert allerdings eine hohe Bearbeitungszeit. Nach Abschluss der Modellierung können die festgelegten Bahnen auf die Steuerung übertragen werden. Zu beachten ist, dass die Qualität der Fräsbahn zu einem großen Teil auch von der Erfahrung des Bedieners abhängt, da dieser die automatisch generierte Bahn der CAM-Software modifizieren und optimieren kann. Dies gilt besonders für das komplexere Fünf-Achs-Fräsen, bei dem zwar die gleichen Abläufe vorkommen, die Anforderungen an Software und Bediener aber aufgrund der größeren Anzahl an Freiheitsgraden ungleich höher liegen. Fünf-Achs-Fräsen werden im Abschnitt 3.2.2 detailliert behandelt.

Einschränkungen des Drei-Achs-Fräsens

Die größte Einschränkung für das Drei-Achs-Fräsen besteht darin, dass es nicht möglich ist, Hinterschnitte zu erzeugen. Für die Erzeugung einer Form mit Hinterschnitt sind auf einer Drei-Achs-Fräsmaschine mehrere Bearbeitungsschritte notwendig, in denen das Werkstück jeweils in verschiedenen Positionen aufgespannt wird. Dazu wird das Werkstück in einem anderen Winkel auf dem Frästisch befestigt, um vorher unerreichbare Bereiche durch die veränderte Lage bearbeiten zu können. Dieses Vorgehen ist allerdings aufgrund verschiedener Eigenschaften problematisch. Das Umspannen benötigt Zeit und macht die Fertigung damit unwirtschaftlicher. Dies ist aber bei Kleinserien häufig tolerierbar. Zusätzlich besteht ein erhöhtes Risiko, dass bedingt durch das Umspannen Ungenauigkeiten entstehen, die außerhalb der Fertigungstoleranzen liegen und somit das Werkstück entweder aufwändig nachbearbeitet werden muss oder Ausschuss ist. Zusätzlich steigen auch die Anforderungen an die CAM-Software, um Fräsbahnen für mehrere Aufspannungen zu erzeugen.

Eine weitere Schwierigkeit für das Drei-Achs-Fräsen ist die Kollisionsvermeidung zwischen Werkstück und Werkzeughalter. Für die Erzeugung sehr tiefer Fräskonturen muss der Werkzeugschaft eine entsprechende Länge aufweisen und ist damit prinzipiell sehr instabil. Dieses ungünstige Verhältnis von Frässchaftlänge zum Durchmesser vergrößert das Risiko, dass es bei hohen Kräften zu einem Bruch des Fräsers kommt. Auch steigt die Anfälligkeit für Schwingungen des Fräsers und einer daraus resultierenden Beschädigung der Werkstückoberfläche.

Fünf-Achs-Fräsen

Das Fünf-Achs-Fräsen vermeidet einige der Schwächen des Drei-Achs-Fräsens, indem der Fräser zusätzlich zu den Bewegungen in den Raumachsen geneigt und um die eigene Achse geschwenkt werden kann. Als Alternative gibt es auch Realisierungen, bei denen der Frästisch geneigt und gedreht werden kann. Zusätzlich existieren Zwischenformen, bei denen nur ein Neigen des Fräskopfes oder des Frästisches möglich ist. Durch die zusätzlichen Werk-

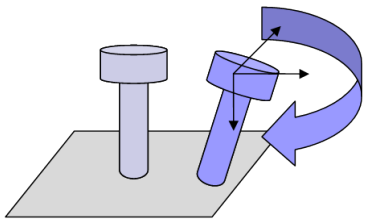


Abbildung 3.23: Bewegungsrichtungen des Fünf-Achs-Fräsen.

zeugachsen ergeben sich sowohl neue Bearbeitungsmöglichkeiten, als auch weitere Schwierigkeiten für die Fräsbahngenerierung. Durch die hinzugekommenen Winkeleinstellungen gegenüber dem Drei-Achs-Fräsen ergeben sich vielfältigere Möglichkeiten auf die Eingriffsrichtung Einfluss zu nehmen.

Es kann durch eine Änderung der Anstellung die benötigte Fräskraft positiv beeinflusst werden. Sehr vorteilhaft ist die Möglichkeit, abhängig von der maximalen Anstellung des Fräskopfes, Hinterschnitte zu erzeugen. Damit können, im Gegensatz zum Drei-Achs-Fräsen, verschiedene Aufspannungen unnötig werden, wodurch sowohl die Fehleranfälligkeit als auch die Prozessdauer sinken. Als vorteilhaft für die Oberflächengüte erweist sich die Möglichkeit, kürzere Werkzeuge verwenden zu können, die beim Drei-Achs-Fräsen zu Kollisionen führen, im Fünf-Achs-Prozess durch eine Veränderung des Anstellwinkels des Fräsers jedoch kollisionsfrei eingesetzt werden können. Durch kurze Werkzeuge sinkt die Schwingungsanfälligkeit des Fräsers und es kann mit höheren Geschwindigkeiten gearbeitet werden.

Bedingt durch die zusätzlichen Freiheitsgrade ergibt sich allerdings eine erheblich gestiegene Problematik der Werkzeugbahngenerierung. Bei der Kollisionsvermeidung muss jetzt zusätzlich die Möglichkeit der Winkeleinstellung des Fräsers beachtet werden. Zusätzlich sollte die Fräsbahn frei von zu abrupten Winkel- oder Richtungsänderungen sein, da solche Bewegungen des Fräsers häufig Marken auf der Werkstückoberfläche hinterlassen.

3.2.3 Optimierproblem

Optimierziele

Die Optimierung einer Fräsbahn setzt sich aus mehreren, teils gegensätzlich wirkenden Kriterien und Restriktionen zusammen und ist damit multikriteriell. Zu den Restriktionen gehören die Beschränkungen der verwendeten Fräsmaschine hinsichtlich des erreichbaren Arbeitsraumes. Zusätzliche maschinenbedingte Einschränkungen sind zum Beispiel die maximale Spindeldrehzahl oder die Motorleistung. Eine weitere Restriktion ist die Kollisionsfreiheit der Bahn bezüglich Werkstück und Werkzeugschaft bzw. Werkzeughalter. Fräsbahnen, auf denen eine oder mehrere dieser Grenzen überschritten werden, sind nicht realisierbar.

Bei der Erzeugung der Fräsbahn müssen zusätzlich noch verschiedene Faktoren, die meist nicht gleichzeitig zu optimieren sind, berücksichtigt werden. Es muss also ein vernünftiger Kompromiss gefunden werden, in dem unter anderen folgende Kriterien einfließen:

- Oberflächengüte: Fehlerfreie Oberflächen sind durch optimale Eingriffsbedingungen sicherzustellen.

- Prozessdauer: Um wirtschaftlich zu arbeiten, ist eine hohe Fertigungsgeschwindigkeit erwünscht. Erreichbar ist dies durch eine Erhöhung der Vorschub- bzw. der Schnittgeschwindigkeit.

- Standzeit: Um eine hohe Lebensdauer des Fräsers zu erreichen, sollten die Kräfte möglichst gering sein, um den Verschleiß zu reduzieren. Dies kann durch Veränderung der Anstellungen oder des Eingriffs der Schneiden erreicht werden.

Zur Lösung des Optimierproblems wird die Möglichkeit der Simulation entweder zum Bewerten einer Lösung oder zum Finden von Alternativen herangezogen. An die Simulation werden verschiedene Anforderungen gestellt, damit diese sinnvoll eingesetzt werden kann. Sie muss ein für den Zweck hinreichend genaues Modell verwenden und eine ausreichend hohe Geschwindigkeit besitzen, um eine große Zahl an verschiedenen Lösungen in vertretbarer Zeit analysieren zu können. Mit Hinblick auf die erweiterten Möglichkeiten der Fünf-Achs-Frästechnik ergibt sich der Wunsch nach einem Verfahren, welches bereits existierende Drei-Achs-Fräsprogramme automatisch in ein optimiertes Fünf-Achs-Programm konvertiert.

Simulationsverfahren

Es existieren verschiedene Simulationsverfahren mit unterschiedlichen Schwerpunkten. Eines dieser Verfahren, beschrieben unter anderem in [MMS05] und [SZ05] sowie in [Zab05], ist auf eine effiziente Erkennung von Kollisionen ausgerichtet und setzt auf eine Kombination einer Modellierung des Werkzeuges mittels der CSG-Technik (=constructive solid geometry) und des Werkstückes als ein so genanntes „Dexelmodell“. Mit dieser Technik ist es zusätzlich möglich, die Veränderung der Winkelanstellungen auf der Fräsbahn auf Harmonie und Gleichmäßigkeit zu optimieren. Bei ausreichend genauer Modellierung sind auch Kräfte bestimmbar und davon ausgehend Verschleißsimulationen durchführbar. Für diese beiden Aufgaben können ebenfalls andere Simulationen wie z. B. die Finite Elemente Modellierung eingesetzt werden. Details hierzu werden beispielsweise in [Bat02] behandelt.

Simulation des Werkzeuges mit CSG Die Modellierung des Werkzeuges bedient sich einer Technik, die den CSG-Algorithmus zur Beschreibung von Körpern nutzt. Hierbei werden komplexe Geometrien aus einfachen Grundkörpern und ihre Verbindungen mit booleschen Operationen, wie Vereinigung und Schnitt, beschrieben. Für genauere Information zur CSG-Technik siehe z. B. [FDFH95]. Bedingt durch die Rotation des Werkzeuges kann der Fräser als rotationssymmetrischer Körper angenommen werden. Zur Modellierung kann der Fräser samt Halter deshalb als Vereinigung von ebenfalls rotationssymmetrischen Volumina wie zum Beispiel Kugeln, Zylindern oder Kegeln zusammengesetzt werden. In Abbildung 3.24 wird die Erzeugung eines Fräsers schematisch dargestellt.

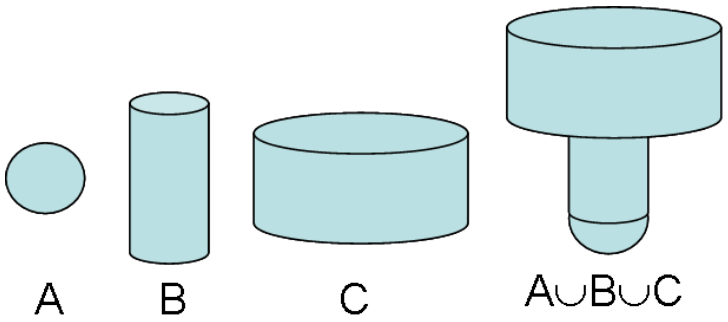


Abbildung 3.24: Schematische Erzeugung eines Fräsers mittels CSG.

Der Vorteil dieses Vorgehens ist die einfache Bestimmung von Kollisionen des Gesamtmodells. Dazu muss eine Kollisionsbestimmung für jedes Einzel-

volumen durchgeführt werden. Dies ist für die verwendeten Grundkörper einfach und effizient implementierbar. Ein großer Nachteil besteht darin, dass das Modell immer nur einer Vereinfachung entspricht und die mit dem Modell gefunden Parameter, wie bei allen Simulationstechniken, experimentell angepasst werden müssen.

Simulation des Werkstückes mit Dexeln Für die Simulation des Werkstücks wird eine Technik verwendet, die auf Grundlage von „Dexeln“, auch Nägel genannt, arbeitet. Das Verfahren ist, wie der Name andeutet, verwandt mit der Voxel-Technik. Allerdings werden hierbei die Volumina nicht durch Punktwolken approximiert, sondern durch Strecken. Das Dexel-Verfahren wurde ursprünglich nur für die Simulation von Drei-Achs-Fräsen benutzt, mit verschiedenen Erweiterungen ist es auch für die Fünf-Achs-Simulationen hinreichend. In Abbildung 3.25 wird schematisch eine Kugel mit der Dexel-Technik approximiert. Dazu wird das zu modellierende Volumen, in diesem Fall eine Kugel, mit Dexeln näherungsweise nachgebildet. Die Vorlage wird durch einzelne Strecken auf einem gewählten Raster nachgeahmt. Durch die Abstände der Dexel auf dem Raster kommt es zu Ungenauigkeiten zwischen Dexelmodell und Vorlage. Im Beispiel der Kugel beschreibt das Dexelmodell in der z-Achse (rot) die Kugel nur unzureichend. Die *„seitlichen“* Kugelschichten werden nicht ausreichend genau abgebildet. Um eine ausreichend hohe Genauigkeit auch für eine Bearbeitung aus mehreren Richtungen zu gewährleisten, werden deshalb drei einzelne Dexelmodelle gebildet, in denen die Dexel jeweils zu einer Raumachse parallel verlaufen. Für das Gesamtmodell werden alle Einzelmodelle kombiniert. Der Materialabtrag durch den Fräser wird simuliert, indem zuerst das Volumen bestimmt wird in dem der Fräskopf mit dem Dexelmodell kollidiert, um anschließend die betroffenen Dexel um den simulierten Abtrag zu verkürzen. Für die Simulation muss es zusätzlich möglich sein, einen Dexel zu unterbrechen und als zwei unabhängige Dexel weiter zu verarbeiten. Mit dieser Technik kann auch die Möglichkeit des Hinterschnittes der Fünf-Achs-Bearbeitung simuliert werden.

Vorteile des Simulationsverfahrens Durch die Kombination der beiden beschriebenen Modelle entsteht eine einfache und schnelle Simulationstechnik. Um eine Kollision zu erkennen, reicht es, einen Schnitttest des Halters und des Frässchaftes mit den ihn umgebenden Dexeln vorzunehmen. Dazu muss ein Schnitttest von einer Strecke, gegeben durch Startpunkt $\overline{p}$ und einer Länge $\overline{d}$, sowie den geometrischen Grundkörpern des CSG-Modells durchgeführt werden. Mathematisch kann die Strecke also durch eine Gleichung

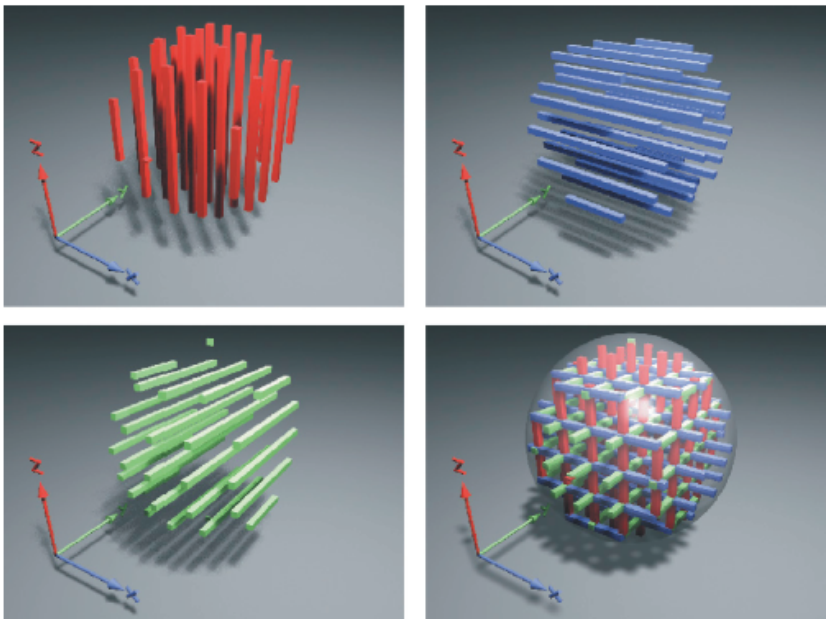


Abbildung 3.25: Modellierung einer Kugel mit Dexeln. [Zab05]

$\overline{q}(t) = \overline{p} + t\overline{d}$ mit $t \in \{0..1\}$ beschrieben werden. Es muss kein vollständiger Richtungsvektor für die Strecke gespeichert werden, sondern nur die Länge. Der benötigte Richtungsvektor wird implizit dadurch festgelegt, zu welcher Raumachse die Dexel parallel verlaufen. Ist ein Schnitttest erfolgreich - es liegt also eine Kollision vor - kann auch das kollidierende Volumen bestimmt werden. Davon ausgehend kann bestimmt werden, ob es eine schwere Kollision oder nur eine leichte Berührung ist. Im letzteren Fall kann häufig durch eine geringe Änderung der Anstellung des Fräsers eine realisierbare Fräsbahn erzeugt werden.

Durch die Modellierung kann das Verfahren in diskrete Einzelschritte unterteilt werden. Nach jedem Schritt liegt ein Zwischenergebnis vor, bei dem das Werkstückvolumen durch die Differenz von altem Dexelmodell und der Fräsbahn bestimmt wird. Durch die Vereinigung dieser beiden Volumina ergibt sich das tatsächlich in diesem Schritt herausgeschnittene Material. Dessen Volumen kann zur Bestimmung der Fräskräfte herangezogen werden. Durch die diskrete Berechnung kann die Kollisionsbestimmung auch zeitliche Veränderungen des Werkstückes berücksichtigen.

Simulationsergebnisse Aus der Simulationstechnik können zusätzlich zur Kollisionserkennung und der Kraftanalyse auch die Gleichmäßigkeit der Bewegung ermittelt werden. Optimal ist eine Bahn, die so wenige Richtungs- und Winkeländerungen wie nötig aufweist. Als Hilfsmittel zur Bewertung kann die Stetigkeit der ersten und zweiten Ableitung der Veränderungen der Winkelanstellung untersucht werden.

Mit Hinblick auf die so gewonnenen Daten ist es möglich, einen evolutionären Algorithmus zu entwickeln, der versucht durch eine Veränderung der Winkelanstellung des Fräsers eine bessere Bahn zu finden. Besondere Bedeutung hat dieses Verfahren, da es damit möglich ist, ein Drei-Achs-Programm automatisch in ein Fünf-Achs-Programm zu konvertieren.

3.2.4 Fazit

Als Ausgangsmaterial für die Darstellung der Optimierkritieren als Paretofront dienen primär die Positionsdaten und Winkelanstellung der NC-Bahn für die Fräsmaschine, welche von den Optimierprogrammen geliefert werden. Zusätzlich können diese noch um Daten ergänzt werden, die aus Bildern und Graphiken gewonnen werden.

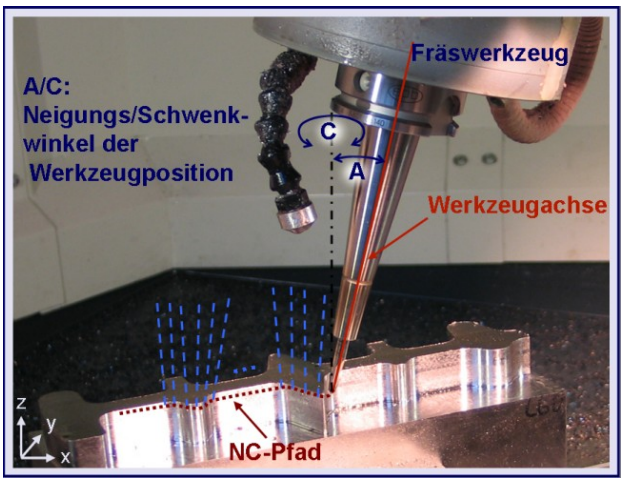


Abbildung 3.26: Darstellung der Position und Winkelanstellung für eine Fräsmaschine.

Die Zielfunktionen werden aus diesen Daten berechnet. Zu diesen Zielfunktionen gehören beispielsweise die Harmonie der Bewegung des Werkzeuges im Verlauf des Fräsprozesses oder die durchschnittliche und maximale Fräskraft.

4 Optimierung

4.1 Evolutionäre Optimierung

Es ist i. A. sehr schwierig, eine genaue Lösung für die Anwendungsfälle, die in den Kapiteln 3.1 und 3.2 beschrieben wurden, zu finden. In der Praxis werden oft Optimierungsmethoden eingesetzt, die nur eine gute Approximation der Lösung finden. Es wird auch erlaubt, dass die Optimierungsmethoden in sehr seltenen Fällen versagen. Die evolutionäre Optimierung ist eine solche Methode.

Das Ziel der Optimierung ist die Suche eines *lokalen Minimums* oder eines *globalen Minimums*. Das lokale Minimum und das globale Minimum werden wie folgt definiert:

Definition 4.1 (lokales und globales Minimum) *Gegeben sei eine Funktion* $F(\vec{x}) : \mathbb{R}^n \longrightarrow \mathbb{R}$ *mit den Restriktionen* $G_i(\vec{x}) > 0$, $i \in \{1, \ldots, q\}$, *gesucht werden:*

- das lokale Minimum $\vec{x}^*$*:*
  $\exists \epsilon \in \mathbb{R}^+$, $\forall \vec{x} \in \mathbb{R}^n$ *mit* $G_i(\vec{x}) > 0$*:* $\quad |\vec{x} - \vec{x}^*| < \epsilon \Rightarrow F(\vec{x}^*) < F(\vec{x})$
- das globale Minimum $\vec{x}^{**}$*:*
  $\forall \vec{x} \in \mathbb{R}^n$ *mit* $G_i(\vec{x}) > 0$ *:* $\quad F(\vec{x}^{**}) \leq F(\vec{x})$

Die Restriktionen $G_i(\vec{x})$ sind die Funktionen, die die ungeeignete $\vec{x}$ eliminieren. Sie werden so definiert, dass es für alle ungeeignete $\vec{x}$ stets $G_i(\vec{x}) \leqslant 0$ gilt.

In der Abbildung 4.1 auf der nächsten Seite sind x_1^* und x_2^* die lokalen Minima und x_2^* gleichzeitig auch das globale Minimum für eine eindimensionale Funktion $F(x)$.
Die evolutionäre Optimierung simuliert die biologische Evolution in der natürlichen Welt. Die biologische Evolution basiert auf der Darwinschen Evolutionstheorie [Wik], welche auf den folgenden drei Prinzipien beruht:

- Vererbung: Die Individuen besitzen eine bestimmte Menge von Einheiten. Solche Einheiten heißen auch Replikatoren. In den Replikatoren werden alle Eigenschaften der Individuen gespeichert. Die Replikatoren müssen fähig sein, Kopien von sich selbst anzufertigen oder andere

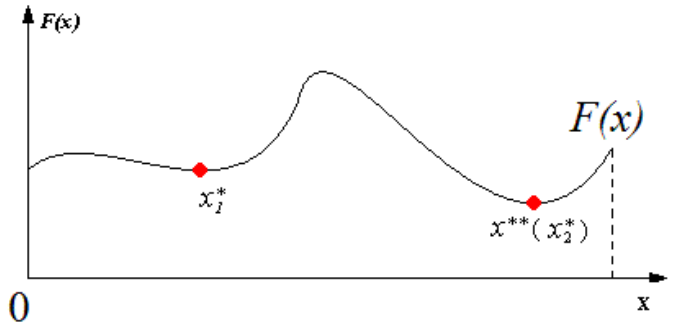


Abbildung 4.1: Ein lokales Minimum x_1^* und ein globales Minimum x^{**}

Einheiten zu veranlassen, entsprechende Kopien zu erzeugen. Die Nachkommen kopieren alle Replikatoren stochastisch von ihren Eltern und erben dadurch die Replikatoren ihrer Eltern.

- Mutation: Bei der Rekombination der Replikatoren können Fehler entstehen, wie z. B. ungenaue Replikation usw. Diese Fehler führen dazu, dass die Nachkommen selbst neue Replikatoren „erfinden“ können.
- Selektion: Vererbte Merkmale müssen die Rekombinationsfähigkeit der Einheiten beeinflussen, entweder durch Überlebensfähigkeit (natürliche Selektion) oder die Fähigkeit, für die Rekombination notwendige Partner zu finden (sexuelle Selektion).

In der evolutionären Optimierung entwickeln sich die Lösungen durch die Simulation von Vererbung, Mutation und Selektion. Die Lösungen können, wie bei der biologischen Evolution in der natürlichen Welt, sich selbst verbessert werden. Es wird gehofft, dass die Lösungen durch die evolutionäre Optimierung in endlicher Zeit verbessert werden können.

Vier berühmte Algorithmen zur evolutionären Optimierung sind (für weitere Information siehe [Fog95]):

- Evolutionäres Programmieren (Fogel, 1962-1966)
- Evolutionsstrategie (Ingo Rechenberg und Hans-Paul Schwefel, 1965)
- Genetischer Algorithmus (John H. Holland, 1975)
- Genetisches Programmieren (John Koza, 1992)

In dieser Ausarbeitung wird Evolutionsstrategie vorgestellt.

4.1.1 Evolutionsstrategie

Die Evolutionsstrategie besteht aus 4 Schritten:

- Initialisierung
- Rekombination
- Mutation
- Selektion

In der Evolutionsstrategie werden zuerst zufällige Daten (allgemein auch Individuen genannt) erzeugt. Diese Daten werden später durch unterschiedliche Methoden rekombiniert und dadurch werden neue Nachkommen erzeugt. Danach werden die neuen Nachkommen zufällig variiert, wodurch eine Mutation simuliert wird. Die besten Individuen werden dann selektiert. Solche besten Individuen bilden zusammen die nächste *Generation* und werden weiter rekombiniert. Dieses Verfahren wird wiederholt, bis eine Abbruchbedingung erfüllt ist.

In [Sch95] wird der Algorithmus der Evolutionsstrategie skizziert. In [Bäc94] und [Sch95] gibt es auch die Pseudo-Programmcodes der Evolutionsstrategie.

Initialisierung

Die Initialisierung ist der erste Schritt der Evolutionsstrategie. In diesem Schritt werden μ zufällige Individuen generiert.

$$\begin{aligned}
\vec{\alpha}_i^{(0)} &= (\vec{x}_i^{(0)}, \vec{\sigma}_i^{(0)}) && i \in \{1, \ldots, \mu\} \\
\vec{x}_k^{(0)} &= \begin{pmatrix} x_{k,1}^{(0)} & x_{k,2}^{(0)} & \cdots & x_{k,n}^{(0)} \end{pmatrix}^T && G_j(\vec{x}_k^{(0)}) > 0, k \in \{1, \ldots, \mu\}, j \in \{1, \ldots, q\} \\
\vec{\sigma}_k^{(0)} &= \begin{pmatrix} \sigma_{k,1}^{(0)} & \sigma_{k,2}^{(0)} & \cdots & \sigma_{k,n}^{(0)} \end{pmatrix}^T && k \in \{1, \ldots, \mu\}
\end{aligned}$$

Ein Individuum besteht aus dem Vektor $\vec{x}$ und dem Vektor $\vec{\sigma}$. Der Vektor $\vec{x}$ stellt eine mögliche Lösung eines Suchprozesses dar. Der Vektor $\vec{\sigma}$ ist ein Hilfsvektor in der Mutationsphase und wird nur in dieser Phase verwendet (siehe auch 4.1.1). Die zwei Vektoren haben die gleiche Dimensionszahl.

Rekombination

In diesem Schritt werden die μ Individuen rekombiniert und es werden λ Nachkommen erzeugt. In der Praxis entspricht λ etwa dem Fünf- bis

Zehnfachen von μ, damit eine große Menge von Individuen für die spätere Selektion (siehe Abschnitt 4.1.1) erzeugt werden kann. Die Nachkommen können durch die folgenden Methoden erzeugt werden:

- Asexuell
  - Keine Rekombination: $\vec{x'}_i^{(g)} = x_{S,i}^{(g)}$
- Sexuell
  - Diskret: $\vec{x'}_i^{(g)} = \vec{x}_{S,i}^{(g)}$ oder $\vec{x}_{T,i}^{(g)}$
  - Zwischenprodukt: $\vec{x'}_i^{(g)} = (\vec{x}_{S,i}^{(g)} + \vec{x}_{T,i}^{(g)})/2$
  - Allgemeines Zwischenprodukt: $\vec{x'}_i^{(g)} = \vec{x}_{S,i}^{(g)} + \chi \cdot (\vec{x}_{S,i}^{(g)} + \vec{x}_{T,i}^{(g)})\chi \in \mathbb{R}$

$x_{S,i}^{(g)}$, $x_{T,i}^{(g)}$ sind zufällige Individuen in der g-ten Generation. Die Vektor $\vec{\sigma}'$ der Individuen werden durch die gleichen Formeln erzeugt.

Bei den asexuellen Methoden entsprechen die Nachkommen einfach einem zufällig ausgewählten Individuum der vorherigen Generation. Die Nachkommen werden später noch mutiert. Deswegen führen die asexuellen Methoden nicht dazu, dass die Individuen in der nächsten Generation den Individuen in der vorherigen Generation entsprechen.

Bei den sexuellen Methoden werden die Nachkommen durch die Rekombination zweier Individuen der vorherigen Generation erzeugt. Wie bei den asexuellen Methoden, werden zwei Individuen zufällig ausgewählt. Alle Individuen haben die gleiche Chance ausgewählt zu werden.

Jetzt gibt es insgesamt $\mu + \lambda$ Individuen. μ Individuen davon sind die Individuen aus der vorherigen Generation, und die anderen λ Individuen sind die in diesem Schritt erzeugten neuen Nachkommen.

Mutation

Im Mutationsschritt werden λ neue Nachkommen zufällig variiert:

$$\vec{x''}_l^{(g)} = \vec{x'}_l^{(g)} + \vec{z} \tag{4.1}$$

mit $G(\vec{x''}_l^{(g)}) > 0$, $l \in \{\mu + 1, \ldots, \mu + \lambda\}$, $\vec{z} = \vec{\sigma} \cdot N(0, 1)$ ein n-dimensional normalverteilter Zufallsvektor.
Die Abweichung sollte in den meisten Fällen nicht zu groß sein. Gleichzeitig soll aber auch die Möglichkeit bestehen, dass sie groß sein kann. Deswegen wird hier die Zufallszahl mit Normalverteilung $\sigma \cdot N(0, 1)$ verwendet.

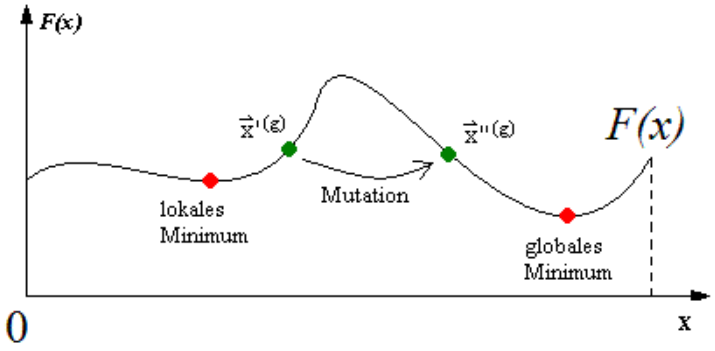


Abbildung 4.2: Mutation

Die Mutation dient dazu, dass Nachkommen nicht nur gegen das lokale Optimum konvergieren können, sondern auch in andere „Wellentäler“ springen können (siehe Abbildung 4.2). Das ist sinnvoll, weil das lokale Optimum in einem anderen Wellental besser sein kann. Der Vektor σ sollte nicht zu klein sein, da sonst die Zufallszahl mit der Normalverteilung $\sigma \cdot N(0,1)$ wahrscheinlich nicht groß genug wäre, als dass der mutierte Nachkomme damit in ein anderes Wellental springen kann. Bei einem zu kleinen σ besteht zwar immer die Möglichkeit, dass die Abweichung irgendwann groß genug ist, aber das zu kleine σ wird dazu führen, dass zu viele Mutationen gebraucht werden und die Optimierung zu lange dauert.

Andererseits bedeutet es auch nicht, dass ein großes σ immer gut ist. Ein zu großes σ führt auch zu Problemen. Falls das σ zu groß wäre, würde die Zufallszahl mit der Normalverteilung $\sigma \cdot N(0,1)$ häufig groß, und es führt dazu, dass der mutierte Nachkomme über das lokale Optimum springt. Das heißt, dass es lange dauern wird, bis der mutierte Nachkomme gegen das Optimum konvergiert.

Deswegen wird ein „passendes“ σ benötigt. Aber das Problem ist, dass i. A. schlecht vorhergesagt werden kann, wie groß das σ sein sollte. In unterschiedlichen Problemen und unterschiedlichen Generationen eines Problems kann das passende σ sehr unterschiedlich sein.

In der Evolutionsstrategie wird die Schrittweitenselbstadaptation verwendet. Ein Individuum besteht aus dem Vektor $\vec{x}$ und dem Vektor $\vec{\sigma}$. Der Letztere wird für die Schrittweitenselbstadaptation eingesetzt.

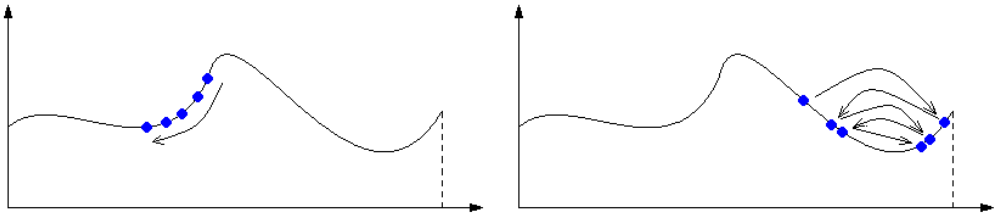


Abbildung 4.3: Zu kleines $\vec{\sigma}$ (links) und zu großes $\vec{\sigma}$ (rechts)

Zuerst wird der neue Vektor $\vec{\sigma'}$ für jeden Nachkommen nach der folgenden Formel berechnet, danach werden die Nachkommen nach dem jeweiligen Vektor $\vec{\sigma'}$ variiert:

$$\vec{\sigma'} = \vec{\sigma} \cdot exp(\tau_0 \cdot N(0,1)) \tag{4.2}$$

$$\vec{x''}_i^{(g)} = \vec{x'}_i^{(g)} + \vec{\sigma'} \cdot N(0,1) \tag{4.3}$$

τ_0 ist eine Erfahrungswert, und entspricht oft $\frac{1}{\sqrt{n}}$, wobei n die Dimensionszahl des Suchraumes ist.

Der neue Vektor $\vec{\sigma'}$ ist eine zufällige Abweichung vom alten $\vec{\sigma}$. Hier wird wieder eine Zufallszahl mit der Normalverteilung verwendet.

Die Idee der Schrittweitenselbstadaptation ist, dass die besten $\vec{\sigma}$ zusammen mit den Nachkommen selektiert werden können. Mit einem passenden $\vec{\sigma'}$ kann ein Nachkomme oft gut mutiert werden. Das heißt, dass der mutierte Nachkomme eine höhere Chance hat, später selektiert zu werden. Falls der mutierte Nachkomme später selektiert wird, wird das passende $\vec{\sigma'}$ zusammen mit diesem Nachkommen vererbt. Falls andererseits der Vektor $\vec{\sigma'}$ zu groß oder zu klein ist, kann der Nachkomme kaum gut mutiert werden. Das heißt, dass der mutierte Nachkomme kaum Chancen hat, später selektiert zu werden. Falls der mutierte Nachkomme bei der Selektion nicht ausgewählt wird, wird das $\vec{\sigma'}$ zusammen mit diesem Nachkommen gelöscht. In der Schrittweitenselbstadaptation kann zwar immer noch nicht vorhergesagt werden, wie groß der Vektor $\vec{\sigma}$ sein sollte, aber die Vektoren $\vec{\sigma}$ können sich selbst adaptieren.

Selektion

In diesem Schritt werden die besten Individuen selektiert. Nach den Selektionsmethoden kann die Evolutionsstrategie in (μ+λ)-*Evolutionsstrategie* und (μ,λ)-*Evolutionsstrategie* unterschieden werden.

In der (μ+λ)-Evolutionsstrategie werden die (μ+λ) Individuen (μ Individuen aus der vorherigen Generation und λ mutierte Nachkommen) sortiert, sodass:

$$F(\vec{x''}_l^{(g)}) \leq F(\vec{x''}_{l+1}^{(g)}) \qquad i \in \{1, \ldots, \mu + \lambda - 1\} \tag{4.4}$$

Danach werden die μ besten Individuen selektiert.

$$\vec{x}_k^{(g+1)} = \vec{x''}_k^{(g)} \qquad k \in \{1, \ldots, \mu\} \tag{4.5}$$

Dagegen werden in der (μ,λ)-Evolutionsstrategie nur die λ mutierten Nachkommen sortiert, sodass:

$$F(\vec{x''}_l^{(g)}) \leq F(\vec{x''}_{l+1}^{(g)}) \qquad i \in \{\mu + 1, \ldots, \mu + \lambda - 1\} \tag{4.6}$$

Danach werden die μ besten Individuen selektiert.

$$\vec{x}_k^{(g+1)} = \vec{x''}_{k+\mu}^{(g)} \qquad k \in \{1, \ldots, \mu\} \tag{4.7}$$

Die beiden Methoden haben Vorteile und Nachteile. In der (μ+λ)-Evolutionsstrategie können die Individuen schneller gegen das (lokale) Optimum konvergieren. Die Individuen sind auf jeden Fall besser als die Individuen in der vorherigen Generation. Aber die Individuen können sehr schwierig zu anderen Wellentälern springen, weil die zufällig variierten Nachfolger meistens nicht so gut sind wie die Individuen aus der vorherigen Generation und deswegen eliminiert werden. Die (μ+λ)-Evolutionsstrategie ist für die Selektion des lokalen Optimums geeignet, aber es ist i. A. sehr schwierig, damit das globale Optimum zu finden.

Dagegen ist die Selektion des globalen Optimums für die (μ+λ)-Evolutionsstrategie etwas leichter. Der Nachteil der (μ+λ)-Evolutionsstrategie ist, dass die Individuen nicht so leicht gegen das Optimum konvergieren.

Wiederholung und Abbruchkriterien

Die Schritte *Rekombination*, *Mutation* und *Selektion* werden wiederholt. Nach jeder Wiederholung wird eine neue Generation erzeugt. Falls eine oder mehrere der folgenden Abbruchbedingungen erfüllt sind, kann vermutet werden, dass das beste Individuum in der aktuellen Generation sich wahrscheinlich nah einem Optimum befindet. Der Vorgang wird deswegen abgebrochen.

- Die Anzahl der Wiederholungen ist groß genug.
- Das beste Individuum ändert sich bei einer bestimmten Anzahl von Wiederholungen nicht mehr.
- usw.

Am Ende wird das beste Individuum als Approximation an ein Optimum ausgegeben.

4.1.2 Fazit

Die evolutionäre Optimierung kann i. A. in einer unendlichen Zeit unter speziellen Voraussetzungen ein globales Optimum finden. Aber das Problem ist, dass weder bekannt ist, wie lange die Optimierung dauert, noch, ob das globale Optimum nach einer endlichen Zeit gefunden wird. Es ist auch unbekannt, ob das beste Individuum in der aktuellen Generation das globale Optimum ist. Des Weiteren ist unklar, wie gut das beste Individuum in der aktuellen Generation ist. Mit anderen Worten, wie nah das beste Individuum in der aktuellen Generation beim globalen Optimum liegt. Deswegen sollte der Optimierungsprozess möglichst lange laufen und die Abbruchbedingung nicht zu eng sein.

Die evolutionäre Optimierung wird oft für schwierige Probleme, für die keine guten Lösungsansätze bekannt sind, eingesetzt. Mit den traditionellen Optimierungsverfahren können solche Probleme eventuell nicht gut gelöst werden. Zudem bietet die evolutionäre Optimierung oft auch Lösungen, die nicht erwartet werden.

4.2 Mehrkriterielle evolutionäre Optimierung

Diese Ausarbeitung behandelt das Thema der mehrkriteriellen evolutionären Optimierung. Während in Kapitel 4.1 die grundlegenden Prinzipien der evolutionären Optimierung mittels Evolutionsstrategie (ES) und Evolutionären Algorithmen (EA) erklärt wurden, soll hier die Anwendung derselben auf Optimierprobleme mit mehreren, sich widersprechenden Zielfunktionen übertragen werden.
Zunächst wird die Thematik an einem Beispiel eingeführt, dann einige wichtige Begriffe definiert und anschließend auf Lösungsansätze eingegangen. Im letzten Teil wird ein konkreter evolutionärer Algorithmus zur mehrkriteriellen Optimierung vorgestellt.

4.2.1 Beispiel: Optimale Lenkradhaltung

Die Optimierung der Lenkradhaltung eines LKW-Fahrers ist ein anschauliches Beispiel für eine mehrkriterielle Optimierung mit konfliktionären Zielfunktionen. Es wird angenommen, dass der LKW-Fahrer eine Strecke mit gleich vielen Rechts- und Linkskurven zu fahren hat. Des Weiteren ist die rechte Hand nur für die rechte Hälfte des Lenkrades zuständig, die linke für die linke Hälfte. Muss das Lenkrad weiter gedreht werden, so ist ein Umgreifen bzw. Handwechsel nötig. Da das Problem symmetrisch ist, wird weiter nur die rechte Seite betrachtet.

Interessant wäre nun die Stelle des Lenkrades, an der die Kraftwirkung des Lenkens optimal ist. Dies ist offensichtlich die Stelle des größten Hebels, in unserem Beispiel die 90° bzw. 3 Uhr Stellung. Formal spielt hier eine Sinusfunktion die entscheidende Rolle. Sei α der Winkel der Handposition und r der Radius.

Dann gilt für das Moment M bzw. für die Zielfunktion F

$$M = M(\alpha) = r \cdot \sin \alpha \tag{4.8}$$

Der Definitionsbereich für α liegt nach den vorgegebenen Beschränkungen zwischen 0° und 180°. Die Formulierung als Optimierproblem sieht dann wie folgt aus:

$$\text{Maximiere } M(\alpha)$$

Als weiteres Kriterium dient die Tatsache, dass der LWK-Fahrer häufiges Umgreifen vermeiden möchte. Es stellt sich also die Frage, welche Stelle des Lenkrades minimales Umgreifen ermöglicht. Aufgrund der Symmetrie der beiden Lenkradseiten ist dies offensichtlich die Position ganz oben.

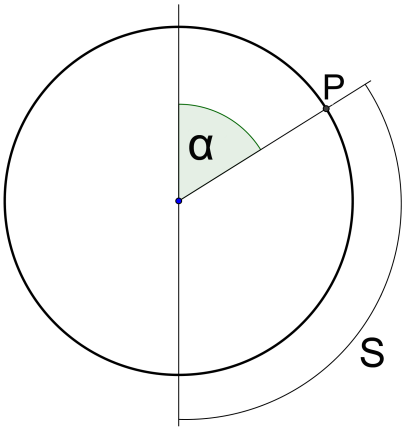


Abbildung 4.4: Abstraktion des LKW-Lenkrades

Um das Problem formal zu fassen, wird die Strecke S bis zum nächsten Umgreifen betrachtet. Abbildung 4.4 verdeutlicht dies. Die Funktion für die Streckenoptimierung sieht wie folgt aus:

$$S = S(\alpha) = r \cdot \frac{\pi - \alpha}{\pi} \tag{4.9}$$

Es ergibt sich also folgender Zielfunktionenvektor :

$$\vec{f} = \begin{pmatrix} M(\alpha) \\ S(\alpha) \end{pmatrix} = \begin{pmatrix} r \cdot \sin\alpha \\ r \cdot \frac{\pi-\alpha}{\pi} \end{pmatrix} \tag{4.10}$$

Es gibt offensichtlich einen Konflikt zwischen den beiden Funktionen. Wird die Lösung bezüglich des Moments verbessert, so wird die Lösung bezüglich des Umgreifens verschlechtert und umgekehrt. Es geht also darum, die Menge „der besten Kompromisse“ zu finden. Im folgenden Unterkapitel werden anhand eines weiteren Beispiels wichtige Begriffe der mehrkriteriellen Optimierung erläutert.

4.2.2 Formulierungen und Definitionen

In Unterkapitel 4.2.1 wurde in erster Linie anschaulich auf die Problematik eingegangen. Hier soll es nun darum gehen, die Problematik zu formalisieren, um anschließend auf die Definitionen der für die Projektgruppe wichtigen Begriffe wie *Paretomenge* und *Paretofront* zu kommen.

Formulierung des Optimierproblems

Ein mehrkriterielles Optimierproblem lässt sich in folgender Art beschreiben:

$$\min_{x \in D} \quad \{f_1(x), f_2(x), .., f_n(x)\}$$

D ist dabei die Menge der zugelassenen Parameter unter Berücksichtigung aller festgelegten Beschränkungen, z. B. $D = \{\alpha \mid 0 \leq \alpha \leq 180\}$ in unserem Lenkradbeispiel.

Es ist eine einheitliche Beschreibung der Art der Optimierung für alle Zielfunktionen zu beachten. Entweder werden alle Funktionen als Minimierungs- oder als Maximierungsproblem beschrieben. Würden beide Schreibweisen vermischt, so würden sich in der Ergebnisdarstellung beispielsweise einzelne Diagrammachsen umdrehen. Dies würde die ohnehin schon schwierige Interpretation der Ergebnisse weiter erschweren.

Ein Maximierungsproblem wird zu einem entsprechenden Minimierungsproblem, wenn die jeweilige Funktion mit -1 multipliziert wird. Je nach Belieben können vor der Optimierung noch weitere Transformationen durchgeführt werden, die das Ergebnis nicht beeinflussen, aber die Rechnung und Darstellung verbessern.

Räume und Mengen

Für die Einführung der Begriffe *Paretomenge* und *Paretofront* ist es notwendig, einen guten Überblick über die zu betrachtenden Räume zu bekommen. Abbildung 4.5 stellt diese im zweidimensionalen Fall graphisch dar.

Zunächst wird zwischen dem sogenannten *Parameterraum* **X** (engl. *decision space*) und dem *Zielfunktionsraum* **Z** (engl. *objective space*) unterschieden. Der Parameterraum ist der Raum, der alle Parametereinstellungen für das Optimierproblem enthält. Im Falle eines Benzinmotors wären einige dieser Parameter die Ventildeckeleinstellung, Einspritzzeiten, Verhältnis von Luft und Kraftstoff, Oktanzahl des Kraftstoffes, Art der Kühlung usw. Jede Konfiguration stellt einen Vektor in X dar.

Natürlich macht es keinen Sinn, alle Konfigurationen zu betrachten. So gibt es Beschränkungen für Parameter. Ein Luft-Kraftstoffverhältis von 1:0 wäre z. B. unsinnig, ebenso ein Kraftstoff von 0 Oktan. Es wird also eine Teilmenge $\mathbf{D} \subseteq \mathbf{X}$ definiert, für die es unter Berücksichtigung aller Beschränkungen eine Abbildung in den Zielraum **Z** gibt. **D** wird auch als *gültige Menge* bezeichnet (engl. *feasible region*, *feasible solutions*).

Der Zielraum **Z** ist der Raum, in den die Vektoren des Parameterraums mittels der so genannten „Zielfunktion“ abgebildet werden. In unserem Ben-

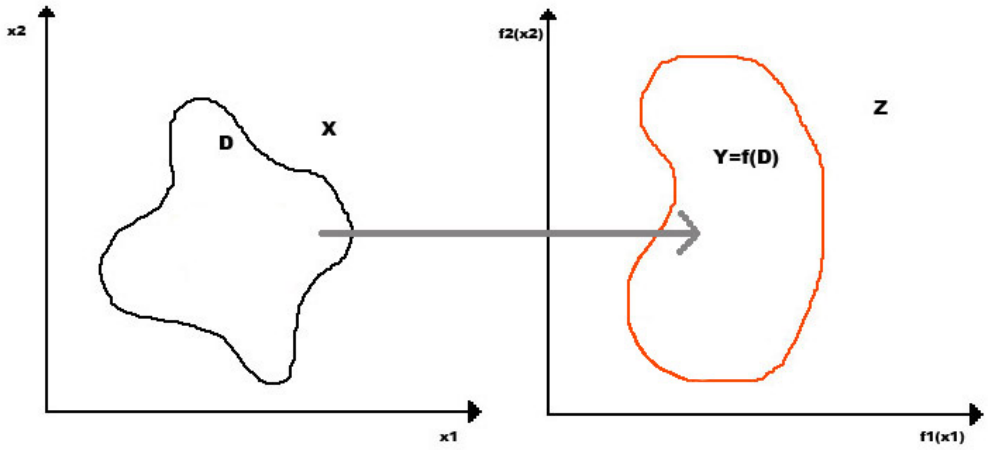


Abbildung 4.5: Parameter- und Zielraum

zinmotorbeispiel können dies Vektoren der Dimensionen Kraftstoffverbrauch, Motorverschleiß, Leistung oder Lautstärke sein.

Auch dieser Raum wird weiter eingeschränkt. Der Raum $\mathbf{Y} \subseteq \mathbb{Z}$ wird als der Raum definiert, für dessen Vektoren die Abbildung $\mathbf{D} \longrightarrow \mathbb{Z}$ existiert, d. h. $\mathbf{Y}$ stellt den Raum dar, der die Ergebnisse der sinnvollen Parametereinstellungen enthält.

Paretomenge und Paretofront Der Unterschied zwischen der Paretomenge und der Paretofront sind die Räume, auf denen sie definiert sind. Während die Paretomenge eine Teilmenge $\mathbf{X_{Par}} \subseteq \mathbf{D}$ ist (Parameterraum), so ist die Paretofront die Teilmenge $\mathbf{Z_{Par}} \subseteq \mathbf{Y}$ (Zielraum). Der Name „Pareto“ stammt von Vilfredo Frederico Pareto (1848-1923), einem italienischen Ingenieur, Ökonom und Soziologen.

Es ist klar, dass nicht gleichzeitig höchste Motorleistung bei vernachlässigbarem Kraftstoffverbrauch erzeugt werden kann. Dies ist der oben erwähnte Konflikt zwischen Zielfunktionen. Es ist also hier einen Kompromiss zwischen Leistung und Verbrauch zu finden. Die Tatsache, dass die Menge der Kompromisse nicht auf einen Einzigen beschränkt ist, ist leicht einzusehen. Es handelt sich vielmehr um eine „Menge“ von Kompromissen im Zielraum, die sog. „Paretofront“, $\mathbf{Z_{Par}}$.

Die Menge der Paretofront ist nicht zu verwechseln mit der Menge ihrer „Ursprünge“ im Parameterraum, der Paretomenge. Ein Vektor dieser Menge stellt eine Motorkonfiguration dar, welche einen besten Kompromiss im Zielraum erzeugt. Abbildung 4.6 auf der nächsten Seite ergänzt die Mengen um die Paretomenge bzw. Paretofront.

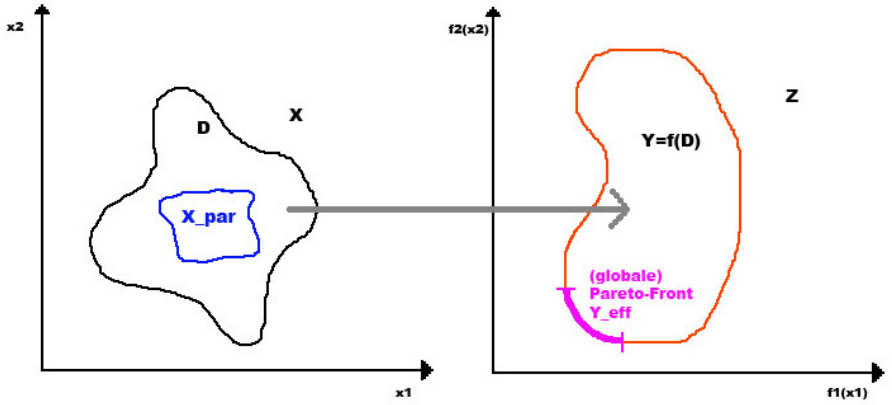


Abbildung 4.6: Paretomenge und Paretofront

Lösungen und deren Struktur

Die Lösungen eines mehrkriteriellen Optimierproblems sind mehrdimensional und somit Vektoren. Daraus folgt direkt das Problem der Vergleichbarkeit von Vektoren . Können bei eindimensionalen Lösungen in $\mathbb{R}$ noch einfache $\leq$ oder $\geq$-Relationen benutzt werden, so erfordert der mehrdimensionale Fall eine komplexere Betrachtung. Als Beispiel seien folgende Vektoren ausgewählt:

$$\vec{a} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \qquad \vec{b} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \qquad \vec{c} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \qquad \vec{d} = \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}$$

Diese Vektoren können nicht ohne Weiteres verglichen werden. Eine Lösung stellt die Einführung von Relationen und Ordnungen da.

Relationen zwischen Vektoren Folgende Ordnungsrelationen bieten Kriterien, mit denen Vektoren verglichen werden können :

Definition 4.2 (Ordnungen auf Vektoren) *Seien $\vec{x}, \vec{y} \in \mathbb{R}^n$ beliebige Vektoren gleicher Dimension, dann sind diese Ordnungen definiert:*

- *schwach komponentenweise:*
  $\vec{x} \leq \vec{y}$ *falls* $\forall i \in \{1, ..., n\} : x_i \leq y_i$
- *komponentenweise:*
  $\vec{x} < \vec{y}$ *falls* $\forall i \in \{1, ..., n\} : x_i \leq y_i, \vec{x} \neq \vec{y}$

- *strikt komponentenweise:*
  $\vec{x} \ll \vec{y} \quad falls\ \forall i \in \{1, ..., n\} : x_i < y_i$
- *lexikographisch:*
  $\vec{x} \leq_{lex} \vec{y} \quad falls\ x_k < y_k, k := \ min_{i:x_i \neq y_i}\ oder\ \vec{x} = \vec{y}$
- *Max-Ordnung:*
  $\vec{x} \leq_{MO} \vec{y} \quad falls\ max_{i=1,...,n}\ x_i \leq max_{i=1,...,n}\ y_i$

Im Beispiel gilt also $\vec{a} \leq \vec{b}$ (schwach komponentenweise Ordnung). Da die Vektoren $\vec{a}$ und $\vec{b}$ nicht identisch sind, gilt sogar $\vec{a} < \vec{b}$, ebenfalls $\vec{c} < \vec{b}$. Die Vektoren $\vec{a}, \vec{b}$ und $\vec{c}$ stehen in strikt komponentenweiser Ordnung unter $\vec{d}$, so ist z. B. $\vec{c} << \vec{d}$. Die lexikographische Ordnung und die Max-Ordnung finden Anwendung in einigen Optimierverfahren.

Sind zwei Vektoren entsprechend der zugrunde gelegten Ordnungen nicht vergleichbar, so werden sie „indifferent“ genannt. Dies ist bei den Vektoren $\vec{a}$ und $\vec{c}$ der Fall. Indifferenz wird mit der Tilde ausgedrückt, also $\vec{a} \sim \vec{c}$.

Dominanz In Abschnitt 4.2.2 wurden die Begriffe Paretomenge und Paretofront auf eine anschauliche Art und Weise eingeführt. Hier soll nun etwas detaillierter auf die mathematischen Eigenschaften sowie die mathematischen Voraussetzungen für die Paretofront eingegangen werden. Die „Dominanz“ ist ein wesentliches Konzept, welches die oben angesprochene Vergleichbarkeit von Vektoren ausnutzt und somit den Grundstein für die Paretofront legt.

Extrempunkte im Zielraum Um eine gute Anschauung zu wahren, seien an dieser Stelle einige Extrempunkte im Zielraum präsentiert. Abb. 4.7 auf der nächsten Seite zeigt die Punkte graphisch.

- **Idealer Vektor:** Der Ideale Vektor $\vec{z}^*$ ist durch die Optima der einzelnen Zielfunktionen bestimmt. In dem Motorbeispiel könnte das der Vektor $\begin{pmatrix} \text{bester Verbrauch} \\ \text{beste Motorleistung} \end{pmatrix}$ sein. Im allgemeinen Fall konfliktionärer Zielfunktionen kann es diesen Vektor nicht geben, da er eine optimale Lösung aller Zielfunktionen darstellt und keine Konflikte enthält. Der ideale Punkt kann, wenn er bekannt ist, als Referenzpunkt für die Beurteilung einer Paretofront genutzt werden.
- **Utopischer Vektor:** Der utopische Vektor $\vec{z}^{**}$ stellt eine untere Grenze für alle Zielfunktionen dar. Manche Algorithmen verlangen eine echt bessere Lösung einer Komponente. Um dies zu erreichen, wird eine

ϵ-Umgebung in Verbesserungsrichtung für jeden Zielfunktionenwert eingeführt. Der utopische Punkt ist ebenfalls eine nicht existierende Lösung.

- **Nadir-Vektor:** Der Nadir-Vektor $\vec{z}_{Nad}$ stellt, im Gegensatz zum idealen Vektor, eine obere Schranke für die einzelnen Zielfunktionen dar. Er bezieht sich *nur* auf die Ausdehnung der Paretofront und stellt nicht den Punkt der „schlimmsten“ *aller* Lösungen dar.

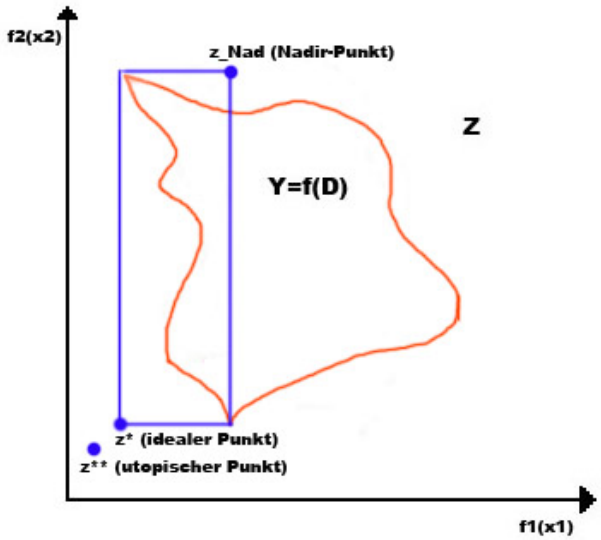


Abbildung 4.7: Idealer, utopischer und Nadir-Punkt

Dominanzkonzept Wie oben verdeutlicht, müssen bei der mehrkriteriellen Optimierung Vektoren verglichen werden. Wann ist ein Vektor „besser“ oder „schlechter“ als ein anderer? Während einige Algorithmen auf die lexikographische oder Max-Ordnung zurückgreifen, wird am häufigsten das von Vilfredo Pareto definierte Pareto-Dominanzkonzept verwendet. Dieses lautet wie folgt:

Ein Vektor $\vec{x}_1$ dominiert einen Vektor $\vec{x}_2$ (geschrieben $\vec{x}_1 \leq \vec{x}_2$) wenn:

- $\vec{x}_1$ ist bezüglich aller Ziele mindestens so gut wie Vektor $\vec{x}_2$ und
- $\vec{x}_1$ ist klar besser als $\vec{x}_2$ in mindestens einem Ziel.

Gilt die Dominanz im Parameterraum, so gilt sie auch für die Bilder im Zielraum.

Diese Definition ermöglicht, eine Ordnung auf den Lösungsvektoren festzustellen. Diese Ordnung ist eine strikte Halbordnung, da die Ordnungsrelation nach der Definition zwar transitiv, aber weder reflexiv noch symmetrisch ist. Diese Tatsache führt direkt auf die Pareto-Optimalität.

Das dargestellte Dominanzkonzept wird auch als „schwache“ Pareto-Dominanz bezeichnet. Bei der so genannten „starken“ Pareto-Dominanz muss ein Vektor $\vec{x}_1$ bezüglich aller Ziele klar besser als $\vec{x}_2$ sein.

Pareto-Optimalität, effiziente Punkte und Paretofront Aus dem Dominanzkonzept folgt, dass es mehrere Vektoren geben kann, die andere dominieren, aber nicht selbst dominiert werden. Diese Lösungen werden als „optimale Lösungen im Pareto-Sinne“ oder „nicht dominierte“ Lösungen bezeichnet. Hier muss unterschieden werden, ob die Pareto-Optimalität für die globale Lösungsmenge gilt oder nur für Teile derselben. Bei der globalen Pareto-Optimalität muss der entsprechende Lösungsvektor die komplette Lösungsmenge dominieren. Bei der lokalen Pareto-Optimalität genügt eine Umgebung der Größe δ, in der alle anderen Lösungen dominiert werden. Das Bild eines Pareto-optimalen Vektors im Zielraum heißt „effizienter Punkt“.

Die Menge der nicht dominierten Lösungen im Parameterraum X wird als Pareto-optimale Menge, kurz Paretomenge, bezeichnet. Die Menge der Bilder der Paretomenge im Zielraum Z (Menge der effizienten Punkte) wird als Paretofront bezeichnet. Die formale Definition der Paretofront lautet daher in Dominanzschreibweise:

Die Paretofront ist die Menge $\mathbf{Y}_{eff} = \{z \in Z \mid \nexists y \in Z : y \leq z\}$. Siehe dazu auch [Meh05, Deb02].

4.2.3 Lösungsansätze

Mehrkriterielle Optimierprobleme sind sehr vielfältig. Sie sind in allen erdenklichen Bereichen des Lebens zu finden und es ist ein Leichtes, Beispiele aus der Biologie, Soziologie, Wirtschaft und vielen anderen Fachrichtungen zu finden. Einige Anwendungsbeispiele bieten [Meh05] und [ZDT+01]. [Deb02] führt in seinem Buch eine Liste mit Studien zu konkreten Anwendungen. In den folgenden Abschnitten soll eine mögliche Strukturierung der mehrkriteriellen Optimierverfahren vorgestellt werden, die bei der Auswahl eines geeigneten Algorithmus für das spezifische Anwendungsproblem helfen kann.

Anwenderpräferenzen

Im Laufe der Zeit haben sich viele Varianten an mehrkriteriellen Optimierverfahren herausgebildet. Es gibt verschiedene Ansätze, die Verfahren zu klassifizieren. Wichtig ist dabei der Einfluss der Anwenderpräferenzen auf das jeweilige Verfahren. [HM79] teilen die Lösungsansätze in folgende vier Klassen ein:

- **Non-Präferenz-Methoden**
  Bei dieser Methode fließen keine Anwenderpräferenzen in den Optimierungsprozess ein. Eine gefundene Lösung stellt lediglich eine Verbesserung zur vorherigen dar. Die Lösung wird dem Anwender präsentiert. Dieser kann sie annehmen oder ablehnen.

- **A-Posteriori-Methoden**
  Es liegen Präferenzinformationen des Entscheiders zu jedem Ziel vor. A-Posteriori-Algorithmen generieren die Paretomenge iterativ unter Berücksichtigung dieser Präferenzen.

- **A-Priori-Methoden**
  Anwenderpräferenzen fließen direkt von Anfang an in den Optimieralgorithmus ein. Dazu müssen sie mathematisch formuliert werden können. Diese Methode eignet sich besonders gut für Entscheider, die die grobe Optimierrichtung vorgeben möchten. Dadurch bleiben neue und innovative Lösungen allerdings oft verborgen. Dem Anwender werden nur wenige Lösungen präsentiert. Er muss nicht aus großen Mengen auswählen.

- **Interaktive Methoden**
  Bei diesem Verfahren findet während des Optimierprozesses Interaktion mit dem menschlichen Entscheider statt. Der Anwender erhält so sehr speziell auf seine Präferenzen zugeschnittene Lösungen. Interaktion kann überdies erforderlich werden, wenn sich die Anwenderpräferenzen nicht formal fassen lassen.

Auswahl bekannter Algorithmen

Im Folgenden soll eine Übersicht über bekannte Algorithmen gegeben werden. Die Verfahren im Einzelnen vorzustellen, würde den Rahmen dieser Arbeit überschreiten. Es sei daher auf die Habilitationsarbeit [Meh05] sowie [Deb02], in denen die unten aufgeführten Algorithmen detaillierter vorgestellt werden, verwiesen.

[Meh05] unterscheidet die Algorithmen wie folgt:

- **Mehrkriterielle deterministische Verfahren**, z. B.
  - Gewichtete Summen
  - ϵ-Constraints
  - Goal Programming
  - Goal-Attainment-Methode
  - Interaktive Methoden (z.B. STEM, GDF, GUESS, NIMBUS)
- **Mehrkriterielle evolutionäre Metaheuristiken** [1]
  Nach [CCVVL02] können die mehrkriteriellen evolutionären Metaheuristiken weiter in Verfahren unterteilt werden, die das Prinzip der Pareto-Dominanz nutzen und diejenigen, die es nicht nutzen. Beispiele für solche Verfahren sind im Folgenden aufgeführt:
  - Nicht-Pareto-Ansätze
    - VEGA: Vector Evaluated GA
    - Spieltheorie
    - PPES: Predator Prey Evolution Strategy
    - VOES: Vector-Optimized Evolution Strategy
  - Pareto-Ansätze
    - Goldbergs mehrkriterieller GA (Goldbergs GA ist prinzipiell monokriteriell)
    - MOGA: Multiobjective Genetic Algorithm
    - NSGA, NSGA-II: Nondominated Sorting Genetic Algorithm
    - ϵ-MOEA, C-NSGA-II
    - NPGA, NPGA 2: Niched-Pareto Genetic Algorithms
    - MOMGA, MOMGA-II: Multiobjective Messy Genetic Algorithms
    - SPEA, SPEA2: Strength Pareto Evolutionary Algorithms
    - PAES: Pareto Archieved Evolution Strategy
    - PESA, PESA-II: Pareto Envelope-based Selection Algorithm
    - micro-GA: Micro Genetic Algorithm for Multiobjective Optimization
    - RPSGAe: Reduced Pareto Set Genetic Algorithm with elitism

Im nächsten Unterkapitel wird ein häufig benutzter Algorithmus, der SPEA bzw. SPEA2, exemplarisch vorgestellt.

[1] Eine Metaheuristik ist eine Heuristik über Heuristiken, siehe dazu [Meh05].

4.2.4 Fallbeispiel

Der SPEA-Algorithmus

Der Strength Pareto Evolutionary Algorithm (SPEA) wurde 1999 von Zitzler und Thiele veröffentlicht [ZT99]. Nach der im vorherigen Abschnitt vorgestellten Kategorisierung der Algorithmen ist der SPEA ein mehrkriterielles Optimierverfahren, welches den Ansatz der Pareto-Dominanz nutzt.

Eine feste Anzahl nicht dominierter Lösungen wird dabei in einer externen Population (Archiv genannt) gespeichert. Dies stellt ein zentrales Konzept dieses Algorithmus dar. Die nicht dominierten Lösungen jeder neuen Generation werden mit denen des Archivs verglichen und die besseren beibehalten [2]. Diese besten Lösungen fließen wieder in die Erzeugung der nachfolgenden Generation ein, um die Optimierung in Richtung „guter“ Gebiete zu bewegen.

SPEA beginnt mit einer zufällig angelegten Population P_0 der Größe N und einer leeren externen Population $\overline{P_0}$ der maximalen Größe $\overline{N}$. In jeder Generation t werden die besten Lösungen aus der Population P_t nach $\overline{P_t}$ kopiert. Die dort vorhandenen, nun dominierten Lösungen, werden gelöscht. Wird die Größe $\overline{N}$ des Archivs überschritten, so werden ähnliche Lösungen verworfen. An dieser Stelle kommen Clusterverfahren zum Einsatz, siehe dazu [Deb02] bzw. die allgemeine Einführung in Clusterverfahren in Kapitel 6.

Nachdem die Eliten einer Generation ins Archiv gelegt wurden, kreiert der Algorithmus mittels genetischer Manipulationen eine neue Generation. Jedem Individuum der aktuellen Generation wird eine Fitness zugewiesen. Da SPEA im Rahmen der genetischen Manipulation auf das Archiv der elitären Lösungen zurückgreift, wird auch den Individuen i des Archivs eine Fitness zugewiesen. Diese Fitness wird „Strength“ genannt und hier mit S_i bezeichnet. Diese Stärke ist nun proportional zur Anzahl der Individuen der aktuellen Population, die von dem externen Individuum i dominiert werden. Um eine Zahl kleiner 1 zu erhalten, wird der Wert jedoch noch durch $N + 1$ dividiert. Je mehr Individuen also dominiert werden, desto besser ist das (externe) dominierende Individuum. Dieser Punkt ist ein weiteres fundamentales Konzept des SPEA.

Der Fitnesswert eines internen Populationsindividuums ergibt sich aus der Summe der Fitnesswerte derjenigen externen Individuen, die dieses schwach [3] dominieren, mit zusätzlicher Addition der Zahl 1 [4]. Je höher der Fitnesswert ist, desto schlechter ist das interne Individuum.

[2] Algorithmen, die dieses Konzept verfolgen, werden in der Literatur auch „elitär“ genannt, siehe [Deb02]

[3] zur schwachen Dominanz sei auf Unterkapitel 4.2.2 verwiesen

[4] damit die Fitness auf jeden Fall größer als die eines jeden externen Individuums ist

Am Ende wird auf der Vereinigung der internen und externen Population eine Auswahl durchgeführt. Die Individuen mit den niedrigsten Fitnesswerten (also die besten) bilden dann nach Mutationen und Rekombination die nächste Generation P_{t+1} der Größe N.

Eine Iteration des SPEA sieht wie folgt aus:

1. Finde die beste nichtdominierte Menge F_1 von P_t. Füge diese Lösungen ins Archiv $\overline{P_t}$ ein.
2. Finde die besten nichtdominierten Lösungen des Archivs und lösche alle dominierten.
3. Hat das Archiv seine Maximalgröße N überschritten, so bilde Cluster. Das Ergebnis ist das Archiv der nächsten Generation.
4. Weise den Archivindividuen Fitnesswerte zu; danach weise den internen Individuen Fitnesswerte zu.
5. Vereinige P_t und $\overline{P_t}$. Suche N Individuen mit kleinsten Fitnesswerten. Führe dann Mutationen und Rekombination durch. Die so entstandene Menge ist die Population P_{t+1} der nächsten Generation.

Wenn eine Lösung der Paretofront gefunden wird, kommt sie auf jeden Fall ins Archiv. Sie kann dort nur wieder entfernt werden, wenn andere Paretooptimale Lösungen eine bessere Verteilung auf der Paretofront aufweisen. Nachteilig ist, dass der Algorithmus einen festen Parameter $\overline{N}$ für die Archivgröße benötigt. Dabei ist auf das Verhältnis zur Populationsgröße N zu achten, da SPEA eventuell nicht richtig funktioniert. Die Laufzeit einer Iteration (Schritte 1-5) beträgt $O(MN^2)$.

Erweiterung SPEA2

SPEA2 ist eine Erweiterung des SPEA um folgende drei Punkte. Diese Punkte sind in dem vorgestellten SPEA z. T. schon enthalten, wurden 2001 aber nochmals in [ZLT01] unter dem neuen Namen zusammengefasst.

1. Die Fitnesszuweisung wurde verfeinert. In SPEA2 wird berücksichtigt, durch wie viele Lösungen eine Lösung dominiert wird und wie viele Lösungen ein Individuum dominiert.
2. Ein neues Dichtemaß zum Erkennen von Nachbarn einer Lösung wurde eingeführt. Dies verbessert die Effizienz der Diversitätsberechnung.
3. Einführung einer Archivgrößenbegrenzung

5 Visualisierung

Die Daten der Anwendungsfälle (siehe Kapitel 3), die mithilfe der evolutionären Optimierung (siehe Kapitel 4) gewonnen wurden, sollen nun für menschliche Betrachter visualisiert werden. Hierzu müssen sowohl die technischen Möglichkeiten als auch die Aspekte der menschlichen Wahrnehmung beachtet werden.

Im ersten Abschnitt (5.1) sollen zunächst die Grundlagen der räumlichen Darstellung erläutert werden, die bei der Entwicklung von Pavel berücksichtigt wurden. Dabei wird auch auf die vielfältigen Tiefenhinweise eingegangen, die das menschliche Gehirn zur räumlichen Wahrnehmung nutzt.

Der zweite Abschnitt (5.2) erläutert zunächst die Grundlagen der stereoskopischen Darstellung, um dann genauer auf die genutzten Parameter einzugehen. Es werden diverse stereoskopische Wiedergabeverfahren vorgestellt und ihre Funktionsweise erklärt.

Der dritte Abschnitt (5.3) führt durch die Grundlagen der Visualisierungstechniken, welche in Pavel umgesetzt wurden.

5.1 Räumliche Darstellung

5.1.1 Darstellungsformen

Im Folgenden werden die verschiedenen, in Pavel verwendeten, Darstellungsformen für dreidimensionale Objekte mit einem zweidimensionalen Medium, wie etwa einem Monitor, erläutert.

Punktewolken

Punktewolken werden durch Messungen, wie z. B. dem Abtasten realer Objekte mit einem 3D-Scanner oder aus Berechnungen gewonnen. Sie bilden die Grundlage für die Erzeugung dreidimensionaler Objekte in der Computergraphik. Stellen diese Punkte ein Objekt dar, so genügt die Darstellung als Punktewolke bei hoher Punktdichte, um das Objekt zu erkennen, besonders wenn das Objekt zusätzlich gedreht wird. Sind nicht ausreichend viele Punkte vorhanden, so können weitere Punkte interpoliert werden.

Werden die Punkte nicht durch Abtasten realer Objekte gewonnen, so wie z. B. die in dieser PG betrachteten Ergebnisse evolutionärer Algorithmen, so stellen sie meist keine Objekte dar und es dürfen keine weiteren Punkte interpoliert werden.

Punkte im mathematischen Sinne besitzen keine Ausdehnung und sind daher für das menschliche Auge nicht sichtbar. Daher werden zur Darstellung anstelle von Punkten Kugeln, Quader oder ähnliche dreidimensionale Formen gewählt. Verschiedene Formen und Farben können dabei dazu dienen, spezielle Bereiche hervorzuheben. Bei der Größe der „Punkte“ sollte die Punktdichte beachtet werden. Liegen die „Punkte“ dicht beieinander, so verlaufen sie bei zu großer Ausdehnung. Diese sollte also an die Größe des betrachteten Bereichs angepasst werden.

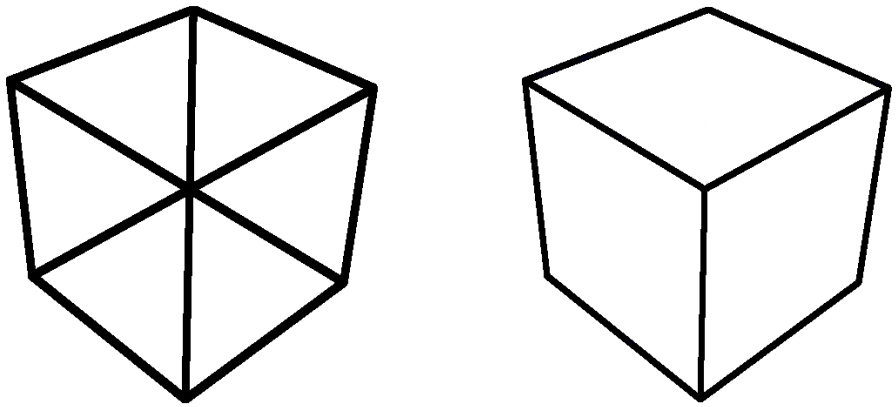
Bei der Darstellung als Punktewolke ist vor allem die geringe Verdeckung von Vorteil. Außerdem ist keine weitere Berechnung (siehe unten) für die Darstellung nötig, wie dies bei den aus Punktewolken gewonnenen Polygonnetzen oder Volumenkörpern der Fall ist.

Für die Erzeugung eines Tiefeneindrucks können allerdings, bis auf die Drehung, kaum Effekte eingesetzt werden. Linearperspektive, relative Größe und depth blur können wegen der begrenzten Ausdehnung der „Punkte“ nur begrenzt eingesetzt werden, da diese eine gewisse Größe aufweisen müssen.

Polygonnetze

Polygonnetze können aus Punktewolken durch Vernetzungsmethoden gewonnen werden. Dafür ist zusätzlicher Aufwand für die Vernetzung und die eventuelle Netzverschönerung nötig. Besonders durchsichtige Netze, bei denen alle Kanten gezeichnet werden, wirken leicht verwirrend und werden schnell als flach wahrgenommen, wie in Abbildung 5.1(a) zu sehen. Diesem Problem kann durch Drehung des Objektes entgegengewirkt werden.

Undurchsichtige Netze, wie in Abbildung 5.1(b) dargestellt, bieten Tiefeninformationen durch die Verdeckung und durch die Winkel der Kanten zueinander. Zusätzlich können die Linearperspektive (siehe Abschnitt: 5.1.2) und der damit verbundene Effekt der relativen Größe eingesetzt werden. Bei genügender Kantendicke sind auch Schatten zur Verbesserung der Tiefenwahrnehmung hilfreich. Luftperspektive und relative Helligkeit (siehe Abschnitt: 5.1.2) wirken allerdings kaum.



(a) (b)

Abbildung 5.1: (a) Sechseck oder Quader? (b) Ein undurchsichtiges Netz

Volumenkörper

Volumenkörper erfordern den höchsten Verarbeitungsaufwand. Auch die Speicherung von Volumenkörpern ist aufwändiger, als die von Punktewolken oder Polygonnetzen.

Im Gegenzug ist die Darstellung von Volumenkörper durch Flächen die realistischste virtuelle Darstellung von Objekten. Sämtliche, im Folgenden genannten Effekte zur Erzeugung eines Tiefeneindrucks können angewandt werden. Größtes Manko von Volumenkörpern ist die Verdeckung. Möglichkeiten dieses Problem zu mindern sind in Abschnitt „Verdeckung“ weiter unten aufgeführt.

Projektion

Da Objekte mit herkömmlichen Monitoren bzw. Leinwänden nicht tatsächlich dreidimensional dargestellt werden können, müssen sie so projiziert werden, dass sie als zweidimensionale Objekte dreidimensional erscheinen. Dafür gibt es im Wesentlichen zwei Methoden, die *Parallelprojektion* und die *perspektivische Projektion*, die in den folgenden Unterabschnitten näher erläutert werden.

Parallelprojektion eignet sich besonders für technische Zeichnungen sowie für die Bearbeitung durch Computer, da sie Form und Skalierung erhält [CS00]. Sie entspricht aber nicht der menschlichen Wahrnehmung, weshalb solche Bilder auf den Betrachter störend wirken und leicht ein Kippeffekt entsteht. Dieser Effekt ist in Abbildung 5.2, bei der die mittlere Ecke entweder als Innenecke oder auch als Außenecke wahrgenommen wird, zu sehen.

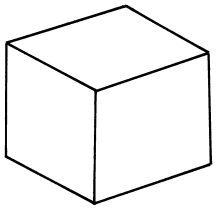


Abbildung 5.2: Parallel projizierter Quader

Mathematisch funktioniert die Parallelprojektion wie im Folgenden beschrieben (siehe auch [Aso96a, Stö03]). Weitere Informationen zur Parallelprojektion sind z. B. in [FKN71] zu finden.

Die Bildebene $ax_0 + by_0 + cz_0 + d = 0$ wird mit der Projektionsgeraden, bestimmt durch den Vektor $\vec{v} = \langle p, q, r \rangle$, geschnitten. Alle Punkte werden entlang paralleler Linien projiziert. Der Normalenvektor der Bildebene sei $\vec{N} = \langle a, b, c \rangle$.

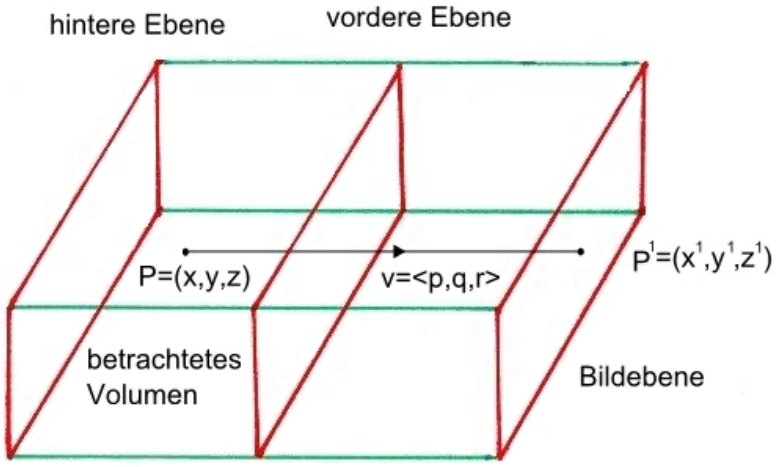


Abbildung 5.3: Bei der Parallelprojektion werden die Punkte $P = (x, y, z)$ des dreidimensionalen Raumes, zwischen der hinteren Ebene und der vorderen Ebene, entlang paralleler Strahlen auf die Bildebene abgebildet. Diese Projektionsgeraden werden durch den Vektor $\vec{v} = \langle p, q, r \rangle$ bestimmt.

Die Projektionsgerade wird wie folgt in Parameterform geschrieben:

$$x_0 = x + pt \tag{5.1}$$
$$y_0 = y + qt \tag{5.2}$$
$$z_0 = z + rt \tag{5.3}$$

Es ist folgendes Gleichungssystem nach t aufzulösen:

$$ax_0 + by_0 + cz_0 + d = 0 \tag{5.4}$$
$$a(x_0 + pt) + b(y_0 + qt) + c(z_0 + rt) + d = 0 \tag{5.5}$$
$$ax_0 + by_0 + cz_0 + t(ap + bq + cr) + d = 0 \tag{5.6}$$

Damit ergibt sich:

$$t = -\left[\frac{ax_0 + by_0 + cz_0 + d}{ap + bq + cr}\right] \tag{5.7}$$

vorausgesetzt $ap + bq + cr \neq 0$.[1] Durch Einsetzen von t in (5.1) bis (5.3) ergeben sich die Gleichungen für den projizierten Punkt:

$$x^1 = x - p\left[\frac{ax_0 + by_0 + cz_0 + d}{ap + bq + cr}\right] \tag{5.8}$$
$$y^1 = y - q\left[\frac{ax_0 + by_0 + cz_0 + d}{ap + bq + cr}\right] \tag{5.9}$$
$$z^1 = z - r\left[\frac{ax_0 + by_0 + cz_0 + d}{ap + bq + cr}\right] \tag{5.10}$$

Dies kann auch in Matrixform geschrieben werden:

$$\begin{pmatrix} x & y & z & 1 \end{pmatrix} \begin{pmatrix} m_{11} & m_{12} & m_{13} & 0 \\ m_{21} & m_{22} & m_{23} & 0 \\ m_{31} & m_{32} & m_{33} & 0 \\ m_{41} & m_{42} & m_{43} & 1 \end{pmatrix} = \begin{pmatrix} x^1 & y^1 & z^1 & 1 \end{pmatrix} \tag{5.11}$$

Mit:

$$m_{11} = \frac{bq + cr}{ap + bq + cr} \qquad m_{12} = \frac{-aq}{ap + bq + cr} \qquad m_{13} = \frac{-ar}{ap + bq + cr}$$
$$m_{21} = \frac{-bp}{ap + bq + cr} \qquad m_{22} = \frac{ap + cr}{ap + bq + cr} \qquad m_{23} = \frac{-br}{ap + bq + cr}$$
$$m_{31} = \frac{-cp}{ap + bq + cr} \qquad m_{32} = \frac{-cq}{ap + bq + cr} \qquad m_{33} = \frac{ap + bq}{ap + bq + cr}$$
$$m_{41} = \frac{-dp}{ap + bq + cr} \qquad m_{42} = \frac{-dq}{ap + bq + cr} \qquad m_{43} = \frac{-dr}{ap + bq + cr}$$

[1] Wäre $ap + bq + cr = 0$, so wäre die Projektionsrichtung senkrecht zur Normalen der Bildebene und der zu projizierende Punkt für die Bildebene nicht sichtbar.

Perspektivische Projektion ist die geeignete Darstellungsform, wenn die Darstellung auf den menschlichen Betrachter realistisch wirken soll, da sie der menschlichen Wahrnehmung entspricht. Auch Kamerabilder entstehen gemäß diesem Prinzip. Nachteil dieser Projektion ist die perspektivische Verkürzung, was besonders bei Punktewolken und anderen kleinen Objekten zu Problemen führt. Je weiter Objekte vom Betrachter entfernt sind, desto kleiner werden sie, sodass „Punkte“ schnell nicht mehr wahrnehmbar sind.

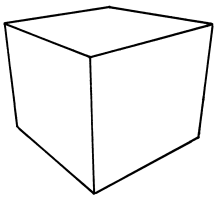


Abbildung 5.4: Perspektivisch projizierter Quader

Die mathematische Berechnung der perspektivischen Projektion funktioniert wie nachfolgend beschrieben (siehe auch [Aso96b, Stö03]). Genauere Informationen über die perspektivische Projektion sind z. B. in [FKN71] zu finden.

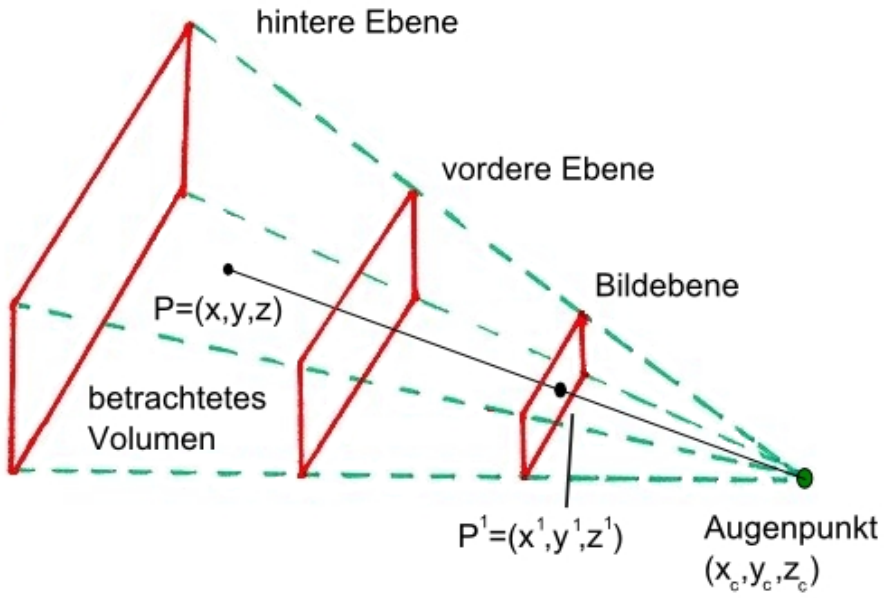


Abbildung 5.5: Bei der perspektivischen Projektion werden die Punkte $P = (x, y, z)$ des dreidimensionalen Raumes, zwischen der hinteren Ebene und der vorderen Ebene, entlang der Geraden durch P und den Augenpunkt (x_c, y_c, z_c) auf die Bildebene projiziert. Der Augenpunkt ist der Punkt, in dem die Projektionsgeraden konvergieren.

Sei $ax_0 + by_0 + cz_0 = 0$ die Bildebene und (x_c, y_c, z_c) der Augenpunkt, also der Punkt, in dem die Projektionsgeraden konvergieren. Es sei (x, y, z) der zu projizierende Punkt. Die Gerade durch (x, y, z) und (x_c, y_c, z_c) wird in Parameterform dargestellt:

$$x_0 = x + (x_c - x)t \tag{5.12}$$
$$y_0 = y + (y_c - y)t \tag{5.13}$$
$$z_0 = z + (z_c - z)t \tag{5.14}$$

Es ist somit folgendes Gleichungssystem nach t aufzulösen:

$$ax_0 + by_0 + cz_0 + d = 0 \tag{5.15}$$
$$a(x_0 + (x_c - x)t) + b(y_0 + (y_c - y)t) + c(z_0 + (z_c - z)t) + d = 0 \tag{5.16}$$
$$ax_0 + by_0 + cz_0 + t(a(x_c - x) + b(y_c - y) + c(z_c - z)) + d = 0 \tag{5.17}$$

Mit dem Ergebnis:

$$t = -\left[\frac{ax_0 + by_0 + cz_0 + d}{a(x_c - x) + b(y_c - y) + c(z_c - z)}\right] \tag{5.18}$$

Angenommen $z = 0$ ist die Bildebene und $(0, 0, e)$ der Augenpunkt.
Dann ist $t = -z/(e - z)$ und

$$x^1 = \frac{ex}{e - z} \tag{5.19}$$
$$y^1 = \frac{ey}{e - z} \tag{5.20}$$
$$z^1 = 0 \tag{5.21}$$

Verdeckung

Es gibt viele Methoden, um das *Verdeckungsproblem*, das besonders bei Volumenkörpern auftritt, wenn ein Objekt ein anderes optisch überlagert, zu beseitigen oder zu umgehen.

Drehung ist wohl die rechenaufwändigste der Methoden zur Beseitigung des Verdeckungsproblems, da eine hohe Zahl von Einzelbildern berechnet werden muss. Sie erlaubt allerdings auch den besten Überblick über das Objekt und erleichtert die Orientierung. Bei sehr unebenen Oberflächen, die sich teilweise selbst verdecken, reicht auch diese Methode nicht für eine vollständige Übersicht aus.

Semitransparenz ist eine sehr gute Ergänzung zur Drehung und anderen Methoden zur Beseitigung des Verdeckungsproblems. Bei mehrfacher Verdeckung ist diese Methode allerdings sehr aufwändig, da Z-Puffertechniken nicht funktionieren. Durch die undurchsichtige Darstellung interessanter Bereiche, sowie die transparente bzw. semitransparente, Darstellung umliegender Gebiete, können auch verdeckte Teile leicht sichtbar gemacht werden. Durch den Einsatz von Farbe können Objektteile zusätzlich hervorgehoben werden.

5.1.2 Monokulare Tiefenwahrnehmung

Der folgende Abschnitt erläutert verschiedene Aspekte der monokularen Tiefenwahrnehmung.

Linearperspektive

Die Linearperspektive wurde in ihrer modernen Form durch Filippo Brunelleschi (1377-1446) begründet. Bedeutende Werke in der systematischen Entwicklung des perspektivischen Zeichnens sind „De prospectiva pigendi“ von Pietro della Francesca (1416-1492) und Leonardo da Vincis (1452-1519) „Traktat der Malerei“ [Bär94]. Es wurde von einem „Brennpunkt“ aus ein Faden zu dem zu zeichnenden Objekt gespannt. Dafür wurde die dazwischen befindliche Leinwand zunächst weggeklappt. Der Schnittpunkt wurde festgehalten und nach Entfernen des Fadens auf der Leinwand markiert. Nachdem auf diese Art hinreichend viele Punkte auf der Leinwand waren, entstand ein Bild, das der perspektivischen Projektion und somit der menschlichen Wahrnehmung entsprach. Albrecht Dürer veranschaulichte dieses Vorgehen in seinem Bild „Der Zeichner der Laute“ in [Dür25] (siehe auch [Bär94]). Bei der Erzeugung von 3D-Computerbildern wird im Prinzip ebenso vorgegangen.

Wie bereits erwähnt, ist diese Art der Darstellung besonders bei Punktewolken problematisch, da „Punkte“ leicht zu klein werden und somit kaum oder gar nicht mehr wahrnehmbar sind. Wichtigster Aspekt der Linearperspektive ist die Konvergenz paralleler, horizontaler Linien oder Strukturen in einem Fluchtpunkt am Horizont (siehe [Keb94]), wie in Abbildung 5.6.
Eng damit verknüpft ist der folgende Effekt: Objekte in Horizontnähe erscheinen weiter entfernt (Abb. 5.7).

Das Zusammenspiel von konvergierenden Linien und relativer Größe wurde von J.J. Gibson in [Gib50] als „Texturgradient“ bezeichnet. Durch den Texturgradienten kann die Ausdehnung von Flächen eingeschätzt werden und horizontale und vertikale Flächen können voneinander unterschieden werden, da bei vertikalen Linien keine Konvergenz auftritt.

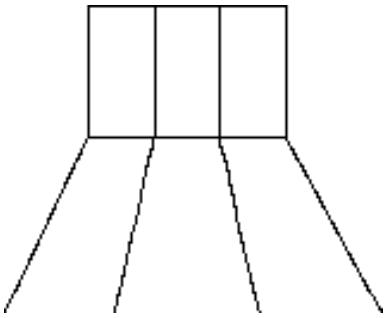


Abbildung 5.6: Skizze der Linearperspektive

Abbildung 5.7: Die Figuren werden als unterschiedlich weit entfernt wahrgenommen.

Interposition

Interposition, oder auch Okklusion, also die Überdeckung von Objekten durch andere Objekte, wurde schon vor rund 4000 Jahren in der ägyptischen Malerei zur Tiefendarstellung eingesetzt [Bär94]. Soll diese Technik bewusst eingesetzt werden, so erfordert dies ein differenziertes räumliches Vorstellungsvermögen wie es etwa Kleinkindern noch nicht besitzen. Das überdeckte Objekt als weiter entfernt zu sehen und es mental zu vervollständigen geschieht automatisch aufgrund von Seherfahrungen im realen Raum [Mie79]. Diese Seherfahrung, das sogenannte Gestaltprinzip [Wie97b], führt dazu, dass z. B. in Abbildung 5.11 drei Quadrate wahrgenommen werden.

Ähnlich wie schon die relative Größe ist auch die Interposition nur ein schwacher Tiefenhinweis und kann ebenso leicht getäuscht werden. Abbildung 5.8 wird als Bild zweier einander verdeckender Quadrate wahrgenommen, obwohl sie auch, wie in Abbildung 5.9 zu erkennen ist, ein Quadrat und eine L-Form darstellt [Fer00].

Ein weiterer Aspekt der Interposition ist das Bestreben des menschlichen Wahrnehmungssystems, Vordergrund und Hintergrund zu bestimmen.
Besonders bei großen farblichen Kontrasten wird eine Unterscheidung erzwungen, was bei uneindeutigen Bildern zu Kippsituationen führt. So ist es in Abbildung 5.10 zwar möglich, zwischen den beiden möglichen Bildern bewusst zu wechseln, aber es ist unmöglich, gleichzeitig die Vase und die Gesichter zu sehen [Mie96].

Abbildung 5.8: Es werden zwei Quadrate wahrgenommen.

Abbildung 5.9: Bild 5.8 könnte ebenso ein Quadrat und ein „L“ darstellen.

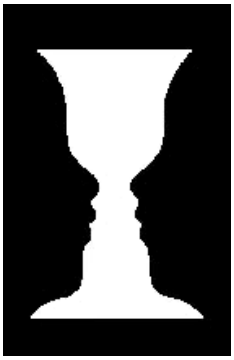


Abbildung 5.10: Es werden entweder eine Vase oder zwei Gesichter wahrgenommen.

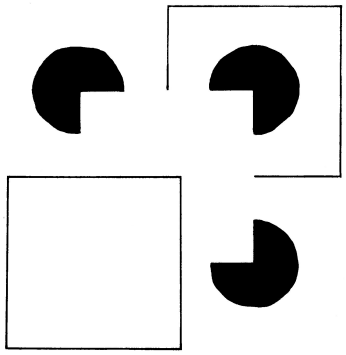


Abbildung 5.11: Das Gestaltprinzip veranlasst den Betrachter drei Quadrate wahrzunehmen.

Relative Helligkeit

Die relative Helligkeit lässt weiter entfernte Objekte dunkler als nahe Objekte erscheinen, da umso weniger Licht in Richtung des Betrachters reflektiert wird je weiter es von einer Lichtquelle entfernt ist (siehe [Wie97b]). Umgekehrt können so Objekte durch eine dunklere Färbung als weiter entfernt dargestellt werden.
Der Effekt der *relativen Helligkeit* wird bei einer Umkehrung der Beleuchtungsverhältnisse, also eine Beleuchtung entfernter Gegenstände und einen dunklen Vordergrund, größtenteils durch andere Tiefenhinweise, vor allem die Linearperspektive, aber auch relative Größe und Interposition, überdeckt. Es kommt nicht zu einer kompletten Umkehrung der Tiefenwahrnehmung. Der Tiefenhinweis der relativen Helligkeit ist eng verknüpft mit der Luftperspektive.

Kinetischer Tiefeneffekt

Das Gegenstück zur Bewegungsparallaxe bildet der *kinetische Tiefeneffekt* (kinetic depth effect, kurz KDE), auch bekannt als motion perspective.

Hierbei wird das Objekt bewegt und der Betrachter bleibt statisch. Objekte, wie zum Beispiel der Quader in Abbildung 5.1(a), wirken ohne Bewegung leicht flach und verworren. Durch die Drehung können einzelne Punkte leichter fixiert werden, wodurch die dreidimensionale Form gut erkennbar wird. Da das Auge niemals vollständig ruhig ist, verliert der Betrachter bei statischen Objekten leicht den fixierten Punkt. Auch bei Volumenkörpern können Kanten und Texturen bei Bewegung leichter fixiert werden [WO53, Wie97b].

Dieser Effekt wurde schon in den 40er und 50er Jahren bei Schattenprojektionen mit Drahtbügeln entdeckt. Dazu wurden verbogene Drahtbügel hinter einer Leinwand aufgehängt und angeleuchtet. Erst als die Drahtbügel gedreht wurden, konnten die Betrachter die dreidimensionale Form erkennen [Fer00].

Akkommodation

Abbildung 5.12: Akkommodation: Bei Anspannung der Ciliarmuskeln, wie im linken Bild, werden Objekte in größerer Entfernung fokussiert. Bei entspannter Linse, wie im rechten Bild, werden näher gelegene Objekte fokussiert.

Die Akkommodation ist die Anpassung der Linsenkrümmung der Augen um Objekte in unterschiedlicher Entfernung zu fokussieren, wie in Abb. 5.12. Dies geschieht durch Anspannung der Ciliarmuskeln bei größerer Entfernung vom Objekt. Da die Linsen nicht beliebig gestreckt werden können, ist die Akkommodation nur bis zu einer Entfernung von 3 Metern als Informationsquelle für Tiefeninformationen nutzbar [Keb94]. [Wik06] nennt hingegen eine Entfernung von 10 Metern für die Nutzung der Akkommodation. In jedem Fall ist sie nur für kurze Distanzen wirksam.

Da Computerbilder flach sind, kann hier die Fokussierung nicht beliebig vom Betrachter durch Anpassen der Linsenkrümmung angepasst werden.

Sollen dennoch verschiedene Bereiche oder Tiefen fokussierbar sein, muss dies simuliert werden. So könnte zum Beispiel ein markiertes Objekt scharf dargestellt werden und der Rest unscharf. Im Gegensatz zu realen Objekten kann so der Fokus des Betrachters auf einen bestimmten Bereich gelenkt werden.

5.1.3 Fazit

Insgesamt kann also gesagt werden, dass es sehr viele optische Hinweise zur räumlichen Wahrnehmung gibt. Um diese Effekte für die Computergraphik zu nutzen muss Verschiedenes beachtet werden. Sämtliche Tiefenhinweise sollten konsistent sein, um den Betrachter nicht zu verwirren. Besonders die starken Hinweise, wie die Linearperspektive, sollten stimmig sein, da sie die anderen Effekte dominieren. Weiterhin sollten diese Techniken bei Computergraphiken nur moderat eingesetzt werden, da gerade kleine Objekte, wie die Punkte in Punktewolken, schnell nicht mehr wahrnehmbar sind, und somit der Gesamteindruck verfälscht würde.

5.2 Stereoskopie

Der folgende Abschnitt widmet sich der binokularen Tiefenwahrnehmung und den verschiedenen Techniken der stereoskopischen Darstellung.

5.2.1 Grundlagen der stereoskopischen Darstellung

Monokulare und binokulare Tiefenwahrnehmung

Die Tiefenwahrnehmung eines Menschen setzt sich aus zwei verschiedenen Effekten zusammen. Zum einen gibt es die monokulare Tiefenwahrnehmung, die bei der Wahrnehmung mit nur einem Auge zum Einsatz kommt. Zu diesen verschiedenen Effekten gehört die Bewegungsparallaxe, die Akkommodation, Größenunterschiede und Helligkeit, um nur einige zu nennen. Diese sind allerdings eher ungenau oder nicht aussagekräftig genug. Sie können nur eine ungefähre Einschätzung der Entfernung liefern.

Weitaus aussagekräftiger ist die binokulare Tiefenwahrnehmung. Hierbei gibt es die Konvergenz und die Stereopsis. Unter Konvergenz wird Folgendes verstanden: Fixiert der Betrachter ein Objekt, bilden die Augenachsen einen bestimmten Winkel zueinander. Weil der Augenabstand konstant ist, lässt sich aus den Winkeln die Entfernung des Objektes bestimmen, welches fixiert ist. Fixieren und Fokussieren von Objekten ist auch bei der monokularen Betrachtung realer Objekte möglich. Herkömmliche Displays können diese Tiefeninformation allerdings vermitteln.

Aber gerade dieses ist die Einschränkung der Konvergenz: Die Tiefe lässt sich nur für alle Objekte mit der gleichen Entfernung zu den Augen bestimmen. Genau diese Schwäche gleicht die Stereopsis aus.

Konvergenz

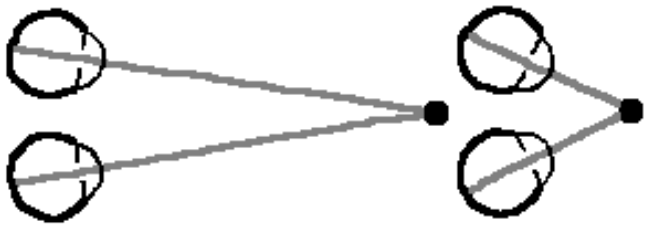


Abbildung 5.13: Je näher das betrachtete Objekt ist, desto stärker müssen die Augen gegeneinander gedreht werden, um es zu fokussieren.

Um die unterschiedlichen Bilder der beiden Augen zur Deckung zu bringen, wird die Blickrichtung durch eine gegenläufige Drehung der Augen angepasst [Keb94]. Bei diesem, Konvergenz genannten, Mechanismus ist die Blickrichtung bei weit entfernten Objekten „parallel“ und bei nahen Objekten gekreuzt.

Wegen der Disparität der beiden wahrgenommenen Bilder kann keine vollständige Deckung erreicht werden. Außerdem können sich wiederholende Muster, wie etwa einen Gartenzaun, nicht eindeutig zur Deckung gebracht werden. Während dies für Computer durchaus ein Problem darstellt, wird das menschliche Gehirn problemlos damit fertig. Es wird vermutet, dass eine enge Verknüpfung zwischen Konvergenz und Akkommodation besteht [Wie97a]. Auch diese Methode ist zur Tiefenwahrnehmung nur für kurze Distanzen nutzbar.

Genau wie die Akkommodation, kann auch die Konvergenz in der Computergraphik nur simuliert werden. Dies kann ebenfalls durch Schärfe bzw. Unschärfe bestimmter Bereiche geschehen.

Stereopsis

Die Stereopsis, oder auch *binokulare Disparität* (retinal disparity), ist für den Menschen eine der wichtigsten Quellen für Tiefeninformationen.

(a) (b)

Abbildung 5.14: Die Bilder für das linke Auge (a) und das rechte Auge (b) sind nicht deckungsgleich. So erhält das Gehirn mehr Informationen als bei der monokularen Wahrnehmung.

Wie in Abbildung 5.2.1 zu sehen ist, nehmen die menschlichen Augen aufgrund ihres Abstandes von 6-7 cm zwei unterschiedliche Bilder auf und übermitteln

sie an das Gehirn, wo sie weiterverarbeitet werden [Keb94].

Der Leser möge sich das einfach verdeutlichen, indem er einen Finger vor sein Gesicht hält und abwechselnd die Augen schließt. Der Finger „hüpft“ hin und her. Es werden die zwei monokularen Bilder wahrgenommen, die bei beidäugiger Betrachtung zu einem binokularen Einzelbild fusionieren.

Die Stereopsis dominiert die meisten anderen Tiefenhinweise und reicht somit für die Tiefenwahrnehmung aus. Allerdings kann auch sie von anderen Effekten überstimmt werden. So überwiegt bei der Vertauschung der Bilder für linkes und rechtes Auge die Linearperspektive, Vordergrund und Hintergrund werden hier nicht vertauscht, wie bei einer Dominanz der Stereopsis zu erwarten wäre. Ein solches Bild wirkt aufgrund der Inkonsistenz der Tiefeninformationen allerdings sehr verwirrend.

Durch die Nutzung beider Augen wird die Informationsdichte erhöht. Da immer noch große Teile des betrachteten Objektes verdeckt werden, wird die Information allerdings nicht verdoppelt, sondern nur leicht erhöht. Diesem relativ geringen Gewinn im Vergleich zur zweidimensionalen Darstellung steht ein vergleichsweise hoher Aufwand für die Darstellung von 3D-Bildern gegenüber.

Die Disparität hängt vom Abstand des Objektes zum Auge ab und kann nur bis zu einer Entfernung von 30 Metern genutzt werden. Diese Entfernung kann zwar in der Computergraphik künstlich erweitert werden, aber die Tiefeneinschätzung leidet darunter [Wie97a].

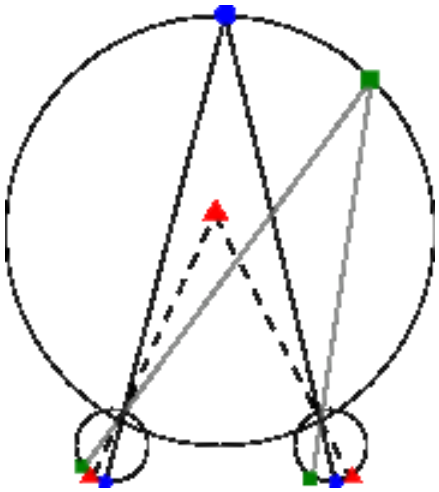


Abbildung 5.15: Vieth-Müller-Kreis: Der runde Punkt ist der Fixationspunkt, der rechteckige wird auf korrespondierenden Punkt abgebildet. Der dreieckige Punkt kann nicht zur Fusion gebracht werden.

Abbildung 5.15 zeigt eine vereinfachte Darstellung bei der Betrachtung eines Objektes. Die schwarzen Linien (ausgehend vom runden Objekt) zeigen die Abbildung des Fixationspunktes auf die einzelnen Augen. Dieser Punkt wird jeweils in der Fovea (der Bereich des schärfsten Sehens der Netzhaut) des linken und rechten Auges abgebildet. Er liegt auf zwei korrespondierenden Punkten.

Wird nun das viereckige Objekt auf dem Kreis mit den dazugehörigen Abbildungspunkten in den Augen betrachtet, wird klar, dass sie ebenso auf korrespondierende Punkte abgebildet werden. Es ist zu beachten, dass der Fixationspunkt immer noch das runde Objekt ist. Alle Punkte, die auf dem Kreis liegen, lassen sich in korrespondiere Punkte auf der Netzhaut abbilden. Dieser Kreis wird Vieth-Müller-Kreis genannt.

Das dreieckige Objekt liegt nicht auf dem Kreis, die Punkte korrespondieren nicht. Daher kann keine Fusion des Punktes stattfinden, der Punkt wird doppelt gesehen. Die Betrachtung des Vieth-Müller-Kreises ist jedoch lediglich eine Vereinfachung. Es lassen sich noch viel mehr Raumpunkte ausmachen, die als ein Punkt wahrgenommen werden und nicht in der Fovea liegen. Aufgrund der nicht ganz kugelartigen Form des Auges liegen diese Raumpunkte auf einer gekrümmten Ebene, die Horopter genannt wird (siehe Abbildung 5.16). In der Abbildung 5.16 sind ebenfalls die Winkel a_r und a_l eingezeichnet. Die

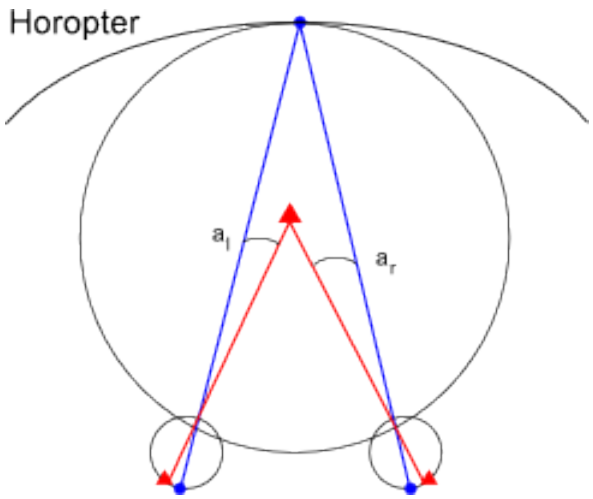


Abbildung 5.16: Vieth-Müller-Kreis mit Horopter und Winkel für einen Punkt

Differenz $a_l - a_r$ wird als Disparität bezeichnet [Han99]. Die Disparität für Punkte, die auf dem Horopter liegen, ist immer 0.

Aber nicht nur Punkte auf dem Horopter selbst können zur Fusion gebracht werden. In einem bestimmten Bereich vor und hinter dem Horopter, Panum genannt, können fusioniert werden. Das Panum hat eine nicht genau eingrenzbare Ausdehnung. Sie ist von Mensch zu Mensch verschieden durch die unterschiedliche Augengeometrie. Üblicherweise wird eine Ausdehnung von 12-50 cm angenommen.

Ausnutzung der Disparität

Die Disparität lässt sich ausnutzen, um auf einem zweidimensionalen Schirm eine Tiefenwirkung zu erzeugen. Abbildung 5.17 zeigt die Umsetzung der Halbbilder für die einzelnen Augen. Abbildung 5.18 zeigt die Situationen für

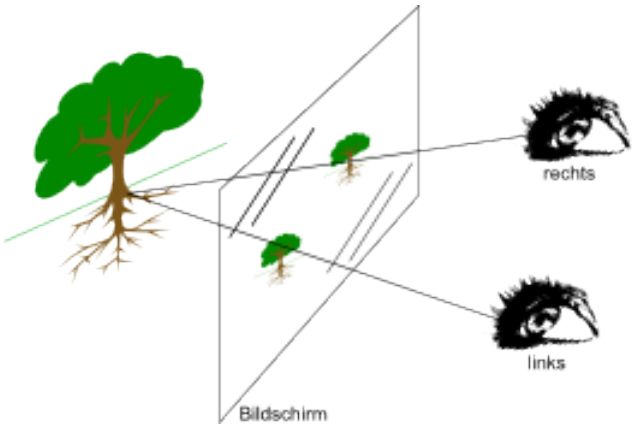


Abbildung 5.17: Disparität auf dem Bildschirm

das Objekt, wenn es sich hinter, auf und vor dem Bildschirm befinden soll. Es ist zu sehen wie sich die Punkte auf dem Bildschirm abbilden. Bei der zweiten Situation beträgt die Disparität 0.

5.2.2 Parameter der stereoskopischen Darstellung

Um die Proportionen einer virtuellen Szene auf die gegebene Realszenengeometrie abzustimmen, ist es notwendig, die passenden Parameter zu bestimmen.

Zunächst sollen die Parameter der Betrachter- und der Kameraszene vorgestellt werden. Der Betrachter (siehe Abbildung 5.19) sieht mit dem Augenabstand (Augenseparation) E den Bildschirm mit dem Abstand Z, wobei der Bildschirm eine Breite W besitzt. Die Panumsgrenzen (N: vordere Grenze, F: hintere Grenze) können/müssen entsprechend an die Bedürfnisse angepasst werden. Ein Betrachter direkt vor einem Bildschirm hat eine andere Panumsgrenze als die Betrachtermenge vor einer Projektionsfläche.

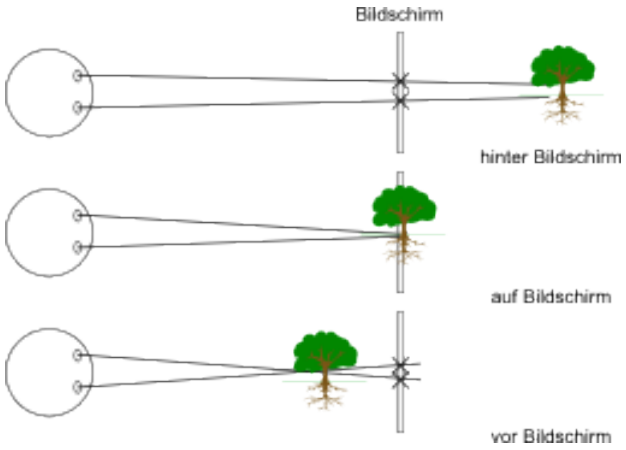


Abbildung 5.18: Disparität für verschiedene Situationen

Berechnung der Parameter

Die jeweiligen Disparitäten d_N und d_F lassen sich aus den anderen festgelegten Parametern berechnen.

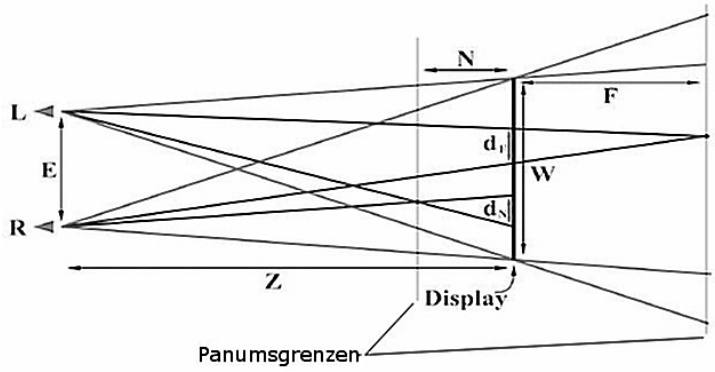


Abbildung 5.19: Betrachterraum, E: Augenabstand, Z: Abstand zum Display, W: Breite des Displays, N: Abstand zur vorderen Panumsgrenze, F: zur entferntesten Panumsgrenze, d_N und d_F: jeweilige Dispariät [JLHE01]

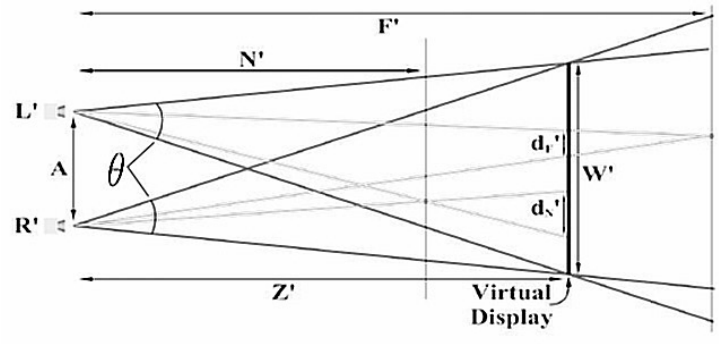


Abbildung 5.20: Kameraraum, A: Separation der Kameras, Z': Abstand zum virtuellen Display, W': Breite des virt. Displays, N': Abstand zur vorderen Darstellungsgrenze, F': zur entferntesten Darstellungsgrenze, d'_N und d'_F: jeweilige Dispariät, θ: Sichtfeldparameter [JLHE01]

Anders sieht es in der Kameraszene aus: Die meisten Parameter müssen wir berechnen, um sie mit der Betrachterszene in Einklang zu bringen. Diese Berechnungsvorschrift hier wurden weitestgehend aus [JLHE01] übernommen.

Die Disparitäten im Betrachterraum lassen sich wie folgt berechnen:

$$d_N = \frac{NE}{Z - N} \qquad d_F = \frac{FE}{Z + F} \tag{5.22}$$

Die Disparitäten in der Kameraszene berechnen sich durch:

$$d'_N = \frac{A(Z' - N')}{N'} \qquad d'_F = \frac{A(F' - Z')}{F'} \tag{5.23}$$

Die Disparitäten beider Räume sind nicht gleich, aber sie sind proportional zueinander. Dies lässt sich durch folgende Gleichung beschreiben:

$$\frac{d_N}{d_F} = R = \frac{d'_N}{d'_F} \tag{5.24}$$

Die Entfernung Z' zum virtuellen Display lässt sich nun bestimmen durch:

$$Z' = \frac{R + 1}{\frac{1}{N'} + \frac{R}{F'}} \tag{5.25}$$

Ist der Sichtfeldparameter θ (auch FOV, Field of View, genannt) der Kamera gegeben, lässt sich hieraus die Breite des virtuellen Bildes bestimmen:

$$W' = 2Z' \tan(\frac{\theta}{2}) \tag{5.26}$$

Mit der Skalierung S vom Bildschirm zum virtuellen Bildschirm

$$S = \frac{W'}{W} \tag{5.27}$$

lässt sich über

$$d'_N = Sd_N \tag{5.28}$$

die Kameraseparation A berechnen:

$$A = \frac{d'_N N'}{Z' - N'} = \frac{Sd_N N'}{Z' - N'} \tag{5.29}$$

Damit sind nun alle erforderlichen Parameter der virtuellen Szene bestimmt. Soll eine Szene konkret berechnet werden, hängt das von den unterstützten Kenngrößen des benutzen Systems ab. OpenGL z. B. besitzt bereits die Möglichkeit, ein „view frustum“ zu erzeugen. Dieses „view frustum“ kann mit dem hier vorgestellten Panum gleichgesetzt werden.

Auswirkung der Parameter

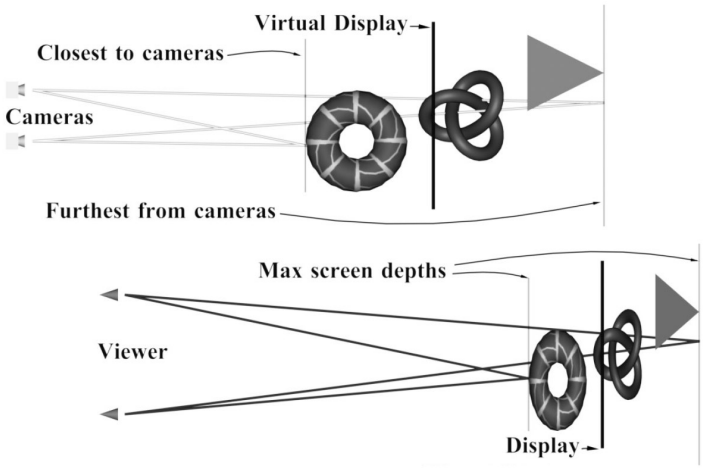


Abbildung 5.21: Tiefenkompression: Entsteht bei falscher Wahl der Parameter [JLHE01]

Die Wahl der Panumsgrenzen ist entscheidend für die Tiefenwirkung der virtuellen Szene. Es kann zur so genannten „Tiefenkompression“ bzw. „Tiefenexpansion“ kommen. Exemplarisch zeigt Abbildung 5.21 die Tiefenkompression. Dort wurden in der Kameraszene (untere Skizze) die Panumsgrenzen zu groß gewählt. Daraus resultiert in der Betrachterszene (obere Skizze), dass die

Objekte komprimiert erscheinen. Umgekehrt kann es auch zu einer Expansion kommen, wenn die Panumsgrenzen in der virtuellen Szene zu klein gewählt werden. Die Objekte erscheinen dann gestreckt.

Einen weiteren Einfluss hat die Größe der Projektionsfläche. Die Darstellung sollte möglichst das komplette Gesichtsfeld des Betrachters ausfüllen. Denn an den Rändern geht die Tiefenwirkung des Bildes verloren, der Betrachter wird verwirrt.

5.2.3 Stereoskopische Wiedergabeverfahren

Die Aufgabe der Wiedergabeverfahren ist es, die stereoskopischen Wiedergabekanäle den Augen des Betrachters getrennt zuzuführen [Lüc00]. Es existieren brillenlose und brillengebundene Verfahren, auf die im Folgenden eingegangen werden soll. Weitere Verfahren existieren, kommen jedoch nicht für unsere PG in Frage.

Brillenlose Verfahren

Diese Verfahrensweise zeichnet sich dadurch aus, dass der Benutzer keine weiteren Hilfsmittel benötigt. Die Bildkanäle für die Augen werden durch Linsen- /Prismenraster oder durch Schlitzmasken getrennt. Diese Art von Display wird autostereoskopisches Display genannt. Abbildung 5.22 zeigt ein Linsenrasterdisplay. Die Wiedergabekanäle werden pixelweise horizontal wechselnd ausgegeben. Durch vertikale Linsen wird das Bild für das linke bzw. rechte Auge gebrochen.

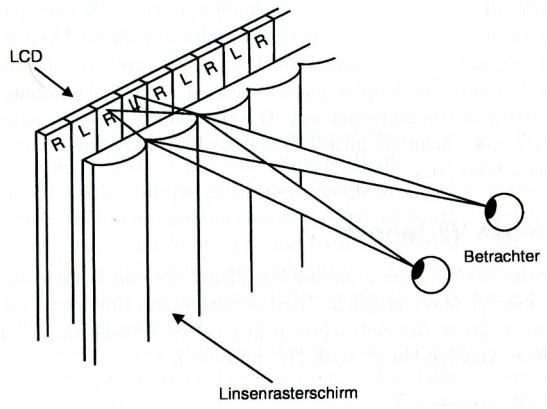


Abbildung 5.22: Linsenraster-Display [Lüc00]

In Abbildung 5.23 wird das Schlitzmaskenverfahren dargestellt. Dabei werden Barrieren geschaltet, die polarisiertes Licht so filtern, dass die beiden Augen ihre verschiedenen Bilder erhalten. Sharp Corp. hat ein Verfahren entwickelt, das es ermöglicht das Display jeweils für die reine 2D-Darstellung und die 3D-Darstellung zu nutzen. Um die reine 2D-Darstellung zu nutzen, werden die Barrieren transparent geschaltet. Dabei lässt sich auch eine der Einschränkungen erkennen: Die Auflösung halbiert sich bei der Aktivierung der 3D-Darstellung. Die Winkelabhängigkeit des Displays beschränkt die

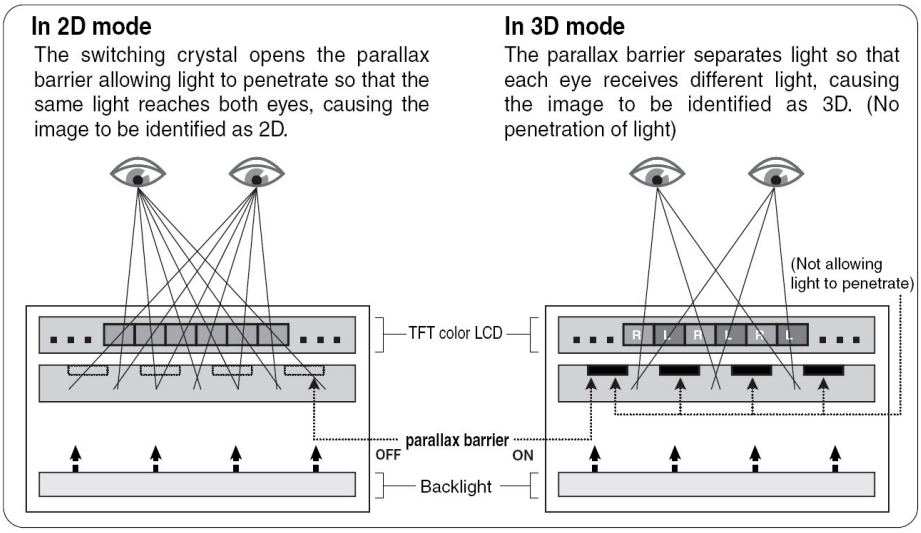


Abbildung 5.23: Schlitzmasken-Display [SE06]

Betrachteranzahl weitestgehend auf eine Person.
Dazu wird meist die Betrachterposition vor dem Display selbst eingeschränkt. Weitere Verfahren ermöglichen es, durch Headtracking mehrere Benutzer zuzulassen. Jedoch wird mit jedem Benutzer die horizontale Auflösung halbiert. Somit beschränkt sich die Nutzung dieses Verfahrens auf eine kleine Betrachtermenge und benötigt ein speziell gefertigtes Display.

Brillengebundene Verfahren

Bei den brillengebundenen Verfahren ist es nötig, eine Brille mit speziellen Gläsern zu tragen. Der Betrachter kann die Position im Raum freier wählen als bei den brillenlosen Verfahren. Ein Nachteil dieser Gruppe ist es, dass die Bildhelligkeit durch die Filterung der Brille abnimmt.

Shutterbrille Diese Brille besitzt zwei LC-Filter, die abwechselnd zwischen transparent und undurchlässig geschaltet werden können.
Dieser Vorgang wird mit dem Bildschirm synchronisiert, sodass jeweils das passende Auge für das dargestellte Bild transparent geschaltet wird. Ein Nachteil ist, dass durch das abwechselnde Anzeigen des Bildes die Bildwiederholfrequenz des Monitors halbiert wird. Bei einem Fernsehgerät mit 50 Hz können nur 25 Bilder pro Auge angezeigt werden, was zu einem leichten Flackern führt. Heutige Röhrenmonitore können jedoch höhere Frequenzen (z. B. 120 Hz) hergeben, was den Komfort um einiges erhöht.

Wegen der LCD-Technik lässt sich dieses Verfahren jedoch nicht bei TFT-Monitoren anwenden. TFT-Bildschirme arbeiten mit polarisiertem Licht. Würde mit solch einer Brille ein TFT-Monitor betrachtet, würde ein Brillenglas immer undurchlässig bleiben. Die Betrachtermenge bleibt hierbei ebenfalls recht gering, aber deutlich größer als bei den Schlitzmaskenverfahren.

Farbfilter Die Farbfiltermethode (auch Anaglyphen-Verfahren genannt) erlaubt es, die Szene auf einer großen Projektionsfläche für eine große Betrachtermenge darzustellen. Hierbei werden die Wiedergabekanäle komplementärfarbig eingefärbt und übereinandergelegt. Die gängigsten Farbkombinationen sind Rot/Grün und Rot/Cyan. Vor das linke Auge wird der rote Farbfilter platziert.

Ist die Abstimmung der Filterung von Brille und Monitor nicht ganz exakt (was sich aber in den meisten Fällen nicht verhindern lässt), entstehen sogenannte „Geisterbilder“. Die Trennung der Kanäle ist hierbei nicht optimal und damit erscheinen Teile des Bildes doppelt.

Die Rot/Cyan-Kombination kann im Gegensatz zu dem Rot/Grün-Verfahren auch Farb-Anaglyphe darstellen. Zudem ist die Helligkeitsverteilung besser, was für das Auge ermüdungsfreier ist.

Insgesamt ist die Farbfiltermethode eine kostengünstige und einfache Methode, um eine dreidimensionale Darstellung auf dem Bildschirm wie auch auf anderen Projektionsflächen zu erreichen. Es werden lediglich die Brillen benötigt. Die Zusammenführung der Bilder erfolgt in Computer.

Polarisationsfilter Bei dem Verfahren mit den Polarisationsfiltern werden die zwei Halbbilder auf je einem Projektor ausgegeben (wobei es mittlerweile Produkte gibt, die beide Projektoren zu einem zusammenfassen). Die Projektoren besitzen um 90° verdrehte Polarisationsfilter, dementsprechend sind die Filter der Brillen ausgelegt. Die Projektionsfläche muss die Polarisation aufrecht erhalten. Der Betrachter darf seinen Kopf nicht zur Seite neigen, da sonst die Filterung nicht korrekt ist. Der PG steht ein solcher Aufbau am ISF zur Verfügung.

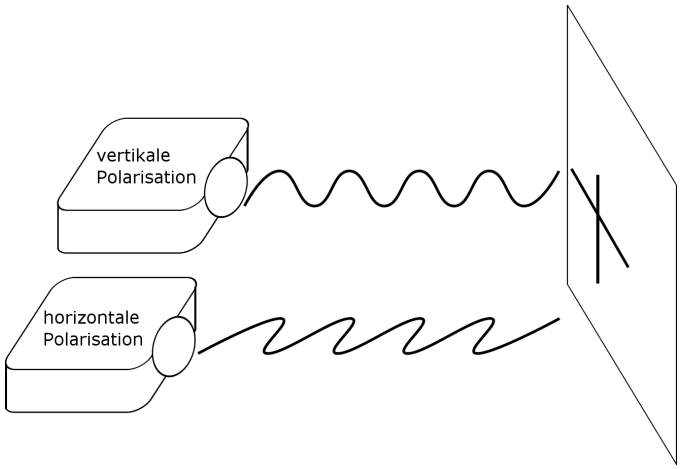


Abbildung 5.24: Polarisationsfiltertechnik: Skizze zur Demonstration der Polarisation

Interferenzfilter Diese Technik wurde 1999 von der DaimlerChrysler AG entwickelt. Hierbei werden die Stereohalbbilder von zwei Projektoren mit verschiedenen Wellenlängen der einzelnen Farben projiziert. Dieses Verfahren nutzt aus, dass das menschliche Auge die Farben über einen bestimmten Wellenlängenbereich als eine Farbe wahrnimmt. Die Brillen verfügen über trennscharfe Interferenzfilter, die die entsprechenden Bilder herausfiltern. [Inf06] Diese Technik ist an der Universität Dortmund in einem neuen Hörsaal (E23 OH14) eingebaut worden.

5.2.4 Fazit

Es gibt vielfältige Möglichkeiten, eine im Computer erstellte 3D-Szene dem Betrachter mit einem räumlichen Eindruck nahe zubringen. Dies ist jedoch nur begrenzt möglich. Zum einen ist die Tiefenwirkung beschränkt, zum anderen sind die Verfahren meist nicht für sehr große Betrachtermengen geeignet. Allerdings bietet die Stereoskopie eine interessante Möglichkeit, mehrdimensionale Daten darzustellen. Der Benutzer ist in der Lage, ein ganz anderes Gefühl für die Daten zu entwickeln.

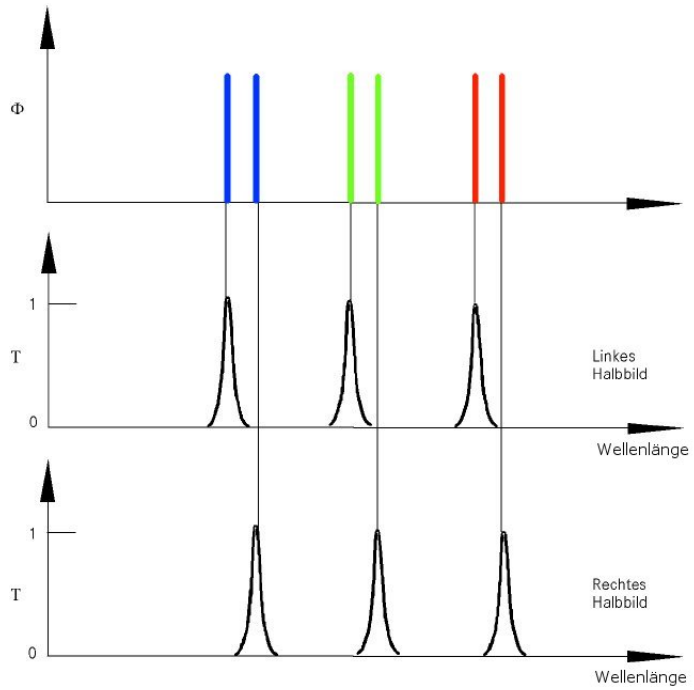


Abbildung 5.25: Interferenzfiltertechnik: Die Projektoren arbeiten mit verschiedenen Wellenlängen. Das oberste Diagramm zeigt die überlagerten Halbbilder. Die unteren beiden Diagramme zeigen jeweils die Wellenlängen der beiden Projektoren.[Inf06]

5.3 Visualisierung hochdimensionaler Datenmengen

Da die in den Unterkapiteln 3.1 und 3.2 beschriebenen Anwendungsfälle sehr große Datenmengen generieren, werden in diesem Unterkapitel die in Pavel zur Visualisierung von Daten genutzten Techniken und ihre Grundlagen beschrieben. Da sich die Daten in den Anwendungsfällen stark unterscheiden, wurden die Verfahren zur Visualisierung darauf abgestimmt. Während bei Temperierbohrungen (siehe Unterkapitel 3.1) wenige hochdimensionale Datensätze erzeugt werden, entstehen bei der Fräsbahnoptimierung (siehe Unterkapitel 3.2) sehr viele Daten mit wenigen Dimensionen.

5.3.1 Scatterplots

Scatterplots sind in ihrer zweidimensionalen Ausprägung schon sehr lange bekannt. Durch ihre einfache Struktur sind sie relativ leicht zu verstehen. Sie basieren auf einem kartesischen System, bei dem auf den zwei Koordinatenachsen jeweils ein Attribut aufgetragen wird. Wie in Abbildung 5.26 zu sehen ist, können sehr einfach Korrelation zwischen den Attributen erkannt werden. So ist in der Abbildung sehr gut die Abhängigkeit des Fitnesswertes eines EA-Laufes und des zu optimierenden Attributs zu erkennen. Je größer der Fitnesswert sein soll, desto größer muss auch das Attribut gewählt werden.

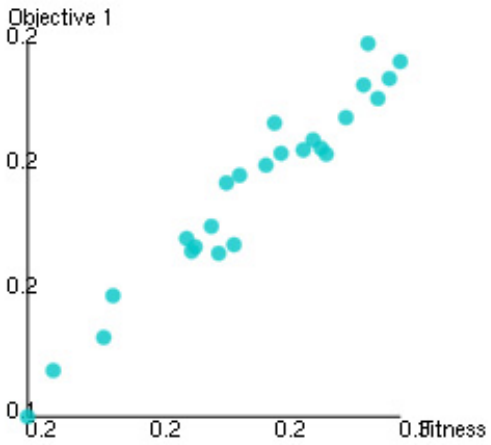


Abbildung 5.26: 2D Scatterplot auf dem der Fitnesswert gegen das Objective1 augetragen wurde

Es zeigen sich allerdings auch Probleme bei der Verwendung von Scatterplots. So besteht gerade bei der Visualisierung von großen Datenmengen eine Schwierigkeit darin, dass sich einzelne Punkte überlappen können. Daher ist es ab einem gewissen Grad schwer zu entscheiden, wie viele Punkte sich an bestimmten Stellen in einem Diagramm befinden. Dies kann so weit führen, dass ein Diagramm nur noch als eine einzige homogene Fläche wahrgenommen wird. Ab diesem Punkt ist es nicht mehr möglich, Korrelationen in den Daten zu erkennen [PKH04].

Weiterhin besteht bei unseren Anwendungsfällen das Problem der hohen Dimensionalität. Denn mit der gezeigten Technik ist es nicht möglich, mehr als zwei Dimensionen darzustellen, da nur zwei Attribut-Achsen zur Verfügung stehen. Zudem ist das Problem der Interaktion des Benutzers mit den Daten nicht geklärt, da Scatterplots in dieser Form keine Interaktion erlauben. Daher wurden in Pavel noch weitere Varianten des Scatterplots implementiert, die auf die einzelnen Probleme eingehen.

3D Scatterplots

Als Erweiterung der Scatterplots ist eine 3D-Version der normalen Scatterplots entstanden. Die Idee des 3D Scatterplots besteht darin, das 2D Koordinatensystem um eine weitere Achse zu erweitern. Diese kann genau wie die ersten beiden Achsen mit einem zusätzlichen Attribut belegt werden. Eine weitere Erhöhung der darstellbaren Attribute kann durch den Einsatz von Farben erreicht werden. Dabei wird eine weitere Eigenschaft auf einen Farbverlauf abgebildet. Mit diesem Farbverlauf werden die einzelnen Punkte des Scatterplots entsprechend ihrer Eigenschaft eingefärbt [KSH02].

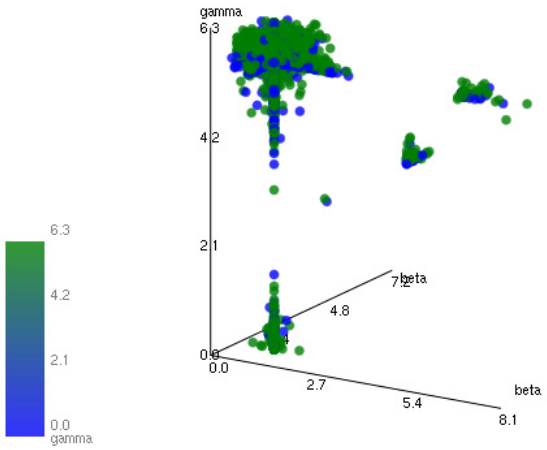


Abbildung 5.27: 3D Scatterplot

Scattermatrix

Als Scattermatrix wird eine Erweiterung des normalen Scatterplots bezeichnet. Sie wird verwendet, um wesentlich mehr als zwei/drei Dimensionen auf einmal darzustellen. Hierzu werden mehrere Scatterplots für jeweils zwei Dimensionen erstellt und in einer Matrix- beziehungsweise Tabellenform, dargestellt.

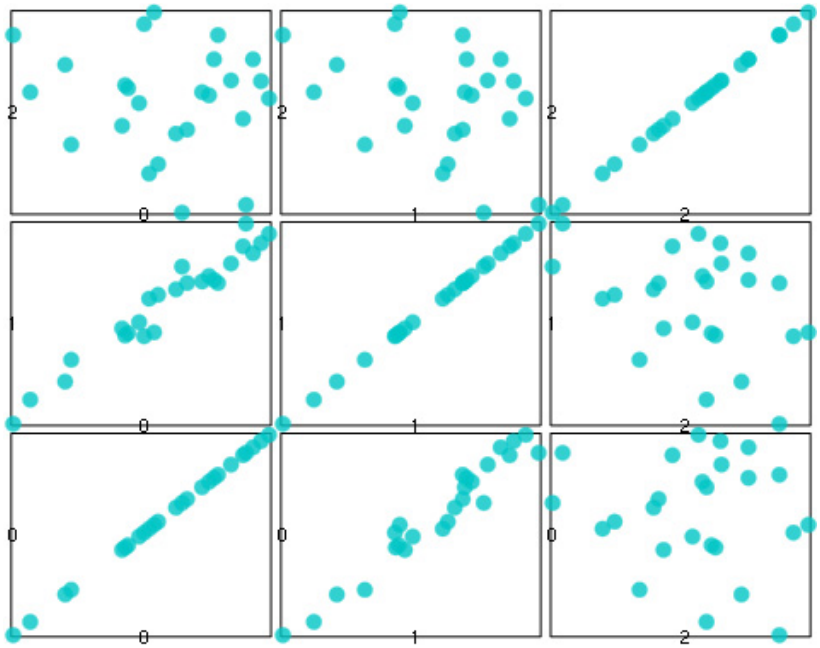


Abbildung 5.28: Scattermatrix

Wie die Abbildung 5.28 veranschaulicht, ist diese Darstellungsform sehr übersichtlich. Es ist möglich mehr als zwei Attribute miteinander zu vergleichen und in diesen Daten Korrelationen zu erkennen.

Interaktion und Tiefenwahrnehmung

Alle bisher aufgeführten Techniken sind statischer Natur. Es gibt keine Interaktion zwischen dem Betrachter und dem Scatterplot. Um diesen Mangel zu beheben stehen mehrere Möglichkeiten zur Verfügung [PKH04], von denen folgende in Pavel integriert wurden:

- Brush-Technik:
  Hierbei werden durch den Betrachter bestimmte Bereiche des Bildes hervorgehoben. Diese Bereiche werden in allen weiteren Ansichten der Daten ebenfalls hervorgehoben.

- Linked-View:
  Linked-View bezeichnet eine Technik, durch die eine Veränderung in einer Ansicht auch in allen anderen Ansichten der Daten visualisiert wird. Diese Methode ist besonders in Kombination mit der oben genannten Brush-Technik effektiv.

- Begrenzung des Datenbereichs
  Hierzu wird interaktiv von dem Betrachter der Datenbereich eingeschränkt. Dies dient der Verbesserung der Übersichtlichkeit. Dadurch müssen weniger Daten visualisiert werden, wodurch die Darstellung deutlich an Verständlichkeit gewinnt. Natürlich sollte der Bereich nicht wahllos eingeschränkt werden; es sollte ein besonderes Augenmerk auf Bereiche gelegt werden, die von besonderem Interesse sind.

5.3.2 Parallelkoordinaten

Idee

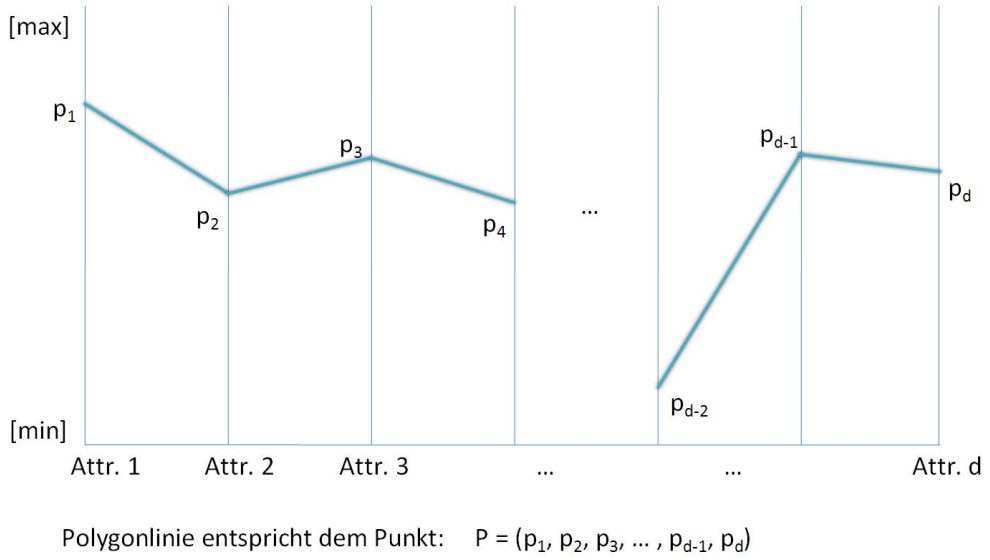


Abbildung 5.29: Parallelkoordinaten

Bei den Parallelkoordinaten werden die Dimensionen vertikalen Achsen zugeordnet. Die Anzahl der Achsen entspricht daher der Anzahl der Dimensionen. Die Achsen werden gleichmäßig horizontal verteilt, sodass die Abstände zwischen den Achsen gleich sind. In welcher Reihenfolge die Dimensionen bzw.

Achsen angeordnet werden, ist ein eigenständiges Problem, auf das später noch eingegangen wird. Jede Achse skaliert für jede Dimension individuell, d. h. der kleinste Wert wird z. B. am unteren Ende der Achse eingetragen und der größte Wert am oberen Ende. Jeder d-dimensionale Punkt wird dann als Polygonlinie dargestellt, die die Achsen an den Punkten schneidet, die den Werten der entsprechenden Attribute entsprechen.

Erweiterungen

Brushing Unter Brushing versteht man das Einfärben von Polygonlinien. So können Polygonlinien farbig markiert werden, die zu einer d-dimensionalen Teilmenge gehören, also z. B. alle Linien, die zu einer Firma, zu einer Automarke etc. gehören [YWR03].

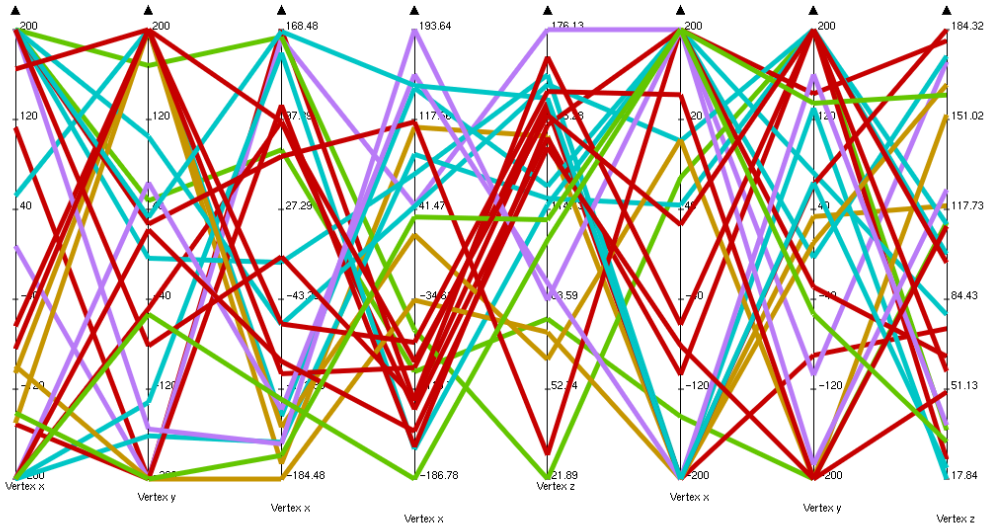
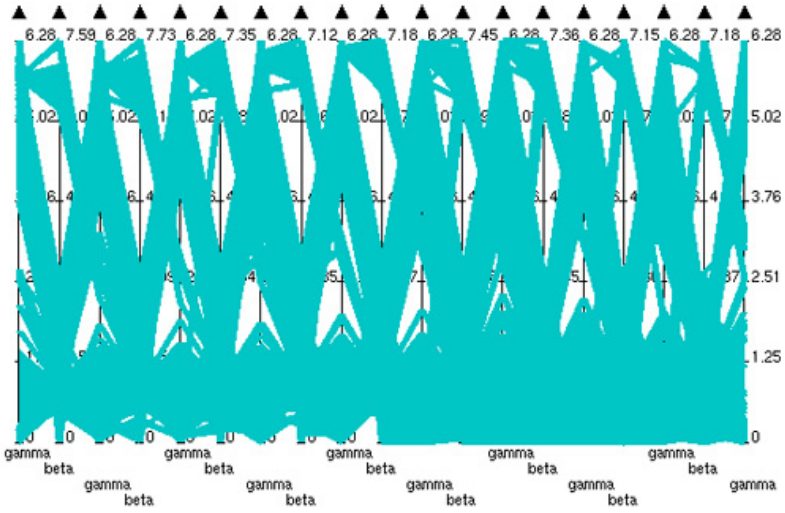
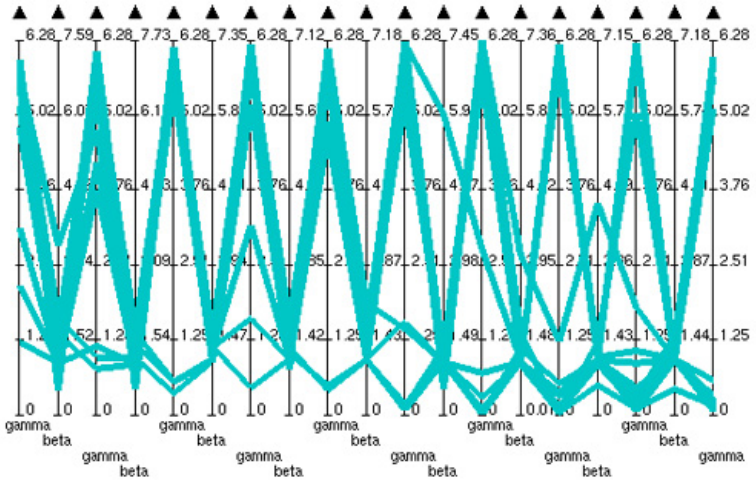


Abbildung 5.30: Brushing

Ausdünnung Das Problem bei Polygonlinien und deshalb speziell bei Parallelkoordinaten ist die Tatsache, dass sie viel Platz brauchen und deshalb schnell zu einer Fläche verschmelzen. Wie an Abbildung 5.31(a) zu sehen ist, muss die Datenmenge schon drastisch reduziert werden, damit die einzelnen Linien unterschieden werden können.
Um die Datenmengen aber sinnvoll reduzieren zu können, werden entsprechende Verfahren gebraucht, auf die im Kapitel 6 (Clustering) eingegangen wird.



(a)



(b)

Abbildung 5.31: Ausdünnung: Vor (a) und nach dem Clustern (b).

Interaktionsmöglichkeiten

Bei den Parallelkoordinaten gibt es viele Interaktionsmöglichkeiten von denen auch einige sehr wichtig zum intuitiven Verständnis der Datenmenge sind. So sollten Achsen leicht vertauscht, gedreht, gelöscht oder die Skalierung geändert werden können, da sich dadurch das Gesamtbild drastisch verändern kann.

5.3.3 Fazit

Natürlich sind dies nicht alle Verfahren, die zur Visualisierung von Datenmengen existieren. Aber diese wurden ausgewählt, weil sie besonders gut die Anforderungen von Pavel erfüllen. So ist es mit den Scatterplots möglich große Datensätze zu visualisieren, wie sie bei der Fräsbahnoptimierung entstehen. Im Gegensatz dazu sind die Parallelplots sehr gut geeignet die Daten der Temperierbohrungen zu visualisieren, da sie besonders gut Datensätze mit vielen Dimensionen darstellen können.

6 Clusteranalyse

Im Rahmen der Projektgruppe gilt es, große Eingabedatenmengen sinnvoll zu visualisieren und den Benutzer beim Auswahlprozess einzelner Objekte zu unterstützen. Da die Semantik der Daten im allgemeinen Fall nicht bekannt ist, eignen sich die Verfahren der Clusteranalyse ideal, um Datenmengen zu strukturieren und repräsentative Objekte für jede Gruppe zu finden.

Im Folgenden werden, nach kurzer Einordnung, zunächst einige Definitionen zum Thema Clusteranalyse gegeben und erläutert. Anschließend folgt der erste Hauptteil dieses Themengebietes, in dem eine Reihe von Standardclusterverfahren vorgestellt werden. Abschließend werden drei weitere Clusterverfahren beschrieben, die speziell für die Projektgruppe von Interesse sein könnten.

6.1 Grundlagen

Mit dem Begriff *Clusteranalyse* wird die Aufgabe bezeichnet, in einer vorgegebenen Menge von Objekten eine Gruppenstruktur zu erzeugen (vgl. [Pru06], S. 289). Dies erfolgt unüberwacht, das heißt, die Ergebnisse sind nicht bekannt, und somit erfolgt kein Eingriff eines Benutzers. Das hat insbesondere zur Folge, dass das Ergebnis nur vom eingesetzten Verfahren (seinen Parametern) und dessen Annahmen abhängt (vgl. [JMF99]). Generell bedeutet dies, dass die verschiedenen Verfahren auf unterschiedlichen Daten vollkommen andere Ergebnisse liefern können. Somit ist ein breites Spektrum an Möglichkeiten notwendig, um flexibel arbeiten zu können.

6.1.1 Clustern

Für die n betrachteten Objekte der Eingabemenge liegen jeweils die Werte von p Merkmalsvariablen vor. Die Datenbasis einer jeden Clusteranalyse ist also eine $n \times p$ Datenmatrix, anhand derer eine Gruppenstruktur innerhalb der Objekte erzeugt werden soll. Hierfür gibt es zwei Kriterien: zum einen die Ähnlichkeit zwischen Objekten, die in eine Klasse einsortiert werden, und zum anderen die Unähnlichkeit zwischen den nicht in eine Klasse eingeordneten Objekten. Um die Unähnlichkeit zwischen Objekten zu messen, werden Distanzmaße benötigt, die im Folgenden näher beschrieben werden.

6.1.2 Distanzmaß

Um einen Begriff von Unähnlichkeit und Ähnlichkeit zu erhalten, wird ein Maß benötigt, welches diese quantifiziert. Dieses ist normalerweise auf jeweils zwei Elementen definiert und wird, in Anlehnung an eine Interpretation im p-Vektorraum, als Distanz bezeichnet. Sind die verwendeten Merkmale intervall-skaliert (metrisch), so ist das am häufigsten verwendete Distanzmaß die Euklidische Distanz. Diese basiert auf den Abstandsformeln für zwei bzw. drei Dimensionen in der Euklidischen Geometrie und wird für die p Dimensionen des Merkmalsraumes folgendermaßen definiert.

Seien x_i und x_j zwei Objekte und sei x_{ih} der Wert der h-ten Merkmalsvariable des Objektes i, dann gilt:

$$\|x_i - x_j\| \equiv \sqrt{\sum_{h=1}^{p} (x_{ih} - x_{jh})^2}. \tag{6.1}$$

Zur Vermeidung der Wurzel wird häufig die quadrierte Euklidische Distanz verwendet. Die Verallgemeinerung des Euklidischen Abstands ist die Minkowski Metrik. Sie verwendet anstatt der Quadrierung und der Quadratwurzel eine variable Potenzierung m-ten Grades und dazu die passende m-te Wurzel. Die Euklidische Distanz ist also der Spezialfall für $m = 2$. Ein anderer oft verwendeter Spezialfall arbeitet mit $m = 1$ und ist dadurch besonders leicht zu berechnen.

Um die Euklidische Distanz auch verwenden zu können, falls die verwendeten Merkmale nicht mit vergleichbaren Skalierungen, Streuungen und Zentrierungen vorliegen, müssen die Werte der Datenmatrix zum einen zentriert und zum anderen standardisiert werden. Die zentrierten und standardisierten Merkmalsvariablen z_{ih} berechnen sich wie folgt:

$$z_{ih} = \frac{x_{ih} - \bar{x}_h}{s_h}. \tag{6.2}$$

Dabei ist $\bar{x}_h$ der Mittelwert und $s_h = \sqrt{s_h^2}$ die Standardabweichung der h-ten Merkmalsvariable:

$$\bar{x}_h = \frac{1}{n} \sum_{i=1}^{n} x_{ih}, \qquad s_h^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_{ih} - \bar{x}_h)^2. \tag{6.3}$$

Eine weitere Unlänglichkeit, die bei der Anwendung der Euklidischen Distanz zu unkorrekten Ergebnissen führt, ist die Korreliertheit der Variablen.

Die Mahalanobis-Distanz stellt die Unkorreliertheit der Variablen her, indem bei der Abstandsberechnung die Inverse der Kovarianzmatrix

$$S = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(x_i - \bar{x})^T \tag{6.4}$$

$$s_{hh'} = \frac{1}{n-1} \sum_{i=1}^{n} (x_{ih} - \bar{x}_h)(x_{ih'} - \bar{x}_{h'}) \tag{6.5}$$

einbezogen wird:

$$\|y_i - y_j\| = \sqrt{(x_i - x_j)^T S^{-1} (x_i - x_j)}. \tag{6.6}$$

y ist dabei das transformierte Objekt. Auf diese Weise lassen sich beliebige Merkmalsvariablen und -kombinationen verarbeiten, solange die Merkmale metrisch sind (vgl. [Pru06], S. 291ff).

6.1.3 Partition

Das Ergebnis der Clusteranalyse liefert im allgemeinen Fall eine Partitionierung der Daten. Bei sehr kleinen Datenmengen könnte deshalb die Idee entstehen, alle möglichen Partitionierungen aufzuzählen und anhand eines im nächsten Abschnitt beschriebenen Gütemaßes die beste Lösung aus dieser Menge auszuwählen. Dieses Vorgehen der Enumeration liefert im allgemeinen Fall - bezüglich eines bestimmten Gütemaßes - natürlich ein ideales Ergebnis. Die Anzahl aller möglichen Partitionierungen (Clusterlösungen ohne leere Cluster) beschreiben die Stirlingschen Zahlen zweiter Art. Für n Objekte und k Cluster gilt die folgende rekursive Formel: $S(n,k) = 0$ für $k > n$, da alle Cluster mindestens ein Element enthalten müssen. $S(n,1) = S(n,n) = 1$ hat nur eine eindeutige Lösung, genau wie $S(0,0) = 1$.

$$S(n+1,k) = k \cdot S(n,k) + S(n,k-1). \tag{6.7}$$

Die folgende Beschreibung verdeutlicht die Rekursionsformel: Wird ein neues Element hinzugefügt $(n+1)$, so kann dieses entweder in eine der k Klassen einsortiert werden (erster Teilterm) oder es wird in einen eigenen Cluster eingeordnet und die restlichen n Objekte werden auf die $k-1$ verbleibenden Cluster aufgeteilt (zweiter Term) (vgl. [Pru06]).

Da die Anzahl der Möglichkeiten schon für relativ kleine n sehr groß werden kann, ist ein solches Vorgehen nicht praktisch anwendbar. Alle beschriebenen Verfahren verwenden demnach nur Heuristiken, die bezogen auf die im Folgenden beschriebenen Gütemaße akzeptable Ergebnisse erzielen.

6.1.4 Gütemaß

Liegt eine Partitionierung als Ergebnis einer Clusteranalyse vor, so lässt sich mithilfe des Varianzkriteriums die Güte eines jeden Clusters messen. Die Heterogenität eines Clusters **A** ist die Summe der quadratischen Abstände

$$\sum_{i \in \mathbf{A}} \|x_i - \bar{x}_A\|^2, \tag{6.8}$$

wobei $\bar{x}_A$ der Gruppenzentroid ist:

$$\bar{x}_A = \frac{1}{n_A} \sum_{i \in \mathbf{A}} x_i. \tag{6.9}$$

Um nun die Heterogenität einer kompletten Clusterlösung zu erhalten, wird die Summe aller oben erwähnten Heterogenitäten der einzelnen Cluster gebildet.

Eine weitere Schwierigkeit ist die Bewertung einer gegebenen Clusterlösung bezogen auf alle möglichen Lösungen. Wie schon erwähnt, ist es für realistische Objektanzahlen unmöglich, eine Clusterlösung mit allen möglichen zu vergleichen. Um dieses Problem zu umgehen und trotzdem eine einigermaßen verlässliche Aussage über die Qualität einer Lösung zu erhalten, werden diese einfach mit einer Anzahl zufällig ausgewählter Clusterlösungen (vgl. [Pru06]) verglichen.

6.2 Standardclusterverfahren

In diesem Abschnitt werden verschiedene Standardclusterverfahren vorgestellt. Dabei wird zunächst zwischen hierarchischen und partitionierenden Verfahren unterschieden, wobei ein Überblick über die verbreitetsten Verfahren dieser beiden Klassen gegeben wird.

Anschließend wird ein kurzer Einblick in die Problemstellung des Clusterns bei kategorialen Daten gegeben, woraufhin abschließend die Idee des *Fuzzy Clusterns* am Beispiel *Fuzzy c-means* angesprochen wird.

6.2.1 Hierarchische Clusterverfahren

Hierarchische Clusterverfahren erstellen Cluster sukzessiv durch vorher erstellte Cluster. Dabei wird ein Distanzmaß benutzt, um in einer gegebenen Aufteilung Cluster zu kombinieren oder zu trennen, um so neue Cluster zu erstellen. Hier kann die Wahl des Distanzmaßes entscheidend für die Qualität der Lösung sein.

Es wird nun zunächst eine Unterscheidung zwischen hierarchischen Verfahren, die agglomerativ und solchen, die divisiv arbeiten, vorgenommen. Abschließend wird außerdem die Vorgehensweise beim dichtebasierten hierarchischen Clustern vorgestellt, die sich keiner dieser beiden Klassen direkt zuordnen lässt. Für die Erstellung dieses Abschnitts wurde hauptsächlich auf [Ach04] zurückgegriffen.

Agglomerative Verfahren

Agglomerative Verfahren arbeiten bottom-up, d. h. initial sind alle n Objekte einelementigen Clustern zugeordnet. Diese Cluster werden dann sukzessiv vereint, bis sich alle Objekte in einem einzigen Cluster befinden oder vorher eine andere definierte Abbruchbedingung erreicht wurde.

Es wird zunächst der HACM-Algorithmus als Standardvorgehensweise bei agglomerativem Clustern vorgestellt und anschließend ein Blick auf verschiedene Möglichkeiten zur tatsächlichen Durchführung dieses Algorithmus geworfen.

HACM-Algorithmus Der HACM-Algorithmus1 (hierarchical agglomerative clustern methods) gibt das grundsätzliche Verfahren des agglomerativen Clusterns an.

Algorithmus 1 HACM-Algorithmus

Input: n Objekte
Output: eine Einordnung der n Objekte in Cluster

```
1: Ordne jedes Objekt einem einelementigen Cluster zu
2: repeat
3:    for all Cluster A do
4:       for all Cluster B ≠ A do
5:          Bestimme die Interclusterdistanz zwischen A und B
6:       end for
7:    end for
8:    Verschmelze die Cluster A und B, deren Interclusterdistanz minimal
      ist
9: until Abbruchbedingung erreicht
```

Mögliche Abbruchbedingungen sind das *number criterion (Anzahlkriterium)* oder das *distance criterion (Distanzkriterium)*. Beim *number criterion* werden solange neue Cluster erstellt, bis eine bestimmte Anzahl k an Clustern erreicht wurde, wobei $1 \leq k \leq n$.

Beim *distance criterion* werden so lange neue Cluster erstellt, bis die aktuelle minimale Interclusterdistanz größer als ein bestimmter Wert d ist, wobei $d \leq d_{end}$ für d_{end} = Interclusterdistanz der letzten beiden Cluster.
Im Folgenden werden verschiedene Verfahren zur Berechnung der Interclusterdistanz in Schritt 5 des HACM-Algorithmus vorgestellt.

Single Link Das Single Link Verfahren ist das bekannteste Verfahren zur hierarchischen Clusteranalyse. Für ein beliebiges Distanzmaß $dist(a, b)$ zur Berechnung der Distanz zweier Objekte a und b ist die Interclusterdistanz zwischen zwei Clustern $\mathbf{A}$ und $\mathbf{B}$ definiert als:

$$dist_{SL}(\mathbf{A}, \mathbf{B}) := min_{(a \in \mathbf{A}, b \in \mathbf{B})} dist(a, b). \tag{6.10}$$

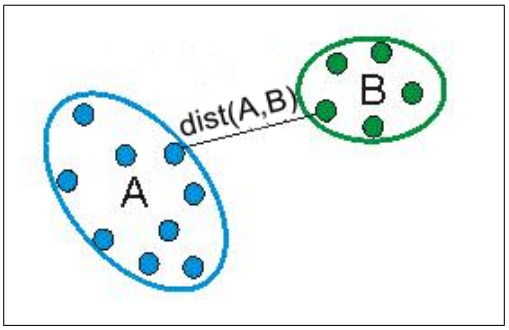


Abbildung 6.1: Single Link: Darstellung der Interclusterdistanz

Mithilfe dieses Verfahrens kann eine Clusterlösung in Laufzeit $O(n^2)$ bei einem Speicherbedarf von $O(n)$ erzeugt werden [Ach04].

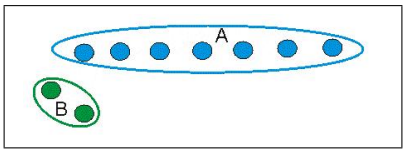


Abbildung 6.2: Single Link: Problem der Verkettung

Ein Nachteil dieses Verfahrens ist das Problem der Verkettung (*chaining*), d. h. dass dieses Verfahren bei großen Datenmengen dazu neigt, kettenförmige Cluster zu erstellen, da nur wenige Objekte erforderlich sind, um zwei Cluster zu vereinen (siehe Abb. 6.2).

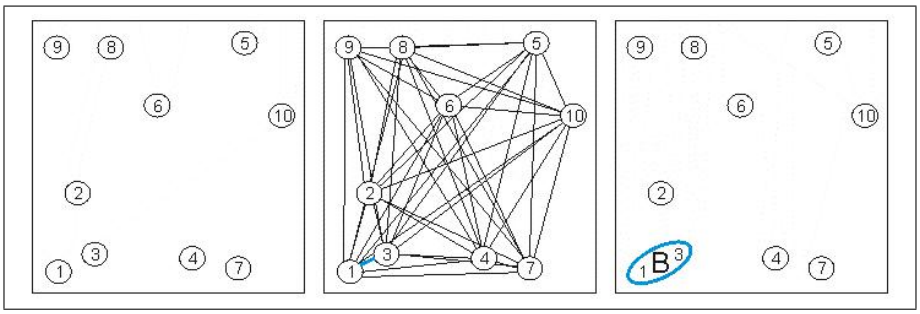


Abbildung 6.3: Single Link: 1. Durchlauf

Hier werden an einem einfachen Beispiel einige Schritte des Algorithmus 1 auf Seite 101 mit dem Single Link Verfahren zur Berechnung der Interclusterdistanzen gezeigt. Die Eingabe besteht aus zehn Objekten ($[1], [2], \ldots, [10]$), die in Cluster eingeteilt werden sollen.

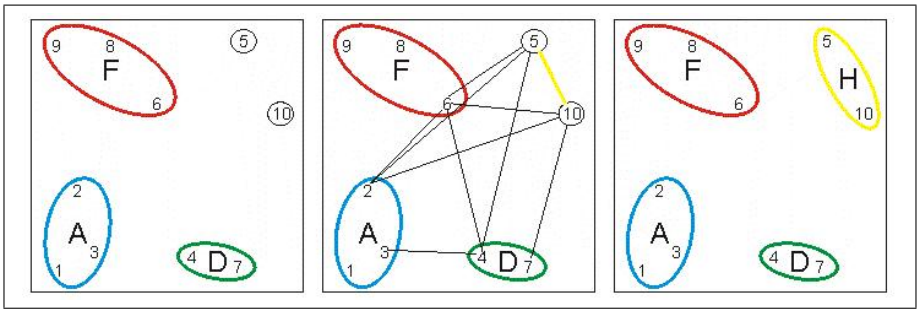


Abbildung 6.4: Single Link: 6. Durchlauf

Initial (siehe Abb. 6.3 a) sind alle Objekte in einzelnen Clustern, woraufhin die Interclusterdistanzen bestimmt werden (siehe Abb. 6.3 b). Dabei ist die Distanz zwischen den Clustern *1* und *3* minimal, sodass diese Cluster zu dem Cluster *B* vereinigt werden (siehe Abb. 6.3 c).

Nach fünf Durchläufen sind die Objekte den Clustern *5*, *10*, **A**, **D** und **F** zugeordnet (siehe Abb. 6.4 a). Nun werden erneut die Interclusterdistanzen berechnet, wobei dieses Mal die Distanz zwischen den Clustern *5* und *10* minimal ist (siehe Abb. 6.4 b). Diese beiden Cluster werden daraufhin vereinigt (siehe Abb. 6.4 c).

Der Algorithmus arbeitet nun maximal so lange weiter, bis alle Elemente in einem Cluster sind.

Complete Link Das Complete Link Verfahren vermeidet die Probleme der Verkettung beim Single Link Verfahren, da es für die Vereinigung zweier Cluster nicht ausreichend ist, dass zwei Objekte der beteiligten Cluster eine geringe Distanz zueinander haben. Für ein beliebiges Distanzmaß $dist(a, b)$ zur Berechnung der Distanz zweier Objekte a und b ist die Interclusterdistanz zwischen zwei Clustern $\mathbf{A}$ und $\mathbf{B}$ definiert als:

$$dist_{CL}(\mathbf{A}, \mathbf{B}) := min_{(a \in \mathbf{A}, b \in \mathbf{B})} dist(a, b). \tag{6.11}$$

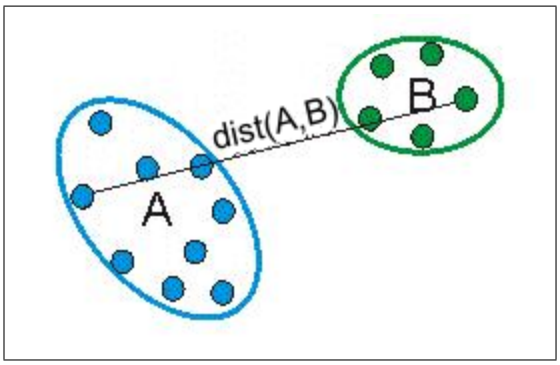


Abbildung 6.5: Complete Link: Darstellung der Interclusterdistanz

Auch mit diesem Verfahren kann eine Clusterlösung in Laufzeit $O(n^2)$ bei einem Speicherbedarf von $O(n)$ erzeugt werden. Allerdings hat dieser Algorithmus in Bezug auf seine Effektivität unbefriedigende Ergebnisse geliefert [Ach04]. Das Complete Link Verfahren tendiert dazu, eine größere Anzahl kleiner, stark voneinander abgegrenzter Cluster zu generieren.
Im Folgenden werden am selben Beispiel wie oben einige Schritte des HACM Algorithmus mit dem Complete Link Verfahren zur Berechnung der Interclusterdistanzen gezeigt. Dabei entspricht der erste Durchlauf dem ersten Durchlauf beim Single Link Verfahren (siehe Abbildung 6.3 auf der vorherigen Seite).

Auch nach dem vierten Durchlauf entspricht die Zuordnung noch der Zuordnung des Single Link Verfahrens. Nun würden beim Single Link Verfahren die Cluster *6* und **G** vereinigt, da der Abstand der Objekte [6] und [8] minimal ist. Nach dem Complete Link Verfahren ist allerdings die Distanz zwischen den Clustern *5* und *10*, repräsentiert durch die Distanz zwischen den Objekten [5] und [10], kleiner als die zwischen den Clustern *6* und **G**, repräsentiert durch die Distanz zwischen den Objekten [6] und [9], sodass die Cluster *5* und *10* vereinigt werden. Dieser Ablauf ist in der folgenden Abbildung zu sehen.

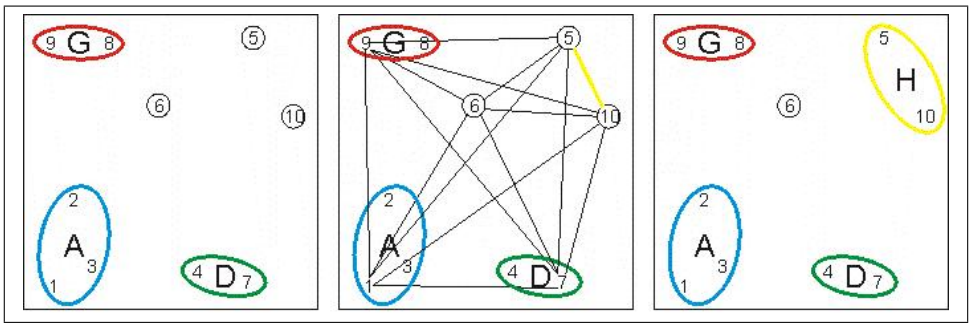


Abbildung 6.6: Complete Link: 5. Durchlauf

Dieser Algorithmus arbeitet nun maximal so lange weiter, bis alle Elemente in einem Cluster sind.

Average Link Average Link ist ein weiteres Verfahren zur Bestimmung der Interclusterdistanzen. Hier ist, für ein beliebiges Distanzmaß $dist(a, b)$ zur Berechnung der Distanz zweier Objekte a und b, die Interclusterdistanz zwischen zwei Clustern $\mathbf{A}$ und $\mathbf{B}$ definiert als:

$$dist_{AVG}(\mathbf{A}, \mathbf{B}) := \frac{1}{|\mathbf{A}|\,|\mathbf{B}|} \sum_{a \in \mathbf{A}} \sum_{b \in \mathbf{B}} dist(a, b). \tag{6.12}$$

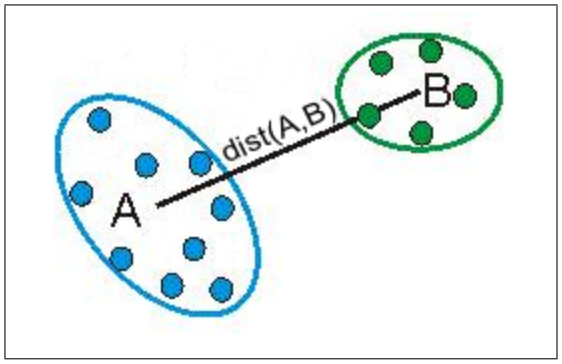


Abbildung 6.7: Average Link: Darstellung der Interclusterdistanz

Dieser Algorithmus generiert eine Struktur, die zwischen den Strukturen der Single und des Complete Link Verfahren liegt. Für dieses Verfahren sind nur Algorithmen bekannt, die unter bestimmten Voraussetzungen eine Laufzeit von $O(n^2)$ bei einem Speicherbedarf von $O(n)$ haben [Ach04].

Divisive Verfahren

Divisive Verfahren arbeiten top-down, d. h. initial sind alle n Objekte einem einzelnen Cluster zugeordnet. Dieser Cluster wird schrittweise zerlegt, bis sich alle Objekte in einelementigen Clustern befinden oder vorher eine Abbruchbedingung erreicht wurde.

Hier können die verschiedenen Methoden des agglomerativen Clusterns angepasst übernommen werden. Allerdings sind diese Verfahren sehr zeitintensiv, da in jedem Schritt alle $O(2k)$ möglichen Zerlegungen eines k-elementigen Clusters in zwei Subcluster untersucht werden müssen (vgl. [Ach04]). Aus diesem Grund werden diese Verfahren hier nicht näher untersucht.

Repräsentation

Die Lösung des hierarchischen Clusterns wird durch eine Baumstruktur, einem sogenannten *Dendrogramm*, repräsentiert. Dabei entspricht jeder Knoten einem Cluster, wobei die n Blätter des Dendrogramms einelementige Cluster mit den einzelnen Objekten sind und die Wurzel ein Cluster darstellt, der die gesamte Objektmenge enthält. Die inneren Knoten repräsentieren Cluster, die die Vereinigung ihrer beiden Kindknoten ist. Dabei werden die Kanten zwischen den einzelnen Knoten proportional zu der Distanz der durch sie verbundenen Cluster dargestellt.

Ein Dendrogramm repräsentiert zunächst eine Menge von mehreren verschiedenen Zuordnungen der Objekte in Cluster. Um eine konkrete Clusterlösung zu wählen, wird an einer festen Tiefe h ein horizontaler Schnitt durch das Dendrogramm gelegt, sodass die so entstandenen Teilbäume Clustern entsprechen, dessen Intraclusterdistanz $\leq h$ ist; hier kann die Wahl von h entscheidend für die Qualität der Lösung sein.
Die Abbildung 6.8 ist das Ergebnis des obigen Beispiels (siehe Abbildung 6.3 auf Seite 103) mit dem Single Link Verfahren. Die 10 Objekte (*[1],[2],. . . ,[10]*) sind zunächst als Blätter gegeben und dann auf verschiedenen Tiefen den Clustern (**A,B,. . . ,H**) zugeordnet; z. B. ist hier der Knoten **A** eine Vereinigung der Cluster *2* und **B** mit den Objekten (*1,2,3*). An der Länge der einzelnen Kanten lässt sich z. B. erkennen, dass die Distanz zwischen den Clustern **A** und **B** drei ist (d. h. die minimale Distanz des Objekts *2* zu einem Objekt aus **B** ist drei). Es wurde $h = 4$ gewählt, sodass die konkrete Lösung aus den Clustern **A,D,F,H** besteht.

Ein Nachteil dieser Form der Darstellung ist, dass der Baum bei großen Datenmengen sehr unübersichtlich werden kann.

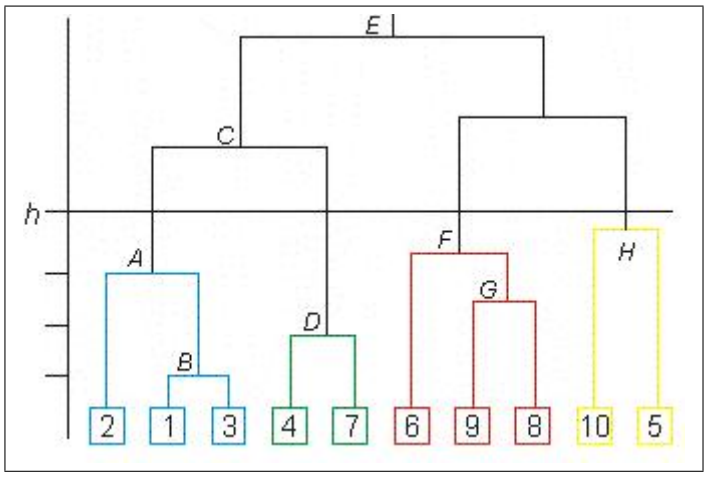


Abbildung 6.8: Beispiel eines Dendrogramms

Dichtebasierte Verfahren

Im Gegensatz zu der Darstellung von Clustern in Form eines Dendrogramms können Cluster auch als Gebiete im d-dimensionalen Raum angesehen werden, in denen eine hohe Objektdichte herrscht. Diese sind durch Gebiete, in denen Objekte weniger dicht beieinander liegen, getrennt. Dichtebasierte Clusterverfahren identifizieren solche dichten Gebiete im Raum, indem sie Objekte so lange um Nachbarn erweitern, wie die Dichte ausreichend groß bleibt. Eine andere Sichtweise als [Ach04] auf diesen Ansatz bietet [Kri02].

Grundbegriffe Seien *o,p,q* Objekte aus einer Objektmenge $\mathbf{O}$, $\epsilon \in \mathbb{R}^+$, $MinPts \in \mathbb{N}^+$ und $dist(a,b)$ ein beliebiges Distanzmaß für zwei Objekte a und b.

- **ϵ-Umgebung eines Objekts:** Die ϵ-Umgebung eines Objekts p umfasst alle Objekte q, deren Distanz zu p kleiner ϵ ist:

$$\mathbf{N}_\epsilon(p) = \{q \in \mathbf{O} | dist(p,q) \leq \epsilon\}\,. \tag{6.13}$$

- **Kernobjekt:** Ein Objekt $p \in \mathbf{O}$ ist Kernobjekt, wenn $|\mathbf{N}_\epsilon(p)| \geq$ MinPts.

- **direkt dichte-erreichbar:** Ein Objekt $q \in \mathbf{O}$ ist direkt dichte-erreichbar von $p \in \mathbf{O}$, wenn $q \in \mathbf{N}_\epsilon(p)$ und p Kernobjekt ist.

- **dichte-erreichbar:** Ein Objekt $q \in \mathbf{O}$ ist dichte-erreichbar von $p \in \mathbf{O}$, wenn es eine Kette direkt dichte-erreichbarer Objekte zwischen p und q gibt.

- **dichte-verbunden:** Ein Objekt $q \in \mathbf{O}$ ist dichte-verbunden mit einem Objekt $p \in \mathbf{O}$, wenn beide dichte-erreichbar von einem dritten Objekt $o \in \mathbf{O}$ sind.

Definition eines Cluster bei dichtebasiertem Clustern Mit diesen Begriffen lässt sich ein Cluster $\mathbf{A}$ als nicht-leere Teilmenge von $\mathbf{O}$ definieren, die folgende Eigenschaften erfüllt:

- **Maximalität:** $\forall p, q \in \mathbf{O} : p \in \mathbf{A}, q$ dichte-erreichbar von $p \Rightarrow q \in \mathbf{A}$

- **Verbundenheit:** $\forall p, q \in \mathbf{A} : p$ ist dichte-verbunden mit q

Standardalgorithmus Der Algorithmus 2 gibt das grundsätzliche Verfahren beim dichte-basierten Clustern an. Die Laufzeit liegt bei $O(n \cdot$ Aufwand zur Bestimmung einer ϵ-Umgebung).

Algorithmus 2 Dichtebasiertes Clustern

Input: n Objekte
Output: eine Einordnung der n Objekte in Cluster

```
1: Alle Objekte o ∈ O sind unklassifiziert
2: for all o ∈ O do
3:   if o unklassifiziert then
4:     if |N_ε(o)| ≥ MinPts (o ist Kernobjekt) then
5:       Klassifiziere o (Erstelle neues Cluster O)
6:       for all p ∈ N_ε(o) (direkt dichte-erreichbar) do
7:         if p unklassifiziert then
8:           Klassifiziere p
9:           Füge p zu dem Cluster repräsentiert durch o hinzu
10:          for all {q ∈ O|q ist dichte-erreichbar von p} do
11:            if q unklassifiziert then
12:              Klassifiziere q
13:              Füge q zu dem Cluster repräsentiert durch o hinzu
14:            end if
15:          end for
16:        end if
17:      end for
18:      das gesamte Cluster O zu Kernobjekt o wurde gefunden
19:      alle Objekte in Cluster O sind klassifiziert
20:    end if
21:  end if
22: end for
```

Das Ergebnis wird dabei nicht in Form eines Dendrogramms repräsentiert, sondern durch ein *Erreichbarkeitsdiagramm*, einer Form der Darstellung, die auch bei großen Datenmengen noch übersichtlich ist. Dafür wird der Begriff des *Erreichbarkeitswerts* benötigt, der für zwei Objekte p und q, wobei q Kernobjekt, die kleinste Distanz ist, bei der p von q aus direkt dichte-erreichbar ist.

Nun sind dichte Gebiete, also Cluster, im *Erreichbarkeitsdiagramm* durch kleine *Erreichbarkeitswerte* erkennbar, während Übergänge zwischen einzelnen Clustern durch große *Erreichbarkeitswerte* gekennzeichnet sind; Cluster sind somit als Vertiefungen im Graphen erkennbar. In der Abbildung 6.9 ist ein Beispiel eines *Erreichbarkeitsdiagramms* mit zwei Dimensionen gezeigt.

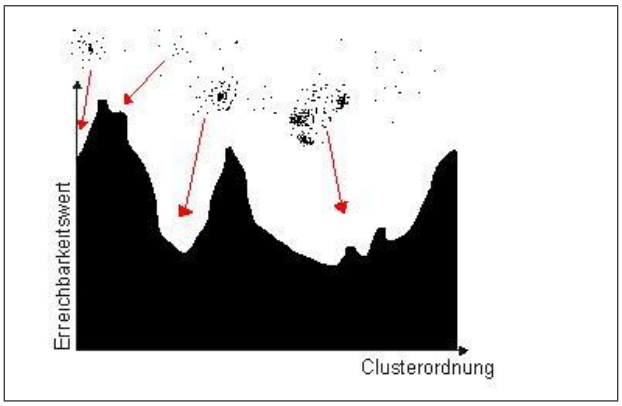


Abbildung 6.9: Beispiel eines Erreichbarkeitsdiagramms

6.2.2 Partitionierende Verfahren

Partitionierende Clusterverfahren erstellen, im Gegensatz zu den hierarchischen Verfahren, nur genau eine Partition der Objektmenge. Dafür wird ein Parameter k vorgegeben, der angibt, in wie viele disjunkte Cluster die Objektmenge zerlegt wird. Jedes der k Cluster muss mindestens ein Objekt enthalten und jedes Objekt muss genau einem Cluster zugeordnet sein.

Die grundsätzliche Vorgehensweise ist bei allen diesen Verfahren identisch. Es werden zunächst (zufällig) k initiale Cluster-Repräsentanten gewählt, die so lange iterativ modifiziert werden, bis ein lokales Optimum der Partitionsgüte abhängig von der initialen Zerlegung gefunden wurde.

Das Ergebnis eines partitionierenden Clusterverfahrens hängt stark von der Wahl des Werts k sowie der initialen Zerlegung ab und bietet daher

zunächst nur lokale Optima. Mehrere Durchgänge der einzelnen Verfahren ermöglichen es, lokale Optima zu verlassen und globale Optima zu finden. Im Folgenden werden nun drei bekannte Verfahren des partitionierenden Clusterns (*k-means*, *PAM* und *CLARA*) vorgestellt.

k-means

Das *k-means* Verfahren ist ein einfaches und schnelles Verfahren zur Durchführung eines partitionierenden Clusterns und wird daher oft benutzt, um einen ersten Einblick in den Aufbau einer großen Datenmenge zu bekommen.

Algorithmus 3 *k-means*

Input: n Objekte, $k \in \mathbb{N}^+$, $\epsilon \geq 0$
Output: eine Einordnung der n Objekte in k Cluster

1: Wähle zufällig k Objekte als Clusterzentren m^j
2: **repeat**
3: **for all** $o \in \mathbf{O}$ **do** {gehe alle Objekte durch}
4: $dist_{min} = \infty$
5: **for** $j = 1$ to k **do** {gehe alle Cluster durch}
6: **if** $dist(o, m^j) < dist_{min}$ **then** {m^j ist das Zentrum, mit der bisher geringsten Distanz}
7: $dist_{min} = dist(o, m^j)$
8: $m = m^j$
9: **end if**
10: **end for**
11: Ordne o dem Cluster $\mathbf{S_m}$ zu
12: **end for**
13: **for** $j = 1$ to k **do**
14: Berechne das *Clusterzentrum* m^j neu
15: **end for**
16: Bestimme die *Intraclustervarianz* V neu
17: **until** $\Delta V \leq \epsilon$

Für Objekte $x_1, \ldots, x_n$ einer Objektmenge $\mathbf{O}$, $k \in \mathbb{N}^+$, Clustern $\mathbf{S_1}, \ldots, \mathbf{S_k}$ mit diesen Objekten und einem beliebigen Distanzmaß $dist(a, b)$ für zwei Objekte a und b sind folgende Begriffe definiert:

Definition 6.1 (Clusterzentrum) *Das Zentrum eines Clusters ist definiert als:*

$$m^j := \frac{1}{|\mathbf{S_j}|} \sum_{x_i \in \mathbf{S_j}} x_i.$$

Definition 6.2 (Intraclustervarianz) *Die Distanz der einzelnen Elemente eines Cluster vom Clusterzentrum wird durch die* Intraclustervarianz *angegeben:*

$$V := \sum_{j=1}^{k} \sum_{i \in \mathbf{S_j}} dist(x_i, m^j)^2.$$

Der k-means Algorithmus minimiert die *Intraclustervarianz* in $O(t \cdot k \cdot n)$ bei t Iterationen (wobei $(t << n)$) [Wan06], abhängig von der initialen Zerlegung. Allerdings kann kein globales Optimum garantiert werden. Ein Beispiel zum Ablauf dieses Algorithmus bietet z. B. [Wan06].

PAM

Der *PAM* (Partitioning Around Medoids) Algorithmus arbeitet ähnlich wie das oben beschriebene *k-means*-Verfahren. Während bei *k-means* die einzelnen Cluster durch Clusterzentren m^j repräsentiert werden, die Mittelwerte der einzelnen Objekte der Cluster sind, werden beim *PAM* die Clusterzentren als Median bestimmt. Damit werden in diesem Fall die Cluster durch eines ihrer echten Objekte repräsentiert, damit ist dieses Verfahren weniger anfällig gegen extreme Werte als das *k-means*-Verfahren.

Allerdings ist dieses Verfahren für große Datenmengen eher ungeeignet, da bei jeder Iteration $O(k(n-k)^2)$ Schritte notwendig sind [Wan06].

CLARA

Der *CLARA* (Clustering LARge Applications) Algorithmus erweitert das *PAM*-Verfahren, sodass es auch bei großen Datenmengen effizient ist. Dafür werden mehrere Stichproben aus der Objektmenge ausgewählt und *PAM* auf jede dieser Stichproben angewandt, um die beste Clusterlösung, die bei verschiedenen *PAM* Durchläufen gefunden wurde, auszugeben.

Die Effizienz dieses Verfahrens hängt dabei stark von der Beschaffenheit der Stichproben ab, da eine gute Clusterlösung einer Stichprobe nicht notwendigerweise auch eine gute Clusterlösung der ganzen Objektmenge darstellt. Daher müssen die Stichproben ausreichend groß sein und Objekte enthalten, die die gesamte Menge gut repräsentieren.

Das *CLARA*-Verfahren liefert dabei zunächst auch nur ein lokales Optimum. Als Alternative steht der *CLARANS* (clustering large application based upon randomized search) Algorithmus zur Verfügung, der durch Randomisierung lokale Optima verlassen kann.

6.2.3 Clustern bei kategorialen Daten

Kategoriale Daten sind Daten, bei denen die einzelnen Werte in verschiedene Kategorien fallen, die jeweils nur eine begrenzte Auswahl an Werten haben, z. B. *Land*, *Geschlecht* oder *Studiengang*. In diesen Fällen lassen sich die normalen Clusterverfahren nicht anwenden, da bei kategorialen Daten z. B. kein Mittelwert definiert ist.

Es stehen verschiedene spezielle Algorithmen wie *k-modes* (ähnlich *k-means*, benutzt statt des Mittelwerts das am häufigsten vorkommende Element) zur Verfügung, auf die an dieser Stelle allerdings nicht weiter eingegangen wird.

6.2.4 Fuzzy Clustern

Im Gegensatz zu normalen Verfahren wird Objekten beim *Fuzzy Clustern* eine prozentuale Zugehörigkeit und keine scharfe Zugehörigkeit zu Clustern zugewiesen. Da es auch in den Ausgangsdaten i. d. R. keine scharfen Grenzen zwischen Clustern gibt, entspricht diese Zuordnung eher den Ausgangsdaten. Das Ergebnis für n Objekte und k Cluster ist eine Matrix U mit Werten u_{ij}, wobei $1 \leq i \leq n$ und $1 \leq j \leq k$, für die gilt:

1. $0 \leq u_{ij} \leq 1$
2. $0 < \frac{1}{n} \sum_{i=1}^{n} u_{ij} < 1$ (kein Cluster darf leer sein)
3. $\sum_{j=1}^{k} u_{ij} = 1$ (jedes Objekt muss komplett zugewiesen sein)

Fuzzy c-means

Der *Fuzzy c-means* Algorithmus ist eine einfache Abwandlung des bekannten *k-means* Algorithmus und der verbreitetste *Fuzzy Clustern* Algorithmus.

6.3 Spezielle Clusterverfahren

Um auf der einen Seite einen Eindruck von der Vielfältigkeit verschiedener Clusterverfahren zu gewinnen und um auf der anderen Seite konkrete, spezialisierte Verfahren in der Projektgruppe zur Verfügung zu haben, wird im

Algorithmus 4 *Fuzzy c-means*

Input: n Objekte, $k \in \mathbb{N}^+$, $\epsilon \geq 0$
Output: eine prozentuale Zuordnung der n Objekte zu den k Clustern

1: Wähle zufällig k Objekte als Clusterzentren m^j
2: **repeat**
3: **for all** $i \in \mathbf{O}$ **do** {gehe alle Objekte durch}
4: **for** $j = 1$ to k **do** {gehe alle Cluster durch}
5: Berechne $dist(o, m^j)$
6: $u_{ij} = \frac{1}{\sum_{r=1}^{k} \frac{d(x_i, m^j)}{d(x_i, m^r)}}$
7: **end for**
8: **end for**
9: **for** $j = 1$ to k **do**
10: Berechne das *Clusterzentrum* m^j neu: $m^j = \frac{\sum_{i=1}^{n} u_{ij}^2 x_i}{\sum_{i=1}^{n} u_{ij}^2}$
11: **end for**
12: **until** $\Delta U \leq \epsilon$

folgenden Abschnitt ein konkretes Clusterverfahren beschrieben. Abschließend folgt ein kurzer Einblick in die Parallelen des Clusterns mit k-means und der Hauptkomponentenanalyse, die dazu verwendet werden können, k-means zu verbessern.

6.3.1 Advanced-Maximum-Linkage Clustering

Der *Maximum-Linkage-Algorithmus* (MLA) wurde 2001 von Matthias Zerbst im Rahmen seiner Dissertation am Fachbereich Statistik der Universität Dortmund entwickelt. Er ist verwandt mit dem weiter oben beschriebenen Single Link Algorithmus, erzeugt jedoch keine Ordnungsrelation zwischen den Clusterlösungen auf verschiedenen Ebenen (vgl. [Zer01]).

Der *Advanced-Maximum-Linkage-Algorithmus* (AMLA) kann als Verallgemeinerung von MLA verstanden werden, da durch die Steuerung eines Parameters, MLA als Spezialfall erreicht werden kann (vgl. [TZ02]). Im Folgenden wird zunächst der konkrete Algorithmus beschrieben, worauf eine kurze Skizzierung der Anwendungsgebiete folgt. Abschließend werden die Vor- und Nachteile des Verfahrens angeführt.

Beschreibung

Als Eingabe nimmt der Algorithmus, neben den eigentlichen Daten $\mathbf{O} = \{x_1, \ldots, x_n\}$ die Clusteranzahl q und den Parameter $\rho \in [0, 1]$ entgegen. $\{x_1, \ldots, x_r\}$ sei die Menge der nicht doppelt vorkommenden Elemente in $\mathbf{O}$ und $h_1, \ldots, h_r$ deren jeweilige Häufigkeit. Zur Verdeutlichung wird zunächst ein Spezialfall des Algorithmus vorgestellt, mit $\rho = 0$ als Teil der Eingabe. Umgangssprachlich bedeutet dies, dass die Häufigkeiten der zu clusternden Objekte ignoriert werden.
Im *ersten Schritt*, wird eine Distanzmatrix Δ erstellt, mit

$$\delta_{ij} = \|x_i - x_j\|_2, \tag{6.14}$$

Das Maximum dieser Werte wird ermittelt und die zugehörigen Elemente i und j werden zur Menge $\mathbf{Z^{(1)}}$ der Zentren des ersten Schrittes. Die Indizes i und j werden zur Menge $\mathbf{Q^{(1)}}$ (Indizes der Zentren) hinzugefügt, während die Indizes aller anderen Objekte die Menge $\mathbf{R^{(1)}}$ (Restindizes) bilden.

Im *zweite Schritt* wird so lange der folgende Ablauf wiederholt, bis $|Z| = q$ gilt: Schlage für jedes Objekt x_i, dessen Index in $\mathbf{R^{(n)}}$ ist, seine Werte δ_{ij} für $j \in \mathbf{Q^{(n)}}$ nach und bestimme das Minimum dieser Distanzen. Gesucht wird also für jedes noch nicht in $\mathbf{Z^{(n)}}$ vorhandene Element das nächstgelegene Zentrum und der Abstand zu diesem. Bestimme anschließend über alle überprüften Objekte das Maximum der bestimmten Minima und füge das resultierende Objekt zur Menge $\mathbf{Z^{(n+1)}}$ hinzu und aktualisiere $\mathbf{Q^{(n+1)}}$ und $\mathbf{R^{(n+1)}}$. Das Objekt, welches als Nächstes zur Menge der Zentren hinzugefügt wird, ist also jenes, das maximal entfernt von allen anderen Zentren ist.

Schritt drei ordnet schließlich die Objekte mit den Indizes aus $\mathbf{R}$ den nächsten Zentroiden zu, wodurch eine Partitionierung der Menge erstellt wird. Durch das beschriebene Vorgehen wird erreicht, dass die Zentroiden maximalen Abstand zueinander einhalten, die resultierenden Cluster also so separiert wie möglich sind. Die hierbei bestimmten Clusterzentren entsprechen nicht den wirklichen Clusterzentren, also dem Mittelwert der im Cluster enthaltenen Objekte. Diese müssen nach Abschluss des Algorithmus, falls gewünscht, berechnet werden.
Abbildung 6.10 zeigt den ersten Schritt des AMLA ($q = 3$), mit der Bestimmung der maximalen Distanz und der Ableitung der ersten beiden Clusterzentren. In Abbildung 6.11 wird die Bestimmung der Minima visualisiert. Abbildung 6.12 zeigt die Feststellung des Maximums dieser Menge, welches zum dritten Clusterzentrum führt. Anschließend erfolgt die Zuordnung der restlichen Objekte zu den Clusterzentren.

Die Spezialität des skizzierten Verfahrens liegt nun darin, dass mit dem Eingabeparameter ρ der Einfluss mehrfach vorkommender Objekte berücksichtigt werden kann.

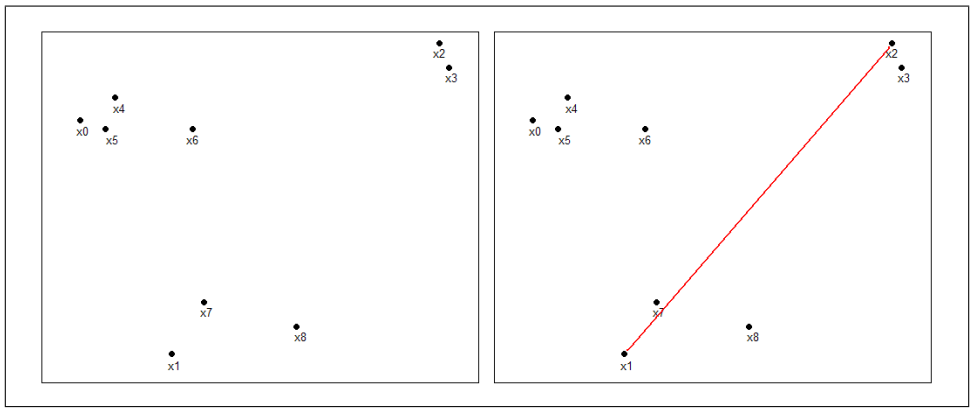


Abbildung 6.10: Advanced-Maximum-Linkage Clustering-Algorithm: Schritt 1

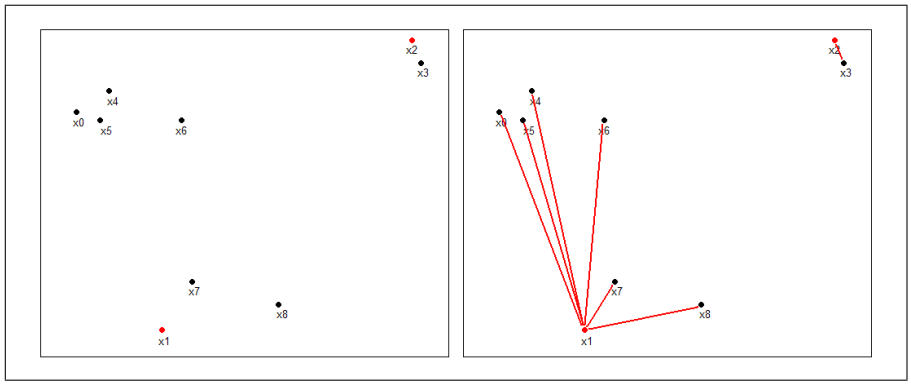


Abbildung 6.11: Advanced-Maximum-Linkage Clustering-Algorithm: Schritt 2

Definiere dazu zunächst

$$\lambda_i := \frac{h_i}{n},\ i \in \{1, \ldots, r\} \tag{6.15}$$

als einfache relative Häufigkeit für jeden Datenwert x_i, der h_i-mal auftaucht. Des Weiteren sei

$$\lambda_{ij} := \frac{h_i \cdot h_j}{n^2},\ i, j \in \{1, \ldots, r\} \tag{6.16}$$

die gegenseitige relative Häufigkeit der Datenwerte x_i und x_j mit den Häufigkeiten h_i und h_j. Zur Gewichtung der Distanzmatrix definiere die Funktion

$$f(\rho, \lambda) := \rho \cdot \lambda + (1 - \rho),\ \rho \in [0, 1]\ , \tag{6.17}$$

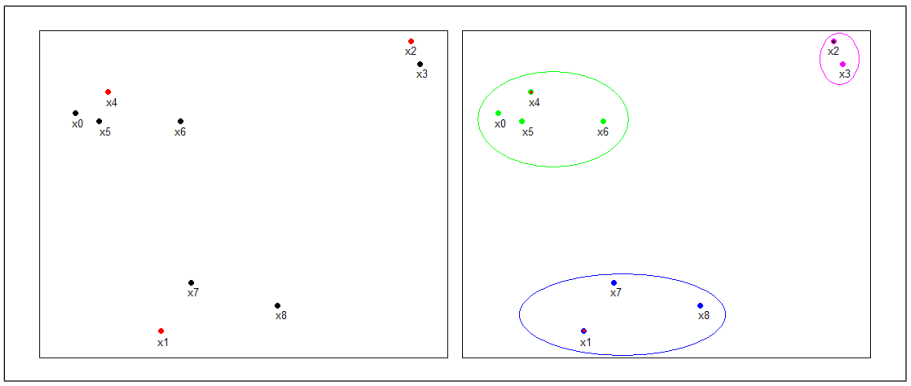


Abbildung 6.12: Advanced-Maximum-Linkage Clustering-Algorithm: Schritt 2/3

die in die Berechnung der Distanzmatrix Δ,

$$\delta_{ij} = f(\rho, \lambda_{ij}) \cdot \|x_i - x_j\|_2 \ , \tag{6.18}$$

genauso wie in die Zuordnung eines Elements i im Schritt 2 zu den Zentren mit eingeht, deren Indizes in $Q^{(s)}$ sind:

$$\delta_{iQ} = f(\rho, \lambda_i) \cdot min_{k \in Q} \|x_i - x_k\|_2 \ , \ i \in R^{(s)}. \tag{6.19}$$

Anwendung

Der beschriebene Gewichtungsfaktor $f(\rho, \lambda)$ ist von besonderer Bedeutung, wenn viele der Datenwerte gleich sind. So wurde das Verfahren für das Clustern von Satellitenaufnahmen entworfen, wo Gebiete anhand der Farbgebung zusammengefasst werden sollten. In dieser konkreten Anwendung ist es üblich, dass ein Farbwert gehäuft vorkommt, wodurch das Verfahren seine Stärken ausspielen kann.

Diskussion

In der konkreten Anwendung wurde das Verfahren für gut befunden, wobei nur für ausgewählte Daten der Parameter ρ von Bedeutung ist. Generell gilt, dass die Laufzeit von AMLA quadratisch mit der Länge der Eingabe wächst. Eine Abgrenzung zu k-means ist, dass keine zufällig gewählten Startwerte Einfluss auf das Ergebnis haben, da das Verfahren vollständig von den Daten bestimmt wird (vgl. [TZ02]).

6.3.2 k-Means Clustern mithilfe der Hauptkomponentenanalyse

Das sehr beliebte Clusterverfahren k-means optimiert die Clusterbildung anhand der Minimierung des folgenden Ausdrucks:

$$J_K = \sum_{k=1}^{K} \sum_{i \in C_k} (x_i - m_k)^2, \tag{6.20}$$

wobei K die Clusteranzahl, C_k die Menge der Objekte in Cluster k und m_k der Mittelwert der Objekte in Cluster k ist. Das Problem bei der Anwendung des Standardverfahrens ist nun, dass der Algorithmus, abhängig von den Startwerten, nicht im globalen, sondern in einem lokalen Optimum landet. In [DH04a] wurde nachgewiesen, dass es einen engen Zusammenhang gibt zwischen dem Clustern mittels k-means und der Hauptkomponentenanalyse. Letzteres ist ein Verfahren, welches diejenigen Achsen im Merkmalsraum berechnet, die maximalen Informationsgehalt tragen. Die Achsen werden dabei absteigend sortiert. Die Daten können daraufhin in einen Raum abgebildet werden, der weniger Dimensionen hat und trotzdem maximal viel Information enthält, oder anders ausgedrückt: Es werden solche Dimensionen vernachlässigt, die wenig Information enthalten.

Praktische Anwendungen der Arbeit von Ding und He werden in [DH04b] angeführt. Zunächst gibt es durch die *Hauptkomponentenanalyse* die Möglichkeit, Schranken anzugeben, die die Qualität einer Clusterlösung durch k-means einschätzbar machen. Darüber hinaus kann mittels der durch die Hauptkomponentenanalyse ermittelten Werte einw 2-Clusterlösung angegeben werden, die nachweislich optimal ist. Durch iteratives Verfeinern gibt es darüber hinaus die Möglichkeit, k-means-Clustern durchzuführen, wodurch das Hauptproblem (die Wahl der initialen Zentroiden) behoben wird. Eine weitere Anwendung besteht darin, eine bestehende k-means-Clusterlösung weiter zu verbessern, indem die Cluster paarweise durch die Hauptkomponentenanalyse erneut einer Clusteranalyse unterzogen werden. Die genannte Arbeit beweist dabei, dass die Clusterlösung dadurch nicht verschlechtert werden kann.

6.4 Fazit

Zusammenfassend ist festzuhalten, dass es eine Vielzahl von Clusterverfahren gibt, die in Pavel eingesetzt werden können. Neben den Standardverfahren sind dies vor allem neuere Entwicklungen, welche moderne Verfahren zu Clusteranalyse nutzen. Auf diese Weise konnten in den letzten Jahren eine Reihe von Verbesserung erreicht und viele nützliche Einsichten gewonnen werden, wie der letzte Abschnitt verdeutlicht.

Die Aufgabe bei der Programmierung von Pavel bestand nun darin, dass für die zu erwartenden Eingabedaten beste mögliche Verfahren auszuwählen und eine möglichst effiziente Implementierung zu finden. [JMF99] zeigt und benennt deutlich, dass es nicht **das** Verfahren zur Clusteranalyse gibt. Darüber hinaus ist und wird es immer eine Frage der Anwendung und der Daten bleiben, welche Methode oder welches Verfahren geeignete Resultate erzeugt. Dies ist auch der Grund, warum es nicht ein fest eingebautes Clusterverfahren in Pavel gibt, sondern viele verschiedene über eine offene Schnittstelle implementiert werden können.

7 Realisierung in Pavel

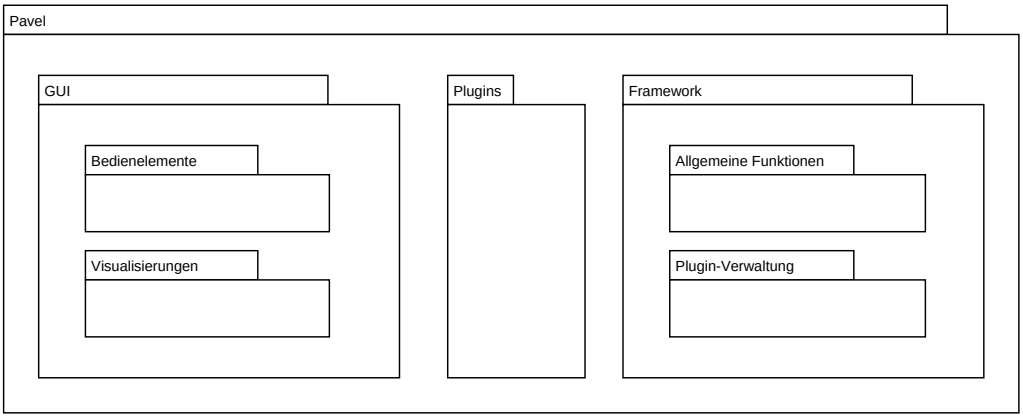


Abbildung 7.1: Die Hauptkomponenten von Pavel

Abbildung 7.1 zeigt eine grobe Übersicht der wesentlichen Komponenten von Pavel. Die GUI teilt sich dabei in einen allgemeinen Part, welcher das Hauptfenster, die Dialogfenster und Steuerelemente enthält und in die Komponenten zur Datenvisualisierung auf. Auf die Strukturen innerhalb der GUI wird in Abschnitt 7.2 näher eingegangen.

Das Framework stellt grundlegende Programmfunktionen wie Controllerklassen und das Datenmodell zur Verfügung. Dazu kommt die PlugIn-Verwaltung, die als dritte Komponente die anwendungsfallspezifischen PlugIns einbindet.

7.1 Datenstrukturen

Der Entwurf geeigneter Datenstrukturen wurde vor allem durch die Ausgabeformate der Anwendungsfälle der Temperierbohrungs- sowie Fräsbahnoptimierung (siehe Abschnitt 4) motiviert. Beide Optimierverfahren produzieren eine Liste von Lösungen, gegeben durch Parameter und Ergebnisse der Simulationen (vgl. Abbildung 3.13 auf Seite 25). Die Lösungen lassen sich

dabei als Vektoren (Punkte) in einem Vektorraum auffassen, der für jeden Parameter und jeden Ergebniswert der Simulationen eine Dimension besitzt. Diese Repräsentation kann gut als Tabelle dargestellt werden, wobei jede Lösung eine Zeile, jede Dimension des Vektorraumes eine Spalte und jede Zelle den numerischen Wert einer Lösung in der entsprechenden Dimension enthält (siehe Abb. 7.2). Verschiedene Dimensionen lassen sich dabei zu Unterräumen gruppieren, wobei im Wesentlichen zwischen *Objective Space* und *Decision Space* unterschieden wird. Der *Decision Space* einer Lösung enthält dabei die Werte, die zur Beschreibung des Phänotyps benötigt werden. Der *Objective Space* beinhaltet die Zielfunktionswerte der Lösung.

Master Space

Decision Space					*Objective Space*		
x_1	y_1	$\cdots$	y_n	d_n	T	V	E
2.83	5.69	$\cdots$	1.68	1.39	4.19	2.54	3.78
4.37	8.25	$\cdots$	0.50	0.01	5.63	3.02	4.40
3.63	2.45	$\cdots$	2.36	1.99	3.41	0.81	5.39
3.02	3.84	$\cdots$	1.76	2.31	4.09	1.17	2.99
$\vdots$					$\vdots$		

Abbildung 7.2: Das logische Datenlayout in Pavel entspricht einer Tabelle: Jede Spalte entspricht einer Dimension, mehrere Spalten werden zu Vektorräumen zusammengefasst. Jede Zeile enthält die numerischen Werte einer einzelnen Lösung.

Implementierung Bei der Implementierung der beschriebenen Datenstruktur mit objektorientierter Modellierung wurde von Anfang an Wert auf Flexibilität und Effizienz gelegt. Obwohl sich für die Datentabelle auf den ersten Blick ein zweidimensionales Array anbietet, genügt dieses lediglich der Forderung nach Effizienz. Um die Beziehungen zwischen Unterräumen angemessen modellieren zu können und das Hinzufügen oder Entfernen von Dimensionen und Lösungen zu ermöglichen, ohne dabei jedes Mal große Datenmengen umkopieren zu müssen, wurde die Tabellenstruktur über mehrere Klassen aufgeteilt. In Abbildung 7.3 sind die diese Klassen dargestellt.

Die kleinste Dateneinheit in Pavel ist der `Point`. Dieser stellt eine der zu untersuchenden Lösungen dar und enthält die numerischen Werte der Lösung. Die Zuordnung der Zahlenwerte zu bestimmten Dimensionen (`Columns`, *Spalten* der Tabelle) des Lösungsraumes geschieht über ein `ColumnSet`.

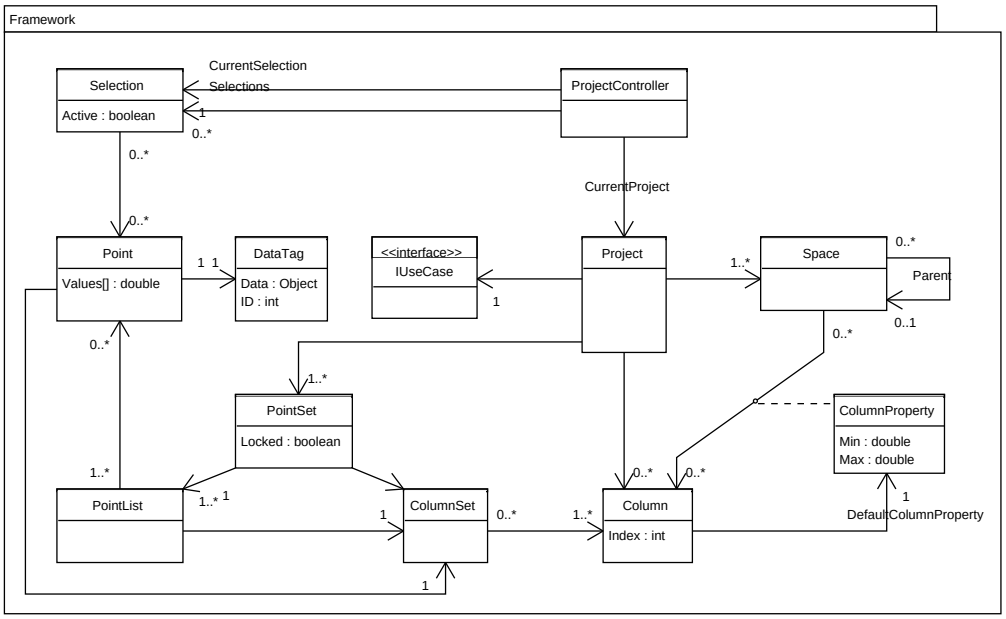


Abbildung 7.3: Die Klassen für die Basisfunktionalität des Frameworks.

Dieses enthält eine Reihe von `Column`s, die in ihrer Anordnung exakt der Anordnung der Werte im `Point` entsprechen. Mehrere `Point`s mit identischem `ColumnSet` werden in einer `PointList`, die über genau dasselbe `ColumnSet` verfügt, zusammengefasst. `PointList`s für unterschiedliche `ColumnSet`s werden in einem `PointSet` vereint, dessen `ColumnSet` dabei eine gemeinsame Untermenge aller `ColumnSet`s der `PointList`en sein muss.

Diese Unterscheidung zwischen `PointList`s und `PointSet`s fußt ausschließlich auf Performancegründen und hat keine Bedeutung für die Bedienung des Programms. Da innerhalb einer `PointList` die Werte-Arrays der `Point`s eine identische Struktur aufweisen, ist ein Zugriff auf sie sehr schnell. Trotzdem ist es oft nötig, `Point`s über unterschiedlichen `ColumnSet`s (und somit unterschiedlicher Struktur der Werte-Arrays) zusammenzufassen. Werden zum Beispiel Daten aus einem Vektorraum A über einen Unterraum B von A geclustert, möchte man sich in der Regel das Ergebnis des Clusterings anzeigen lassen. Dazu werden die Ursprungspunkte über A in einer `PointList` mit `ColumnSet` A, die Clusterrepräsentanten über B in einer `PointList` mit `ColumnSet` B gespeichert und beide `PointList`en zu einem `PointSet` mit `ColumnSet` B zusammengefasst. Dieses `PointSet` kann anschließend über `ColumnSet` B visualisiert werden. Die Zusammenfassung von `PointList`s ohne gemeinsamen Unterraum in einem `PointSet` ist deshalb nicht sinnvoll

(und daher nicht möglich), da Clusteralgorithmen oder Visualisierungen keine `Column` zum Verarbeiten hätten, für die alle enthaltenen `Points` Werte bereit hielten.

Da die `ColumnSet`s lediglich der strukturellen Organisation von `Column`s dienen, sind sie sehr strikt aufgebaut und lassen weder eine Umsortierung noch doppeltes Vorkommen von `Column`s zu. Um diese Beschränkungen umgehen zu können, arbeiten andere Programmteile in der Regel auf `Space`s. In diesen gelten die Beschränkungen der `ColumnSet`s nicht, darüber hinaus bieten `Space`s die Möglichkeit, die Visualisierung einzelner `Column`s anzupassen, indem der darzustellende Wertebereich mithilfe einer `ColumnProperty` eingeschränkt wird.

Weitere Klassen Alle beschriebenen Datenstrukturen werden in einem `Project` zusammengefasst und können über den `ProjectController` geladen oder abgespeichert werden. Der `ProjectController` verwaltet außerdem zentral die `Selection`s, die Mengen von `Point`s darstellen, ohne diese dabei – anders als bei `PointSet`s oder `-List`s – irgendwelchen Beschränkungen zu unterwerfen. Diese zentrale Verwaltung ist nötig, um das Crosshighlighting und die Übergabe von Selektionen an Programmteile zur Weiterverarbeitung zu ermöglichen.

Die in Abbildung 7.3 außerdem dargestellte Klasse `DataTag` dient dazu, einem `Point` beliebige zusätzliche Daten anzuhängen. In Pavel wird dies derzeit genutzt, um Simulationswerte für die Visualisierung eines einzelnen `Point`s bereitzustellen.

7.2 GUI

Abbildung 7.4 zeigt die Hauptklassen der GUI und die wichtigsten Verknüpfungen zum Framework. Die zentrale Klasse ist hierbei `MainWindow`, die alle GUI Elemente des Hauptfensters von Pavel beinhaltet und verschiedene Zugriffe auf zentrale Funktionen erlaubt. Im Folgenden wird der Aufbau der GUI und ihrer wichtigsten Elemente vorgestellt.

Die wichtigsten Elemente in `MainWindow` sind das `PropertyControl` und das `VisualizationWindow`. Das `PropertyControl` ist die zentrale Steuereinheit von Pavel, über die Einstellungen zu den verschiedenen Visualisierungen möglich sind. Je nach ausgewähltem `Space`, `PointSet` und der Visualisierung bietet das `PropertyControl` per Reflection zur Laufzeit Felder an, über die sich die Werte der öffentlichen Eigenschaften der dargestellten Visualisierung verändern lassen.

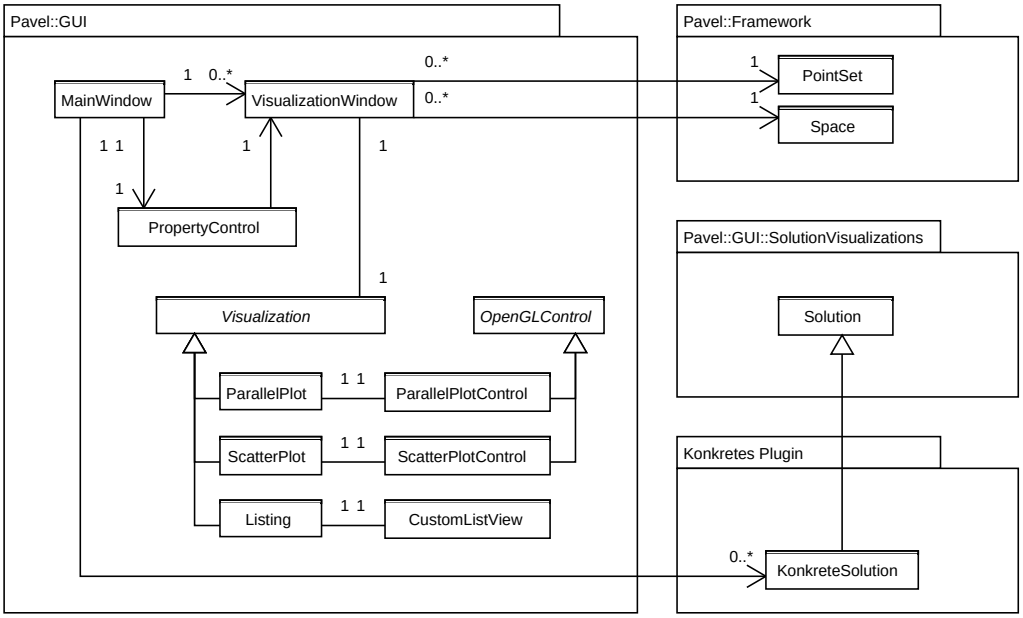


Abbildung 7.4: Die Klassen für die Basisfunktionalität der GUI.

`MainWindow` stellt auch Funktionen zum Öffnen und Schließen eines neuen `VisualizationWindow` bereit. Es lassen sich beliebig viele unterschiedliche Visualisierungen gleichzeitig öffnen. Jedes `VisualizationWindow` enthält eine Visualisierung vom Typ `Visualization`, einen darzustellenden Space und ein PointSet.

Bei `Visualization` handelt es sich um eine abstrakte Klasse, die Basisfunktionalitäten für alle Visualisierungen bereitstellt. Vier solcher Visualisierungen sind in Pavel implementiert, eine einfache tabellarische Liste, ein Scatterplot, ein Parallelplot sowie eine Scattermatrix (nicht dargestellt). Mit Ausnahme der Tabelle wird für die Darstellung aller Visualisierungen OpenGL verwendet, gemeinsam benutzte Funktionalität stellt dabei die abstrakte Klasse `OpenGLControl` bereit.

Zusätzlich zu `PointSet`s lassen sich in Pavel auch einzelne `Point`s genotypisch oder phänotypisch (falls ein entsprechendes PlugIn existiert) darstellen. Diese Aufgabe erfüllen Ableitungen der Klasse `Solution`, Näheres dazu wird in Abschnitt 7.8 erläutert.

7.3 Stereoskopie

Pavel bietet für Visualisierungen in einem dreidimensionalen Raum die Möglichkeit einer stereoskopischen Betrachtung an. Zu diesen Visualisierungen zählt der Scatterplot sowie die phänotypischen Darstellungen der Fräsbahnen und der Temperierbohrungen in den Einzellösungsvisualisierungen. Derzeit beschränkt sich die stereoskopische Darstellung auf die Anaglyphentechnik.

Zur Betrachtung stereoskopischer Darstellungen wird eine Rot-Cyan Brille empfohlen. Eine Rot-Grün Brille ist alternativ möglich, allerdings ist hiermit der Farbeindruck in den Einzellösungsvisualisierungen geringer. Um den dreidimensionalen Eindruck individuell an den Betrachter anzupassen, ist es möglich den Augenabstand zu modifizieren.

Für die Erzeugung der stereoskopischen Ansicht wird die in [Mül05] beschriebene Off-Axis-Methode eingesetzt. Dabei wird die aktuelle Szene zweimal aus verschiedenen Perspektiven, jeweils unter Einsatz unterschiedlicher asymetrischer Sichtpyramiden, für jedes Auge erzeugt und nach Anwendung eines Rot-Filters für das Bild des linken Auges sowie eines Blau-Grün-Filters für das Bild des rechten Auges gleichzeitig auf dem Bildschirm dargestellt.

7.4 Visualisierungen

In Pavel wurden Scatterplot, Scattermatrix und Parallelplot als Visualisierungsmethoden für die Daten implementiert. Diese drei Varianten sind besonders geeignet, den divergierenden Ansprüchen der beiden Anwendungsfälle *Temperierbohrungen* und *Fräsbahnoptimierung* gerecht zu werden. Auf der einen Seite treten dabei Datensätze in wenigen Dimensionen mit vielen Lösungen auf, auf der anderen Seite existieren wenige Lösungen in einem hochdimensionalen Raum. Dabei können in den Scatterplots besonders gut die Daten mit wenigen Dimensionen visualisiert werden, wohingegen Paralleplots gut für Daten mit vielen Dimensionen geeignet sind. Als Zwischenweg dient die Scattermatrix, die sich für Daten eignet, welche nicht in einem der beiden Extrema liegen.

Einzellösungsvisualisierung Basierend auf den Werten des *Decision Space* lassen sich für Lösungen der beiden umgesetzten Anwendungsfälle phänotypische Darstellungen erzeugen. In diesen wird der Bohrungs- bzw. Fräswerkzeugverlauf über dem Werkstück mittels OpenGL gerendert (siehe Abbildungen 8.13, 8.14 auf Seite 146ff.). Zusätzlich stellen die Einzellösungsvisualisierungen Mechanismen bereit, die eine Gegenüberstellung von Lösungen erlauben

und so einen Vergleich erleichtern. Zum einen lassen sich die gerenderten Darstellungen ineinander überblenden, zum anderen stellt ein Glyph solche Aspekte der Lösung dar, die in die 3D-Darstellung nicht einfließen.

7.5 Datenreduktion/Clustering

Bei der Verarbeitung von Daten in Pavel ist sowohl die Datenreduktion als auch die Strukturierung der Daten eine Aufgabe, für die Clustering gut geeignet ist. Bei der Datenreduktion geht es vor allem darum, sehr große Datensätze (10.000 - 100.000 Punkte) zur schnelleren Verarbeitung zu verkleinern (Einteilung in viele kleine Cluster).

Im Gegensatz dazu steht bei der Strukturierung von großen und von kleinen Datenmengen die Einteilung der Daten in einige wenige große Cluster im Vordergrund. In jedem Fall ist es notwendig, zwischen akkurat berechneten und schnell verfügbaren Lösungen abzuwägen, da die Datenanalyse, zu der das Clustering gehört, ein rechenintensiver Vorgang sein kann. Für die beiden beschriebenen Anwendungen des Clusterings eignen sich bekannte Verfahren gut. So ist k-Means (siehe Abschnitt 6.2) sehr schnell beim Einteilen von (auch großen) Datenmengen in wenige Klassen von ähnlichen Objekten. Ist die Menge der Eingangsdaten eher gering und werden akkuratere Ergebnisse benötigt, so spielt die Gruppe der HACM (Hierarchical-Agglomerative-Clustering-Methods: Single-Link, Complete-Link) ihre Stärken aus. Sie erlauben es zusätzlich, die Granularität der Klassifizierung nachträglich auszuwählen (hierarchisches Clustering).

Dem gegenüber steht die Aufgabe der Datenreduktion, für die keines der beiden Verfahren geeignet ist, da die HACM quadratisch viele Distanzberechnungen voraussetzen und k-Means linear mit der Anzahl der Klassen skaliert. Aus diesem Grund wurden die HACM erweitert, sodass diese nur auf einer zufällig ausgewählten Teilmenge der Distanzen arbeitet. Dadurch ergibt sich selbstverständlich lediglich eine Approximation der korrekten Lösung, für die Ansprüche der Datenreduktion ist jedoch weiterhin gewährleistet, dass nur ähnliche Datenpunkte zusammengefasst werden. Auf diese Weise können auch sehr große Datensätze verarbeitet werden und durch einstellbare Parameter (Stichprobengröße) skaliert der Algorithmus gut mit der Rechenleistung, sodass genauere Ergebnisse erzielt werden können.

Ein Aspekt, der in Pavel besondere Beachtung fand, war die Berücksichtigung der hohen Dimension der Lösungsmengen. Durch sie ist es nur schwer möglich, Distanzmaße mit sinnvoller Bedeutung zu belegen, da bei großen Dimensionszahlen die Unterschiede in den Distanzen im Rauschen der Werte untergehen. Aus diesem Grund lassen sich in Pavel Distanzberechnungen

durch Wahl relevanter Spalten auf Unterräume beschränken, während die Werte der übrigen Dimensionen beibehalten werden. So ist es beispielsweise möglich, einen Datensatz nach den Parameterwerten zu clustern, und die resultierenden Gruppierungen hinterher im Zielraum zu betrachten. Zusätzlich können alle Berechnungen nicht nur auf den Originaldaten durchgeführt werden, sondern beliebige lineare Skalierungen, die auch zur Darstellungsanpassung gebraucht werden, können als Datenquelle herangezogen werden. Auf diese Weise ist es möglich, Gewichtungen der Werte, die mit Vorwissen oder aus Experimenten begründet werden, unter visueller Kontrolle einzustellen, das Clustering durchzuführen und die Ergebnisse mit verschiedenen Methoden erneut zu visualisieren.

Zusammenfassend ergibt sich durch die beschriebenen Verfahren und Frameworkfunktionen eine große Fülle an Möglichkeiten, sowie enormes Potenzial zur Erweiterung, da konsequent eine offene, gut dokumentierte Struktur geschaffen wurde.

7.6 Paretofront

Wie in der Einleitung und in Abschnitt 4.2 beschrieben, unterstützt Pavel das Arbeiten mit „besten Kompromisslösungen“, den *Paretooptimalen Lösungen*. Die Definition dazu sowie weitere Informationen zur Theorie der Paretooptimalität findet sich ebenfalls in Abschnitt 4.2. Das Berechnen der Paretofront funktioniert für zwei und dreidimensionale Darstellungen im Scatterplot. Dabei unterscheiden sich die verwendeten Algorithmen grundlegend.

Im zweidimensionalen Fall genügt eine nichtdominierende Sortierung nach einem Kriterium. Haben mehrere Lösungen für dieses Kriterium den gleichen Wert, so wird auf dem zweiten Kriterium sortiert. Diese Sortierung lässt sich effizient mit dem Quicksort Algorithmus durchführen, die erwartete Laufzeit der zweidimensionalen Berechnung beträgt daher O(n log n).

Die Berechnung im dreidimensionalen Fall ist komplizierter. Hier kommt ein Divide-and-Conquer-Algorithmus nach [Ben80] zum Einsatz. Eine 3D-Punktemenge S wird rekursiv in zwei Mengen A und B aufgeteilt. Für diese werden die nichtdominierten Punkte separat gesucht. Unter der Annahme, dass kein Punkt von B durch einen Punkt von A dominiert wird, genügt es nun, die jeweiligen nichtdominierten Punkte der Teilmengen mit dem Verfahren für den zweidimensionalen Fall zu vergleichen.

7.7 Individuelle Columns

Für einige Anwendungen kann es hilfreich sein, `Column`s zu definieren, die es in den Rohdaten nicht gibt. Angenommen, bei dem zu optimierenden Kriterium handelt es sich um einen Winkel. Es kann dann z. B. durchaus Sinn ergeben, nicht den Wert des Winkels zu visualisieren, sondern dessen Sinus-Wert. In anderen Fällen kann es nötig sein, Quotienten zweier Kriterien zu ermitteln und diese zu visualisieren.

Zum einen können neue `Column`s, die mittels mathematischer Ausdrücke mehrere andere Kriterien kombinieren, die darstellungstechnisch auf drei Dimensionen begrenzte Visualisierungen erweitern. Eine Column entspricht dabei einer Funktion über eine Reihe anderer `Column`s. Zum anderen lässt sich mit individuellen `Column`s die Gewichtung bestimmter Kriterien beeinflussen. Werden die Werte eines Kriteriums multipliziert, so wandern die Punkte an der entsprechenden Columns je nach Faktor entweder weiter auseinander oder rücken enger zusammen. Dabei kann es sich auch um nichtlineare Funktionen wie Polynome oder Exponentialfunktionen handeln.

Wie Abbildung 7.5 zeigt, hat diese Skalierung Auswirkung auf die Semantik des Clusterings. Man erkennt deutlich, dass sich nach einer Achsentransformation die Clusterzentren an anderen Stellen bilden. Pavel bietet daher an Stellen, an denen die Semantik gefährdet ist, an, die Daten unter Berücksichtigung der geänderten Umstände neu zu clustern. Nach Änderungen, die die Semantik beeinflussen können, bietet Pavel daher an, die Daten unter Berücksichtigung der geänderten Umstände neu zu clustern.

Die Implementierung des Moduls für die individuellen `Column`s befindet sich in der statischen Klasse `IndividualColumns`. In dieser Klasse wird ein von der GUI übergebener String mit dem mathematischen Ausdruck geparst und ausgewertet. Jeder `Point` im aktuellen `PointSet` wird dann durch eine neuen `Point` ersetzt, der um eine zusätzliche `Column` und dem dazugehörigen berechneten Wert ergänzt wurde.

7.8 PlugIns

Die PlugIn-Struktur ist dazu gedacht, zusätzliche Anwendungsfälle leicht in Pavel einbinden zu können. Im Wesentlichen besteht ein PlugIn aus einer Klasse die vom `IUseCase` Interface abgeleitet ist. Diese Klasse stellt alle weiteren Komponenten bereit:

- Hierzu gehört ein *Parser*, der die Daten in Pavels interne Struktur einliest. Der Parser kann selbst geschrieben werden, um spezielle Dateiformate zu verarbeiten, oder es wird der Standardparser genutzt, der in Pavel bereits integriert ist.

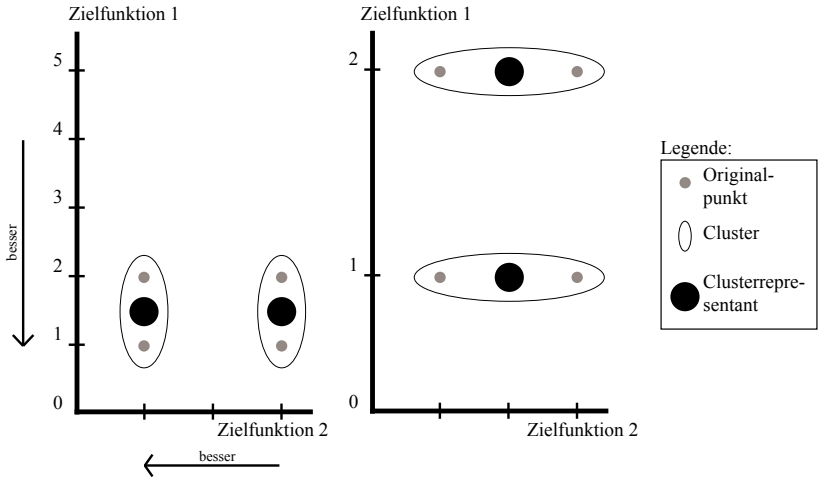


Abbildung 7.5: Da die Column zu Zielfunktion 1 im rechten Diagramm skaliert wurde, liefert der Clusteralgorithmus andere Cluster sowie unterschiedliche Repräsentanten zurück.

- Weiterhin gehört zu einem `IUseCase` eine anwendungsfallspezifische *Einzellösungsvisualisierung*, die durch Ableitung von `Solution` implementiert werden muss. Eine solche Einzellösungsvisualisierung kann zum Beispiel eine phänotypische Darstellung einer Lösung präsentieren, standardmäßig existiert lediglich eine genotypische Darstellung.

- Die *StarterPages* definieren das Aussehen des Assistenten zum Anlegen eines Projektes in diesem Anwendungsfall.

7.9 Fazit

Einer der Umstände, die die Entwicklung und Anwendung agiler Entwicklungsmethoden vorangetrieben haben, war die Erkenntnis, dass es so gut wie unmöglich ist, die Anforderungen des zu modellierenden Systems zum Projektbeginn vollständig zu erfassen [BA04]. Dies traf auch auf Pavel zu. Die Datenstrukturen des Frameworks erfuhren während des ersten Projektsemesters zwei umfangreiche Refaktorisierungen, um neu gewonnene Kenntnisse über Performanceprobleme und eine bessere Repräsentation bestimmter Abhängigkeiten einfließen zu lassen. Die GUI wurde anfänglich nur prototypisch umgesetzt. Nachdem sich mit der Zeit bestimmte Strukturen zwischen einzelnen Elementen als vorteilhaft, andere als störend oder überflüssig erwiesen,

wurden auch hier große Teile des Codes neu geschrieben. Selbst gegen Ende des Projektes fielen immer wieder Konstrukte auf, zuletzt in der PlugIn-Schnittstelle, deren zugrunde liegende Ideen sich mittlerweile als falsch oder überholt erwiesen hatten.

Solche Änderungen wurden jedoch von Anfang an erwartet und eingeplant und bereiteten daher auch keine nennenswerten Probleme. Die Verwendung von C# als Implementierungssprache hat sich dabei durch die gewonnene Flexibilität als enorm hilfreich erwiesen. Das Endergebnis von Entwurf und Implementierung ist zwar nicht perfekt, hinsichtlich Architektur und Geschwindigkeit allerdings durchaus zufriedenstellend.

8 Anwenderhandbuch

Im Folgenden wird die Nutzung des Programmes Pavel genauer beschrieben.

8.1 Datenstrukturen

Für das Verständnis vieler Abläufe bei der Bedienung von Pavel ist es hilfreich, die zugrunde liegenden Datenstrukturen zu verstehen. Eine Vorstellung aller wichtigen Klassen und ihrer Beziehungen wurde in Abschnitt 7.1 bereits gegeben und wird an dieser Stelle deshalb nicht wiederholt.

8.2 Arbeitsablauf

Zu Beginn eines Arbeitsablaufes in Pavel wird ein neues Projekt angelegt oder ein altes geöffnet. Dazu muss im Hauptmenü oder im ersten ToolStrip das Feld bzw. der Button „New Project“ betätigt werden (siehe Abschnitt 8.3).

Danach sind alle notwendigen Daten eingeladen und es ist möglich, die Räume in verschiedenen Ansichten zu betrachten. Pavel bietet die Ansichten Tabelle, ParallelPlot, ScatterPlot und ScatterMatrix (Abschnitt 8.5.2 - 8.5.5), die über den Button im MainWindow oder den Menüpunkt „Ansichten“ zu erreichen sind.

Ist mindestens ein Punkt in einer Ansicht markiert, ist der Button „Display Single Solution“ freigeschaltet (siehe Abschnitt 8.5.6). Es ist möglich die Darstellung des Punktes im „decision space“ (siehe Kapitel 4.2.2) zu betrachten und bei Markierung mehrerer Punkte diese auf verschiedene Arten zu vergleichen. Ein Punkt (`Point`) entspricht einem Individuum der Eingabedatei (siehe Kapitel 8.9.3) und somit einer Lösung des für die Erstellung der Eingabedatei genutzten Algorithmus.

Ist ein Datensatz sehr groß und unübersichtlich, kann er durch Clustering (siehe Kapitel 8.8) zu Clusterzentren zusammengefasst werden. Dazu stehen verschiedene Algorithmen (extended HACM oder k-Means siehe Kapitel 6) und Einstellmöglichkeiten zur Verfügung. Liegt die Größe der eingelesenen

Daten über 10.000 Individuen, so wird der Benutzer nach dem Anlegen eines neuen Projektes automatisch gefragt, ob er die Daten einem Clustering unterziehen möchte.

Im SpaceManager (siehe Abschnitt 8.7), der über einen Button im MainWindow erreicht wird, können die `Columns` (siehe Kapitel 8.9.3 und 9.1.1), die in den oben beschriebenen Ansichten dargestellt werden, explizit ausgewählt oder gelöscht werden. Außerdem können eigene `Columns`, sogenannte IndividualColumns (siehe Abschnitt 8.7.1), hinzugefügt werden. Im PointSetManager (siehe Abschnitt 8.6) steht die Verwaltung der Punktemengen im Vordergrund.

Als weiteres Feature wurde der ParetoFinder (siehe Abschnitt 8.5.4) eingebaut, der bei Betätigung des entsprechenden Buttons dem Benutzer die Paretofront im zwei- oder dreidimensionalen Raum anzeigt. Mehr Dimensionen sind vorerst nicht implementiert, da die Berechnung eine übermäßig lange Zeit in Anspruch nehmen würde.

8.3 Projekt neu/öffnen

Im Abbildung 8.1 auf der nächsten Seite kann bei (1.) ein neues Projekt angelegt werden. In der ListBox werden die zur Verfügung stehenden PlugIns gelistet. Unter (2.) kann ein bereits gespeichertes Projekt (3.) geöffnet werden. Ist dies nicht aufgelistet, kann es über einen OpenDialog geöffnet werden. Der Next-Button (4.) wird jeweils auf den einzelnen Seiten fortgefahren werden.

Beim Fenster 8.2 auf Seite 134 kann mit den Browse-Buttons (1.) eine entsprechende Datei gewählt werden. Entsprechen die Dateien den Mustern, werden alle Felder entsprechend ausgefüllt. Mit den X-Buttons (2.) kann das jeweilige Feld geleert werden. Alle Felder müssen ausgefüllt werden.

Auf der in Abbildung 8.3 auf Seite 135 zu sehenden Seite müssen jeweils die Columns ausgewählt werden, die zur Darstellung der Einzelvisualisierung (siehe Abschnitt 8.5.6) benötigt werden. Als Voreinstellungen sind die Columns des „Decision Space“ eingestellt, sofern ein Space mit diesem Namen existiert, sodass der Benutzer keine weiteren Einstellungen tätigen muss.

8.4 Aufbau des MainWindow

Das MainWindow setzt sich aus 3 Teilbereichen zusammen: dem Menü und dem ToolStrip, dem PropertyControl sowie dem Arbeitsbereich, indem die Visualisierungen angezeigt werden.

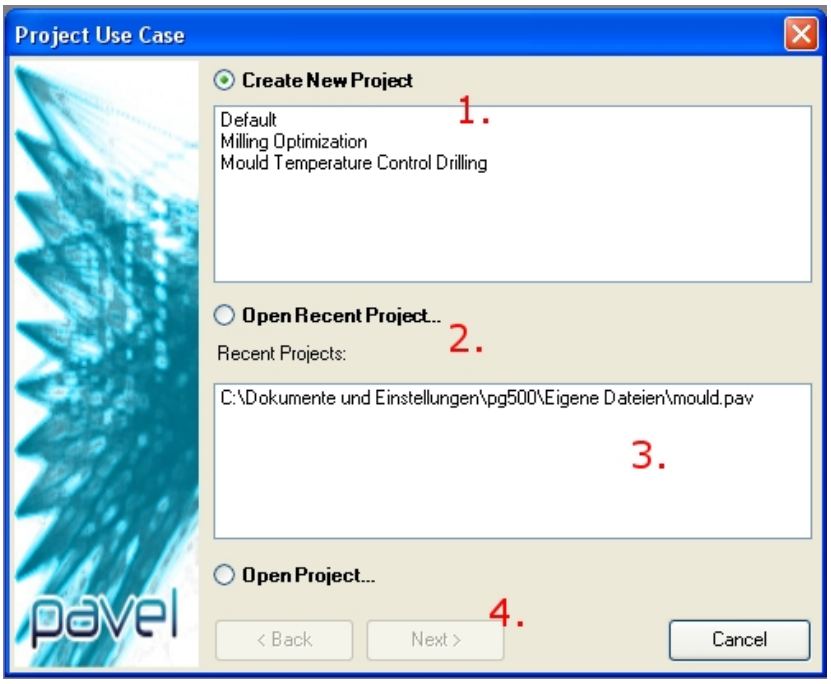


Abbildung 8.1: Es kann ein neues oder ein bestehendes Projekt geöffnet werden

8.4.1 Menü und ToolStrips

Das Menü enthält dieselben Funktionen wie die ToolStrips, weshalb hier nur die ToolStrips beschrieben werden.

Mit dem ersten Button kann ein neues Projekt angelegt oder ein bereits bestehendes geöffnet werden. Nähere Details dazu sind im Abschnitt 8.3 zu finden. Solange kein Projekt eingeladen ist, sind alle weiteren Buttons der ToolStrips und somit ebenfalls die Einträge des Menüs inaktiv. Der zweite Button schließt das aktuelle Projekt, wobei nachgefragt wird, ob Änderungen vorher gespeichert werden sollen. Dies geschieht jedoch lediglich, wenn eine Speicherung notwendig ist. Um das explizite Speichern zu realisieren, muss der dritte Button mit dem Disketten-Icon betätigt werden. Er ist jedoch nur aktiviert, wenn das Projekt tatsächlich geändert wurde.

Der zweite ToolStrip ist für die Visualisierungen zuständig. Mit dem Button „Open New Visualization Window“ (Button 4 in Abb. 8.5) kann ein neues `VisualizationWindow` geöffnet werden. Wird dieser Button geklickt, erscheint zunächst ein Dialog für das PointSet und den Space (siehe Abschnitt 8.5.1) und nach dieser Auswahl wird die letzte benutzte Visualisierungsart angezeigt. Durch Anklicken des nebenstehenden Dreiecks sind die anderen Visualisierungen auswählbar, genau wie im Menü unter „Visualization“.

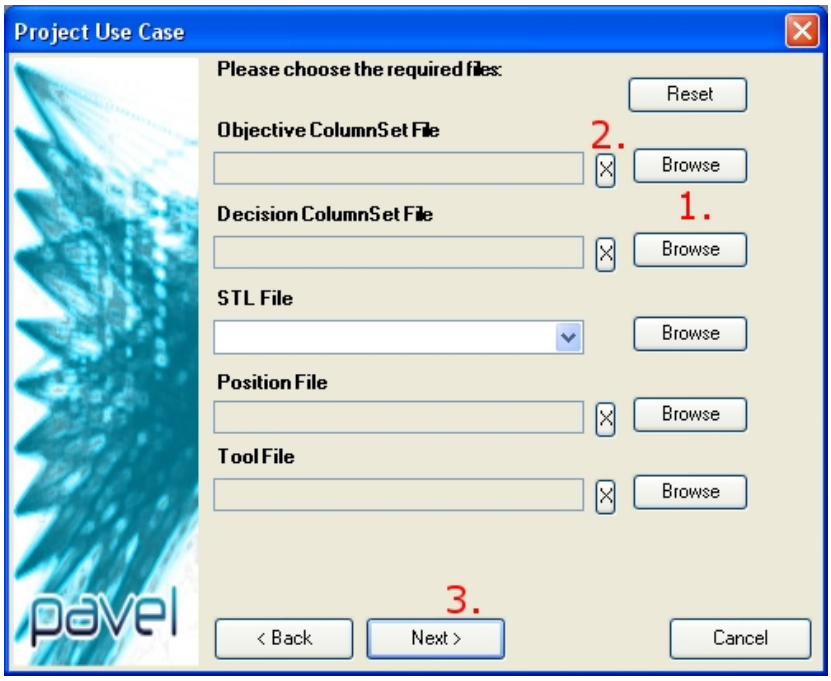


Abbildung 8.2: Neues Projekt: Spaceauswahl beim Anwendungsfall Mould-Temperature

Der Button 5 des zweiten ToolStrips ist lediglich aktiviert, wenn mindestens ein Punkt im Visualisierungsfenster markiert ist. Damit ist es möglich die Einzelvisualisierung (siehe Abschnitt 8.5.6) des bzw. der Punkte anzuzeigen.

Mit dem Button 6 wird ein Dialog angezeigt, der ein Clustering (siehe Abschnitt 8.8) erlaubt.

Button 7 öffnet ein neues Fenster mit dem SpaceManager (siehe Abschnitt 8.7), während der darauf folgende Button 8 eines mit dem PointSetManager (siehe 8.6) anzeigt.

Die ComboBox 9 beinhaltet gespeicherte `Selection`s, die bei Auswahl die entsprechenden Punkte farbig markiert. Um eine `Selection` zu erhalten, müssen Punkte im Visualisierungsfenster markiert werden, ein Name in die leere ComboBox geschrieben und der nachfolgende Buttton 10 „Save Selection“ geklickt werden. Die gespeicherte `Selection` ist über das Dreieck der ComboBox auswählbar (siehe Abb. 8.6).

Zusätzlich zu den Funktionen der ToolStrips beinhaltet das Menü unter dem Punkt „Window“ Anzeigemöglichkeiten der geöffneten Fenster. Es ist möglich, diese horizontal, vertikal oder kaskadierend, also ineinander geschachtelt, anzuordnen.

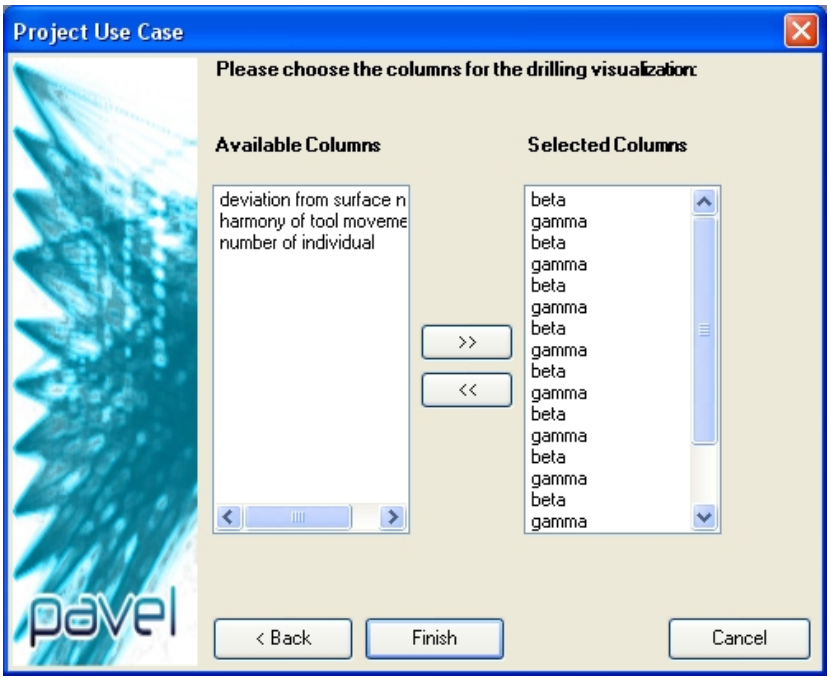


Abbildung 8.3: Auswahl der Columns für die Einzelvisualisierung

Des Weiteren ist unter Help -> Dokumentation oder Help -> Index eine Hilfedatei zu den einzelnen Klassen zu finden. Unter View -> PropertyControl ist das PropertyControl aus- bzw. einblendbar. Um die Logbucheinträge zu betrachten, muss Datei -> Log betätigt werden.

Falls eine Visualisierung geöffnet und aktiviert ist, erscheint ein weiterer ToolStrip, der mindestens drei Buttons beinhaltet (siehe Bereich 11 in Abb. 8.5). Mit dem ersten Button können die aktuell markierten Punkte aus dem PointSet entfernt werden, während der zweite diese Aktion wieder rückgängig macht, falls dieses versehentlich geschehen sein sollte. Der letzte Button dieses ToolStrips dient dazu, einen Screenshot des aktuell aktiven VisualizationWindow zu erstellen. Dieser kann als Bitmap, JPEG oder PNG in einer neuen Datei gespeichert werden.

Jede Visualisierung hat ihre eigenen speziellen ToolStrips, die hinter die eben beschriebenen angehängt werden (siehe Bereich 12 und 13 in Abb. 8.5). Die Beschreibung der einzelnen Funktionen dieser ToolStrips finden sich unter den jeweiligen Visualisierungsabschnitten (8.5.2 - 8.5.5).

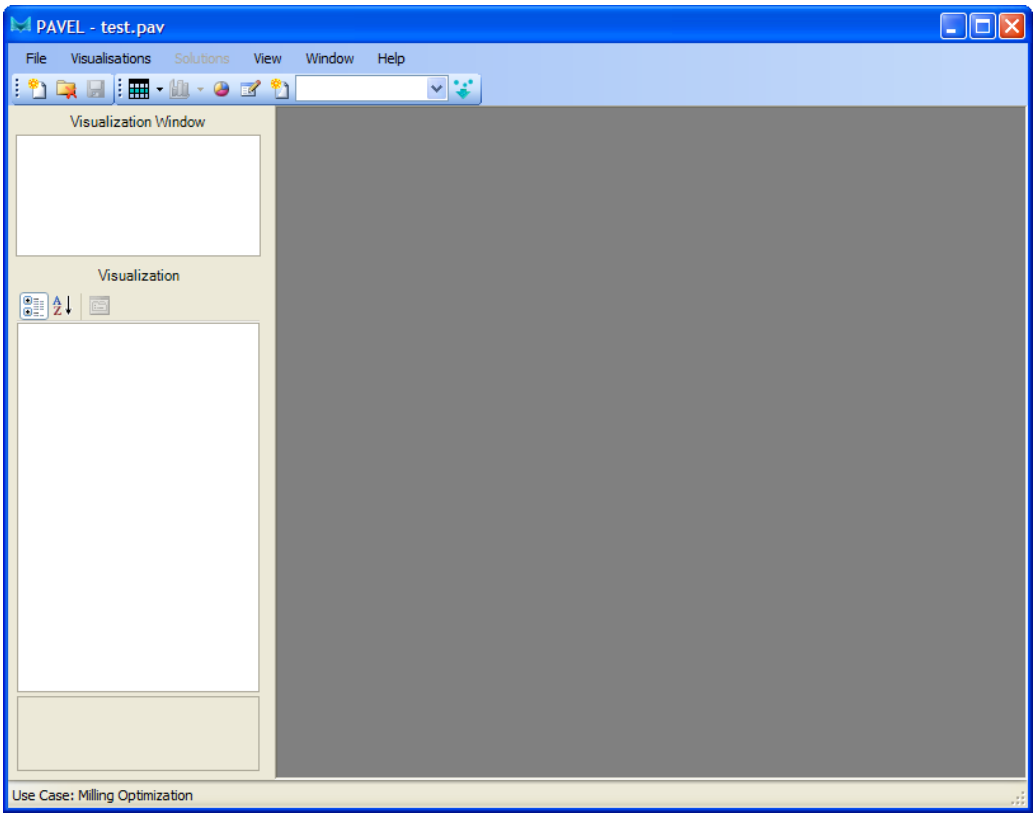


Abbildung 8.4: MainWindow von Pavel

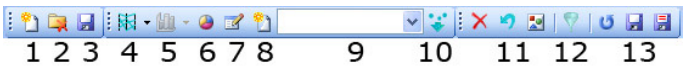


Abbildung 8.5: ToolStrip: (1) Neues Projekt (2) Schließen des Projekts (3) Speichern (4) Neues Visualisierungsfenster (5) Einzelvisualisierung (6) Clustering (7) SpaceManager (8) PointSetManager (9) Selektionen (10) Speichern der Selektionen (11) allgemeiner Visualisierungstoolstrip (12) visualisierungsspezifischer ToolStrip (13) weiterer visualisierungsspezifischer ToolStrip

8.4.2 PropertyControl

Das PropertyControl ist in zwei Bereiche unterteilt: in Visualization Window und Visualization. Im Bereich Visualization Window sind allgemeine Eigenschaften einstellbar, wie z. B. welcher Space und welches PointSet für die aktuelle Visualisierung benutzt wird. Des Weiteren ist es dort möglich die Visualsierungsart unter der Kategorie Display zu ändern, also z. B. einen ScatterPlot in einen ParallelPlot umzuwandeln.

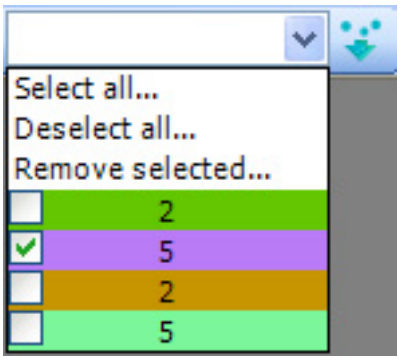


Abbildung 8.6: ComboBox für gespeicherte Selections

Der Visualization-Bereich beinhaltet Einstellmöglichkeiten, die speziell für die jeweilig aktuelle Visualisierung sind. Am unteren Ende befindet sich eine grau hinterlegte Ausgabe, in der Beschreibungen zu dem gerade ausgewählten Attribut stehen, sodass hier nicht jedes im Detail beschrieben sein wird.

Ist die aktive Visualisierung eine Einzelvisualisierung, so besteht das PropertyControl lediglich aus einem Solution-Bereich, der Einstellmöglichkeiten für die Einzelvisualisierung, abhängig vom gewählten UseCase, bietet.

8.5 Visualisierung

Der folgende Abschnitt beschreibt die Interaktionsmöglichkeiten der Visualisierungen in Pavel, deren Grundlagen in Kapitel 5.3 beschrieben wurden.

8.5.1 Dialog für PointSet und Space

Bevor ein neues Visualisierungsfenster geöffnet wird, erscheint der Dialog zur Auswahl von `PointSet` und `Space`. In ihm sind im oberen Abschnitt alle aktuell im Projekt enthaltenen PointSets aufgelistet, von denen der Benutzer genau einen markieren muss. Im unteren Abschnitt muss analog ein dazu passender Space ausgewählt werden.

Erst wenn ein PointSet und ein Space ausgewählt sind, kann durch Betätigung des OK-Buttons ein Visualisierungsfenster mit den zuvor gemachten Einstellungen geöffnet werden. Zur leichteren und schnelleren Benutzung des Programms sind das MasterPointSet (`PointSet` über alle `Points` (siehe Kapitel 9)) und der MasterSpace (`Space` über alle `Columns` (siehe Kapitel 9)) als Vorauswahl eingetragen. Die in diesem Dialog erfolgten Einstellungen können bei einer bereits geöffneten Visualisierung im PropertyControl verändert werden.

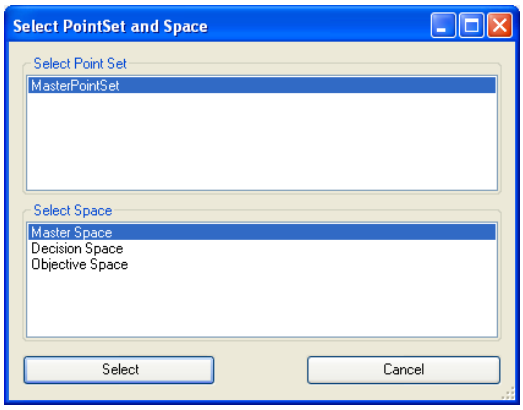


Abbildung 8.7: Dialog zur Auswahl von PointSet und Space für eine Visualisierung

8.5.2 Listing

Das Listing ist eine einfache Visualisierung in Form einer Tabelle, bei der exakte numerische Werte dargestellt werden. Diese Tabelle unterliegt keinerlei Beschränkungen hinsichtlich der Anzahl gleichzeitig darstellbarer `Points` und `Columns`. Durch einen Klick auf den Spaltenkopf werden alle `Points` in der Tabelle entsprechend der gewählten Spalte sortiert. Die Sortierreihenfolge orientiert sich an den `ColumnProperty` der entsprechenden Spalte, deren minimaler Wert oben in der Tabelle angezeigt wird.

Es besteht die Möglichkeit aus verschiedenen Darstellungsformen zu wählen, die den Focus auf verschiedene Eigenschaften legen. Die Einstellung „Selections“ färbt alle dargestellten Punkte entsprechend ihres Selektionszustandes, identisch zu der Farbdarstellung des ScatterPlots oder des ParallelPlots. Eine andere Variante ist „Value“. Hierbei wird jede Zelle entsprechend ihres Wertes relativ zu den zugehörigen ColumnProperties eingefärbt. In dieser Darstellungsform sind Werte am einfachsten zu vergleichen. Minimale Werte in Bezug auf die ColumnProperty werden grün, maximale Werte rot eingefärbt, dazwischen wird linear interpoliert. Zusätzlich existiert die Variante „None“, bei der einzig die aktuelle Selektion hervorgehoben wird.

Das Listing bietet weiterhin mehrere Möglichkeiten den dargestellten Space zu manipulieren. Durch Drag&Drop der Spaltenköpfe kann die Reihenfolge der Spalten verändert werden. Nach einem Rechtsklick auf die Spalte kann sowohl die Spalte gelöscht werden, als auch die ColumnProperty angepasst werden. Um den zusammengestellten Space dauerhaft speichern zu können, ist es im ToolStrip möglich, diesen unter gleichem oder neuem Namen zu speichern oder die kompletten Änderungen durch einen Reset wieder rückgängig zu machen.

8.5.3 ParallelPlot

Der ParallelPlot nutzt das Prinzip der Parallelkoordinaten zur Visualisierung hochdimensionaler Datenmengen (siehe Kapitel 5.3). Je nach Größe des ausgewählten Spaces und PointSets, ist die Darstellung der Datenmenge durch Parallelkoordinaten besser oder schlechter geeignet. So kann diese Visualisierung gut mit hohen Dimensionen umgehen, wobei aber schon vergleichsweise wenige Datenpunkte zu einer unübersichtlichen Darstellung führen. Um auch bei hohen Dimensionen den Überblick zu behalten, ist es notwendig, die vorhandenen Achsen möglichst optimal vertikal zu positionieren und unwichtige Achsen komplett aus der Darstellung zu löschen. Es empfiehlt sich bei sehr großen Datensätzen, vor der Arbeit mit dem ParallelPlot ein Clustering durchzuführen. Auch die Orientierung der einzelnen Achsen kann bei intelligenter Wahl viel zur Übersicht beitragen. Wie bei allen Visualisierungen ist es mit der linken Maustaste möglich, einzelne oder mehrere Points, repräsentiert durch Linien, auszuwählen. Für all diese Funktionalitäten bietet der ParallelPlot eine schnelle und intuitive Bedienung an.

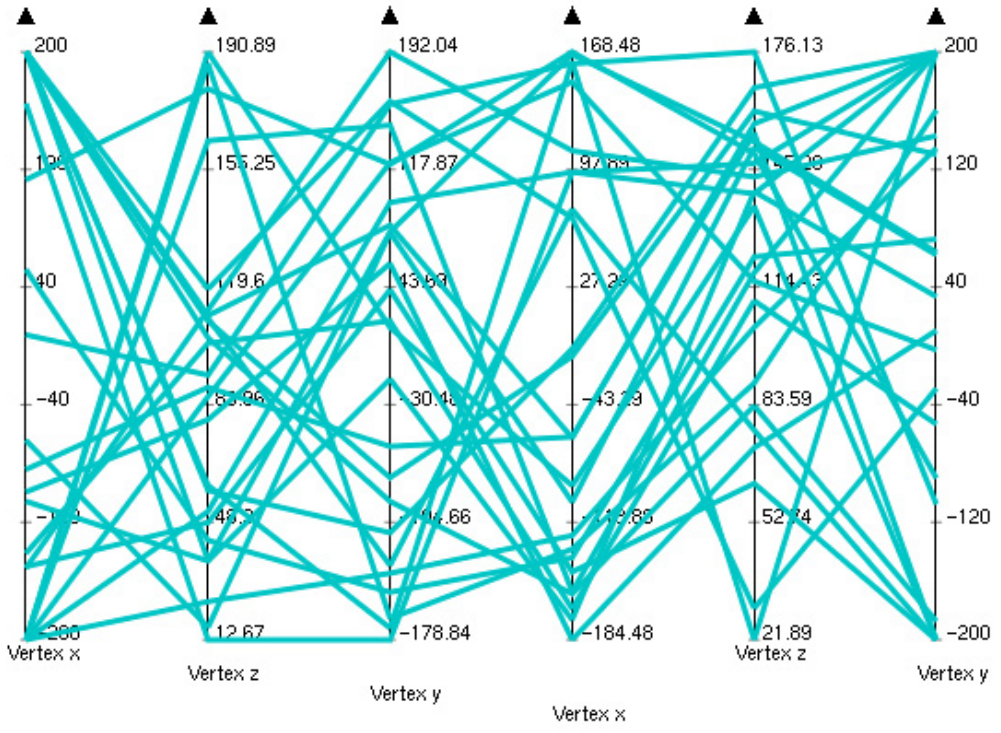


Abbildung 8.8: ParallelPlot

Über den einzelnen Achsen werden Pfeile angezeigt, um die Orientierung der jeweiligen Achse zu verdeutlichen. Mit einem Klick der linken Maustaste auf einen dieser Pfeile kehrt sich die Orientierung dieser Achse um. Ein Klick der rechten Maustaste öffnet eine Dialogbox, in der Ober- und Untergrenze der Achse eingestellt werden können. Oberhalb der Pfeile tauchen rote Kreuze auf, wenn sich der Mauszeiger dort befindet, ein Klick auf das Kreuz entfernt die entsprechende Achse aus der Darstellung. Achsen lassen sich umsortieren, indem bei gedrückter linker Maustaste der Pfeil oberhalb einer Achse nach links oder rechts verschoben wird. Durch einen Klick mit der rechten Maustaste zwischen zwei Achsen lässt sich an diese Stelle eine neue Achse einfügen. Dabei stehen sämtliche Columns aus dem ColumnSet des dargestellten PointSet zur Auswahl.

Um den auf diese Weise bearbeiteten Space dauerhaft speichern zu können, ist es im ToolStrip möglich, diesen unter gleichem oder neuem Namen zu speichern oder die kompletten Änderungen durch einen Reset wieder rückgängig zu machen. Im PropertyControl können die Linienbreite, Linientransparenz und Breite eines Clusterzentrums im ParallelPlot eingestellt werden. Wenn ein geclustertes PointSet ausgewählt ist, wird durch diese Einstellung das Clusterzentrum besser hervorgehoben.

8.5.4 ScatterPlot

Im Gegensatz zum ParallelPlot ist der ScatterPlot ideal dafür geeignet, große Datensätze darzustellen. Allerdings können nur maximal 4 Dimensionen gleichzeitig visualisiert werden, die im PropertyControl ausgewählt werden können. Standardmäßig sind nur die X-, Y- und Z-Achse zugewiesen. Es ist darüber hinaus möglich, noch eine vierte Achse auszuwählen (C-Achse), die auf eine Farbskala abgebildet wird. Bei gedrückter rechter Maustaste ist es möglich, die Ansicht um den Mittelpunkt des Koordinatensystems zu drehen, mit dem Mausrad kann man in ihn hineinzoomen. Im ToolStrip des ScatterPlots ist es möglich, eine beliebige Auswahl von Punkten als neuen Dreh- bzw. Zoompunkt auszuwählen, um zum Beispiel eine Anhäufung von Punkten besser untersuchen zu können. Eine andere Möglichkeit, viele dicht aneinander liegende Punkte besser analysieren zu können, ist das Skalieren einer ausgewählten Menge von Punkten auf das komplette Koordinatensystem. Auch diese Funktionalität ist über den ToolStrip abrufbar und lässt sich dort auch wieder rückgängig machen. Wird die rechte Maustaste nicht gedrückt gehalten sondern nur kurz angeklickt, öffnet sich ein Menü, in dem sich die Orientierung der dargestellten Achsen umkehren lässt. Alle auf diese Weisen durchgeführten Änderungen betreffen nicht nur die Anzeigen sondern den

dargestellen Space. Um diesen samt der Änderungen dauerhaft speichern zu können, ist es im ToolStrip möglich, ihn unter gleichem oder neuem Namen zu speichern oder die kompletten Änderungen durch einen Reset wieder rückgängig zu machen.

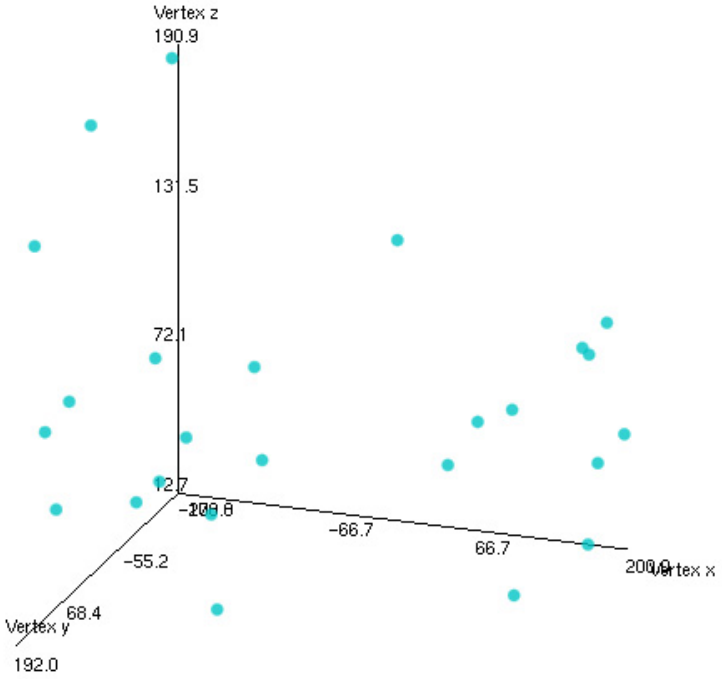


Abbildung 8.9: ScatterPlot

Ein besonderes Feature beim ScatterPlot ist die Möglichkeit der stereoskopischen Darstellung der Punktmenge durch Aktivieren der entsprechenden Option im PropertyControl. Mit einer Rot-Grün Brille wird die Illusion der Dreidimensionalität verstärkt und die räumlichen Beziehungen zwischen Punkten besser sichtbar.

Eine weitere Besonderheit des Scatterplots besteht in der Möglichkeit, die Paretofront der gesamten oder einer markierten Punktmenge berechnen zu lassen. Diese wird bei einem entsprechenden Klick auf den ToolStrip Button als Selection gespeichert und automatisch angezeigt. Im folgenden Abschnitt wird dieser Vorgang noch genauer beschrieben.

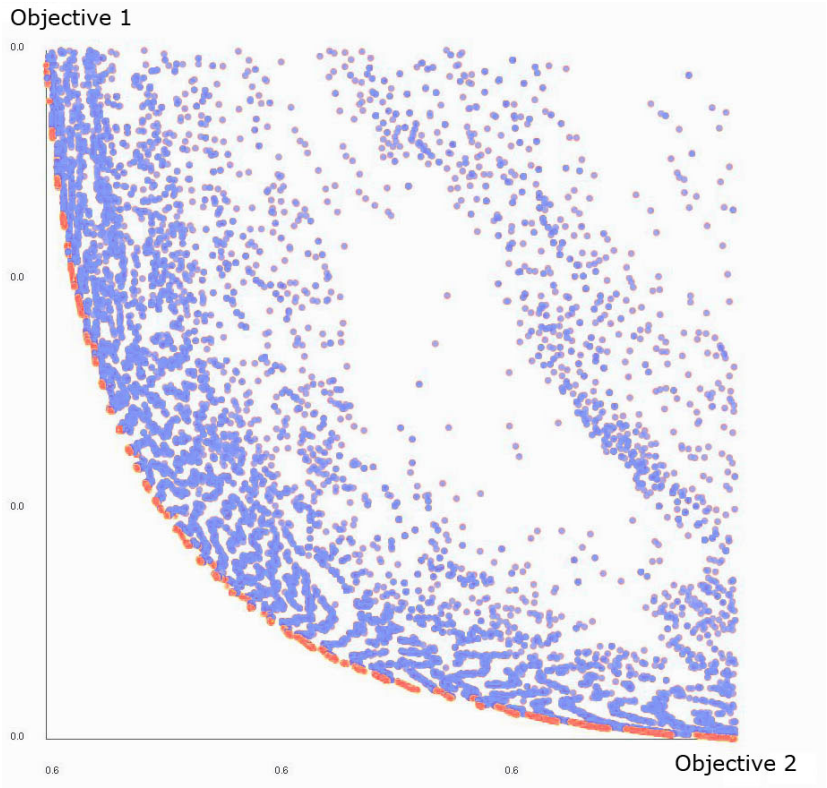


Abbildung 8.10: Hervorhebung der Paretofront einer zweidimensionalen Punktmenge

Arbeiten mit der Paretofront

Wie in der Einleitung und in Abschnitt 4.2 beschrieben, unterstützt Pavel das Arbeiten mit „besten Kompromisslösungen“, den Paretooptimalen Lösungen. Die Definition sowie weitere Informationen zur Theorie der Paretooptimalität finden sich ebenfalls in Abschnitt 4.2. Auf die Implementierung in Pavel wird in 7.6 eingegangen.

Im Visualisierungstoolstrip für den ScatterPlot befindet sich ein Button „Evaluate the Paretofront“. Durch einen Klick auf diesen Button wird die Paretofront aus den angezeigten Lösungen berechnet und als aktivierte `Selection` zurückgegeben. Soll eine lokale Paretofront berechnet werden, muss mit der Maus eine Teilmenge des angezeigten PointSets ausgewählt werden. Nach der Berechnung kann mit „Create a new PointSet containing the currently selected Points“ im PointSetManager ein passendes PointSet erzeugt werden. Nach Schließen des Dialoges muss nun nur noch bei den Eigenschaften

des Visualisierungsfensters das eben erzeugte PointSet ausgewählt werden. Ein Klick auf „Evaluate the Paretofront“ gibt die lokale Paretofront, wie in Abbildung 8.10 dargestellt, als `Selection` zurück.

8.5.5 ScatterMatrix

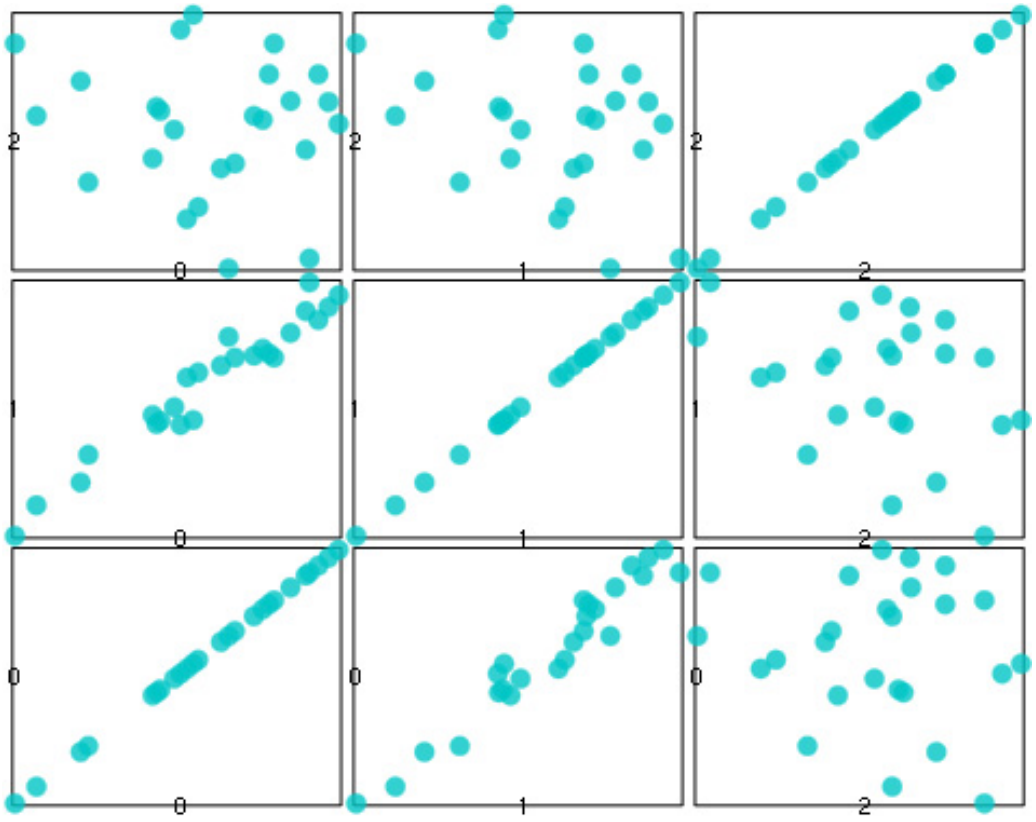


Abbildung 8.11: ScatterMatrix

Ist im PropertyControl, bei „Picking“, „Plots“ ausgewählt, so können durch Anklicken mit der linken Maustaste oder durch Aufziehen eines Rechtecks bei gedrückter linker Maustaste ein oder mehrere Scatterplots ausgewählt werden. Bei gedrückter Shift-Taste können weitere Scatterplots zur Auswahl hinzugefügt werden, wie in Abbildung 8.11 zu sehen ist. Ausgewählte Scatterplots werden in der im PropertyControl wählbaren „selection color“ markiert. Durch Klicken neben die Scatterplots werden sämtliche Scatterplots abgewählt.

Ist im PropertyControl, bei „Picking“, „Points“ ausgewählt, so können durch Anklicken mit der linken Maustaste oder durch Aufziehen eines Rechtecks bei gedrückter linker Maustaste ein oder mehrere Punkte ausgewählt werden. Bei gedrückter Shift-Taste können weitere Punkte zur Auswahl hinzugefügt werden. Die ausgewählten Punkte werden in allen Scatterplots markiert.

Die Punktgröße sowie die Transparenz der Punkte kann ebenfalls im PropertyControl ausgewählt werden.

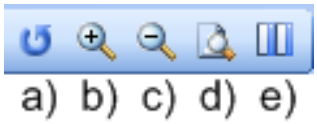


Abbildung 8.12: ScatterMatrix ToolStrip: (a) ResetView (b) Zoom In (c) Zoom Out (d) Show Selected in New Window (e) Show Complete Legend

Über den ToolStrip (Abb. 8.12) können die gewählten Scatterplots herangezoomt werden (b). Dies geschieht wie in Abschnitt 5.3.1 beschrieben. Es kann auch zur letzten Zoomstufe zurückgegangen werden (c) oder sofort zur Ausgangsdarstellung zurückgekehrt werden (a). Die ausgewählten Scatterplots können auch in einem neuen Fenster angezeigt werden(d). Ist dabei nur ein Scatterplot ausgewählt, wird statt einer ScatterMatrix ein ScatterPlot geöffnet. Mit dem letzten Button (e) kann schließlich eine Legende angezeigt werden, die die in der ScatterMatrix angezeigten Indizes den Labels der Columns zuordnet.

Durch Klicken mit der rechten Maustaste auf einen Scatterplot werden die Labels der Columns, die die Achsen dieses Scatterplots bilden, direkt am Scatterplot angezeigt. Zusätzlich wird das jeweilige Minimum und Maximum der zugehörigen `ColumnProperty` angezeigt. Wie beim kartesischen Koordinatensystem üblich, wird dabei das Minimum links unten und das Maximum rechts oben angezeigt. Das Minimum der `ColumnProperty` gibt lediglich den Grenzwert an, der im „Nullpunkt“ liegen soll und kann auch größer sein als das Maximum.

8.5.6 Einzelvisualisierungen

Einzelvisualisierungen können über den Button „Show Single Solution“ angezeigt werden, sobald in einer Visualisierung mindestens ein Punkt markiert ist. Zur Auswahl der Darstellung stehen zwei Varianten zur Verfügung: als einfaches Textfeld oder als anwendungsfallspezifische Darstellung des Decision Space. Im letzteren Fall ist das Fenster unterteilt in einen abschaltbaren

Legendencontainer und die eigentliche phänotypische Visualisierung des Punktes/der Lösung.

Den oberen Teil der Legende nehmen das Glyph und dessen Einstelloptionen ein (siehe Abb. 8.13). Ein Glyph ist eine Darstellungsform, bei der ausgehend von einem Mittelpunkt die Achsen sternförmig nach außen zeigen. Dabei liegen die Minima auf einem Kreis um den Mittelpunkt, während sich die Maxima am äußeren Ende der Achsen befinden. Hierbei entsprechen die Minima und Maxima den als ColumnProperties ausgewählten Minima und Maxima einer Column. Die einzelnen Werte der Punkte werden auf den Achsen in Relation zu dem Mini- bzw. Maximum markiert und untereinander verbunden, sodass das gesamte Glyph wie ein Spinnennetz erscheint. Falls die Werte eines Punktes über eine ColumnProperty hinauslaufen, wird das Minimum in den Mittelpunkt des Netzes gelegt und das Maximum außerhalb des äußeren Kreises platziert. Je nach der Wahl des Spaces für die Glyphs im `PropertyControl` werden in der ListBox die zugehörigen Columns aufgelistet, die einzeln durch Checkboxen aus- und eingeblendet werden können. Damit ein Glyph angezeigt wird, müssen mindestens zwei Columns ausgewählt sein.

Der mittlere Teil der Legende besteht aus einer Auflistung der einzelnen Columns und deren exakte Werte für den dargestellten Punkt. Der untere Teil beinhaltet Einstellmöglichkeiten, die bei dem entsprechenden „Mode“ nötig werden, weitere Möglichkeiten werden im Abschnitt 8.5.6 erläutert.

Die eigentliche Visualisierung des Punktes erfolgt im rechten Teil des Einzellösungsfensters und ist anwendungsfallspezifisch. In den Abbildungen 8.13 und 8.14 sind beispielhaft Einzelvisualisierungen für die beiden implementierten Anwendungsfälle „Temperierbohrungen“ und „Fräsbahnoptimierung“ dargestellt. In beiden Darstellungen ist es möglich, die Szene stereographisch mithilfe einer Rot-Grün- bzw. Rot-Blau-Brille zu betrachten. Der Augenabstand kann dabei vom Benutzer individuell angepasst werden.

Bei der MouldTemperature-Visualisierung wird für die Simulationswerte, also die Temperaturwerte des Werkstücks, eine Skala angezeigt, an der die Werte abgelesen werden können.

Vergleich von Einzellösungen

In der Einzellösungsdarstellung gibt es drei Modi, die auf verschiedene Weise den Vergleich von Lösungen unterstützen.

Zapping Beim *Zapping* lassen sich über zwei Tasten „Back“ und „Next“ nacheinander alle ausgewählten Lösungen betrachten.

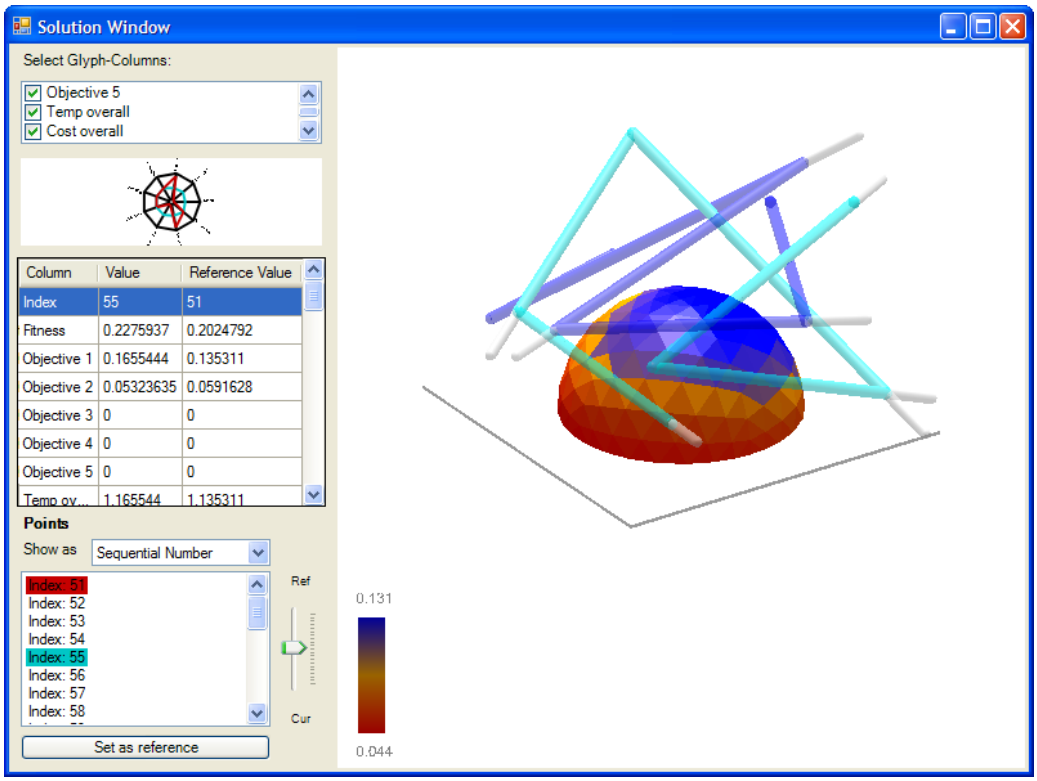


Abbildung 8.13: Einzelvisualisierung für den Anwendungsfall „Temperierbohrungen“ im Modus „CompareToRef“. Bei der Überblendung werden beide Temperierbohrungen mit unterschiedlichen Farben übereinandergelegt. Die Simulationswerte (Oberflächentemperatur) werden in dieser Abbildung durch Addition bestimmt.

CompareToRef Wird in den *CompareToRef*-Modus geschaltet, erscheint eine Liste der ausgewählten Lösungen (zu sehen in Abbildung 8.13). Durch einen Doppelklick auf eine Lösung oder durch Betätigen des Buttons „Set as Reference“ wird diese Lösung als Referenzlösung ausgewählt. Jede weitere markierte Lösung wird mit dieser Referenzlösung im Eins-zu-Eins-Vergleich dargestellt.

Über die Auswahlliste *Show as* lässt sich steuern wie die Liste der Lösungen dargestellt wird. Neben einer einfachen sequenziellen Nummerierung zur Identifikation der verschiedenen Lösungen lassen sich sämtliche Spalten des angezeigten Space auswählen. Er hat lediglich im CompareToRef-Modus beim MouldTemperature-Anwendungsfall eine Wirkung.

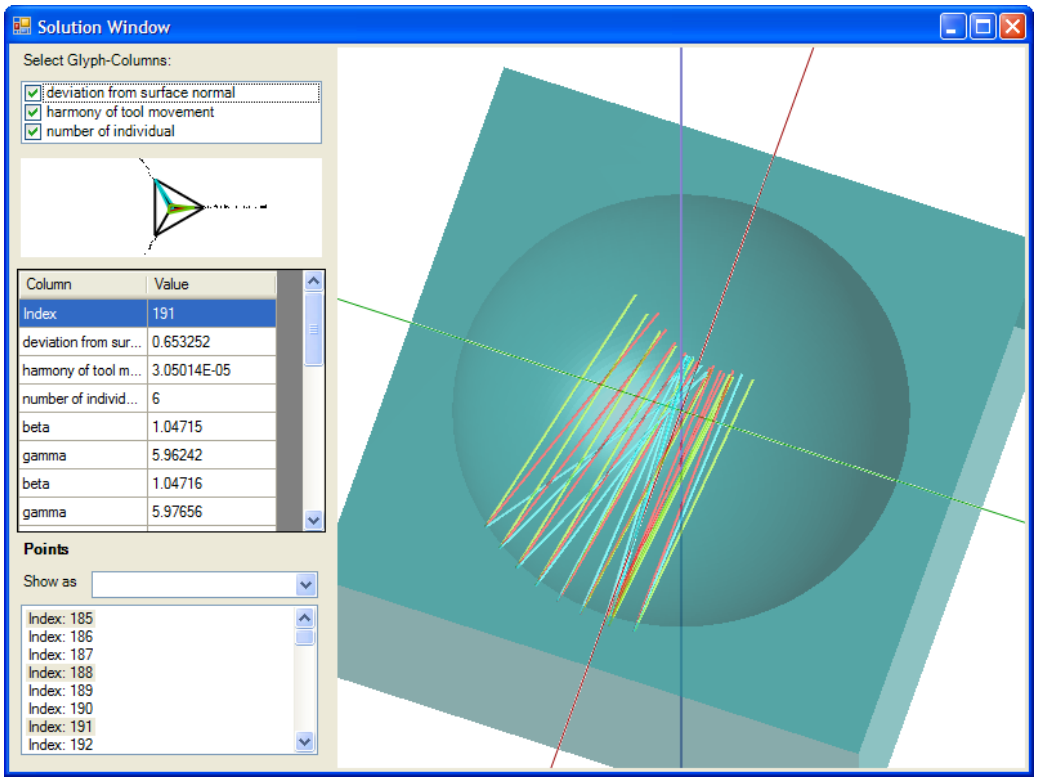


Abbildung 8.14: Einzelvisualisierung für den Anwendungsfall „Fräsbahnoptimierung“ im Modus „CompareMany“. Die drei markierten Lösungen (unten links) werden in verschiedenen Farben gleichzeitig dargestellt (3D-Ansicht und Glyph)

Des Weiteren wird die Tabelle der numerischen Werte in der Mitte der Legende um eine Spalte erweitert, die stets die Werte des Referenzpunktes anzeigt. Das Glyph zeigt beide Lösungen in den selben Farben an, mit denen sie in der Liste markiert sind, sodass sie leicht unterscheidbar bleiben.

Beim Anwendungsfall *MouldTemperature* werden die Bohrungen der Referenzlösung und die der aktuell markierten Lösung übereinandergelegt und mithilfe des Schiebereglers neben der Lösungsliste ineinander überblendet. Die hierbei entstehenden Simulationswerte ergeben sich durch den Comparison-Modus (wählbar im PropertyControl), die Gewichtung wird dabei durch die Position des Schiebereglers bestimmt.

Es stehen folgende Vergleichsmodi zur Verfügung:

- Diff (A-B)
- AbsDiff (|A-B|)
- Max (maxA, B)
- Add (A+B)

Dabei sei A ein Simulationswert der Referenz- und B der der markierten Lösung. Hierbei werden die Werte zusätzlich normiert.

Beim Anwendungsfall *MillingOptimization* wird die Referenzlösung als Verbindungsfläche zwischen den einzelnen Fräserrichtungen dargestellt, während die Fräserrichtungen der aktuellen Lösung vereinfacht als Linien angezeigt werden (siehe Abbildung 8.15). Dadurch werden Unterschiede zwischen den Fräseranstellungen deutlicher erkennbar.

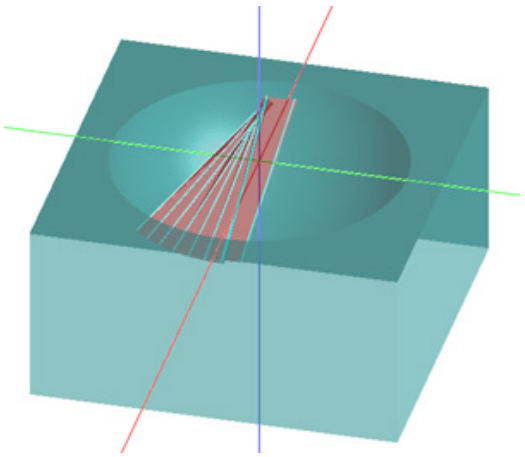


Abbildung 8.15: Beim Vergleichen eines Referenzpunktes bei MillingOptimization wird der Referenzpunkt als Fläche dargestellt.

CompareToMany Beim letzten Modus *CompareToMany* erscheint in der Legende eine Auswahlliste der Lösungen (Abbildung 8.14). In dieser Liste können mehrere Lösungen gleichzeitig mithilfe der STRG- oder Umschalttaste markiert werden. Alle gewählten Lösungen werden dabei im Glyph und in der 3D-Darstellung in verschiedenen Farben angezeigt.

Dieser Modus kommt derzeit nur beim *MillingOptimization*-Anwendungsfall zum Einsatz. In Abbildung 8.14 ist ein Beispiel mit drei markierten Lösungen zu sehen.

8.6 PointSetManager

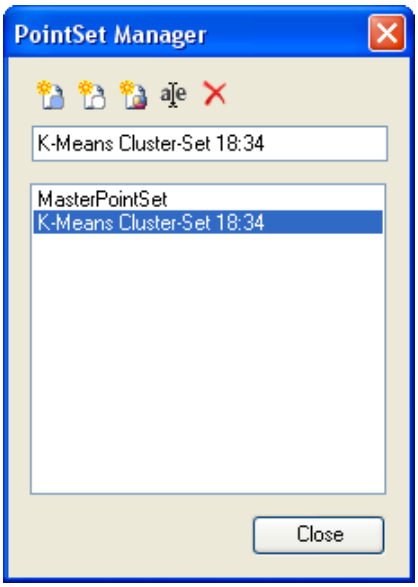


Abbildung 8.16: Der PointSetManager ermöglicht die Manipulation von PointSets.

Befindet sich mindestens ein Punkt in der CurrentSelection, wenn der PointSetManager aufgerufen wird, so kann ein neues PointSet, basierend auf dem in der Liste ausgewählten PointSet, aus den ausgewählten Punkten mit dem linken Button in Abbildung 8.16 erzeugt werden. Der zweite Button von links erzeugt ein PointSet aus den Punkten, die nicht in der CurrentSelection enthalten sind, basierend auf dem in der Liste ausgewählten PointSet. Handelt es sich bei dem gewählten PointSet um ein ClusterSet, so kann mit dem mittleren Button ein PointSet daraus erzeugt werden, das aus den Punkten besteht, die die enthaltenen Cluster bilden. Mit dem zweiten Button von rechts kann das gewählte PointSet umbenannt werden. Bei all diesen Aktionen ist zu beachten, dass der eingegebene Name eindeutig sein muss. Mit dem rechten Button schließlich kann jedes PointSet bis auf das MasterPointSet gelöscht werden. Über den Close Button wird der PointSetManager geschlossen.

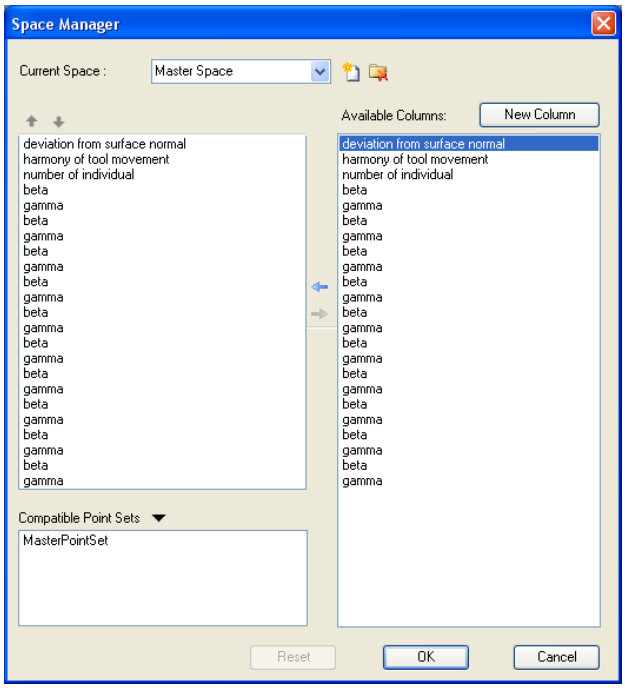


Abbildung 8.17: Der SpaceManager ermöglicht die Manipulation von Spaces. Neue Spaces können erstellt werden. Spaces können umbenannt oder gelöscht werden. Es können Columns aus den Spaces entfernt oder neue hinzugefügt werden. Über den Button „New Column“ können neue Columns erzeugt werden.

8.7 SpaceManager

Der SpaceManagers (siehe Abbildung 8.17) dient der Manipulation der Spaces des aktuellen Projekts. Im oberen Teil muss dazu zunächst der zu bearbeitende Space ausgewählt werden. Neben der ComboBox zur Space-Auswahl befinden sich drei Icons. Das linke öffnet den NewSpaceDialog (siehe Abbildung 8.18 auf der nächsten Seite). Das mittlere öffnet einen Dialog zum Umbenennen des gewählten Spaces. Das rechte Icon entfernt den gewählten Space permanent aus dem Project.

Im NewSpaceDialog muss ein eindeutiger Name für den neuen Space eingegeben werden. Ein neuer Space kann als leerer Space erzeugt werden oder als Kopie eines bereits vorhandenen Spaces.

Wurde der gewünschte Space gewählt, so werden links unten die dazu passenden PointSets angezeigt. Diese Anzeige kann durch Klicken auf das zugehörige Dreieck minimiert werden. In der linken ListBox werden die im

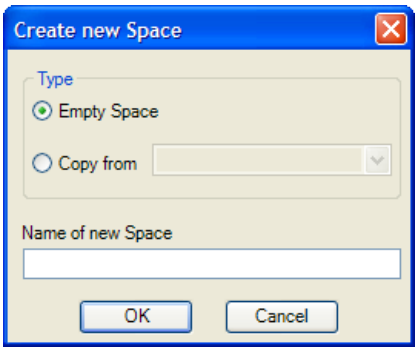


Abbildung 8.18: Der NewSpaceDialog dient zur Erzeugung neuer Spaces.

gewählten Space enthaltenen Columns angezeigt. Nachdem eine Column ausgewählt wurde, kann ihre Position in der Liste mithilfe des Auf- bzw. Ab-Pfeils verändert werden. Durch Klicken des Rechts-Pfeils wird die Column aus dem Space entfernt. In der rechten ListBox stehen sämtliche im Projekt enthaltenen Columns. Diese können durch Klicken des Links-Pfeils beliebig oft in den Space übernommen werden. Die Columns können auch per Drag&Drop aus dem Space entfernt oder zu diesem hinzugefügt werden. Alle Änderungen an einem Space können mit dem „Reset“-Button rückgängig gemacht werden, solange sie nicht durch Auswahl eines anderen Spaces oder durch Verlassen des SpaceManagers über den „OK“-Button gespeichert wurden. Beim Verlassen des SpaceManagers werden alle Spaces gelöscht, die weniger als zwei Columns enthalten, da diese ohnehin sinnlosen Spaces in Pavel zu diversen Problemen führen würden. Durch einen Klick auf den „New Column“-Button wird das Dialogfeld für individuelle Columns geöffnet, das im Folgenden näher beschrieben wird.

8.7.1 Individuelle Spalten

Zur Erzeugung individueller Columns müssen zunächst sämtliche Visualisierungen geschlossen werden. Dies ist über einen Dialog, der beim Klick auf den „New Column“-Button angezeigt wird, direkt möglich. Das Dialogfenster in Abbildung 8.19 auf der nächsten Seite ermöglicht die Erzeugung der in Kapitel 7.7 beschriebenen individuellen Columns. Im oberen Teil müssen zunächst die PointSets ausgewählt werden, in denen die neue Column verfügbar sein soll. Darunter müssen Name und Formel eingegeben werden. Durch Klicken auf eine der im linken Bereich angezeigten Columns wird das entsprechende Kürzel in das Formelfeld übernommen. Für sämtliche möglichen

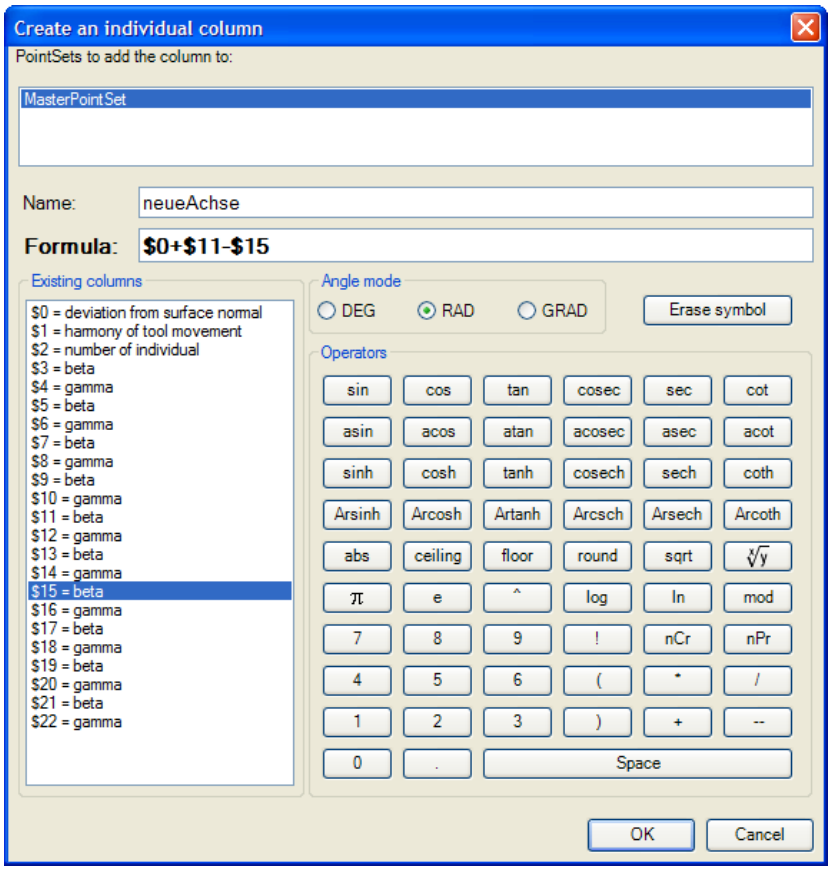


Abbildung 8.19: Dialogfenster zur Erzeugung individueller Columns.

Operationen stehen rechts Buttons zur Verfügung, die den passenden Wert in das Formelfeld einfügen.

Nach einem Klick auf OK können die ColumnProperties für die Column verändert werden. Danach erscheint die neue Column bei den „Available Columns“ des SpaceManagers. Um eine Visualisierung mit der neuen Column zu erhalten kann wie gewohnt vorgegangen werden.

8.8 Clustering

Da die Datenmengen, die im Programm üblicherweise verarbeitet werden, sehr groß sind, ist eine Reduktion der Datenmenge nicht nur sinnvoll, sondern kann auch notwendig sein, um überhaupt mit den Daten arbeiten zu können. Um dies zu ermöglichen, können alle im Programm verfügbaren Clusteralgorithmen über einen zentralen Dialog ausgeführt und deren Ergebnisse uneingeschränkt weiterverarbeitet werden.

Im folgenden Abschnitt werden zunächst der Dialog und die speziellen Parameter der implementierten Clusteringalgorithmen erklärt. Anschließend erfolgt eine Beschreibung der weiteren Funktionen, die speziell für geclusterte Daten in Pavel zur Verfügung stehen.

8.8.1 Clusterdialog

Durch das Vorhandensein von mehr als 10000 Individuen oder durch manuelles Anstoßen über den Button „Clustering“ erscheint der Clusterdialog, der verschiedenste Einstellungsmöglichkeiten und die Auswahl eines Algorithmus bietet.

Wie in Abbildung 8.20 auf der nächsten Seite zu sehen ist, lässt sich im Clusterdialog zuallererst ein Name für das resultierende ClusterSet angeben. Da jedes ClusterSet auch ein PointSet ist, sind mit den Ergebnissen des Clusterns alle Operationen in Pavel möglich. Das Clustern kann also als eine Transformation einer Datenmenge (PointSet) gesehen werden. Dazu muss zunächst ein vorhandenes PointSet (oder ein ClusterSet) im Bereich „Data“ ausgewählt werden, sowie ein Space, für dessen Columns das resultierende ClusterSet, Werte beinhalten soll. Um später die resultierenden Daten möglichst uneingeschränkt benutzen zu können, sollte hier ein möglichst umfassender Space gewählt werden. Beachtet werden sollte jedoch auch, dass die Performance unter einer großen Anzahl von Dimensionen leiden kann. Um trotz der genannten Flexibilität Einschränkungen auf den Einflussbereich für Distanzberechnungen in den Algorithmen vornehmen zu können, ist es möglich unter dem Punkt „Relevant Columns“ nur die Dimensionen auszuwählen, die für die Distanzberechnungen Berücksichtigung finden sollen. Auf diese Weise ist es möglich, dem Problem der abnehmenden Aussagekraft von Distanzen bei steigender Dimensionszahl entgegen zu wirken. Als Beispiel sei eine Datenmenge eines evolutionären Algorithmus gegeben, der Werte eines Entscheidungsraumes permutiert und auf Werten des Lösungsraumes Zielfunktionen auswertet. Mit den genannten Optionen ist es demnach möglich, den Oberraum beider Räume für das ClusterSet zu wählen, aber lediglich die Werte des Entscheidungsraumes in die Distanzberechnung mit einzubeziehen. Nach Abschluss des Vorgangs liegt ein ClusterSet vor, das eine Visualisierung im Lösungsraum ermöglicht, um Einblicke in die Zusammenhänge zwischen „Formen“ im Entscheidungsraum und Fitnesswerten im Lösungsraum zu bekommen.

Im Bereich „Clustering Parameter“ stehen zuallererst alle im Programm bekannten Clusteringalgorithmen zur Auswahl. Abhängig vom gewählten Algorithmus werden außerdem weitere Optionen eingeblendet, die im Folgen-

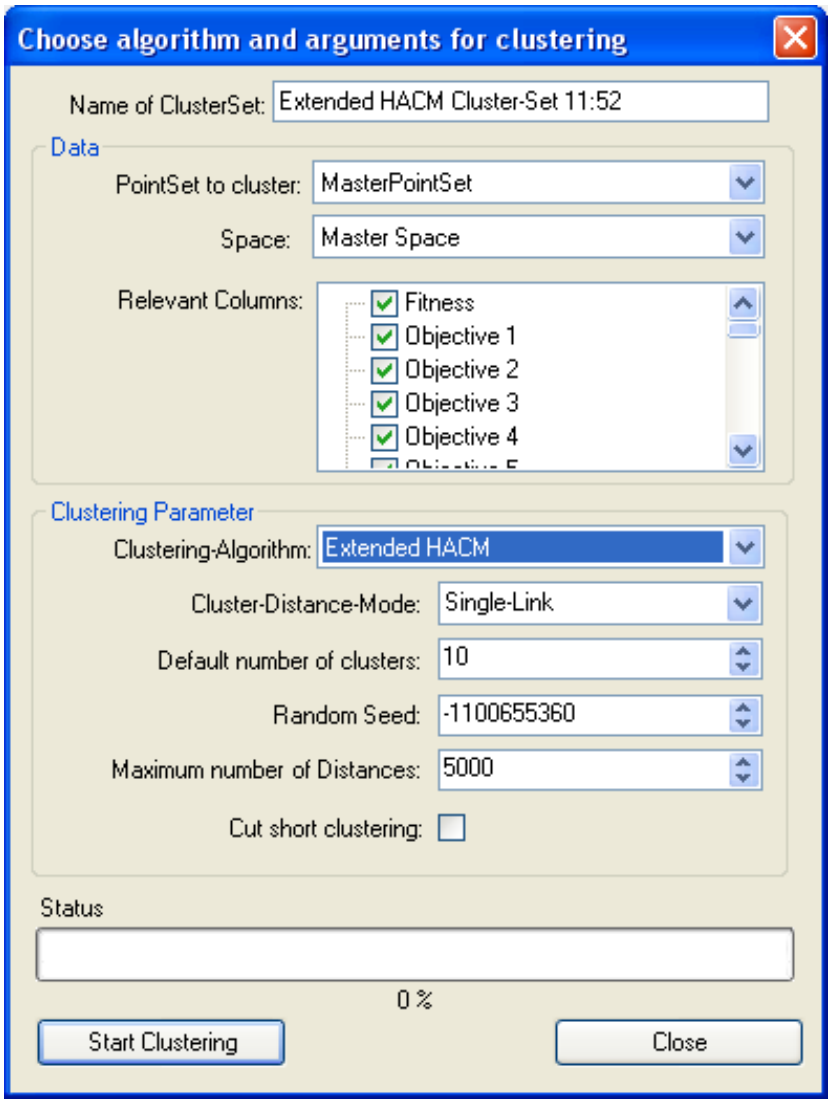


Abbildung 8.20: Dialogfenster zur Erzeugung von Clusterlösungen

den für die beiden Standardclusterverfahren k-Means und Extended HACM erläutert werden sollen.

k-Means Einstellungsmöglichkeiten

- Number of Clusters:
  Mit dieser Einstellung wird festgelegt, wie viele Datenpunkte das resultierende ClusterSet noch beinhalten soll. Nach dem Prinzip von k-Means werden dann die Originalpunkte diesen Clusterzentren zugeordnet. Achtung: die Laufzeit wächst linear mit der Anzahl der Cluster.

- Convergion bound:
  Dieser Wert dient als Abbruchbedingung für den Algorithmus. Er gibt die Summe aller Veränderungen von Clusterzentren an, die unterschritten werden muss, damit der Algorithmus vorzeitig endet. Ein Wert von 0.0 entspricht also dem Iterieren des Vorgangs bis sich kein Clusterzentrum mehr ändert.

- Maximum iterations:
  Da die genannte Abbruchbedingung nicht immer erreicht werden kann und um eine sinnvolle Laufzeitbeschränkung zu gewährleisten, ist es mit dieser Option zusätzlich möglich, die Anzahl der Durchgänge zu beschränken. Auch dieser Wert geht im schlechtesten Fall also linear ein.

- Random seed:
  Diese Option dient dazu, Ergebnisse, trotz des Vorhandenseins von Zufallsentscheidungen beim Clustern, reproduzieren zu können. Durch das Angeben eines bestimmten Startwertes für den Zufallszahlengenerator entstehen bei gleichen Eingangsdaten die gleichen Ausgangsdaten.

Beispielkonfiguration k-Means: Um zu verdeutlichen, welchen Einfluss die Parameter auf das Clusterergebnis haben, so eine beispielhafte Konfiguration näher erläutert werden:
Der klassische Fall, in dem k-Means zur Anwendung kommt, ist die Strukturierung von Daten. Dabei entstehen relativ wenige Klassen, denen die Objekte zugeordnet werden. Für diesen Fall sei angenommen, dass eine Eingabedatenmenge mit 100.000 Punkten vorliegt. Des Weiteren sei bekannt, dass es Strukturen in diesen Daten gibt, die die Daten in 10 Klassen einteilen. Solches Vorwissen ist sehr speziell und in der praktischen Anwendung muss dies meist durch mehrfaches Clustern mit verschiedenen Parametern ermittelt werden. Ist die Klassenanzahl völlig unbekannt oder sehr variabel eignet sich HACM möglicherweise besser.
In diesem Fall wählen wir „Number of clusters“ gleich 10. Der Parameter Convergion-Bound kann ohne weiteres 0.0 betragen, so ergeben sich für den Clustering-Prozess zwar mehr Durchgänge, doch bei dem geringen zu erwartenden Zeitbedarf ist eine Qualitätverbesserung in den meisten Fällen gewichtiger. Der zweite limitierende Parameter „Maximum iterations“ sollte dagegen in jedem Fall eine sinnvolle Größe aufweisen, da ansonsten der Clusteringvorgang endlos laufen kann. Zu erwarten ist ein Konvergieren erfahrungsgemäß innerhalb von 10 bis 20 Durchläufen. Eine sinnvolle Begrenzung nach oben ist also z. B. 50.

Der letzte Parameter sollte wie beschrieben nur dann gesetzt werden, wenn vorherige Clusterergebnisse reproduziert werden müssen. In den meisten Fällen ist also die zufällige Voreinstellung sinnvoll.

Extended HACM Einstellungsmöglichkeiten

- Cluster-Distance-Mode:
  Mit dieser Option kann die Inter-Cluster-Distanz festgelegt werden. Da der Algorithmus sukzessive Cluster kombiniert, ist die Reihenfolge entscheidend von dieser Einstellung abhängig.

- Default number of Clusters:
  Im Gegensatz zum k-Means entsteht bei diesem Algorithmus nicht einfach eine Partitionierung, sondern es wird eine Clusterhierarchie (Dendrogramm) aufgebaut. Diese Angabe legt dabei fest, welche der vielen enthaltenen Clusterlösungen standardmäßig verwendet werden soll. Auf diese Weise wird es möglich, auch die Ergebnisse dieses Clusterverfahrens, wie jedes andere zu benutzen, ohne jedoch die Flexibilität der Ergebnisse zu verlieren. Für Details siehe Abschnitt 8.8.2 auf Seite 158.

- Random seed:
  Diese Option dient dazu, Ergebnisse, trotz des Vorhandenseins von Zufallsentscheidungen beim Clustern, reproduzieren zu können. Durch das Angeben eines bestimmten Startwertes für den Zufallszahlengenerator entstehen bei gleichen Eingangsdaten die gleichen Ausgangsdaten.

- Maximum number of Distances:
  Durch diese spezielle Option ergibt sich die Möglichkeit, auch große Datensätze mit diesem Verfahren zu clustern. Sie gibt an, welche Menge von Distanzen (zufällige Auswahl) überhaupt betrachtet wird, wodurch sich natürlich eine Approximation des Ergebnisses ergibt. Achtung: die Anzahl an Distanzen geht linear in die Algorithmenberechnung ein, jedoch ist sie selbst (standardmäßig) das Quadrat der Menge der Eingangsdaten!

- Cut short clustering:
  Ist diese Option ausgewählt, wird das Dendrogramm nicht bis zur Wurzel erstellt, sondern nur bis die „Default Cluster count“ erreicht wurde. Danach wird das Dendrogramm wahllos zusammengeführt. Durch diese Option kann eine enorme Zeitersparnis erreicht werden, da vor allem die letzten Vereinigungsschritte sehr viel Zeit benötigen.

Beispielkonfiguration Extended-HACM: Der erste Parameter „Cluster-Distance-Mode“ bestimmt hauptsächlich die Form der resultierenden Cluster. Während Single-Link eher kettenförmige Cluster bildet, entstehen beim Complete-Link tendenziell kompaktere Klassifizierungen. Die konkrete Entscheidung muss experimentell erschlossen werden. Als Einstieg ist Complete-Link sicherlich eine gute Wahl. Im konkreten Beispiel sei zunächst eine Punktmenge mit 100.000 Individuen angenommen. Diese soll auf 10.000 Individuen reduziert werden. Die Qualität ist in diesem Beispiel also weniger entscheidend, sondern eher eine schnelle Reduktion der Datenmenge. „Default number of clusters“ wird entsprechend der Vorgabe auf 10.000 gesetzt. „Number of distances“ wäre beim normalen HACM in diesem Fall automatisch bei 10.000.000.000, was zu einer unendlichen Laufzeit und zu einem untragbaren Hauptspeicherbedarf führen würde. Zugute kommt uns in diesem Fall allerdings, dass eine Approximation völlig ausreichend ist, wir also mit gutem Gewissen lediglich z.B. 10.000 einstellen können. Dadurch reduziert sich der Ressourcenbedarf auf einen Bruchteil. Des weitern sollte der Haken bei „Cut short clustering“ gesetzt sein, da dieser Clustervorgang nur zur Datenreduktion dient. Ein weiteres Aufbauen des Dendrogramms zur Wurzel hin wird hierdurch extrem beschleunigt.

Nach dem alle gewünschten Parameter gewählt bzw. angegeben wurden, startet ein Klick auf „Start Clustering“ den eigentlichen Vorgang. Dabei können, je nach Algorithmus, verschiedene Phasen durchlaufen werden, den Fortschritt des Hauptteils der Zeit gibt jedoch der Fortschrittsbalken an. Dabei ist zu beachten, dass es sich dabei lediglich um eine Schätzung handelt, da viele Algorithmen keine vorhersehbare Laufzeit besitzen. Während des Vorgangs können aufgrund der sich ergebenden Abhängigkeiten keine weiteren Funktionen in Pavel genutzt werden. Es ist jedoch jederzeit möglich, die Berechnung vorzeitig abzubrechen. Dafür muss nur auf „Stop Clustering“ geklickt werden. Im sich öffnenden Dialog gibt es nun drei Möglichkeiten:

- Cancel bricht die Unterbrechung ab und lässt des Algorithmus weiterlaufen.
- „Nein“ unterbricht die Berechnung so schnell wie möglich. Dies hat zur Folge, dass mögliche Zwischenergebnisse verworfen werden. Beim erneuten Starten des Algorithmus beginnt er die Arbeit von Vorne.
- Die dritte Möglichkeit besteht darin, den Algorithmus lediglich aufzufordern, seine Arbeit zu beenden („Ja“). Dabei werden bereits berechnete Ergebnisse (ClusterSet) so behandelt, als wären sie das Endergebnis des Algorithmus. Dies können Approximationen der optimalen Lösung,

aber auch Teilhierarchien eines Dendrogramms sein. Ob, wann und mit welchem Ergebnis der Algorithmus endet, hängt dabei allein von der Implementierung des Algorithmus ab. Lässt sich der Algorithmus nicht vorzeitig beenden, so ist ein Abbruch über den „Stop Clustering“-Button und die Wahl von „Nein“ möglich.

8.8.2 Arbeiten mit ClusterSets

Nach erfolgreichem Clustering kann das resultierende ClusterSet wie jedes andere PointSet verarbeitet werden. Zusätzlich zeigt ein Klick auf Cluster-Mode in den Visualisierungen „ScatterPlot“ und „ParallelPlot“ nicht nur die Clusterzentren, sondern auch die dazugehörigen Originaldaten an. Außerdem wird für jedes Clusterzentrum eine Selektion erstellt und eingefärbt, wodurch es möglich ist, eine Auswahl von Clusterzentren (mit Originalpunkten) zu treffen und diese mithilfe des PointSet-Managers weiterzuverarbeiten (neues PointSet erstellen).

Handelt es sich beim ausgewählten PointSet um ein hierarchisches Cluster-Set (Extended HACM), so gibt es zusätzlich die Möglichkeit, nachträglich die „Default Number of Clusters“ zu ändern. Dies ist gleichzusetzen mit einer Veränderung der Schnittebene durch das Dendrogramm, wodurch beobachtet werden kann, wie Cluster zusammenwachsen (kleinere Clusterzahlen) oder zerfallen (größere Clusterzahlen). Dazu kann die Zahl der gewünschten Clusterzentren in das Feld „Default number of clusters“ eingetragen werden.

8.9 Datenformate

Hier werden die von Pavel akzeptierten Eingabe- und Ausgabeformate beschrieben.

8.9.1 Projektdateien (PAV)

Gespeicherte Projekte in Pavel besitzen die Endung „.pav“. Diese Datei enthält alle projektbezogenen Daten in binärer Form. Zur einfachen Realisierung wurde die Speicherfunktion mittels der Serialisierungsfunktion aus dem .Net-Framework realisiert. Dazu wird ein Objekt der Klasse `Pavel.Framework.Project` mittels eines BinaryFormatters serialisiert und danach mittels GZip komprimiert. Für die Ladefunktion wird diese Reihenfolge invertiert. Zuerst wird die Datei dekomprimiert und anschließend zu einem `Project`-Objekt deserialisiert.

8.9.2 STL-Dateien

STL (Surface Tesselation Language oder Standard Triangulation Language) ist eine Standardschnittstelle vieler CAD-Systeme. In einer STL-Datei wird die Oberfläche eines 3D-Körpers mithilfe von Dreiecksfacetten beschrieben. Jede Dreiecksfacette wird durch die drei Eckpunkte und den Normalenvektor des Dreieckes charakterisiert. Die Eckpunkte sind dabei üblicherweise entgegen dem Uhrzeigersinn angeordnet. Pavel kann sowohl STL-Dateien im ASCII-Format als auch binäre STL-Dateien einlesen.

ASCII-STL

Eine ASCII STL-Datei muss mit folgender Zeile beginnen:

```
solid NAME
```

wobei „NAME“ ein optionaler Bezeichner ist, der von Pavel ignoriert wird. Es folgt eine beliebige Zahl von Dreiecken, die wie folgt angegeben werden:

```
facet normal n1 n2 n3
  outer loop
    vertex v11 v12 v13
    vertex v21 v22 v23
    vertex v31 v32 v33
  endloop
endfacet
```

und endet mit:

```
endsolid NAME
```

Die Wörter und Zahlen müssen dabei jeweils durch Leerzeichen getrennt werden.

Binär-STL

Da ASCII-STL-Dateien sehr groß werden können, existiert eine binäre Version von STL. Eine binäre STL Datei beginnt mit einem 80 Byte langen Kopf. Dieser wird von Pavel ignoriert, sollte aber weder „vertex“ noch „facet“ enthalten. Im Gegensatz zu vielen anderen Programmen liest Pavel auch binäre STL-Dateien, deren Kopf „solid“ enthält. Auf den Kopf folgt ein vorzeichenloses vier Byte langes Integer, welches die Zahl der Dreiecksfacetten

in der Datei angibt. Danach werden nacheinander die Dreiecke beschrieben. Nach dem letzten Dreieck endet die Datei.

Jedes Dreieck wird durch zwölf Gleitkommazahlen (single, 32-Bit): zunächst drei für den Normalenvektor und dann jeweils drei für die X/Y/Z Koordinaten jedes Eckpunktes - genau wie bei ASCII-STL. Auf die zwölf Gleitkommazahlen folgt ein vorzeichenloses zwei Byte langes Integer, der „attribute byte count“, dieser wird von Pavel ignoriert, sollte aber null sein, da die meisten Programme andere Werte nicht akzeptieren.

8.9.3 Standard-Eingabedateien

Eine Standard-Eingabedatei ist eine .txt Datei bestehen aus zwei Teilen, dem Kopf-Teil und dem Daten-Teil. Diese werden durch eine beliebige Anzahl von Leerzeilen getrennt. Im Daten-Teil werden Leerzeilen und Zeilen die mit # beginnen von Pavel ignoriert. Zeilentrenner kann sowohl ein Zeilenvorschub („`\n`“) als auch ein Zeilenumbruch gefolgt von einem Zeilenvorschub („`\r\n`“) sein.

Der Kopf-Teil muss die Bezeichner aller Spalten (Columns) der gegebenen Daten enthalten. Eine Spaltenbezeichner wird dabei durch drei # identifiziert, gefolgt vom Schlüsselwort „Column i“ und der Bezeichnung für die Spalte. Eine Zeile steht dabei für eine Column, es müssen also genau so viele Zeilen wie Columns vorhanden sein. Zusätzlich können zu Beginn beliebige Kommentare hinter einem # angegeben werden.

Individuen im Daten-Teil werden durch Leerzeilen getrennt. Die Werte eines Individuums werden durch ein oder mehrere Leerzeichen getrennt. Die Werte werden als Double-Werte (64-Bit) interpretiert. Als Dezimaltrenner muss der Punkt verwendet werden. Die Werte können auch im Exponential-Format angegeben werden.

```
[-] m.dddddd E+xx
[-] m.dddddd E-xx
[-] m.dddddd e+xx
[-] m.dddddd e-xx
```

Die Zahlen können mit einem negativen Vorzeichen versehen werden. Ein positives Vorzeichen ist nicht erlaubt. Vor dem Dezimaltrenner können ein oder mehrere Ziffern stehen („m“). Der Exponent besteht aus einem Plus oder Minus gefolgt von mindestens einer Ziffer. Es sind Fließkommazahlen mit doppelter Genauigkeit gemäß IEEE 754 erlaubt. Verständlicherweise müssen für jedes Individuum genau so viele Werte angegeben werden, wie es Columns gibt.

Beispiel:

```
#<Comments>
#
#<more comments>
#
### Column 0 Fitness <- Column #1 Name
### Column 1 Objective 1 <- Column #2 Name
### Column 2 Objective 2 <- Column #3 Name
### Column 3 Objective 3 <- Column #4 Name
### Column 4 Objective 4 <- Column #5 Name

# Individual 0
0.2243532 0.1620628 0.05360335 0 0
#^-Column #1
#         ^-Column #2
#                   ^-Column #3
#                              ^-Column #4
#                                ^-Column #5

# Individual 1
0.2955322 0.2362838 0.04792459 0 0

# Individual 2
0.2245764 0.1627162 0.05320324 0 0

# Individual 3
0.2922837 0.2326404 0.04838658 0 0

# Individual 4
0.2264219 0.1647798 0.05292171 0 0
```

8.9.4 SIM Dateien

Eine Datei mit der Bezeichnung „NAME“-sim.txt enthält die Simulationswerte für die Individuen. Das Format entspricht dem der Standard-Eingabedateien ohne den Kopf-Teil. Jedes Individuum muss dreimal so viele Werte haben, wie das Werkstück (in der STL-Datei) Dreiecke hat.

8.9.5 POS Dateien

Eine Datei mit der Bezeichnung „NAME“-pos.txt enthält die Positionen des NC-Pfads. Es müssen genau so viele Zeilen (ohne Kommentarzeilen) enthalten sein, wie Positionen auf dem NC-Pfad berechnet werden. Zeilen, die mit # beginnen, enthalten auch hier Kommentare und werden ignoriert. Jede Zeile muss drei Werte für die X/Y/Z Koordinaten enthalten, die durch Leerzeichen getrennt sind. Die Koordinaten müssen zu denen des Werkstücks (in der STL-Datei) passen.

8.9.6 TOO Dateien

Eine Datei mit der Bezeichnung „NAME“-too.txt enthält nur eine Zeile in der der Radius und die Länge des Fräskopfes, getrennt durch ein Leerzeichen, stehen. Zusätzlich können auch hier Kommentarzeilen, die mit # beginnen, enthalten sein.

8.9.7 Die Anwendungsfälle

Die in Pavel bereits implementierten Anwendungsfälle erfordern verschiedene Eingabedateien, die im Folgenden beschrieben werden.

Default

Es kann eine beliebige Anzahl von Dateien im Standard Eingabeformat eingebunden werden. Die Daten jeder Datei werden als eigenständiger Space eingebunden.

Mould Temperature Drilling Control

Es sind mindestens die folgenden vier Dateien nötig. „NAME“ sollte für zusammengehörige Dateien immer gleich sein. Zusätzlich kann eine beliebige Anzahl weiterer Space durch Hinzufügen weiterer Dateien im Standard Eingabeformat eingebunden werden.

„NAME“-dec.txt Decision Space Datei. Enthält die Werte des Decision Space im oben angegebenen Standard Eingabeformat.

„NAME“-obj.txt Objective Space Datei. Enthält die Werte des Objective Space im oben angegebenen Standard Eingabeformat.

„NAME“-sim.txt Simulation Datei im oben angegebenen SIM Eingabeformat.

„ANY NAME“.stl Geometrie des Werkstücks im oben angegebenen STL-Format (ASCII oder binär).

Milling Optimization

Es sind fünf Dateien folgender Art nötig. „NAME“ sollte für zusammengehörige Dateien immer gleich sein.

„NAME“-dec.txt Decision Space Datei. Enthält die Werte des Decision Space im oben angegebenen Standard Eingabeformat.

„NAME“-obj.txt Objective Space Datei. Enthält die Werte des Objective Space im oben angegebenen Standard Eingabeformat.

„NAME“-pos.txt Positions Datei im oben angegebenen POS Eingabeformat.

„NAME“-too.txt Fräskopf Datei im oben angegebenen TOO Eingabeformat.

„ANY NAME“.stl Geometrie des Werkstücks im oben angegebenen STL-Format (ASCII oder binär).

8.10 Erweiterungen

Pavel wurde so ausgelegt, dass sehr einfach zusätzliche spezifische Anwendungsfälle hinzugefügt werden können. Ein Anwendungsfall besteht auf einer Einheit, die die Daten einliest und einem Visualisierungsteil. Für Informationen zur Programmierung eigener PlugIns und Anwendungsfälle sei auf Abschnitt 9.2 verwiesen.

9 Entwicklerhandbuch

Dieser Teil des Handbuches soll dazu dienen, grundlegende Informationen für Entwickler bereitzustellen, die Pavel erweitern möchten. Es werden daher grobe Überblicke über die Implementierung der einzelnen Programmteile gegeben, für detaillierte Informationen zu einzelnen Klassen oder Methoden sollten die Dokumentationskommentare im Quelltext herangezogen werden.

9.1 Framework

Das Framework beinhaltet das komplette Datenmodell für Pavel, sowie einige Controllerklassen, die beispielsweise Projekte verwalten. Abbildung 7.3 auf Seite 121 enthält eine Übersicht der wichtigen, zum Framework gehörenden Klassen.

9.1.1 Column

Eine `Column` repräsentiert eine Dimension des Vektorraumes in dem die geladenen Lösungen definiert sind und kann mit einem Namen versehen werden. Programmintern und für den Benutzer nicht sichtbar, werden alle je erzeugten `Columns` durchnummeriert um eine vollständige Ordnung auf ihnen gewährleisten zu können. Diese Ordnung wird benötigt, um in anderen Programmteilen binäre Suchen in Mengen von `Columns` zu ermöglichen. In jeder `Column` wird außerdem eine `DefaultColumnProperty` gespeichert, die den initialen Wertebereich für diese Dimension enthält.

9.1.2 ColumnSet

Ein `ColumnSet` ist eine geordnete Menge von `Columns`, für die zwei Regeln gelten

1. Eine `Column` darf nicht zweimal enthalten sein.

2. Die `Columns` sind nach ihrer internen ID sortiert.

Die Sortierung nach der – im Prinzip willkürlichen – internen ID der `Columns` erfüllt keinen semantischen Zweck, sondern dient der Beschleunigung von Zugriffsoperationen. Da außerdem jede `Column` höchstens einmal im `ColumnSet` enthalten sein kann, handelt es sich hier um eine echte Menge.

9.1.3 Point

Der `Point` ist die kleinste Dateneinheit in Pavel. Er beschreibt einen einzelnen mehrdimensionalen Datensatz über einem bestimmten `ColumnSet`. Die Werte für jede Dimension sind in einem Array abgelegt, ihre Reihenfolge entspricht denen der `Columns` im `ColumnSet`. Jeder `Point` verfügt außerdem über ein `DataTag`, in dem eine Identifikationsnummer und ein beliebiges Datenobjekt gespeichert werden kann. Momentan wird das `DataTag` benutzt um Simulationswerte für die Einzellösungsdarstellung zu speichern.

9.1.4 Maps und Beziehungen zwischen ColumnSets

Da `ColumnSets` Mengen darstellen, lassen sich zwischen ihnen Teilmengenbeziehungen ausdrücken sowie Mengenoperationen durchführen. Eine besondere Funktion kommt den Maps zwischen zwei `ColumnSets` zu, die an verschiedener Stelle im Programm verwendet werden. Ein motivierendes Beispiel für die Verwendung einer Map und eine anschauliche Darstellung ihrer Funktionsweise ist folgendes Szenario:

Angenommen es sollen Punkte dargestellt werden, die Werte für das `ColumnSet` A enthalten. Es soll allerdings nur `ColumnSet` $B \subseteq A$ visualisiert werden. Für die Visualisierung muss nach und nach der Wert jeder `Column` von B aus einem Punkt gelesen werden. Da die Punkte aber nicht über B sondern über A definiert sind, können die Werte nicht einfach nacheinander entnommen werden. Spalten die in A, nicht aber in B vorhanden sind, müssen übersprungen werden. Die Maps stellen diese Funktionalität bereit. Es handelt sich bei ihnen um `int[]` Arrays, die für die Position jeder Spalte in B die Position der gleichen Spalte in A enthalten. Angenommen A enthält die Spalten $[a, b, c, d, e, f]$, B die Spalten $[a, b, d, f]$. Die Map hat dann die Form $[0, 1, 3, 5]$:
Das Element 0 (a) in B befindet sich an Stelle 0 in A.
Das Element 1 (b) in B befindet sich an Stelle 1 in A.
Das Element 2 (d) in B befindet sich an Stelle 3 in A.
Das Element 3 (f) in B befindet sich an Stelle 5 in A.
Eine solche Map kann durch Aufruf der `ColumnSet.SuperSetMap()` Funktion erzeugt werden. Im Code sähe die korrekte Verwendung einer Map so aus:

```
ColumnSet A,B; // B subset A
PointList points = new PointList(A);
points.AddRange(...);

int[] map = B.SuperSetMap(A);

foreach(Point point in points)
   for(int ci = 0; ci < map.Length; ci++) {
      point[ map[ci] ]
   }
}
```

9.1.5 PointList

PointLists sind Listen von Points, die alle über dem gleichen ColumnSet definiert sind:

$$\forall Point_i \in PointList : Point_i.ColumnSet = PointList.ColumnSet$$

Dadurch muss bei der Iteration über eine PointList höchstens einmal eine Map berechnet werden, die dann auf alle Punkte in der Liste angewandt werden kann.

9.1.6 PointSet

PointSets sind Vereinigungen mehrerer PointLists. Wie diese sind auch PointSets an ein ColumnSet gebunden, jedoch dürfen die ColumnSets der enthaltenen PointLists beliebige Obermengen des ColumnSets des PointSets sein:

$$\forall PointList_i \in PointSet : PointList_i.ColumnSet \supseteq PointSet.ColumnSet$$

9.1.7 Spaces und ColumnProperties

Ein Space stellt eine Ansicht auf ein ColumnSet dar. In einem Space können die Columns beliebig geordnet oder auch mehrfach eingefügt werden. Die Columns werden nicht direkt im Space gespeichert sondern mittelbar über ColumnProperties. Eine ColumnProperty enthält zwei Werte, Min und Max, welche einen Wertebereich für die Column angeben. Dieser Bereich wird nicht direkt in der Column gespeichert, damit verschiedene Spaces die gleiche Column mit verschiedenen Wertebereichen darstellen können.

So kann es sein, dass der komplette Bereich einer Temperaturspalte mehrere hundert Grad Celsius umfasst. Soll nur eine kleine Auswahl von Punkten untersucht werden, deren Werte in dieser Spalte sich nur um wenige Grad unterscheiden, kann für die Darstellung dieser Auswahl der Wertebereich so skaliert werden, dass die Unterschiede zwischen den Punkten deutlicher werden.

Beim Laden von Daten zu Beginn eines Projektes werden normalerweise vordefinierte `Space`s angelegt und initialisiert. Wird eine Visualisierung geöffnet, arbeitet diese auf einer Kopie des gewählten `Space`s. Die Kopie enthält in der `Parent`-Property einen Verweis auf den `Space` von dem sie abstammt. Änderungen am `Space`, die in der Visualisierung durchgeführt wurden, können somit entweder beim Schließen der Visualisierung verworfen oder in ihren Ursprungs-`Space` zurückgeschrieben werden.

9.1.8 Paretofinder

Die statische Klasse `ParetoFinder` stellt die Methode `EvaluateParetoFront(PointSet, ColumnProperty[])` zum Berechnen der Paretofront eines `PointSet`s bereit. Diese Methode liefert eine `Selection` namens „ParetoFront“ zurück, welche dann in einer Visualisierung angezeigt werden kann. `EvaluateParetoFront` entscheidet anhand der Dimensionszahl des übergebenen `ColumnProperty`-Arrays, ob intern die zwei- oder dreidimensionale Variante der Paretoberechnung durchgeführt werden soll. Die Berechnung der Paretofront in mehr als drei Dimensionen ist zurzeit nicht vorgesehen, kann jedoch nach Bedarf implementiert werden. Die Arbeitsweise der verwendeten Algorithmen wird in Abschnitt 7.6 erklärt.

9.1.9 Individual Columns

Die in 7.7 und 8.7.1 beschriebenen individuellen `Column`s werden in der statischen Klasse `IndividualColumns` erzeugt. Diese Klasse enthält eine Methode `CreateColumn`, welche im Wesentlichen die in der GUI eingegebene mathematische Formel, sowie die PointSets für die neu zu erzeugende Column übergeben bekommt. Der Rückgabewert dieser Methode ist dementsprechend vom Typ `Column`. Die in den übergebenen `PointSet`s enthaltenen `Point`s werden kopiert und dabei modifiziert. Die Kopien der `Point`s enthalten in ihrem Werte-Array einen weiteren, der neuen `Column` entsprechenden Wert. Dieser wird mithilfe des in `IndividualColumns` integrierten Formelparsers berechnet. Die private Methode `Evaluate` wertet die als Zeichenkette übergebene Formel für einen `Point` aus und liefert einen neuen `double`-Wert zurück. Wurde in der GUI nichts anderes eingegeben, erzeugt `CreateColumn`

automatisch anhand der Min-/Max-Werte `DefaultColumnProperties` für die neue Column.

Im weiteren Verlauf der Methode `IndividualColumns` werden dann aus den alten `PointSet`s und `PointList`s neue `PointSet`s mit erweiterten `ColumnSet`s angelegt und im `ProjectController` registriert. Danach wird die neu erzeugte `Column` zurückgegeben.

9.1.10 Selektionen

Um markierte Punkte über verschiedene Visualisierungen hervorzuheben, ist es nötig, diese Punktmengen zentral zu verwalten. Diese Funktion übernehmen die `Selection`s. Eine `Selection` besteht aus einer Hashtabelle (generisches `Dictionary` aus dem .NET Framework), in der die selektieren Punkte als Schlüssel hinterlegt werden. Die Hashtabelle wurde gewählt, weil es häufig nötig ist, für einen gegebenen Punkt zu prüfen ob er in der `Selection` enthalten ist, eine Operation die hier in $O(1)$ durchgeführt werden kann. Eine `Selection` verfügt über eine Vielzahl von Methoden um einzelne Punkte oder Punktmengen hinzuzufügen oder zu entfernen ohne die Konsistenz der `Selection` zu beeinträchtigen. Vor allem bedeutet dies, dass Punkte nicht doppelt in einer einzelnen `Selection` vorkommen. Falls es bei diesen Operationen zu einer Veränderung der Punktmenge in der `Selection` kommt, wird ein `SelectionModified`-Event geworfen, damit Visualisierungen darauf reagieren können.

Um auf eine Selection zuzugreifen, steht die Methode `Contains(Point p)` bereit. Ein typischer Zugriff auf die Selection sieht folgendermaßen aus:

```
foreach ( Point p in pointList ){
  if ( ProjectController.CurrentSelection.Contains(p) ) {
    //Markiere Punkt
  }
}
```

Pavel ist in der Lage, verschiedene Selektionen parallel zu verwalten. Eine besondere Bedeutung kommt dabei der `CurrentSelection` zu, die über den `ProjectController` erreichbar ist. Mit dieser Selektion werden nicht nur visuelle Hervorhebungen realisiert, sondern sie wird auch für Operationen wie das Löschen von selektierten Punkten verwendet. Zusätzlich zu dieser ständig vorhandenen Selektion können beliebig viele weitere Selektionen erzeugt werden, die in der statischen `Selections`-Liste im `ProjectController` gespeichert werden. Diese werden beispielweise zur Hervorhebung von Clustergruppen oder einer Paretofront eingesetzt. Meist ist dabei allerdings nicht

von Interesse in welcher Selection ein Punkt gespeichert ist, sondern nur in welcher Farbe er dargestellt werden muss. Hierzu kann die Methode `ProjectController.GetSelectionColor(Point p)` benutzt werden. Diese Methode liefert die Farbe als `ColorOGL`-Objekt zurück.

9.2 PlugIns

Pavel wurde so ausgelegt, dass sehr einfach zusätzliche Anwendungsfälle hinzugefügt werden können. Ein Anwendungsfall (UseCase) besteht aus einer Klasse, die die Daten einliest (Parser) und einer Einzellösungsvisualisierung (Solution). Im Folgenden wird genauer auf die Struktur und die nötigen Klassen eingegangen.

9.2.1 PlugIn-Struktur

Abbildung 9.1 enthält eine Übersicht über die Klassenstruktur eines PlugIns.

Solution

Eine von `Solution` abgeleitete Klasse ist zuständig für die Visualisierung einer Einzellösung. Ebenso kann diese dazu genutzt werden, um verschiedene Einzellösungen untereinander zu vergleichen. `Solution` erbt von `System.Windows.Forms.Form`, stellt also ein komplettes Fenster dar. Außerdem implementiert die Klasse `Solution` das `ICustomTypeDescriptor Interface`, um nur solche Properties im `PropertyControl` anzuzeigen, die das `ShowInProperties` Attribut besitzen.

Da eine Instanz der von Solution abgeleiteten Klasse bereits bei der Wahl eines UseCases, beim Öffnen eines Projektes, erzeugt wird, müssen die darzustellenden `Pavel.Framework.Point`s als Array an die Methode `Initialize` übergeben werden, bevor die Solution mit der Methode `Show` angezeigt wird. Eine solche Klasse muss eine Property `Label` besitzen, die einen String zurückgibt, der die Solution beschreibt. `Solution` enthält die abstrakte Methode `ChangePoint`, der ein Bool-Wert übergeben werden muss, und die einen Integer-Wert zurückgibt, die die Position des nächsten anzuzeigenden `Pavel.Framework.Point`s im Array angibt. Des Weiteren muss die Methode `ChangeMode` implementiert werden, die die Visualisierung an den gewählten Modus (siehe Abschnitt 8.5.6) anpasst.

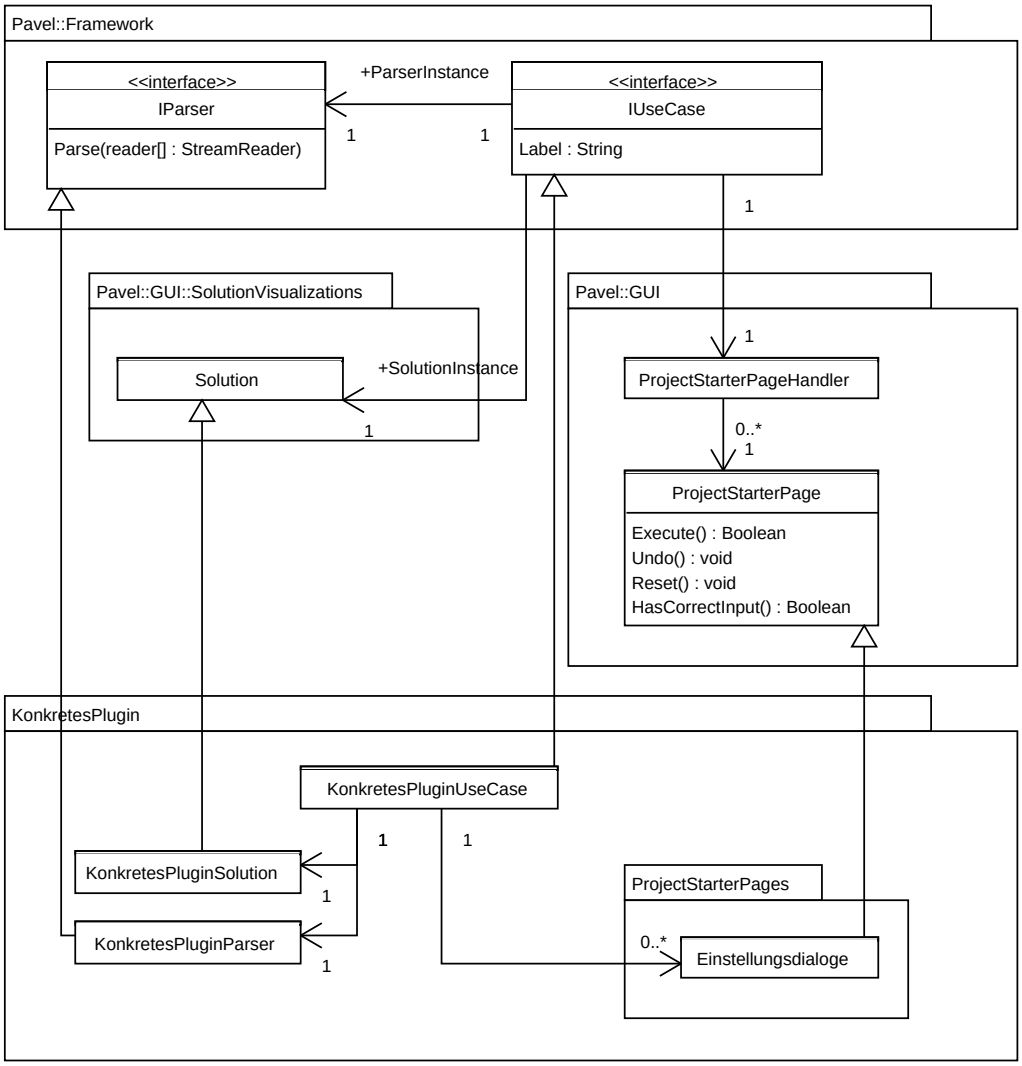


Abbildung 9.1: Die PlugIn-Struktur als grobe Übersicht.

IParser

Eine auf `IParser` basierende Klasse ist zuständig für das Einlesen der Daten. Dieses Interface legt eine Property `Label` fest, die den Namen des Parsers als String zurückgibt. Zusätzlich werden die beiden Methoden `Initialize` und `Dispose` festgelegt. Die wichtigste durch dieses Interface festgelegte Methode ist die Methode `Parse`, die für das Einlesen der Daten zuständig ist. Als Parameter darf eine beliebige Anzahl von `Streamreadern` übergeben werden.

IUseCase

Dieses Interface definiert die Verbindung der einzelnen Komponenten eines Anwendungsfalles. Über die Eigenschaft `ParserInstance` wird eine Instanz des Parsers für diesen Fall zurückgeliefert. Über `SolutionInstance` wird eine Instanz für die Einzelvisualisierung zurückgegeben. `ProjectStarterPages` gibt den Handler für die Einstellungsdialoge des Projektes zurück. Diese werden beim Anlegen des Projektes benötigt, um die Dateien einzulesen.

PluginManager

Der `PluginManager` hält alle verfügbaren PlugIns, die auf `IParser`, `IUseCase` oder `ClusteringAlgorithm` basieren, bereit. Beim Start eines Projektes in Pavel wird ein Anwendungsfall festgelegt, der sich, einmal festgelegt, nicht ändern lässt. Im Suchpfad der PlugIns werden die verfügbaren DLLs durchsucht und, sortiert nach den Interfaces (`IParser`, `IUseCase`) und den von `ClusteringAlgorithm` erbenden Klassen, abgespeichert. Die gefundenen `IUseCase`s werden beim Anlegen eines neuen Projektes angezeigt.

9.2.2 ProjectStarterPage

Beim Anlegen eines neuen Projektes ist es notwendig, Daten einzulesen und Einstellungen zu treffen. Dazu sind die ProjectStarterPages da (siehe Abbildung 9.1 oben rechts). Eine Klasse, die von `ProjectStarterPage` abgeleitet ist, kann mit Bedienelementen, wie z.B. Buttons oder Eingabefeldern, versehen werden. In der Methode `HasCorrectInput()` sollten die Eingabedaten überprüft werden und ein `false` zurückgegeben werden, wenn die Daten nicht korrekt sind. Die Methode `Execute()` sollte schließlich die Werte verarbeiten, d.h. unter anderem die Eingabedaten einlesen. Über den Rückgabewert wird angezeigt, ob die Daten korrekt verarbeitet werden konnten oder nicht. `Reset()` sollte den Inhalt der Bedienelemente zurücksetzen. Mit `Undo()` sollten sämtliche seit dem Start der ersten `ProjectStarterPage` vorgenommenen Änderungen rückgängig gemacht werden.

Es kann eine beliebige Anzahl an StarterPages für einen Anwendungsfall angelegt werden. Die Pages sollten dann in der richtigen Reihenfolge im PluginUseCase dem `ProjectStarterPageHandler` hinzugefügt werden. Dieser übernimmt dann die Steuerung der Pages beim Anlegen des Projektes.

9.3 GUI

Abbildung 7.4 enthält einen groben Überblick des Aufbaus der GUI. Kleine Hilfsklassen sowie Toolbars etc. sind darin nicht enthalten. Oben links in der Abbildung befindet sich das `MainWindow`, die „Zentrale“ der GUI, das Hauptprogrammfenster. Das MainWindow enthält mehrere `VisualizationWindows` und `SolutionWindows`. Die `VisualizationWindows` verwalten die dargestellten `PointSets` und `Spaces`, sowie Parameter, die die Darstellung dieser beeinflussen. Die eigentliche Aufgabe des Darstellens und Renders wird von den `Visualizations` erledigt. Der Sinn dahinter ist zum einen, die Verwaltung der `PointSets` und `Spaces` und deren Darstellungsparameter zu zentralisieren, zum anderen wird es so möglich, die selben dargestellten Inhalte im gleichen Fenster mit einer anderen `Visualization` darzustellen.

Da innerhalb der Visualizations in der Regel zur Darstellung von einer anderen Klasse geerbt werden muss (ParallelPlot/ScatterPlot: `OpenGLControl`, Listing: `ListView`) und C# keine Mehrfachvererbung unterstützt, wird das eigentliche Rendern an eine weitere Klasse delegiert. `Visualizations` funktionieren somit vor allem als Container für das rendernde Control und einen `ToolStrip` und sind darüber hinaus dafür vorgesehen, Events zu verarbeiten und dem `PropertyControl` Eigenschaften bereitzustellen.

Der Aufbau der Einzellösungsvisualisierungen (`SolutionWindows`) lässt sich ähnlich begründen, diese werden ausführlicher in Abschnitt 9.3.3 beschrieben.

9.3.1 MainWindow

Die Klasse `MainWindow` ist die Zentrale der GUI. `MainWindow` ist MDI-Container für die `VisualizationWindows` und die `SolutionWindows`, stellt über verschiedene ToolStrips Zugriff auf Controller- und Modelfunktionen bereit und enthält das `PropertyControl`, welches die Einstellungen des gerade aktiven Fensters zugänglich macht.

Toolstrips Die ToolStrips des MainWindow sind als eigene Klassen, abgeleitet von `ToolStrip`, implementiert. Dadurch können Sie Code zur Eventan- und abmeldung, sowie Eventbehandlung selbst enthalten, ohne die `MainWindow` Klasse damit aufzublähen. Eine Ausnahme bildet der `MainWindow.VisualizationToolStrip`. Dieser wird einfach als leerer ToolStrip initialisiert, aktivierte `VisualizationWindows` fügen dort ihre eigenen Buttons ein wenn sie aktiv werden.

PropertyControl Um die Optionen von VisualizationWindows (dargestellter `Space` und `PointSet`, Art der Visualisierung) für den Anwender bereitzustellen, werden zwei `Windows.Forms.PropertyControl`s verwendet, eines für das `VisualizationWindow`, eines für die `Visualization`. Beide sind in `Pavel.GUI.PropertyControl` zu einem `Control` zusammengefasst. `Pavel.GUI.PropertyControl` registriert sich als EventHandler für das `MdiChildActivate` Event im `MainWindow` und passt die `SelectedObject` Properties der beiden `Windows.Forms.PropertyControls` an, sobald ein anderes `VisualizationWindow` oder `SolutionWindow` angewählt wird.

Die `Windows.Forms.PropertyControl`s ermitteln per Reflection die öffentlichen Properties ihres `SelectedObjects` und stellen Editierfelder dafür bereit. Um zu verhindern, dass *alle* Properties angezeigt werden anstatt nur derer, die für die Arbeit mit Pavel wichtig sind, implementieren alle Klassen, die Ziel eines PropertyControls werden können (`Visualization`, `VisualizationWindow`, `Pavel.GUI.Visualizations.Solution`) das `ICustomTypeDescriptor Interface`. Dessen Methoden werden alle mittels Delegation an `System.ComponentModel.TypeDescriptor` so behandelt wie sie es normalerweise auch würden, mit der Ausnahme von `GetProperties(Attribute[])`. `GetProperties(Attribute[])` liefert eine Liste aller Properties eines Objektes zurück, die über die übergebenen Attribute verfügen, und wurde so implementiert, dass zu dieser Liste das Attribut `Pavel.GUI.Visualizations.PropertyControl.ShowInPropertiesAttribute` hinzugefügt wird. Wenn nun ein `PropertyControl` die Attribute ermittelt, werden nur solche zurückgegeben, die im Quellcode mit `[ShowInProperties]` ausgezeichnet wurden.

9.3.2 Visualisierungen

Visualisierungen werden in Pavel durch Erweitern von `Visualization` implementiert. Per Reflection im `DataToolStrip`/`MainMenuStrip` werden alle Visualisierungen automatisch gefunden und dem Anwender bereitgestellt. In `Visualization` werden Methodensignaturen für EventHandler deklariert, `ICustomTypeDescriptor` für die Zusammenarbeit mit dem PropertyControl implementiert sowie die Anbindung an ein VisualizationWindow vorbereitet.

Im Folgenden werden einige der zu implementierenden Methoden beschrieben:

VisualizationStandardToolStrip Diese Property liefert einen ToolStrip zurück, der Steuerungselemente für die Visualisierung enthält.

Der ToolStrip wird vom VisualizationWindow in das ToolStripPanel im MainWindow eingefügt. Wird kein ToolStrip benötigt, kann `null` zurückgegeben werden.

Control Ein `System.Windows.Forms.Control`, oder eine abgeleitete Klasse, die die eigentliche Darstellung übernimmt.

UpdateSpace Diese Methode wird vom VisualizationWindow aufgerufen wenn der dargestellte Space geändert wurde. Sollte so implementiert werden, dass entsprechend auf die Änderung reagiert wird.

SelectionModified Dieser EventHandler wird automatisch beim `ProjectController` registriert und sollte so implementiert werden, dass entsprechend reagiert wird, wenn die aktuelle Auswahl geändert wird.

9.3.3 Einzelvisualisierungen

Die Erstellung von Einzelvisualisierungen geschieht durch Klassen, die von `Solution` implementieren. Eine genauere Beschreibung befindet sich in Abschnitt 9.2.1.

9.3.4 OpenGLControl

Um die Arbeit mit dem in Pavel verwendeten Tao OpenGL-Framework zu erleichtern, wurde eine Control Klasse entwickelt, die es erlaubt schnell und sauber 3D-Visualisierungen zu realisieren. Sie basiert zum größten Teil auf dem `SimpleOpenGLControl` aus Tao, wurde jedoch um einige zusätzliche Methoden erweitert, die in abgeleiteten Klassen verwendet werden können um wiederkehrende Aufgaben zu erledigen.

Viewports, Sichtkegel, Zeichenbereiche Es wird empfohlen, sämtliche OpenGL Zeichenoperationen durch Skalierung auf den Einheitswürfel um den Nullpunkt zu beschränken und diesen dann nach Bedarf zu drehen. Grund dafür ist, dass die Projektionsberechnung und andere Hilfsfunktionen in OpenGLControl von eben diesem Bereich ausgehen, bei Abweichungen können weit reichende Anpassungsarbeiten nötig werden. Da die Berechnung der Projektionsmatrix vorgegeben ist, wird empfohlen, ein eventuell benötigte Zoomfunktion mittels Skalierung zu realisieren.

RenderScene Beim Design von `OpenGLControl` wurde darauf geachtet, gleichförmige Abläufe schon fertig bereitzustellen. Dazu gehört vor allem das Rendern mit dem Setup der ModelView- und Projektionsmatrizen. Die einzigen Bestandteile dieses Ablaufes die von einer erbenden Klasse implementiert werden müssen, sind die Berechnung der Modelview-Matrix (Translation, Drehung, Skalierung der Punkte auf den Einheitswürfel) in der Methode `SetupModelViewMatrixOperations` sowie die Beschreibung der Szene in der Methode `RenderContent`.

Felder und Eigenschaften

keepAspect Eine Bool Variable die steuert, ob beim Berechnen der Parameter für glOrtho und glFrustrum das Seitenverhältnis gemäß dem Seitenverhältnis des Viewport angepasst wird. Ist diese Variable true, führt eine Verbreiterung des Viewports auch zu einer Verbreiterung des OpenGL Sichtbereiches und eine Verzerrung der dargestellten Szene wird verhindert.

halfHeight Hier wird die halbe Höhe des Sichtbereiches von glFrustum/glOrtho angegeben. Ist standardmäßig so gewählt, dass ein Einheitswürfel in jeder beliebigen Orientierung noch ungefähr in den orthogonalen Sichtbereich passt.

halfEyeDistance und stereoMode Diese beiden Eigenschaften steuern den Steroskopischen Modus.

Window Aspect Eigenschaften

WindowAspect Das Seitenverhältnis Breite : Höhe. Ist `keepAspect` false, wird das Seitenverhältnis immer als 1:1 angenommen, was dazu führt, dass keine Korrektur des Sichtbereiches vorgenommen wird.

HalfHeight und HalfWidth Geben `halfHeight` und ein entsprechend `keepAspect` angepasstes `HalfWidth` zurück.

AspectCap Wenn `WindowAspect` kleiner als 1 ist, ist `HalfWidth` kleiner als `HalfHeight`. Dies kann zu unerwünschtem Abschneiden des Bildinhaltes an den linken und rechten Bildrändern führen. Um das zu verhindern muss `HalfWidth` aufskaliert werden. Damit dann das Seitenverhältnis noch stimmt, muss `HalfHeight` ebenfalls aufskaliert werden. AspectCap ist genau dieser Skalierungsfaktor.

HalfHeightCapped und HalfWidthCapped Entspricht `HalfHeight` und `HalfWidth` mit einer bereits vollzogenen Skalierung gemäß `AspectCap`.

Setup Methoden

SetupViewPort Passt den OpenGL Viewport an die aktuelle Größe des Controls an. Sie muss normalerweise nur bei einem Resize-Event aufgerufen werden.

SetupProjection Bereitet eine isometrische Projektionsmatrix vor.

SetupProjectionFlat Bereitet eine zweidimensionale Projektionsmatrix vor, die hilfreich beim Zeichnen von GUI Elementen oder Labels ist. In der so generierten Perspektive können zweidimensionale Koordinaten im Viewport entweder relativ $[0, 1]$ oder absolut $[width, height]$ angegeben werden.

SetupProjectionStereo Bereitet eine perspektivische Projektionsmatrix vor, abhängig vom `mode` Parameter für das linke oder rechte Auge.

SetupModelView Bereitet die ModelView Matrix vor. Die eigentlichen Matrixoperationen stecken in der Funktion SetupModelViewMatrixOperations, die von der erbenden Klasse zu überschreiben ist.

SetupModelViewMatrixOperations Ist von der erbenden Klasse zu überschreiben und sollte nur OpenGL Matrixoperationen enthalten, die die ModelView Matrix für die Darstellung der Szene erzeugen.

Hilfsmethoden

PushMatrices, PopMatrices Führt `Pushmatrix` und `Popmatrix` parallel für Projektions- und ModelviewMatrix aus.

RelFromAbs/AbsFromRel Rechnen relative in absolute 2D-Koordinaten um und umgekehrt.

9.3.5 OpenGL Funktionen

CopyPixels Die `CopyPixels`-Methode wurde geschrieben, um schnell und einfach Teile eines Buffers in einen anderen zu kopieren. Damit ist es möglich, einen Teil der Szene im Frontbuffer zu übermalen, beispielsweise mit dem Picking-Rechteck und anschließend das Original aus dem Backbuffer pixelweise wieder herzustellen ohne den gesamten Bereich neu rendern zu müssen.

MakeCurrentContext Setzt den aktuellen OpenGL-Kontext auf das offene Fenster.

CheckErrors Überprüft den Errorzustand von OpenGL und gibt eine Fehlermeldung aus, falls ein Fehler aufgetreten ist.

PrintText Rendert einen Text in die Szene.

9.4 Clustering

Jegliche Implementierung für das Clustering befindet sich im Namespace `Clustering` und lässt sich relativ unabhängig vom Framework und der GUI erweitern. Für eine effiziente Umsetzung und die strukturierte Rückgabe der Daten für Sonderfälle (Beispiel: Hierarchisches Clustern) ist jedoch eine genaue Kenntnis der Datenstrukturen notwendig. Auf der anderen Seite bringt das vorhandene Clustering-Framework schon viele nützliche Funktionen mit, die den effizienten Umgang mit großen Mengen von Daten erleichtern, sodass ohne größere Vorkenntnisse die vorhandenen Algorithmen erweitert/abgeleitet werden können.

Zu Beginn erfolgt eine ausführliche Auslistung und Erläuterung der Clustering-Klassen und Funktionen. Anschließend soll beispielhaft gezeigt werden, wie ein neuer Clustering-Algorithmus entsteht.

9.4.1 Einbettung ins Framework

Der Namespace `Clustering` enthält zunächst die benötigten Datenstrukturen:

Cluster Variante eines `Point`, der zusätzlich ein Label und ein angehängtes `PointSet` enthält. Ein `Cluster` ist also ein Datenpunkt, der als

Repräsentant für eine beliebige Menge von Punkten steht. Um verschiedene Funktionalitäten beim Clustern zu unterstützen ist es außerdem möglich, den Wert eines `Cluster`s zu verändern (neu zu definieren).

ClusterSet Eine Variante eines `PointSet`, die zusätzlich einen Verweis auf das zugrunde liegende `PointSet` enthält, sowie den Algorithmus, der zum Erzeugen des `ClusterSet` benutzt wurde. Außerdem stellt es die Funktion `PointSetFromClusters(String name)` bereit, die alle über `Cluster` dieses `ClusterSet`s iteriert und deren angehängte `Point`s wieder zu einem `PointSet` vereinigt. Sie ermöglicht es, die strukturierten Clusterdaten zu bearbeiten (z.B. ganze `Cluster` löschen) und anschließend die übrig bleibenden Punkte unter Aufgabe der Clusterstruktur weiterzuverarbeiten.

HierarchicalClusterList Diese Variante der `PointList` verhält sich durch den gesetzten Wert `defaultClusterCount` wie eine normale `PointList`. Gespeichert wird aber ein komplettes Dendrogramm, bestehend aus Instanzen der Klasse `Node`. Vom `rootnode` ausgehend, ist dies ein binärer Baum, in dem ein horizontaler Schnitt eine Clusterlösung repräsentiert. Dazu werden in den `Node`s Indizes gespeichert, die die Reihenfolge der Zerlegung der initialen Punktmenge angeben.

Daneben gibt es die Hilfsklassen:

ClusteringSelectDialog Stellt den Auswahldialog dar.

ArgsDescriptionAttribute Beschreibt individuelle Parameter für Clusteralgorithmen und deren Darstellung im Dialog.

ClusteringArgumentControl Ein Standard-Control um mittels `ArgsDescriptionAttribute` beschriebene Parameter in einem Control anzuzeigen. Abweichend von diesem Mechanismus, kann jeder Clusteringalgorithmus ein beliebiges eigenes Control an den Auswahldialog übergeben.

ClusteringAlgorithm

`ClusteringAlgorithm` ist die Basisklasse für die eigentlichen Algorithmen. Die Eigenschaften und Funktionen dieser Basisklasse sollen im Folgenden beschrieben werden:

Eigenschaften

ErrorMessage Kann von abgeleiteten Klassen überschrieben werden um Fehlerinformationen vor einem Abbruch festzuhalten. Wird benutzt, um diese nach Abbruch des Clusterns anzuzeigen.

SaveAbortRequested Gibt an, ob der Algorithmus zum Abbruch aufgefordert worden ist. Dies sollte regelmäßig im Algorithmus überprüft werden. Hat es den Wert `true` sollte der Algorithmus seine Berechnungen abbrechen und ein Zwischenergebnis wie ein Endergebnis zurückliefern.

ArgumentControl Dieses Property kann überschrieben werden um Nichtstandard-Controls im Clustering-Dialog anzuzeigen.

ColumnSet Enthält das `ColumnSet` des für das Clustern ausgewählten `Space`.

RelevantSpace Enthält eine Kopie des für das Clustern ausgewählten `Space` mit ausschließlich relevanten `Columns`.

ScaledData Enthält eine Kopie der relevanten Werte des `PointSets`, die entsprechend des `Space` skaliert wurden. In der ersten Hierarchie befinden sich die `PointLists`, in der zweiten die Punkte und in der dritten die Werte.

PointSet Das `PointSet` für den Clustervorgang.

Space Der `Space` für den Clustervorgang.

RelevantColumns Ein `Bool`-Array, das für jede `Column` im `Space` angibt, ob sie bei der Distanzberechnung berücksichtigt werden soll.

Name Der Name für das resultierende `ClusterSet`.

Methoden

ToString() Zu überschreibende Methode für die Bezeichnung des Algorithmus.

SignalProgress(int progress, string message) Diese Methode sollte regelmäßig aufgerufen werden, um Informationen über den Fortschritt der Clustervorgang auszugeben. `progress` ist dabei ein Wert zwischen 0 und 1000.

DoClustering() Diese Methode muss überschrieben werden und beinhaltet den eigentlichen Clusteringvorgang. Als Eingabedaten stehen die Felder und Properties von `ClusteringAlgorithm` bereit, in denen zusätzlich Daten vorberechnet werden. Der Rückgabewert sollte eine gültige `PointListe` mit `Clustern` sein oder `null` bei einem Abbruch.

Start() Diese Funktion wird von außen zum Starten des Clusterns aufgerufen und führt Vorberechnungen aus. Anschließend wird `DoClustering` aufgerufen, das Ergebnis schließlich ausgewertet und in ein `ClusterSet` verpackt. Diese Methode sollte immer in einem unabhängigen Thread ausgeführt werden.

InterruptClustering() Diese Methode wird von außen aufgerufen um den Clusteringalgorithmus aufzufordern, seine Arbeit zu beenden.

9.4.2 Einen neuen Clusteralgorithmus erstellen

Um zu verdeutlichen, wie ein neuer Clusteringalgorithmus implementiert werden kann, soll dies im Folgenden beispielhaft gezeigt werden. Dazu wird eine Variante des AMLA (siehe 6.3.1 auf Seite 113) implementiert und Schritt für Schritt das Vorgehen erklärt.
Zunächst wird eine neue Klasse mit dem Namen MLA.cs angelegt, die die folgenden Eigenschaften aufweist:

- Der Namespace sollte `Pavel.Clustering` lauten.
- Die Klasse muss `public` sein.
- Sie muss Unterklasse der abstrakten Klasse `ClusteringAlgorithm` sein.
- Die Klasse muss das Attribut: `[Serializable()]` besitzen.

Durch die Ableitung von der Klasse `ClusteringAlgorithm` müssen mindestens die Methode `ToString` implementiert werden, sowie die Klasse `protected abstract PointList DoClustering()`. Die Erste sieht folgendermaßen aus:

```
public override string ToString() { return "MLA"; }
```

In `DoClustering` wird der eigentliche Algorithmus implementiert. Zusätzlich benötigt MLA noch Parameter, für die Felder und Properties angelegt werden, wobei letztere mit Ableitungen des `ArgsDescriptionAttribute` ausgezeichnet werden sollten, damit sie automatisch im Clusterdialog erscheinen. In diesem Fall wird zunächst ein Property angelegt:

```
[Spinner("Number of Clusters",
                "Specifies the number of clusters.",
                1.0, 1000000.0, 0, 1)]
public int NumberOfClusters {
    get { return numberOfClusters; }
    set { numberOfClusters = value; }
}
```

In der Methode `DoClustering()` wird das eigentliche Clustering durchgeführt. Als Input dienen die Felder und Properties von ClusteringAlgorithm, die Ausgabe ist eine `PointList`, die `Cluster` enthält. Konkret implementieren wir zunächst eine Sicherheitsabfrage und es wird die `PointList` für die Ausgabe erstellt:

```
PointList clusterList = new PointList(ColumnSet);

//Check:
if (PointSet.Length < NumberOfClusters) {
    ErrorMessage = "Number of Clusters is greater than
                    the size of the PointSet!";
    return null;
}
```

Um nicht im ersten Schritt alle n^2 Distanzen untersuchen zu müssen, wählen wir als erstes ein zufälliges Zentrum. Dazu benötigen wir ein weiteres Property:

```
[Spinner("Random Seed", "Initializes the Clustering-Algorithm
                         Random-Generator.
                         Use for repeatable behaviour",
        int.MinValue, int.MaxValue, 0, 1)]
public int RandomSeed {
    get { return randomSeed; }
    set { randomSeed = value; }
}
```

Außerdem fügen wir folgenden Code in `DoClustering()` ein:

```
Random r = new Random(RandomSeed);
int[] centers = new int[NumberOfClusters];
centers[0] = r.Next(0, PointSet.Length);
```

Zur besseren Kontrolle über die Daten werden hier zunächst nur die Indizes der Punkte verwendet. Während des Clusterings ist gewährleistet, dass die Indizierung stabil bleibt und mit den Werten in `ScaledData` und `ScaledFlatData` synchronisiert ist.

Im Weiteren wird nach Vorgabe durch den Algorithmus noch die endgültige ClusterList erstellt. Dabei sollte sichergestellt werden, dass falls Zwischenergebnisse vorliegen und das Flag SaveAbortRequest gesetzt ist, der Clustervorgang mit der momentanen Lösung beendet wird. Im konkreten Fall ist dies möglich sobald die Clusterzentren bestehe und der Algorithmus lediglich noch diese iterativ anpasst.

Das weitere Vorgehen ist sehr spezifisch und nur auf den eigentlichen Algorithmus zugeschnitten. Die grundlegenden, beschriebenen Schritte können für eigene Algorithmen fast unverändert übernommen werden. Weitere Informationen finden sich in den reichhaltigen Klassendokumentationen, diesem Handbuch oder auf Anfrage.

Abbildungsverzeichnis

Literaturverzeichnis

[Ach04] Achtert, E.: *Inkrementelles hierarchisches Clustering*, Ludwig Maximilians Universität München, Diplomarbeit, 2004.

[Aso96a] Asokarathinam, P.: *Introduction to parallel projection.* 1996 Florida Institute of Technology, Tutorial: The wonderful world of computer graphics. WWW: `http://www.cs.fit.edu/~wds/classes/cse5255/thesis/paraProj/paraProj.html` [Stand: Oktober 2006].

[Aso96b] Asokarathinam, P.: *Perspective projections.* 1996 Florida Institute of Technology, Tutorial: The wonderful world of computer graphics. WWW: `http://www.cs.fit.edu/~wds/classes/cse5255/thesis/persProj/persProj.html` [Stand: Oktober 2006].

[BA04] Beck, Kent; Andres, Cynthia: *Extreme Programming Explained: Embrace Change (2nd Edition).* Boston: Addison-Wesley, 2004.

[Bat02] Bathe, K.J.: *Finite-Elemente-Methoden.* 2., vollst. neu bearb. und erw. Aufl. Springer-Verlag, 2002.

[Bay06] Bayer MaterialScience: *Werkzeugtemperierung.* 2006. WWW: `http://plastics.bayer.com/plastics/emea/de/technology/1207/article.jsp?docId=6449&cid=1207` [Stand: Oktober 2006].

[Bäc94] Bäck, T. *Evolutionary Algorithms in Theory and Practice.* 1994.

[Ben80] Bentley, J. L.: Multidimensional Divide-and-Conquer. In: *Communications of the ACM* 23. 1980.

[Bic06] Bichler, M.: *Kunststoffteile fehlerfrei spritzgießen.* Hüthing GmbH, 2006.

[Bär94] Bärtschl, W.A.: *Geometrische Linear- und Schattenperspektive*. Vieweg, 1994.

[CCHAZ05] Coello Coello, C.A.; Hernandez Aguirre, A. ; Zitzler, E.: *Evolutionary Multi-Criterion Optimization, LNCS 3410*. Springer, 2005.

[CCVVL02] Coello Coello, C.A.; Van Veldhuizen, D.A. ; Lamont, G.B.: *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, 2002.

[CS00] Chai, J.; Shum, H.: Parallel Projections for Stereo Reconstruction. In: *Proceedings of IEEE Computer Vision and Pattern Recognition 2000*, 2000.

[Deb02] Deb, K.: *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Ltd, 2002.

[DH04a] Ding, C.; He, X.: K-means Clustering via Principal Component Analysis. In: *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, ACM Press, 2004, S. 29.

[DH04b] Ding, C.; He, X.: Principal Component Analysis and Effective K-Means Clustering. In: *SDM*, 2004.

[Dür25] Dürer, A. *Underweysung der messung mit Zirckel un richtscheyt in linien ebnen vnd gandzen corporen*. 1525.

[FDFH95] Foley, J.D.; van Dam, A.; Feiner, S.K. ; Hughes, J.F.: *Computer Graphics: Principles and Practice*. 2nd Edition. Addison Wesley, 1995.

[Fer00] Ferwerda, J.A.: *Perceiving Size and Distance*. 2000 Cornell University. WWW: `http://www.graphics.cornell.edu/~jaf/projects/pn/space.html` [Stand: Oktober 2006].

[FKN71] Fucke, R.; Kirch, K. ; Nickel, H.: *Darstellende Geometrie*. Verlag Harri Deutsch, 1971.

[Fog95] Fogel, D.B. *Introduction to parallel projection*. 1995.

[Gas75] Gastrow, H.: *Der Spritzgieß-Werkzeugbau in Beispielen*. 2. Auflage. Carl Hanser Verlag, 1975.

[Gas98] Gastrow, H.: *Der Spritzgieß-Werkzeugbau in Beispielen*. 5. Auflage. Carl Hanser Verlag, 1998.

[Gib50] Gibson, J.J.: *The Perception of the Visual World*. Houghton Mifflin, 1950.

[Han99] Hansen, M.: *Stereosehen - Ein verhaltensbasierter Zugang unter Echtzeitbedingungen*, Christian-Albrechts-Universität zu Kiel, Dissertation, 1999.

[HM79] Hwang, C.; Masud, A.: *Multi Objective Desicion Making - Methods and Applications: A state-of-the-art Survey*. Springer, 1979.

[Inf06] Infitec GmbH: *Infitec*. 2006. WWW: `http://www.infitec.net/infitec.pdf` [Stand: Oktober 2006].

[IS05] Illschner, B.; Singer, R.: *Werkstoffwissenschaften und Fertigungstechnik; Eigenschaften, Vorgänge, Technologien*. Springer Verlag, 2005.

[JLHE01] Jones, G.; Lee, D.; Holliman, N. ; Ezra, D. *Controlling Perceived Depth in Stereoscopic Images*. 2001. WWW: `http://www.dur.ac.uk/n.s.holliman/Presentations/EI4297A-07Protocols.pdf` [Stand: Oktober 2006].

[JMF99] Jain, A.K.; Murty, M.N. ; Flynn, P.J.: Data Clustering: A Review. In: *ACM Computing Surveys* 31, Nr. 3, S. 264–323. 1999.

[Keb94] Kap. 1.5 in: Kebeck, G.: *Wahrnehmung: Theorien, Methoden und Forschungsergebnisse der Wahrnehmungspsychologie*. Juventa Verlag, 1994, S. 60–69.

[KK97] König, W.; Klocke, F.: *Fertigungsverfahren Drehen, Fräsen, Bohren*. 5.Auflage. Springer-Verlag, 1997.

[Kri02] Kriegel, H.-P.: Hauptseminar KDD - Session: Clustering. Ludwig Maximilians Universität München, 2002. Forschungsbericht.

[KSH02] Kosara, R.; Sahling, G.N. ; Hauser, H.: Linking Scienfitic and Information Visualization with Interactive 3D Scatterplots.

In: *Technical Report TR-VRVis-2002-044 at the VRVis Research Center in Vienna, Austria* 2002.

[Lüc00] Lück, M.: *Echtzeitfähige Zwischenbildinterpolation für die stereoskopische Bewegtbildwiedergabe*, Universität Dortmund, Dissertation, 2000.

[LTDZ01] Laumanns, M.; Thiele, L.; Deb, K. ; Zitzler, E.: On the Convergence and Diversity-Preservation Properties of Multi-Objective Evolutionary Algorithms. In: *Tech. Report, Swiss Federal Institute of Technology, ETH Zürich* 108. 2001.

[Meh05] Mehnen, J.: *Mehrkriterielle Optimierverfahren für produktionstechnische Prozesse*. Vulkan Verlag Essen, 2005.

[Mie79] Mietzel, G.: *Wege in die Psychologie*. Klett, 1979.

[Mie96] Mietzel, G.: *Wege in die Psychologie*. Klett-Cotta, 1996.

[Mül05] Müller, H. *Skript zur Vorlesung Mensch-Maschine-Interaktion, Universität Dortmund*. 2005.

[Mül06] Müller, H. *PG-Antrag für PG500, Universität Dortmund*. 2006.

[MM06] Mehnen, J.; Michelitsch, Th. *Optimization of Producion Engineering Problems with Discontinuous Cost-Functions*. Proceedings of the 5th CIRP International Seminar on Intelligent Computation in Manufacturing Engineering. 2006.

[MMBBS04] Mehnen, J.; Michelitsch, Th.; Bartz-Beielstein, Th. ; Schmitt, K.: Evolutionary optimization of mould temperature control strategies: encoding and solving the multiobjective problem with standard evolution strategy and kit for evolution algorithm. In: *Journal of Engineering Manufacture, Vol.218, Professional Engineering Publishing* 2004.

[MMM99] Menges, G.; Michaeli, W. ; Mohren, P.: *Spritzgießwerkzeug, Anleitung zum Bau von Spritzgieß-Werkzeugen*. Carl Hanser Verlag, 1999.

[MMS05] Mehnen, J.; Michelitsch, Th. ; Stautner, M.: Multiobjective Optimization of Mould Temperature Control Designs and

Milling Strategies. In: *Proceedings of the First Invited COST 526 (APOMAT)* , S. 78–87. 2005.

[PKH04] Piringer, H.; Kosara, R. ; Hauser, H. *Interactive Focus+Context Visualization with Linked 2D/3D Scatterplots*. 2004. WWW: `http://www.vrvis.at/via/research/2d3dscatterplots/index.html` [Stand: Oktober 2006].

[Pru06] Pruscha, H.: *Statistisches Methodenbuch*. 1. Springer-Verlag, 2006.

[PW90] Parrish, R.V.; Williams, S.P. *Determination of Depth-Viewing Volumes for Stereo Three-Dimensional Graphic Displays*. 1990. WWW: `http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19900013649_1990013649.pdf` [Stand: Oktober 2006].

[Sch95] Schwefel, H.-P.: *Evolution and Optimum Seeking*. John Wiley & Sons, Ltd, 1995.

[SE06] Sharp Electronics: *User Manual LL-151-3D 15" XGA LCD Monitor*. 2006.

[Stö03] Stöcker, H.: *Taschenbuch mathematischer Formeln und moderner Verfahren*. Verlag Harri Deutsch, 2003.

[SZ05] Stautner, M.; Zabel, A.: Optimizing the Multi-Axis Milling Process via Evolutionary Algorithms. In: *Proceeding of the 8th CIRP International Workshop on Modeling of Machinig Operations* , S. 363–370. 2005.

[TD04] Tönshoff, H.K.; Denkena, B.: *Spanen Grundlagen*. 2. erweiterte und neu bearbeitete Auflage. Springer-Verlag, 2004.

[TZ02] Tschiersch, L.; Zerbst, M.: The Advanced-Maximum-Linkage Clustering-Algorithm / Universität Dortmund. 2002. Technischer Bericht.

[VDW05] VDW. *Magazin des Verbands Deutscher Werkzeug- und Formenbauer e.V.* 2005.

[Vog04] Vogel, H.: Körper-Bau, Teil 4: Computer Aided Manufacturing. In: *c't Magazin für Computer Technik* 11, S. 222–229. 2004.

[Wan06] Wang, W.: Research Seminar - Clustering. University of North Carolina at Chapel Hill, 2006. Forschungsbericht.

[Wei00] Weinert, K. *Skriptum zur Vorlesung „Fertigungsverfahren I“.* Vorlesungsskript Institut für Spanende Fertigung, Universität Dortmund. 2000.

[Wie97a] Schulz zur Wiesch, J. *Primäre Faktoren für Tiefenwahrnehmung.* 1997. WWW: `http://www.jszw.de/3d_wahrnehmung/tiefe.html` [Stand: Oktober 2006].

[Wie97b] Schulz zur Wiesch, J. *Sekundäre Faktoren für Tiefenwahrnehmung.* 1997. WWW: `http://www.jszw.de/3d_wahrnehmung/tiefe.html` [Stand: Oktober 2006].

[Wik] Wikipedia.de. *Darwinismus.* WWW: `http://de.wikipedia.org/wiki/Darwinismus` [Stand: Oktober 2006].

[Wik06] Wikipedia. *Raumwahrnehmung — Wikipedia, Die freie Enzyklopädie.* 2006. WWW: `http://de.wikipedia.org/w/index.php?title=Raumwahrnehmung&oldid=19195528` [Stand: 20. November 2006].

[WO53] Wallach, H.; O'Connell, D.N.: The kinetic depth effect. In: *Journal of Experimental Psychology* 45, S. 205–217. 1953.

[YWR03] Yang, J.; Ward, M.O. ; Rundensteiner, E.A.: Interactive hierarchical displays: a general framework for visualization and exploration of large multivariate data sets. In: *Computers & Graphics* 27, Nr. 2, S. 265–283. 2003. WWW: `http://dx.doi.org/10.1016/S0097-8493(02)00283-2` [Stand: Oktober 2006].

[Zab05] Zabel, A.: Einsatzfelder der mehrachsigen Frässimulation. In: *wt Werkstattstechnik online* 95, S. 56–61. 2005.

[ZDT+01] Zitzler, E.; Deb, K.; Thiele, L.; Coello Coello, C.A. ; Corne, D.: *Evolutionary Multi-Criterion Optimization.* Springer, 2001.

[Zer01] Zerbst, M.: *Die pixelbasierte Clusterung von Luftaufnahmen im Rahmen von Erosionsuntersuchungen*, Universität Dortmund, Dissertation, 2001.

[ZLT01] Zitzler, E.; Laumanns, M. ; Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In: *Tech. Report, Swiss Federal Institute of Technology, ETH Zürich* 103. 2001.

[ZT99] Zitzler, E.; Thiele, L.: Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. In: *IEEE Trans. Evol. Computing* 3. 1999.

Index

Symbole

A

B

C

D

E

N

O

P

Q

R

S

T

U

V

W

Z