Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Vitalij Kagadij

Beispiel-basierte Generierung von Silhouetten

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Vitalij Kagadij

Beispiel-basierte Generierung von Silhouetten

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Philipp Jenke
Zweitgutachter: Prof. Dr. Wolfgang Fohl

Eingereicht am: 1. Februar 2016

Vitalij Kagadij

Thema der Arbeit
Beispiel-basierte Generierung von Silhouetten

Stichworte
Prozedurale Generierung/Synthese, Hauptkomponentenanalyse

Kurzzusammenfassung
Prozedurale Generierung spielt heutzutage eine große Rolle in der Welt von Computergrafik. Besonders in Computerspiele finden die prozedurale Verfahren Anwendung, um den Inhalt schnell und platzsparend zu produzieren. Modellierung von Silhouetten ist ein weiteres Bereich, wo die prozedurale Generierung den Computer-Grafik-Künstler helfen könnte, ihre Aufgaben schneller und effizienter zu machen.
Vitalij Kagadij

Title of the paper
Example-based generation of silhouettes

Keywords
Procedural generation/synthesis, principal component analysis

Abstract
Procedural generation nowadays plays an important role in the world of computer graphics. Procedural methods are especially good in computer games to produce content quickly and compactly. Modelling of silhouettes is a further area where the procedural generation could help the computer graphic artists to make their tasks more quickly and efficiently.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einfürung

1.1. Motivation

Prozedurale Synthese ist ein mächtiger Mechanismus in Computergrafik, welcher die investierte Zeit und den Aufwand bei der Modellierung eines 3D-Objektes mäßig verringert. Anstatt alles manuell zu kreieren, braucht man nur ein Paar vom wichtigen Parameter, welche man manipulieren kann, um eine Menge von verschiedenen Objekten zu erzeugen. Das ist etwa für die 3D-Künstler relevant, die ihre Aufgaben effizienter machen können. Modellierung von Silhouetten ist ein weiteres Bereich, wo man prozedurale Synthese anwenden kann.

Das Ziel meiner Masterarbeit ist es, einen neuen Ansatz für die prozedurale Modellierung der Silhouetten zu erarbeiten. Als Silhouette wird die zweidimensionale Darstellung vom Umriss eines Gegenstandes bezeichnet. In dieser Arbeit werde ich die Funktionalität meines Ansatzes am Beispiel von Kraftfahrzeug-Silhouetten zeigen. Ebenso werde ich zeigen, dass dieser Ansatz auch für andre Domäne erweitert werden kann. Als zweite Domäne dienen die Schmetterling-Silhouetten. Gewünscht ist es, mit diesem Verfahren eine riesige Menge der verschiedenen Ergebnissen sehr schnell und mit wenig Aufwand zu erzeugen.

1.2. Prozedurale Generierung

Die Prozedurale Synthese oder die prozedurale Generierung bezeichnet in der Informatik solche Verfahren, welche in Echtzeit und zur Laufzeit des Computerprogrammes die Programminhalte (wie Texturen, virtuellen Welten, 3D-Objekte und sogar Musik) erzeugen, ohne dass diese Inhalte vorher fest angelegt oder vorprogrammiert sind. Diese Inhalte werden nicht zufällig erzeugt, sondern die Generierung folgt deterministischen Algorithmen, damit die Inhalte immer gleich sind, wenn die Ausgangsbedingungen auch gleich waren.

Die Schlüsseleigenschaft der prozeduralen Generierung ist die Tatsache, dass sie das Objekt (Geometrie, Textur oder Effekt) als eine Sequenz von Anweisungen beschreibt, und nicht als ein statischer Datenblock. Diese Anweisungen können dann aufgerufen werden, um eine Instanz von diesem Objekt zu erstellen. Die Beschreibung kann parametrisiert werden, damit diese

Instanzen variierte Charakteristiken haben. Ein typisches Beispiel eines solchen Verfahrens ist die Bevölkerung von einem Wald mit prozedural erstellten einzigartigen Bäume [SpeedTree].

Abbildung 1.1.: [SpeedTree] Screenshot demonstriert prozedural generierte und in Echtzeit gerenderte Bäume

Solche prozedurale Verfahren verwendet man, um abwechslungsreiche Objekte zu erzeugen. Eine der Grundtechniken, die verwendet werden könnte, ist die Generierung von 3D-Primitiven mit zufälligen Parameter. Zum Beispiel ein Quader mit zufälliger Höhe. Die einfachen Algorithmen, welche die pseudo-zufälligen Funktionen verwenden, können benutzt werden, um "rauschen"zu generieren. DIeses Verfahren wird zum Beispiel in Texturierung gebraucht. Die komplexeren rekursiven Algorithmen, wie z. B. Fraktale oder L-Systeme, kann man verwenden, um organische Strukturen aus der Natur wie die Schneeflocken oder Bäume nachzubauen.

Alle diese Verfahren müssen eine bestimmte Reihe von Eigenschaften erfüllen. [Ebert et al.] identifizieren die folgenden wichtigen Eigenschaften von prozeduralen Verfahren:

- **Abstraktion:** die Daten sind nicht im herkömmlichen Sinn angegeben, stattdessen sind die Details in einen Algorithmus oder einer Reihe von Prozeduren abstrahiert. Diese

werden von Computer bearbeitet und aufgerufen, falls ein Bedarf entsteht. So kann der Operator mit dem Minimum von Daten das Model sehr einfach manipulieren, ohne dass er viel über die Implementierung weiß.

- **Parametrisierte Steuerung:** Parameter sind definiert und angepasst, was einem spezifischen Verhalten in prozeduraler Generierung genau entspricht. Der Entwickler kann so viele nützliche Steuerelemente definieren, dass der Anwender effektiv arbeiten kann. Beispiel für einen Parameter: Die Höhe der Berge in einem Gelände-Algorithmus oder die Nummer der Segmente in einer prozeduralen Kugel.
- **Flexibilität:** ein Objekt muss nicht unbedingt im Rahmen der realer Welt liegen. Das heißt, die Parameter können unbegrenzt verändert werden, um eine Menge von Ergebnissen zu produzieren, welche unsere Weltvorstellungen oder gewöhnlichen Physikgesetze widersprechen.

Die präzise Beschreibung für die generierten Objekte ist also möglich und kann durch wenige einfache Parameter ausgedrückt werden. Diese formale Beschreibung wird verwendet, um eine große Anzahl von detaillierten Texturen, Geometrien bzw. andere Objekte zu kreieren. Dieser Effekt ist als Datenverstärkung bekannt und erlaubt es den Entwicklern die ganze Spielwelt zu erschaffen, die sehr einfach über das Netzwerk verteilt werden kann.

Die Flexibilität und Kontrolle der prozeduralen Generierung geben dem Designer eine Plattform für künstlerische und Experimenten Freiheit. Neue visuelle Effekte und Originalobjekte können durch das Experimentieren mit Parameterwerten, die normalen Grenzen überschreiten.

1.3. Aufbau dieser Arbeit

Nach dieser Einführung werden im zweiten Kapitel zunächst verwandte Arbeiten vorgestellt.

In dem dritten Kapitel «Grundlagen» erläutere ich die für meine Arbeit wichtigsten Themen. Ich erkläre die Idee meines Ansatzes. Außerdem erzähle ich, wie man mit Kurven umgeht und was "principal component analysis"(PCA) ist.

Dann kommt das vierte Kapitel «Design», wo ich das Konzept meiner Entwicklung erläutere. Ich beschreibe hier ein formales Modell eines zu generierenden Objektes, ein konkretes Fahrzeug-Modell und ein konkretes Schmetterling-Modell. In diesem Kapitel wird auf die Datenbank der zur Analyse benutzten Objekte eingegangen. Insbesondere wird hier auch gezeigt, wie man mit Hilfe von PCA die wichtigsten Merkmale eines Modells extrahieren kann, um die Dimensionierung des Problems zu reduzieren.

In dem fünften Kapitel «Implementierung» zeige ich, wie ich mein theoretisches Konzept praktisch umgesetzt habe. Es werden hier die Java-Klassen beschrieben und die wichtigsten Merkmale der Implementierung.

Die Ergebnisse finden sich im sechsten, und eine abschließende Zusammenfassung und ein Ausblick auf zukünftige Arbeit in siebtem Kapitel.

2. Verwandte Arbeiten

Die prozeduralen Verfahren sind sehr beliebt in der Welt von Computergrafik. Es gibt zahlreiche Papers, welche die neuen prozeduralen Lösungen für die bekannte Probleme vorschlagen oder die alten Algorithmen verbessern und erweitern. In diesem Kapitel möchte ich einige Arbeiten sowie die Verfahren vorstellen.

2.1. Prozedurale Verfahren

2.1.1. Fractals

Benoît Mandelbrot ist betrachtet als der "Vater von Fraktale". Dieser Begriff hat er in seiner Arbeit [Mandelbrot] eingeführt (aus lateinisch $fractus$ - gebrochen). Die Fraktal Mathe hilft, die natürlichen Formen durch die herkömmlichen geometrischen Formeln zu beschreien.

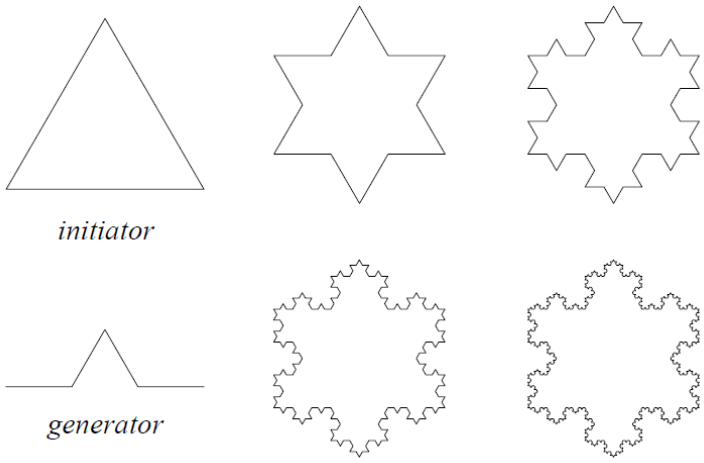


Abbildung 2.1.: Kochsche Schneeflocke. (Entnommen aus: [Prusinkiewicz und Lindenmayer].)

Das Grundkonzept der Fraktale ist ein hohes Maß an Selbstähnlichkeit. Das heißt, dass sie in der Regel kleine Kopien von sich selbst enthalten. Die Fraktale besitzen außerdem unendliches

Detail. Je näher jedes Einzelne betrachtet wird, desto mehr Detail wird es zeigen. Die Kochsche Schneeflocke (Abb. 2.1) ist ein gutes Beispiel der Anwendung von den Fraktalen.

Wie bei jedem prozeduralen Verfahren wird die Form des Fraktales durch einen Algorithmus definiert. In diesem Fall sind diese Algorithmen rekursiv und die nachfolgenden Rekursionen ergeben detaillierte Versionen von der Grundform. Selbstähnlichkeit wird durch die Erzeugung der gleichen Formen oder Muster zu immer kleineren Skalen erreicht. Es gibt keine theoretische Grenze für die Anzahl der Rekursion und damit existiert unendliche Detailebene innerhalb einer Form.

Fraktal-ähnliche Formen wie die Bäume oder Farne können mit Hilfe von sehr einfachen prozeduralen Algorithmen generiert werden. Fraktal-Algorithmen bieten eine große Abstraktion von der strukturellen Komplexität der natürlichen Objekte, die sie darstellen, und sie können die Rekursion verwenden, um verschiedenes Detailniveau anzubieten.

Mit Fraktal kann man aber nur die selbst-ähnlichen Objekte darstellen. Es gibt andere viel flexibleren Algorithmen wie formale Grammatiken, zum Beispiel L-Systeme.

2.1.2. L-Systems

Lindenmayer-Systeme, oder L-Systeme, sind die formale Grammatik, die von dem Biologen A. Lindenmayer als mathematische Theorie für biologische Entwicklung erfunden wurde. L-Systeme wurden ursprünglich entwickelt, um die Replikation von Bakterien und die Wachstum-Muster von einfachen Organismen zu studieren ([Lindenmayer]). Das System und die Anwendungen sind entwickelt und werden nun in der Computergrafik und insbesondere für die Generierung von Fraktalen und realistische Modellierung von Pflanzen benutzt.

Das zentrale Konzept von L-Systemen ist “rewriting ”. Das ist eine Technik zur Definition von komplexen Objekten durch Ersetzen von Teilen eines einfachen Ausgangsobjekts mit einer Reihe von Ersetzungsregeln.

Die Komponente eines L-Systems sind:

- **V** (*das Alphabet*) ist eine Menge von Symbolen, welche die Elemente zum Ersetzen beinhaltet (*variables*)
- **S** ist eine Menge von Symbolen, welche die Elemente beinhaltet, die fest bleiben (*constants*).
- ω (*start, axiom* oder *initiator*) ist eine Kette von Symbolen und Konstanten, welche den Anfangszustand des Systems definieren.

- **P** ist eine Menge von Regeln(oder *productions*), welche es definieren, wie die Variablen durch Kombinationen von Konstanten und anderen Variablen ersetzt werden können. Besteht aus zwei Ketten – *predecessor* und *successor*.

Ein Anfangszustand, ω, ist angegeben. Danach wird er durch eine Reihe von Regeln P umgeschrieben. Die werden iterativ angewendet, so dass komplexere Objekte mit einem einfachen Satz von Regeln definiert werden kann.

Die L-Systeme wurden entwickelt, um die Pflanzen und anderen natürlichen Strukturen zu definieren und zu visualisieren. Die bedeutendsten Fortschritte wurden gemacht und heutzutage benutzt man die L-Systeme für die Generierung von vielfältigen Landschaften mit detaillierter Flora. Die L-Systeme sind ein gutes Beispiel von prozeduraler Generierung für eine Reihe von Gründen. Mit deren Hilfe kann man komplexere Modelle und organische Strukturen nur mit einer knappen Reihe von Regeln definieren, modellieren und visualisieren. Eine unterschiedliche Komplexität kann mit den Parameter, wie z. B. die Rekursionstiefe, unterstützt werden. Die Algorithmen können in einer kompakten und intuitiven Art und Weise definiert werden als auch die rekursive Struktur von vielen Naturphänomenen abstrahieren. L-Systeme können sehr einfach durch externe Parameter eingestellt werden und sind den anderen formalen Grammatiken ähnlich erweiterbar.

2.1.3. Perlin Nosie

Perlin Noise wurde ursprünglich entwickelt, um die "natürlich aussehenden"Texturen schaffen zu helfen. Die Technik wurde von Ken Perlin zur Verwendung in dem Film “Tron ”in 1982 entwickelt ([Perlin]). Um Rausch zu erschaffen, wird zuerst von einer pseudo-zufälligen Funktion eine Reihe von Werten generiert, welche danach in einen kohärenten Rausch interpoliert werden. Mehrere Schichten werden danach zusammengesetzt, um eine “natürlich ”aussehende Textur mit Fraktal ähnlichen Details zu erstellen.

Rauschfunktion

Eine Rauschfunktion generiert Zufallsdaten. Also jedes Mal, wenn die Funktion aufgerufen wird, wird eine neue Zahl zurückgeliefert. Dies ist nützlich, erlaubt jedoch nicht die Kontrolle über die Ergebnisse. Um eine parametrisierte Kontrolle über Rauschgenerator zu erhalten, wird eine Zufallsfunktion mit einem “Seed ”verwendet.

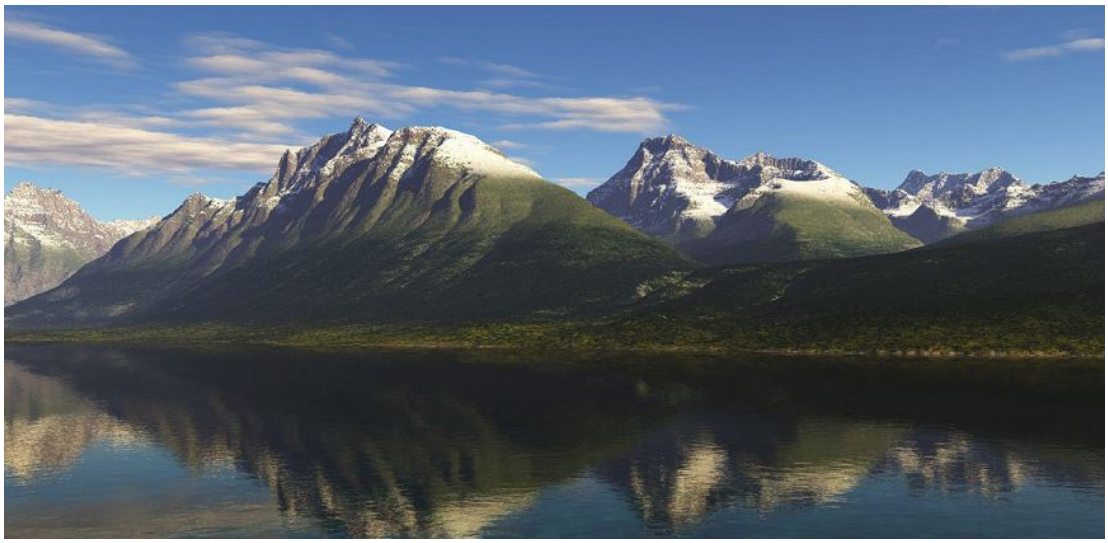


Abbildung 2.2.: Mit "Terragen"prozedural erstellte Textur. © 2003 M. GIULI Terragen Artist.

Interpolation

Interpolation ist ein Prozess der Kurvenanpassung, in dem eine Funktion aufgebaut wird, die genau die Datenpunkte schneidet. Diese Funktion kann die neuen Datenpunkte erzeugen, indem die Eingangspunkte durch die Rauschfunktion erzeugt werden. Für eine endliche Menge von Data-punkten kann eine Funktion generiert werden, die ein unendliches Angebot von Punkten liefert. Mehrere verschiedene Algorithmen stehen zur Verfügung, um diese Interpolation durchzuführen. Die Algorithmen unterscheiden sich in der Anzahl der Datenpunkte, die sie als Eingabe nehmen, der Genauigkeit, die sie bieten, ihrer Rechenkomplexität und der Glätte der Kurve, die sie erzeugen.

Turbulence

Ergebnisse von interpoliertem Rausch haben zufällige Eigenschaften, scheinen aber ziemlich künstlich und nicht natürlich. In der Natur gibt es viele verschiedene Details. Um eine bessere Textur-Quelle zu liefern, die der Natur sich ähnelt, wird mehr Turbulenz bei Kombinieren von mehreren Rauschtexturen angewendet. Die Schichten werden mit unterschiedlichen Amplituden und Frequenzen kombiniert.

Als Prozedurale Verfahren, Perlin Noise bietet eine umfassende Reihe von Vorteilen. Parametrisierte Kontrolle erlaubt es den Entwicklern mit Hilfe von “high-level ”Parameter die Ausgabe zu kontrollieren. Die mit dem Algorithmus erstellte reproduzierbare Geometrie und Textu-

ren haben die minimalen Speicheranforderungen, können sehr effizient erstellt werden und können durch ein Paar einfache Parameter definiert werden. Perlin Noise ist eins der von den nützlichsten prozeduralen Verfahren und ist sehr nützlich in Computer-Grafik-Anwendungen.

2.1.4. Voronoi Texture Basis

Voronoi Diagramme wurden von [Worley] als eine Methode der prozeduralen Generierung präsentiert. Es war ein Algorithmus, der den Raum in einer zufälligen Array von Zellen aufteilt, was die Zellen-ähnliche Texturen darstellt. Die Voronoi Diagrammen werden in wissenschaftlichen Anwendungen benutzt, wie z. B. räumliche Analyse, Stadt-Siedlung Analyse, Robotik und Ökologie.

Die Voronoi Diagramm ist die Zerlegung eines metrischen Raums in eine bestimmte diskrete Menge von Objekten in diesem Raum. Die Abb. 2.3 zeigt ein Gebiet, das mit Hilfe von Linien in die Zellen aufgeteilt ist, welche mit Hilfe von Punkten auf der Karte gekennzeichnet sind. Jede Grenzlinie ist zwischen zwei benachbarten Punkten positioniert. Mehrere Zellstrukturen können mit unterschiedlichen Konfigurationen der Punkten-Setzung erstellt werden.

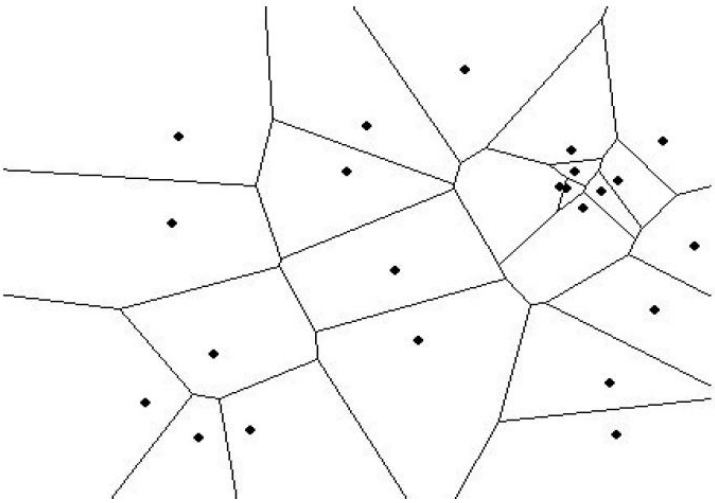


Abbildung 2.3.: Voronoi Diagram.

Mit diesem Algorithmus kann man eine große Abstraktion für die Generierung von zellulären Oberflächen erzielen, und zwar mit kleinem Set von Parameter. Die besten Beispiele dafür sind natürliche Oberflächen wie Papier, Haut, Stein, Baumrinde oder Schlamm.

2.2. Morphable Model

2.2.1. Synthese von 3D-Gesichtern

Bei meiner Arbeit habe ich mehr Rücksicht auf das Paper von [Blanz u. Vetter] genommen, welches sich mit Synthese von 3D-Gesichter beschäftigt. Ich verwende ein ähnliches Verfahren. In ihrem Paper wird ein Prinzip erläutert, wie man eine Reihe von typischen Elementen (in diesem Fall – Gesichter) kombinieren kann, um ein neues Element zu generieren.

Das Gesichtsmodell von [Blanz u. Vetter] basiert sich auf Daten Set von 3D-Gesichter (die Laser scnans von 200 menschlichen Köpfen, repräsentiert als 70000 Eckpunkte und genau so viel Farbwerten). Die Geometrie von einem Gesicht wird durch einen Form-Vektor S = $(X_1, Y_1, Z_1, X_2, \dots, Y_n, Z_n)^T$ aus R^{3n} repräsentiert, welcher die X, Y, Z-Koordinaten von seinen n Eckpunkten enthält. Genauso werden die Farbwerte dargestellt (die Farbwerte sind für meine Arbeit nicht relevant, deswegen werde ich die weiter nicht erwähnen). Ein morphable Gesichtsmodell hat man aus einem Set von m Beispielgesichter konstruiert, repräsentiert durch Formvektor S_i. Die neuen Formen S_{mod} kann man in baryzentrischen Koordinaten ([Laval]) als eine lineare Kombination der Formen von m Beispielgesichter ausdrücken:

$$S_{mod} = \sum_{i=1}^{m} a_i S_i,$$

wobei

$$\sum_{i=1}^{m} a_i = 1;$$

Ein morphable Modell ist als Set von Gesichter ($S_{mod}(\vec{a})$) definiert, welche durch Koeffizienten $\vec{a} = (a_1, a_2, \dots, a_m)^T$ parametrisiert sind. Die neuen Gesichter kann man dementsprechend durch Variieren von Parameter a generieren. Dasselbe Verfahren verwende ich für die Erstellung der neuen Fahrzeugen bei meinem Ansatz. Ein Set von Beispielfahrzeugen wird analysiert, um ein Durchschnittsauto und die sinnvolle Parameterwerte für $\vec{a}$ zu finden.

2.2.2. Ein Morphable Model für Fische

Die Arbeit von [Maier] ist ein weiteres Beispiel für ein morphable Modell. Der Autor stellt ein Modell für Fische dar. In Rahmen dieser Arbeit hat der Autor ein eigenes Modell für Fische vorgestellt. Dafür hat er erstmal die verschiedenste Bilder von Fische analysiert und die Korrespondenzen zwischen diesen gefunden. Dafür wurden eigene Methoden zur Korrespondenzbestimmung entwickelt und zusammen mit dem zur Korrespondenzfindung häufig verwendeten optischen Fluss evaluiert. Als Ergebnis wurde eine Kombination aus manuell

vorgegebenen Korrespondenzen, sowie daraus gewonnenen Konturkorrespondenzen, welche mittels Interpolation zu einem dichten Korrespondenzfeld ergänzt wurden, entwickelt. Danach hat der Autor ein Algorithmus vorgestellt, welcher die Feature-Korrespondenzen automatisch approximiert. Um die Fische, deren Korrespondenz bekannt ist, zu annähern, wurde die Eigenraumprojektion mittels PCA verwendet. Hieraus ergab sich, dass bereits eine kleine Anzahl von Fischen geeignet ist, weitere Fische ausreichend zu approximieren, was die Möglichkeit eines generativen Modelles für viele Fische auf Basis einer begrenzten Anzahl von Fischen nahelegt. Zum Ende hat der Autor weitere Anwendungen zum Morphing und Übertragen von Forminformationen und Merkmalen eines Fisches auf andere Fische vorgestellt.

Für das Aufbauen des Modells hat der Autor das Bild eines Fisches durch einen Formvektor F und einen Texturvektor T wie folgt beschrieben:

$$F_i = (x_{0,0}, y_{0,0}, x_{1,0}, y_{1,0}, \ldots, x_{n-1,m-1}, y_{n-1,m-1})^t$$

F_i gibt für jedes Pixel des Referenzbildes an, um welchen Wert in X-/Y-Richtung verschoben das korrespondierende Pixel im i-ten Eingabebild zu finden ist.

$$T_i = (g_{0,0}, g_{1,0}, \ldots, g_{n-1,m-1})^t$$

T_i speichert den Grauwert für jedes Pixel. T_i enthält also die Grauwerte des mittels F_i auf das Referenzbild gewarpten i-ten Eingabebildes. Diese Modell (Form- und Texturdaten) werden einer Eigenwertzerlegung unterzogen. Je größer sind die Eigenwerte, desto stärker wirken die Änderungen an den Koeffizienten auf Form und Textur des Fisches. Dieses Wissen wird dafür verwendet, um das Modell weiterhin automatisch aufzubauen.

3. Grundlagen

In diesem Kapitel werde ich die für meine Arbeit wichtigsten Themen erläutern. Ich erkläre die Idee meines Ansatzes. Außerdem erzähle ich, wie man mit Kurven umgeht und was "principal component analysis"(PCA) ist.

3.1. Idee

Die in Kapitel 2 eingeführte prozedurale Verfahren sind sehr mächtige Mechanismen und lassen viele Probleme Prozedural lösen. Trotzdem sind die nicht für alle Fälle geeignet. Für die Spezialfälle (was bei Generierung von Silhouetten der Fall ist) sollte man neue Ansätze entwickeln.

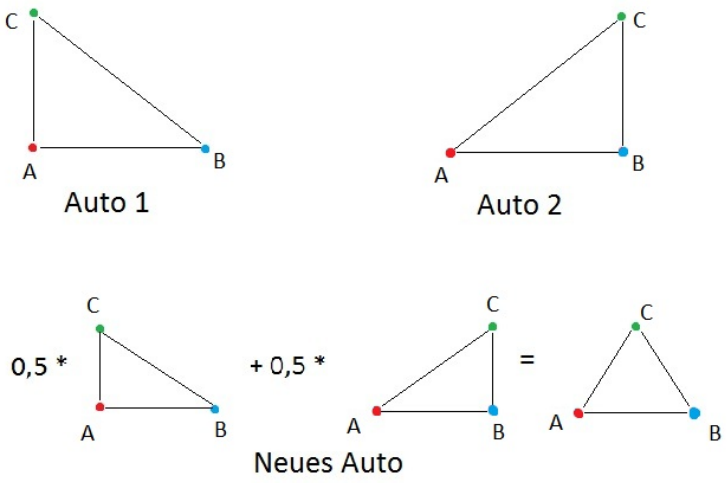


Abbildung 3.1.: Das Erzeugen neues Autos durch lineare Kombination

Wie schon erwähnt wurde, basiert die Grundidee meines Verfahrens auf der Arbeit von [Blanz u. Vetter]. Zuerst wird ein Set von typischen Repräsentanten einer Domäne analysiert. Die neuen Objekte kann man modellieren, indem man lineare Kombinationen aus diesen

Prototypen bildet. Die Idee ist relativ einfach: Angenommen, wir haben zwei typische Objekte, der Einfachheit halber bestehen die aus nur drei Punkten – A, B und C. Die Punkte A und B von den beiden Objekten sind identisch und der Punkt C hat jeweils verschiedene Koordinaten. Um ein neues Objekt zu kriegen, bilden wir die Summe von diesen zwei Objekten, multipliziert mit einem Skalar-Koeffizient. Das Ergebnis ist ein neues Objekt mit den gleichen A und B und einem neuen C (Abb. 3.1).

3.2. Bézier-Kurven

Bézier-Kurven wurden in 60er Jahren des XX Jahrhunderts unabhängig von einander von Pierre Bézier und Paul de Faget de Casteljau erarbeitet und wurden für das Design von Autos verwendet.

Die Bézier-Kurve, welche ich in meiner Realisierung verwende, ist ein Spezialfall der Bernsteinpolynome [Bernstein]. Das sind die komplexe parametrisierte Kurven, welche den Vorteil haben, mit einer sehr geringen Datenmenge komplizierte Kurven beschrieben werden können. Man betrachtet so nicht jeden einzelnen Punkt der Kurve, sondern nur die vorgegebenen Bézier-Punkte im Zusammenspiel mit der Berechnungsvorschrift, die sich aus der Benutzung der Bernsteinpolynome ergibt [Holldack].

Eine Bezie-Kurve n-ten Grades zu gegebenen $n + 1$ Kontroll- oder Bézier-Punkten b_0, b_1,…,b_n, die das so genannte Kontrollpolygon bilden, ist für λ aus [0, 1] definiert als

$$x(\lambda) = \sum_{i=0}^{n} b_i B_i^n(\lambda)$$

wobei

$$B_i^n(\lambda) = \binom{n}{i}\lambda^i(1-\lambda)^{n-1}$$

Das Tupel (b_0,b_1, …, b_n) heißt Bézier-Polygon, und die Bézier-Kurve kann nur innerhalb von diesem Polygon verlaufen.

Für die Randpunkte $x(0)$ und $x(1)$ einer Bézier-Kurve, die durch die Bézier-Punkte b_0, . . . , b_n gegeben ist, gilt:

$$x(0) = b_0;$$

$$x(1) = b_n;$$

Die Tangenten an den Randpunkten sind nur durch die benachbarten Bézier-Punkte bestimmt.

$$x'(0) = n(b_1 - b_0);$$

$$x'(1) = n(b_n - b_{n-1});$$

Wenn man diesen Satz betrachtet, ist es zu sehen, dass die Bézier-Kurve im ersten Punkt b_0 des Polygons beginnt und im letzten Punkt b_n endet. Das bedeutet insbesondere, dass durch die Aneinanderreihung mehrerer Bézier-Polygone stetige und sogar glatte (differenzierbare) Kurven dargestellt werden können. Zur Veranschaulichung (Abb. 3.2) betrachten wir zwei Bézier-Kurven, die durch die beiden Bézier-Polygone (b_0, b_1, b_2) und (b_2, b_3, b_4) gegeben sind. Für einen differenzierbaren Übergang der beiden Kurvenstücke müssen die beiden Nachbarpunkte und der gemeinsame Bézier-Punkt auf einer Gerade liegen.

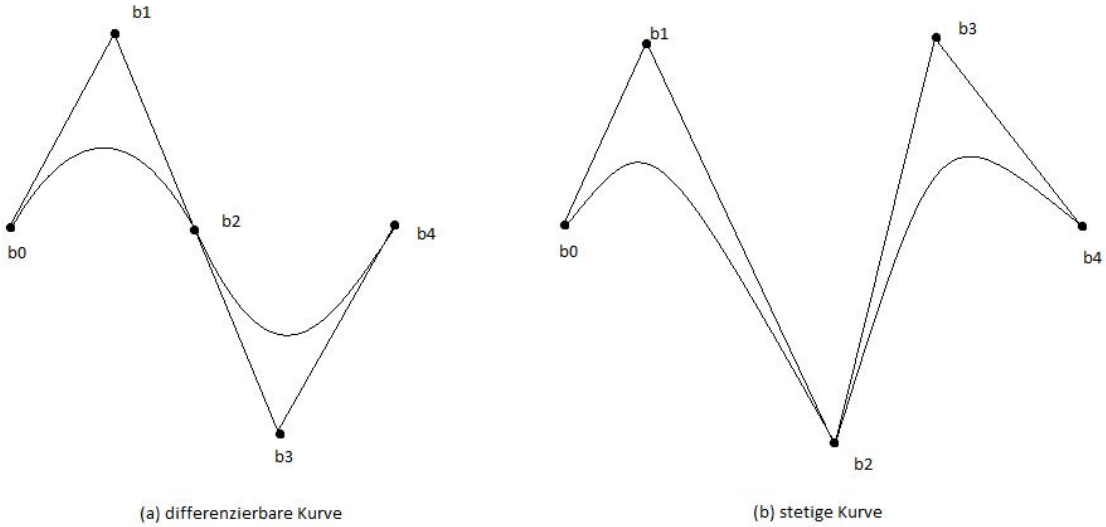


Abbildung 3.2.: Darstellung komplizierterer Kurven durch Aneinanderreihung der Bézierpolygone

Die Punkte einer Bézier-Kurve können aus gegebenen Bézier-Punkten b_i für i = 0..n schrittweise berechnet werden. Um das geometrisch vorzustellen, muss folgender Algorithmus verwendet werden [Holldack]:

- **Schritt 1** Zeichne das n-eckige Bézier-Polygon ($b_0, \ldots, b_n$). Dies entspricht den Konvexkombinationen (1 - λ) * b_i + λ * b_i für i = 0..(n - 1).
- **Schritt 2** Teile nun die Seiten des Bézier-Polygons in einem gewählten Verhältnis λ und zeichne nun Strecken zwischen jeweils zwei benachbarten "Mittelpunkten", sodass ein (n-1)-Eck entsteht. Dies entspricht der Konvexkombination der in Schritt 1 entstandenen Hilfspunkte mit je zwei linearen Polynomen als Koeffizienten.
- **Schritt k=2..n** Wiederhole den vorherigen Schritt mit dem neu entstandenen Polygon.
- **Schritt n+1** Es ist nur noch eine Strecke zwischen zwei Punkten übrig. Teile diese wieder in dem anfangs gewählten Verhältnis λ, um den Punkt x(λ) der Bézier-Kurve zu erhalten. x(λ) ist somit die Konvexkombination der letzten entstandenen Hilfspunkte mit Polynomen vom Grad n - 1.

Für den Fall n=3 und λ = 1/2 ergeben sich somit die folgenden Zeichnungen (Abb. 3.3).

3.3. Principal Component Analysis

Principal Component Analysis (PCA) ist schon seit längere Zeit als mathematisches Verfahren bekannt. Die verschiedenste Papers beschreiben sowohl das Prinzip von PCA selbst ([Jolliffe]) als auch die einzelne Probleme, welche mit PCA gelöst werden können ([Aguilera], [Aitchison], [Ali et al.]). In diesen Papers werden die Grundlagen von Hauptkomponentenanalyse vorgestellt, was kritisch für das Verständnis ist, wie und für welche Zwecke man PCA überhaupt verwenden kann.

Die zentrale Idee von PCA, oder auch Hauptkomponentenanalyse, ist es, die Dimension eines Datensatzes zu reduzieren, welcher aus zusammenhängender Variablen besteht [Jolliffe]. Dabei sollte man so viel wie möglich in dem Datensatz vorgestellter Variationen behalten. Das erreicht man durch die Transformation in einen neuen Satz der Variablen, Hauptkomponenten, welche nicht korreliert und so geordnet sind, dass die erste $wenige$ davon die $meiste$ Variation von originalen Variablen darstellen. Die PCA verwendet man meistens für die statistische Analyse von Daten als eine Methode für die Dimensionsreduktion und für das Auffinden von Mustern.

Angenommen, haben wir einen m Vektoren $V_i = (v_0, \ldots v_{n-1})^t$ der Dimension n. Die Hauptkomponente lassen sich aus Kovarianzmatrix ableiten [Aitchison], welcher wie folgt berechnet wird:

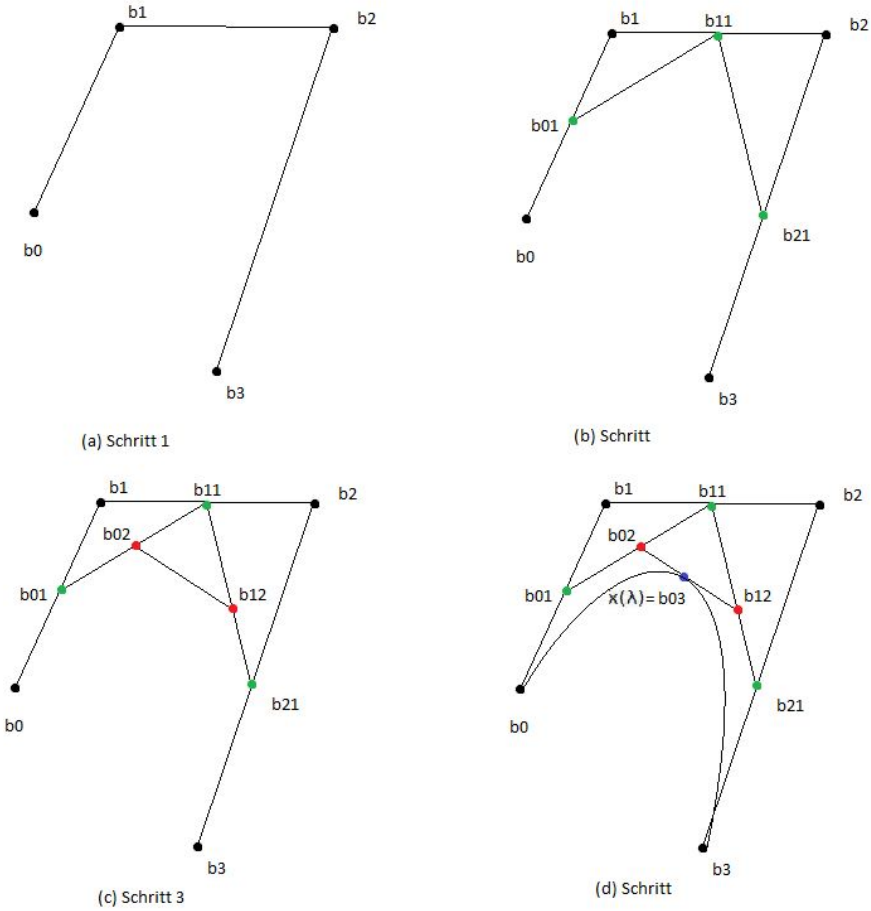


Abbildung 3.3.: Schritte des De-Casteljau-Algorithmus

$$C = \frac{1}{m} \sum_{i=1}^{m} (v_i - \bar{v}) \cdot (v_i - \bar{v})^t.$$

wobei $\bar{v}$ den Schwerpunkt der Daten darstellt und wie folgt berechnet wird:

$$\bar{v} = \frac{1}{|V|} \sum_{i=1}^{m} v_i.$$

Die Kovarianzmatrix repräsentiert die Verteilung der Daten in neuem Raum ($Eigenraum$) bezüglich des ursprünglichen Raums ($EuklidischerRaum$). Diese ist eine mehrdimensionale Normalverteilung [Mardia et al.]. Um die neue Hauptachse in neuem Eigenraum zu finden,

bestimmt man die Eigenvektoren der Kovarianzmatrix. Für die Eigenvektoren $\bar{e_i}$ und die korrespondierenden Eigenwerte λ_i einer Matrix A gilt

$$A\bar{e_i} = \lambda_i \bar{e_i}$$

Die Eigenwerte und Eigenvektoren lassen sich über eine Singulärwertzerlegung von A berechnen [Gene u. Kahan], falls A symmetrisch und positiv-semidefinit ist (was bei der Kovarianzmatrix der Fall ist). Jede $M \times N$-Matrix kann man in eine $M \times M$-Matrix U, eine $N \times M$-Matrix W und eine $N \times N$-Matrix V derart zerlegen, dass gilt:

$$A = UWV^t$$

wobei W eine Matrix mit nicht negativen Werten auf der Diagonalen und 0 sonst ist. Die Singulärwerte sind dann die Eigenwerte und die Zeilen in V die Eigenvektoren. Führt man dies für $n \times n$-Matrix C, kann man die Eigenvektoren e_i und die zugehörigen Eigenwerte λ_i berechnen. Die Eigenvektoren e_i haben Dimension m, die Eigenwerte sind Skalare.

Von hier an nehmen wir an, dass die Eigenwerte und damit die zugehörigen Eigenvektoren der Grö"ße nach absteigend sortiert sind. Der größte Eigenwert und damit sein zugehöriger Eigenvektor werden als λ_0 und e_0 bezeichnet. Diese repräsentieren die wichtigste Komponente im Eigenraum.

Definiert man eine Matrix B mit Eigenvektoren in absteigender Reihenfolge als Spalten, so kann man ein Punkt aus dem Ursprungsraum durch eine Rotation um den Schwerpunkt mit B^t in den Eigenraum überführen

$$x_{Eigen} = B^t(x_{Euklid} - \bar{v}) \tag{3.1}$$

Analog ergibt sich die Rücktransformation vom Eigenraum in den Euklidischen Raum als

$$x_{Euklid} = B \cdot x_{Eigen} + \bar{v} \tag{3.2}$$

Bei B handelt es sich um eine $n \times n$-Matrix. Dabei werden alle Eigenvektoren benutzt. Mit Hilfe der Gewichtung durch die Eigenwerte lässt sich eine Dimensionsreduktion erreichen. Wählt man B_o mit $o < n$, sieht die Hintransformation wie folgt aus:

$$x_{pca} = B_o^t(x_{Euklid} - \bar{v})$$

Der neue Datensatz besteht jetzt in dem Eigenraum nur aus o wichtigsten Hauptkomponenten. Die übrige Information geht verloren. Die Rücktransformation sieht wie folgt aus:

$$x = (B_o x_{pca}) + \bar{v}$$

Die Matrix B_o ist jetzt eine $n \times o$-Matrix, welche die bestmögliche o-dimensionale Basis für die gegebenen Daten darstellt. In so einem Weg rekonstruierter Datensatz fehlen die Daten mit kleineren Eigenwerten ($x \neq x_{Euklid}$). Die Standardabweichung der Datenpunkte entlang dieser Achsen fällt somit zunehmend geringer aus. Abb. 3.4 veranschaulicht das Prinzip der Dimensionsreduktion an einem schlichten zweidimensionalen Beispiel.

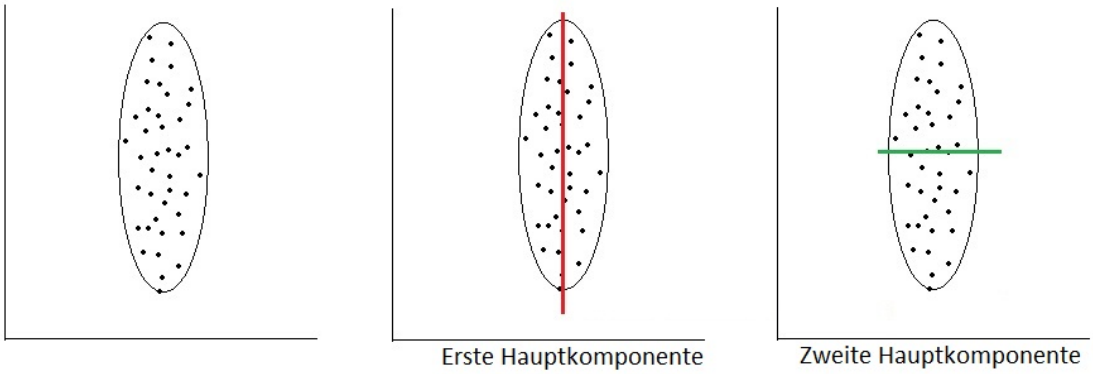


Abbildung 3.4.: Links: Im ursprünglichen zweidimensionalen Raum verteilte Daten. Mitte: Die erste Hauptkomponente. Projiziert man zur Dimensionsreduktion sämtliche Datenpunkte der Verteilung auf diesen Eigenvektor und macht diese Projektion rückgängig, liegen sämtliche Datenpunkte auf der roten Geraden. Rechts: zweite Hauptkomponente. Projiziert man zur Dimensionsreduktion sämtliche Datenpunkte der Verteilung auf diesen Eigenvektor und macht diese Projektion rückgängig, liegen sämtliche Datenpunkte auf der grünen Geraden

4. Design

In diesem Kapitel geht es um das Konzept meines Verfahrens. Es wird hier ein formales Modell eines zu generierendes Objektes beschrieben, sowie ein konkretes Modell des Fahrzeugs und ein konkretes Modell des Schmetterlings. Insbesondere wird hier auch gezeigt, wie man mit Hilfe von PCA die wichtigsten Merkmale eines Modells extrahieren kann, um die Dimensionierung des Problems zu reduzieren.

4.1. Formales Modell

In meiner Realisierung arbeite ich mit den Objekten, die durch Bézier-Kurven gegeben sind. Die Kurven werden jeweils durch $n + 1$ Kontrollpunkte repräsentiert (in meiner Realisierung meist drei pro Bézier-Kurve). Der Grad der Kurve ist also n. Jede Bezierkurve hat die Form:

$$p(t) = \sum_{i=0}^{n} p_i B_i^n(t).$$

Es werden k Bézier-Kurven zusammengefügt. Damit ergibt sich ein Vektor x von Kontrollpunkten. Der Vektor beinhaltet jeweils 3D-Vektoren (die Kontrollpunkte) als Koordinaten. Der Vektor hat die Dimension $m = k \cdot (n + 1)$. Ein solcher Vektor definiert eindeutig ein Objekt.

Meine Implementierung (Kapitel 5) erlaubt es so einen Vektor grafisch als zweidimensionale Silhouette darzustellen. Es ist auch möglich ein 2D-Modell als Vektor zu speichern. So einen R^{28} Vektor enthält die Kontrollpunkte aller Bézier-Kurven, welche das Objekt bestimmen. Die Kontrollpunkte sind in einer bestimmte Reihenfolge sortiert. Kennt man die Reihenfolge, kann man die Kurven wiederherstellen, um ein Objekt anzuzeigen.

4.2. Analyse und Synthese

Es werden nun Wege gesucht, neue Objekte zu synthetisieren, also jetzt müssen neue (sinnvolle) Vektoren von Kontrollpunkten gefunden werden. Der Einfachheit halber trennen wir dazu die Vektoren für die drei Achsen x, y und z auf. Es ergeben sich also drei Vektoren v^x, v^y und v^z jeweils der Dimension m (in meiner Realisierung ist die Dimension von jedem Vektor m = 28).

Im Folgenden wird der Bezeichner v als Repräsentant für einen beliebigen der drei Teilvektoren v^x, v^y und v^z verwendet. Dabei handelt es sich also um einen m-dimensionalen Vektor mit Skalaren als Komponenten.

4.2.1. PCA

Ein Mittel, wichtige Hauptrichtungen aus einer Menge von Vektoren zu extrahieren, ist PCA. Um damit zu arbeiten, brauche ich eine Menge von typischen Objekten der zu analysierenden Domäne (4.5). Es steht also eine Liste $V = v_i$ von Vektoren zur Verfügung, die jeweils ein Objekt repräsentieren (genau genommen repräsentieren sie jeweils eine Komponente der Kontrollpunkte der verwendeten Bézier-Kurven).

Zunächst wird für die Vektoren den Schwerpunkt $\bar{v}$ berechnet und die Kovarianzmatrix C bestimmt (Kapitel 3.3). Bei C handelt es sich also um eine $m \times m$-Matrix. Für diese Matrix sind weiterhin die Eigenvektoren e_i und die zugehörigen Eigenwerte λ_i zu berechnen. Die Eigenvektoren e_i haben eine Dimension m, die Eigenwerte sind Skalare.

Von hier an nehmen wir an, dass die Eigenwerte und damit die zugehörigen Eigenvektoren der Größe nach absteigend sortiert sind. Der größte Eigenwert und damit sein zugehöriger Eigenvektor werden als λ_0 und e_0 bezeichnet. Diese repräsentieren die wichtigste Komponente im Eigenraum zur Generierung von Fahrzeugen.

4.2.2. Transformation

Wie im Kapitel 3.3 beschrieben ist, ergeben sich nun zwei Räume. Der Raum, in dem die Vektoren v zur Repräsentation der Objekte existieren, ist der Euklidische Raum. Durch die Eigenvektoren ergibt sich ein zweiter Raum, der Eigenraum. Die Eigenvektoren können verwendet werden, um einen Vektor vom Euklidischen Raum in den Eigenraum und zurück zu transformieren (Gleichungen 3.1 und 3.2)

Was ist damit erreicht? Man kann die Eigenkoordinaten a_i eines Objektes berechnen, indem man den Euklidischen Objektvektor in den Eigenraum transformiert (für je ein Objekt ist also $a_i = v^{Euklid}$). Diese Eigenkoordinaten lassen sich als Gewichtungen der entsprechenden Eigenvektoren interpretieren, um das Objekt im Eigenraum zu repräsentieren. Insbesondere ist es möglich alle manuell erstellten Objekte in den Eigenraum zu transformieren und beobachten, welche Eigenkoordinaten typisch sind. Hat man l Fahrzeuge, so erhält man beispielsweise l Werte für a_0 und damit eine Verteilung der Gewichtungen für den ersten Eigenwert. Beispielsweise kann man den Mittelwert der verschiedenen a_0 verwenden, um den *typischen* Gewichtungsfaktor für den ersten Eigenwert zu bestimmen (gegeben unsere Testobjekte).

Macht man das für alle Eigenkoordinaten a_i so kann man im Eigenraum das *Durchschnittsobjekt* generieren und dann wieder in den Euklidischen Raum zurück transformieren (z.B. um es anzuzeigen). Für eine gegebene Belegung der Eigenkoordinaten a_i ergibt sich das Objekt im Eigenraum natürlich wieder als

$$v^{Eigen} = \sum_{i=1}^{m} a_i e_i \tag{4.1}$$

Außerdem lässt sich ein Objekt, das im Eigenraum repräsentiert ist, zurück in den Euklidischen Raum transformieren.

4.2.3. Dimensionsreduzierung

Mit Hilfe der Gewichtung durch die Eigenwerte lässt sich eine Dimensionsreduktion erreichen. So kann man die Repräsentation eines Objektes im Eigenraum auf $o < m$ reduzieren. Dazu verwendet man natürlich die o größten Eigenwerte und deren Eigenvektoren. Die Matrix B wird damit zur Matrix B^o, also einer $m \times o$-Matrix. Es werden zur Erstellung von B^o nur die ersten o Eigenvektoren verwendet und als Spalten in die Matrix geschrieben. Entsprechend ergeben sich nach der Transformation in den Eigenraum auch nur noch o Eigenkoordinaten: $v^{Eigen} \in R^o$. Bei der Rücktransformation in den Euklidischen Raum kommen die fehlenden Koordinaten aber wieder dazu und es ergeben sich wieder m-dimensionale Vektoren v^{Euklid}.

Will man nun Objekte bei reduzierter Dimension im Eigenraum erzeugen, dann verwendet man entsprechend auch nur die Eigenkoordinaten der verwendeten Eigenvektoren. Gleichung 4.1 reduziert sich also zu:

$$v^{Eigen} = \sum_{i=1}^{o} a_i e_i$$

4.2.4. Synthese neuer Objekte

Ein neues Objekt wird also synthetisiert, indem man im Eigenraum geeignete Eigenkoeffizienten a_i findet, die dann zusammen einen Vektor a bilden. Dieser Vektor hat im Normalfall die Dimension m. Verwendet man aber die Dimensionsreduktion aus einem Abschnitt 4.2.3, hat er zunächst nur die Dimension $o < m$. Die fehlenden $m - o$ Koordinaten werden dann einfach mit Nullen aufgefüllt. Den zugehörigen Fahrzeugvektor berechnet man wieder nach Gleichung 3.2.

4.3. Fahrzeug-Modell

Als Basis für meine Realisierung verwende ich das Kraftfahrzeug-Modell. In diesem Abschnitt werde ich beschreiben, wie dieses Modell aufgebaut wird.

4.3.1. Box styling

Laut der Internetseite www.drivespark.com gibt es die folgenden Karosseriearten: Sedan, Hatchback, SUV(Sports Utility Vehicle), MPV(Multi-Purpose Vehicles), Kombi, Coupe, Kabrio ud Pickup. Wenn man alle diese Fahrzeuge von der Seite anschaut, kann man die in drei Blöcke teilen – Motorraum, Fahrgastraum und Kofferraum. Diese Verteilung nennt man "Box styling", und man unterscheidet drei verschiedene Designs.

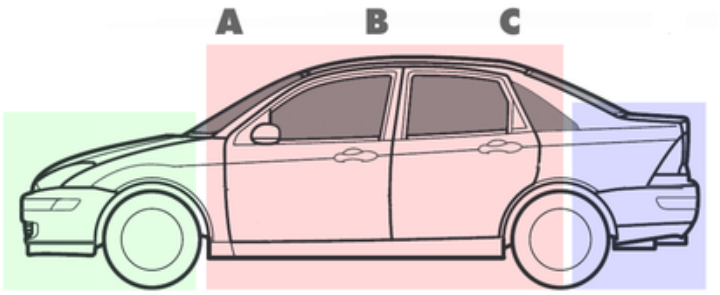


Abbildung 4.1.: Typische Fahrzeugsäule-Konfiguration von Sedan(three-box). [Wikipedia]

One-Box Design (Monospace or Monovolume)

Dieses Design erweicht alle Unterschiede zwischen separaten und umschließt den gesamten Innenraum eines Fahrzeugs in eine einzige Form.

Two-box Design

Dieses Design betont den Motorraum und setzt den Fahrgast- und Kofferraum zusammen.

Three-box Design

Dies ist ein Design, bei dem alle drei Räume(Motor-, Fahrgast- und Kofferraum) sehr auffällig sind und man deswegen sie ganz genau unterscheiden kann.

Das three-box Design betrachte ich als Grunddesign für meine Realisierung (Abb. 4.1), da man mit diesen drei Räumen fast alle Fahrzeuge beschreiben kann(Jedes Fahrzeug hat Motor, Fahrgastraum und Kofferraum).

4.3.2. Parameter eines Fahrzeugs

Hier möchte ich die durchschnittlichen Zahlen vorstellen, welche die Abmessungen eines Fahrzeugs beschreiben. Diese Zahlen hängen von der Karosserieart ab. Die geometrischen Kenndaten, die mich interessieren, sind: Länge, Radstand, Höhe und Breite.

Typ	Länge	Randstand	Höhe	Breite
Sedan	4,77	2,85	1,75	1,91
Hatchback	4,47	2,66	1,52	1,79
SUV	4,85	2,85	1,5	1,85
MPV	4,78	2,85	1,73	1,88
Kombi	4,47	2,66	1,52	1,79
Coupe	4,07	2,51	1,53	1,73
Kabrio	4,68	2,76	1,38	1,84
Pickup	5,15	3,23	2,06	1,93

Abbildung 4.2.: Geometrische Kenndaten des Fahrzeugs. [Schuster et al.]

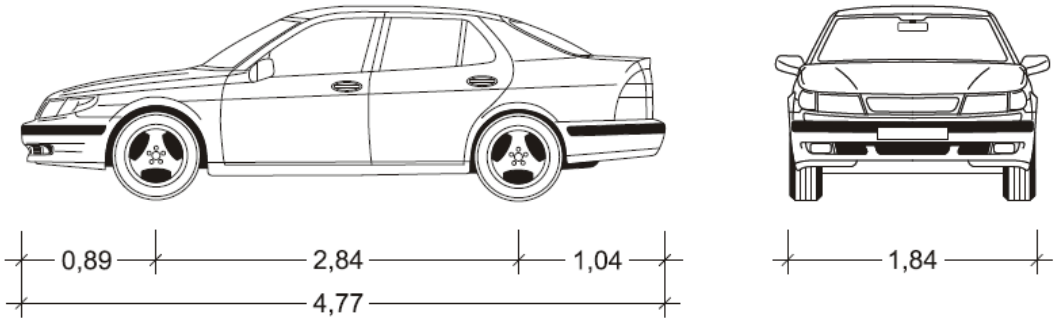


Abbildung 4.3.: Geometrische Kenndaten des Fahrzeugs.

Wenn man diese Zahlen betrachtet, sieht man, dass der Randstand(Kernteil des Modells) von allen Fahrzeugtypen fast gleich ist. Nur der Randstand von Pickup ist größer, als bei anderen, und der von Coupe - kleiner. Deswegen klassifiziere ich den Rest nach der Türanzahl:

- *Groß*: Pickup
- *Normal*: Sedan, Hatchback, SUV, MPV, Kombi
- *Klein*: Coupe, Kabrio

4.3.3. Konzept

Wie ich schon erwähnt habe, betrachte ich mein Modell des Fahrzeugs als three-box Design. Es gibt also drei Hauptteile: der Motorraum, der Gastraum und der Kofferraum. Ich füge noch ein Chassis hinzu, welches als Fundament für das ganze Fahrzeug dient.

So sieht das primitivste Modell eines Fahrzeugs aus. Alle Blöcke (Motor, Gast, Koffer und Chassis) sind Rechtecke. Das betrachte ich als Skelett eines Fahrzeugs. Ein rechteckiger Block dient als ein "Behälter"für den tatsächlichen Umriss eines Fahrzeug-Bestandteiles. Die Rechtecke eignen sich am besten, weil ich mit deren Hilfe sehr einfach die wichtigsten Daten speichern kann: die Höhe und die Länge eines Blockes, die Position im Raum (Zentrum eines Blocks) und Verhältnis zu anderen Blöcke.

Wie bereits erklärt, betrachte (darstelle) ich ein Fahrzeug als Modell, das aus vier Blöcken besteht – Chassis, Motorraum, Fahrgastraum und Kofferraum. Diese Blöcke dienen als "Container"für die Realisierung des Modells durch Kurven. Das Zentrum von Chassis-Block wird immer im Nullpunkt gerendert, damit das Finden von anderen Koordinaten erleichtert wird.

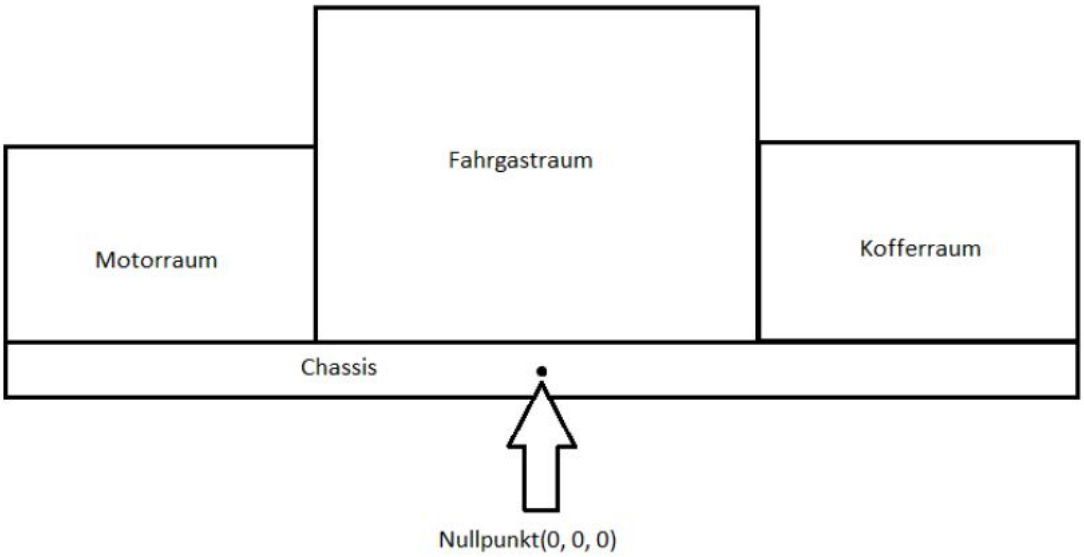


Abbildung 4.4.: Block-Modell des Autos

Allgemein betrachtet, um das Modell einfacher manipulieren zu können, brauchte man ein globales Koordinatensystem (in 2D-Darstellung). Die Linien, wo sich die verschiedene Blöcke

treffen/schneiden sind meine Achsen. Dementsprechend werden die wichtigsten Punkte wie folgt definiert:

- Schnittpunkt des Chassis mit dem Motorraum
- Schnittpunkt des Chassis mit dem Motorraum und mit dem Fahrgastraum
- Schnittpunkt des Chassis mit dem Fahrgastraum und mit dem Kofferraum
- Schnittpunkt des Chassis mit dem Kofferraum

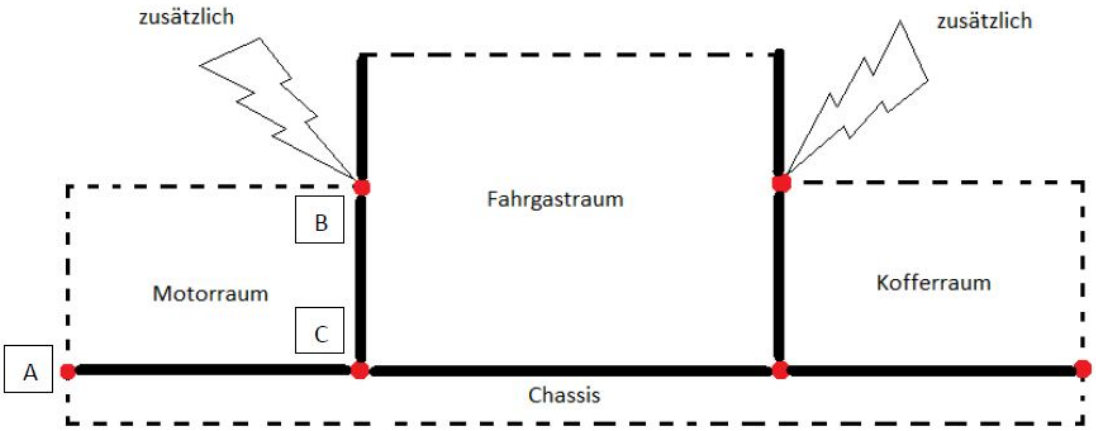


Abbildung 4.5.: Lokales Koordinatensystem

Zusätzlich, kann ich die Schnittpunkte des Fahrgastraums mit dem Motorraum und dem Kofferraum benutzen. Diese helfen mir später die Höhe der entsprechenden Blöcke zu bestimmen. Außerdem, weiß ich so, wo sich die Linie, der ein Block repräsentiert endet und die Linie, welche den Nachbarn-Block repräsentiert anfängt. Somit wird der Übergangspunkt bekannt. Die Punkte A und B (Abb. 4.5), zum Beispiel, werden dafür benutzt, um eine Kurve zu berechnen, die den Motorraum darstellt (Abb. 4.6).

Die Abb. 4.6 zeigt, wie eine Kurve für die Darstellung des Motorraums benutzt werden kann. Die größeren Punkte des globalen Koordinatensystems sind die Grenzpunkte der Kurve, und der kleine Punkt ist der Kontrollpunkt, der für das Konstruieren der Krümmung der Kurve gebraucht wird. Genauso kann man die anderen Komponente des Fahrzeugs erzeugen.

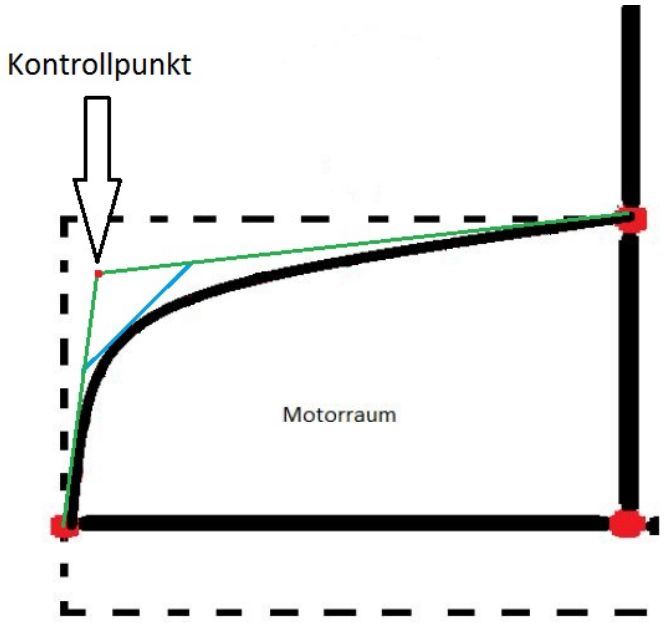


Abbildung 4.6.: Kurvendarstellung

4.4. Schmetterling-Modell

Um zu beweisen, dass mein Verfahren tatsächlich funktioniert, und nicht nur für die Generierung von Fahrzeugen geeignet ist, werde ich den auf eine weitere Domäne anwenden. Für die zweite Domäne wählte ich die Silhouette von Schmetterling aus. In diesem Abschnitt werde ich den Aufbau des Schmetterling-Modells beschreiben.

4.4.1. Körperbau

Die Abb. 4.7 demonstriert den Körperbau eines erwachsenen Schmetterlings. Es ist zu sehen, dass ein Schmetterling aus zu viel kleineren Körperteilen und Details besteht. Mein Ziel ist es nicht das genauere Modell des Insektes zu reproduzieren, sondern eine Silhouette darzustellen, als ob man den Schmetterling von oben betrachtet.

Somit reduziert sich die Anzahl von Körperteilen auf fünf (siehe Abb. 4.8). Das ist der Körper selbst, der linke obere und der linke untere Flügel, und der rechte obere und der rechte untere Flügel. In der Natur sind die Schmetterlinge nicht absolut symmetrisch. Einfachheitshalber werde ich aber mein Modell symmetrisch aufbauen.

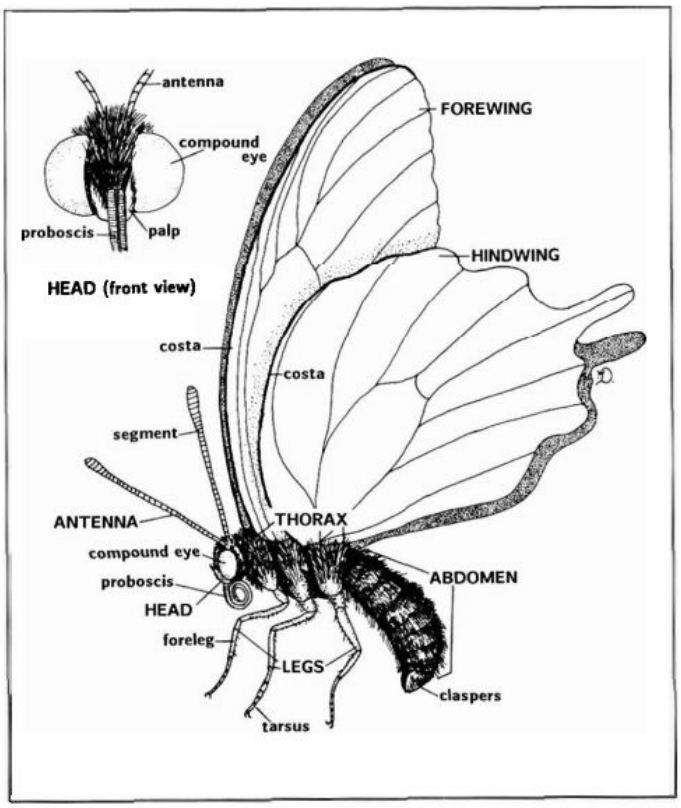


Abbildung 4.7.: Körperbau eines Schmetterlings ([Klass u. Dirig]).

Die Größe der Schmetterlinge variiert: vom kleinsten Mikroschmetterling mit 3 mm Spannweite bis zum größten Atlasspinner mit einer Spannweite von über 25 cm. In meiner Darstellung werde ich die relative Größe betrachten, sodass die Flügel zu einander und zu dem Körper maßstabgerecht sind.

4.4.2. Konzept

Das Modell des Schmetterlings baue ich genauso auf, wie das Model von einem Fahrzeug. Erstmal definiere ich die Rechteckigen Blöcke, welche die Körperteile des Insektes repräsentieren. Genau wie beim Fahrzeug-Modell dienen die Blöcke als "Behälter"für die Kurven. Das primitivste Modell eines Schmetterlings besteht also aus fünf rechteckigen Blöcken (Abb. 4.9).

Der Körper-Block ist immer mit dem Zentrum im Nullpunkt dargestellt und die anderen Blöcke werden dementsprechend positioniert. Auf dieser Abbildung kann man die wichtigsten Korrespondenzpunkte erkennen (rot markiert). Das sind die Punkte, wo sich die oberen und unteren Flügel treffen, und die Schnittpunkte von Flügeln mit dem Körper.

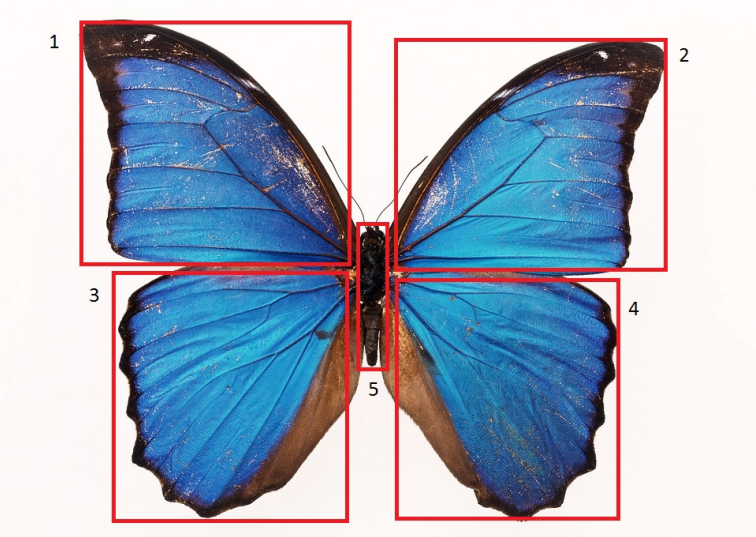


Abbildung 4.8.: Reduzierter Körperbau eines Schmetterlings: 1. Der linke obere Flügel; 2. Der rechte obere Flügel; 3. Der linke untere Flügel; 4 Der rechte untere Flügel; 5 Der Körper. (https://pixabay.com, Creative Commons CC0 Lizenz.)

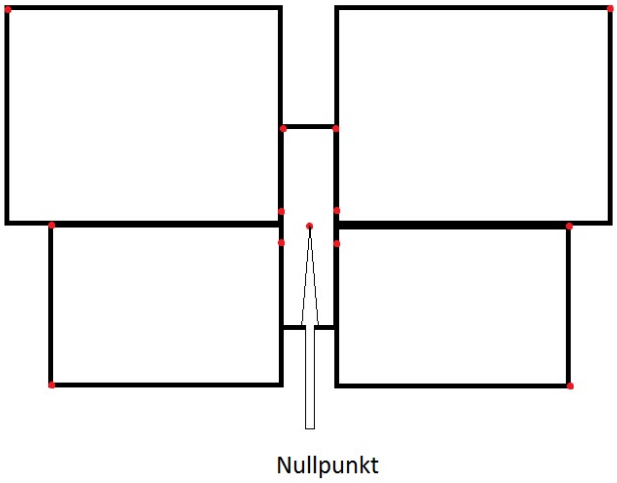


Abbildung 4.9.: Das primitivste Modell eines Schmetterlings mit Korrespondenzen

Außerdem, um den Schmetterling besser darstellen zu können, wurde jeder Flügel mit drei Bézier-Kurven dargestellt. Die Kurven wurden genau wie bei dem Fahrzeug-Modell innerhalb

von dem Block gezeichnet. Eine vollständige Schmetterling-Silhouette wurd mit 14 Bézier-Kurven dargestellt.

4.5. Fahrzeug-Datenbank

Um es zu analysieren, wie das prozedural synthetisierte Fahrzeug tatsächlich aussehen soll, wurde ein Set von typischen Fahrzeugen gesammelt [www.carlook.net]. Das sind 30 Fahrzeuge verschiedener Größen und Karosserietypen. Alle Beispielautos haben den gleichen Maßstab.

Alle 30 Fahrzeuge aus der Datenbank wurden manuell mit Hilfe von einem Editor als zweidimensionale Modelle nachgebildet. Als nächstes wurde jedes Auto als R^{28} Vektor mit Kontrollpunkten gespeichert. So entstanden 30 von solchen Vektoren mit den zu analysierenden Daten.

Abbildung 4.10.: Mit dem Editor manuell nachgebildetes Fahrzeugmodell (https://pixabay.com, Creative Commons CC0 Lizenz.)

4.6. Schmetterling-Datenbank

Um es zu analysieren, wie der prozedural synthetisierte Schmetterling tatsächlich aussehen soll, wurde ein Set von typischen Schmetterlingen gesammelt [www.babochkiz.narod.ru]. Das sind 30 Schmetterlinge verschiedener Formen.

Alle 30 Schmetterlinge aus der Datenbank wurden manuell mit Hilfe von ähnlichem Editor, wie bei dem Fahrzeug-Modell, als zweidimensionale Modelle nachgebildet. Als nächstes wurde jeder Schmetterling als R^{32} Vektor mit den Kontrollpunkten gespeichert. So entstanden 30 solcher Vektoren mit den zu analysierenden Daten.

5. Implementierung

In diesem Kapitel erläutere ich die Hauptpunkte der Implementierung meines Verfahrens. Hier bekommt man Einblick auf eine Klassenübersicht und manche Algorithmen mit Codebeispielen.

5.1. Entwicklungsumgebung und benutzte Software

Für die Realisierung meines Verfahrens wurde von mir Java 8 und Eclipse Mars als Entwicklungsumgebung verwendet. Außerdem arbeite ich mit einem Projekt namens «ComputerGraphics»-Framework.

Die «ComputerGraphics»-Framework ist ein von Prof. Dr. Philipp Jenke entwickeltes Projekt, welches die Entwicklung von Grafik-basierten Anwendungen erleichtert. Das Framework beinhaltet eine Menge von Klassen, welche die wichtigsten geometrischen Datenstrukturen und mathematische Operationen, zusammenfasst. Die Klassen, die ich in meinem Projekt benutze, sind zum Beispiel $BzierCurve$, PCA, $IVector3$, $IMatrix$ u.a..

Das Framework verwendet und erweitert die Bibliothek JOGL (Java-OpenGL). Dieses Projekt fasst alle CG-Objekte zusammen, indem es für jedes Objekt ein Node-Element erstellt. Ein Netz aus Nodes wird definiert und auf dem Bildschirm gerendert. Jedes Node kann man bearbeiten und/oder entfernen.

5.2. GUI

Die Bedienoberfläche ist ein sehr wichtiges Element meines Prototyps, weil an dieser Stelle die Parametrisierung des Fahrzeugs und des Schmetterlings stattfindet. Hier findet man eine Menge von Sliders, die für die Parameter des Objektes zuständig sind. Ändert man die Position von Sliders, wird das Obkejkt in Echtzeit neu generiert. Die Parameter, von Sliders abhängig sind, kann man in folgende Gruppen unterteilen:

Für die Fahrzeuge:

- **Automobilabmessungen:** das sind die Höhe, die Länge und die Breite des Fahrzeugs.

- **Art des Fahrzeugs:** das sind die Parameter, welche die Karosserieart des Fahrzeugs bestimmen. Dazu zählt man die Höhe von der Haube und die Höhe von dem Kofferraum, sowohl auch die Länge der einzelnen Blöcke relativ zueinander.
- **Umriss des Fahrzeugs:** diese Parameter beschreiben die Kurven, die das Umriss des Fahrzeugs bestimmen.

Für die Schmetterlinge:

- **Abmessungen des Schmetterlings:** die Höhe und die Länge des Insektes.
- **Körperparameter:** das sind die Parameter, welche die Breite und die Länge des Körpers bestimmen. Außerdem werden hier die Schnittpunkte von den Flügeln und Körper definiert.
- **Die Form von den Flügeln:** diese Parameter beschreiben die Kurven, welche den Umriss von den Flügeln bestimmen.

5.3. Klassenmodell

In diesem Abschnitt werde ich kurz die Klassen beschreiben, welche ich für die Implementierung des Modells geschrieben habe.

5.3.1. Fahrzeug

Da die vier Blöcke, die mein Fahrzeug-Modell beschreiben, für das Konzept sehr wichtig sind, hat jeder Block seine eigene Java-Klasse: $Chassis2D$, $Front2D$, $Gats2D$ und $Heck2D$. Außerdem gibt es eine Klasse $Auto2D$, die ein komplettes Fahrzeug-Modell beschreibt und die verschiedenen Blöcke verwaltet.

Jede Block-Klasse hat mehrere Variablen (Felder), die das Manipulieren und Erstellen von Instanzen vereinfachen. Da jeder Block im Prinzip ein Rechteck ist ("Behälter"),wurde dieser mit vier Punkten (*IVector3*) definiert. Die Punkte wurden bei jedem Block gleich verteilt: Punkt A (links unten), Punkt B (links oben), Punkt C (rechts oben), Punkt D (rechts unten). Demnächst wurden die Punkte mit entsprechenden Kanten verbunden (*Line3D*): AB, BC, CD und DA. Außerdem hat jeder Block zwei Variablen, die seine Höhe und seine Länge speichern. Mit Hilfe dieser Daten konnte man den Block einfacher manipulieren und rendern.

Die Klasse *Auto2D* nimmt eine Menge von Parametern an und generiert auf der Basis von denen vier Bestandsblöcke, die ein Auto repräsentieren. Die Klasse hat die Methoden

generateChassis(), *generateFront()*, *generateGast()* und *generateHeck()*, welche aus Parametern die konkreten Zahlen berechnen und dementsprechend die Böcke generieren.

Die Klasse Car definiert das Model des Fahrzeugs als ein Vektor mit Kontrollpunkten. Diese Klasse erlaubt es, die Fahrzeuge zu speichern und zu wiederherstellen.

5.3.2. Schmetterling

Genau wie bei dem Fahrzeug wurde jeder Körper-Block eines Schmetterlings durch eine eigene Java-Klasse repräsentiert. Die Klassen $LeftTopWing$, $LeftBottomWing$, $RightTopWing$, $RightBottomWing$ und $Body$ wurden nach gleichem Prinzip beschrieben - die vier Punkte A, B, C und D definieren die Kanten ($Line3D$). Die weiteren Variablem definieren die Höhe, Die Breite und das Zentrum jedes Blockes.

Die Klasse $Butterfly$ beschreibt das Block-Modell eines Schmetterlings, nimmt mehrere Parameter auf und erstellt aus denen ein Modell-Objekt. Die Klasse $ButterflyModel$ beschreibt einen Schmetterling als ein Vektor mit Kontrollpunkten. Die Klasse erlaubt es ebenso, die Objekte zu speichern und zu wiederherstellen.

5.3.3. Analyse

Die Klasse $Analyzer$ sammelt alle Methoden zusammen, welche für PCA zuständig sind. Hier findet die Hauptkomponentenanalyse statt. Der Methode $applyPCA()$ der Klasse wird eine Datei mit einer Menge Objekten übergeben. Die Objekte sind als Vektoren mit Kontrollpunkten gespeichert. Die Klasse $Amalyzer$ zerlegt die Daten in X-, Y- und Z-Daten. Für jedes Daten-Set werden die Eigenvektoren und Eigenwerte mit Hilfe von PCA berechnet. Dann werden die Eigenwerte analysiert und entsprechend der Dimensionsreduzierung werden die Matrizen B und B_o erstellt (Kapitel 3.3).

5.4. Kurven

Wie früher schon eingeführt, wurden für meine Realisierung die Bézier-Kurven verwendet.

Die rechteckigen Blöcke sind nur die “Behälter”für die tatsächliche Realisierung von Umrissen eines Fahrzeugs (bzw. Schmetterlings). Diese werden mit Hilfe von Kurven erstellt. Zur Realisierung wurden die Bézier-Kurve benutzt. Die Bézier-Kurve hat einen Grad und eine Reihe von Kontrollpunkte (Grad+1). Die Einfachste Bézier-Kurve erstes Grades hat nur zwei Kontrollpunkte: Start- und Endpunkt. Das ist eigentlich keine Kurve, sondern eine Gerade. Wenn man mehrere Kontrollpunkte hinzugefügt, kann man den Verlauf der Kurve kontrollieren.

Jede Block-Klasse des Fahrzeugs besitzt vier Bézier-Kurven (*left*, *top*, *right* und *bottom*), die man zur Darstellung seines Umrisses benutzen kann. Es wurden die Kurven zweites Grades verwendet. Das bedeutet, dass diese zwei Kontrollpunkte den Start und das Ende der Kurve definieren, und die Krümmungswert von noch einem Kontrollpunkt steuert, wie diese Kurve verläuft.

Zum Beispiel für die Windschutzscheibe ist die Bézier-Kurve *left* des Gast-Blockes zuständig. Ihr Startpunkt ist der Schnittpunkt von dem Front-Block und dem Gast-Block und der Endpunkt ist der Schnittpunkt der Kurve *top* von dem Gast-Block. Der dritte Kontrollpunkt, der die Krümmung bestimmt, ist standardmäßig der Punkt B dieses Blockes, ist aber mit Parameter änderbar.

Die Block-Klassen des Schmetterlings besitzen jeweils drei Bézier-Kurven zweites Grades. Genau wie bei dem Fahrzeug-Modell benutzt man diese, um den Umriss darzustellen.

6. Evaluation

In diesem Kapitel möchte ich die Ergebnisse von PCA evaluieren und zeigen, wie man diese für das Erzeugen von neuen Objekten verwenden kann.

6.1. Analyse der Hauptkomponenten

Wie im Kapitel 4.2.4 bereits beschrieben, um ein neues Objekt zu synthetisieren, muss man im Eigenraum die geeigneten Eigenkoeffizienten finden und aus diesen einen Vektor a zusammen bilden. Diese Eigenkoeffizienten lassen sich mit Hilfe von PCA finden, indem man alle Beispielobjekte in Eigenraum transformiert (Kapitel 4.2.2). Demnächst wird es analysiert, wie verschiedene Koeffizienten bei verschiedenen Eigenvektoren auswirken. Die Werte aller Beispielobjekte wurden für jeden Eigenvektor (Hauptkomponente) miteinander verglichen(Abb.6.1 - 6.2). Es wurde das Minimum, das Maximum und der Mittelwert für jede Hauptkomponente gefunden. Die Werte zwischen das Minimum und das Maximum wurden als Koeffizienten für die Synthese neuer Objekte benutzt. Den Mittelwert kann man verwenden, um das typische durchschnittliche Objekt einer Domäne zu erzeugen.

6.1.1. Fahrzeug

Die Analyse hat gezeigt, dass die meisten Werte der X-Achse zwischen -0,5 und +0,5 liegen und der Mittelfaktor nähert sich dem Wert von 0. Die Unterschiede zwischen den Werten sind ziemlich klein. Das bedeutet, dass sogar eine kleine Änderung an den Koeffizienten wird das Modell verändern und so ein neues Ergebnis generieren. Die Variation von resultierenden Ergebnissen ist aber klein, da die Unterschiede minimal sind. Das ist aber legitim für die Fahrzeuge.

Für die Y-Achse ist die Verteilung noch kleiner (zwischen -0,1 und +0,1). Der Mittelwert nähert sich wieder der 0. Daraus folgt, dass die Y-Koeffizienten nicht so eine große Auswirkung auf das Modell haben, wie die X-Koeffizienten.

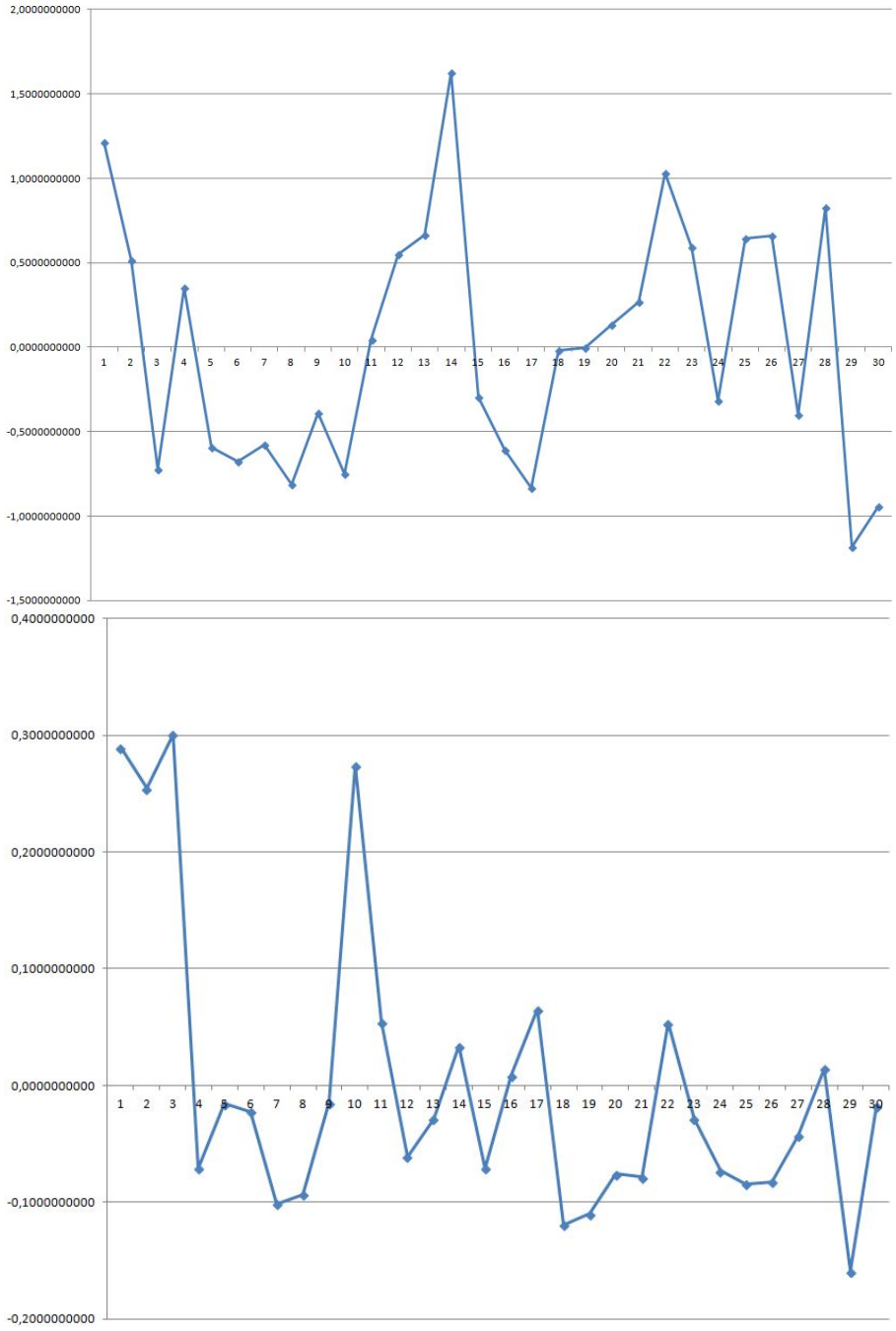


Abbildung 6.1.: Fahrzeug-Hauptkomponente 1: X- und Y-Werte.

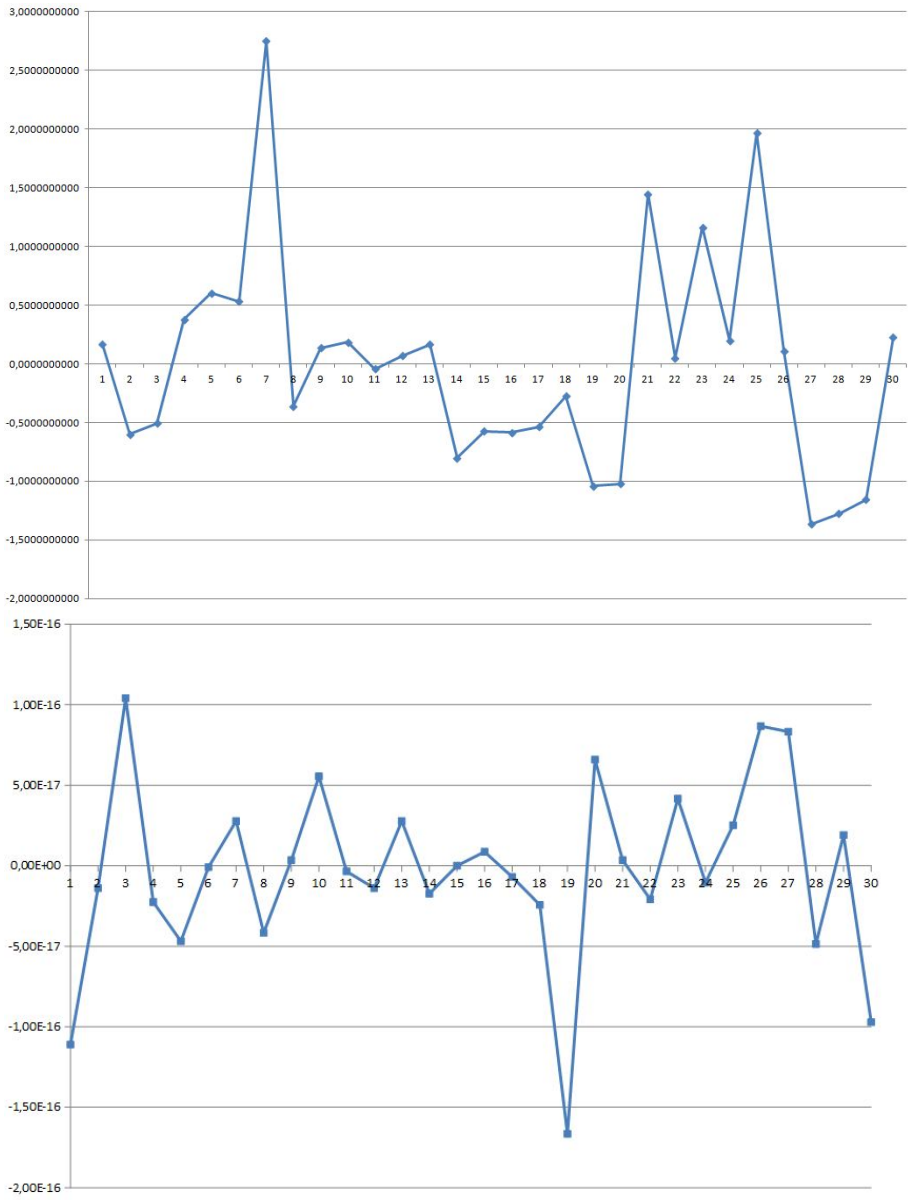


Abbildung 6.2.: Schmetterling-Hauptkomponente 1: X- und Y-Werte.

6.1.2. Schmetterling

Die Analyse hat gezeigt, dass die meisten Werte der X-Achse zwischen -1 und +2 liegen und der Mittelfaktor nähert sich wieder dem Wert von 0. Im Unterschied zu den Fahrzeugen, haben die Werte in diesem Fall den größeren Unterschiedsfaktor. Daraus folgt eine größere Variation von resultierenden Ergebnissen, als bei den Fahrzeugen. Das wird auch durch eine größere Anzahl

von den Hauptkomponenten erreicht. Dies entspricht auch der Realität, da die Schmetterlinge sich viel mehr unterscheiden, als die Fahrzeuge.

Für die Y-Achse ist die Verteilung extrem klein (zwischen -1×10^{-16} und $+1 \times 10^{-16}$). Der Mittelwert nähert sich wieder dem 0-Wert. Trotzt solcher kleiner Unterschiede, die Änderungen an dem Y-Koeffizienten beeinflussen stark das Ergebnis.

Theoretisch kann man auch die größeren bzw. die kleineren Werte als den Eigenkoeffizienten verwenden. Und diese würden auch sinnvolle Ergebnisse liefern. Die Werte, welche ich in meiner Realisierung verwendet habe, erfolgten aus der Analyse von 30 typischen Objekten der Domäne.

Die Beispiel-Ergebnisse meiner Realisierung kann man im Anhang sehen.

6.2. Dimensionsreduzierung

In meiner Realisierung habe ich die im Kapitel 4.2.3 eingeführte Dimensionsreduzierung angewandt. Für die Synthese von neuen Objekte habe ich nur die wichtigsten Hauptkomponenten verwendet. Das waren 7 Hauptkomponenten für die Autos und 10 Hauptkomponenten für die Schmetterlinge. Alle andere Werte waren Nullen und hatten keinen Einfluss auf das resultierendes Model.

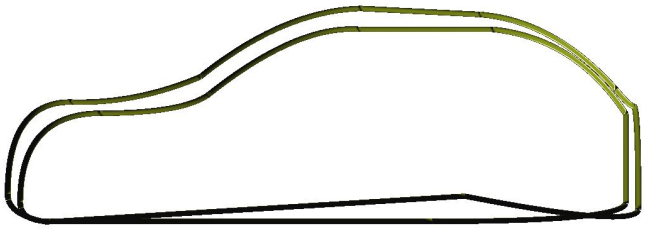


Abbildung 6.3.: Beispielauto und nur mit einem Hauptkomponente zurück transformiertes Auto.

Für die Evaluation habe ich versucht, weniger Hauptkomponenten zu verwenden, um ein Beispielobjekt zurück zu transformieren. Für die Autos habe ich festgelegt, dass sogar mit nur einem Hauptkomponente sind die Unterschiede zwischen dem Beispielobjekt und dem zurück transformiertem Objekt nicht so groß (Abb. 6.3). Es ist aber nicht sinnvoll weniger

Hauptkomponenten zu verwenden, weil dann hat man weniger Flexibilität bei der Erzeugung eigener Autos.

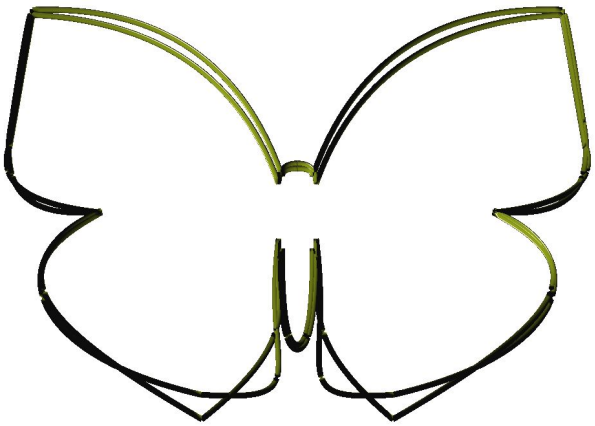


Abbildung 6.4.: Beispielschmetterling und mit zehn Hauptkomponenten zurück transformierter Schmetterling.

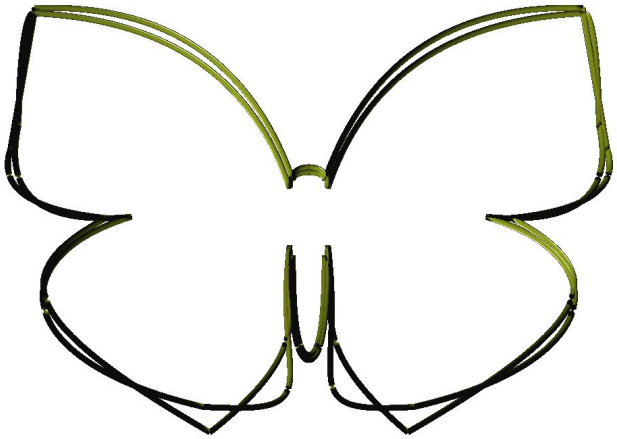


Abbildung 6.5.: Beispielschmetterling und mit fünf Hauptkomponenten zurück transformierter Schmetterling.

Die Ergebnisse sehen bei Schmetterlinge anders aus. Je weniger Hauptkomponenten man verwendet, desto mehr Unterschiede gibt es zwischen Beispiel- und zurück transformiertem Objekt (Abb. 6.4 - 6.7).

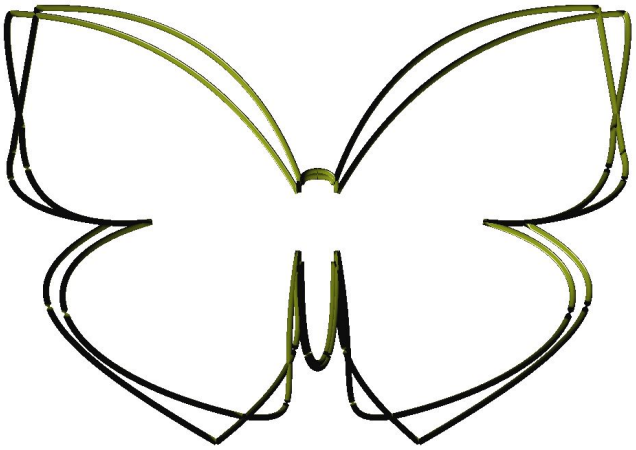


Abbildung 6.6.: Beispielschmetterling und mit drei Hauptkomponenten zurück transformierter Schmetterling.

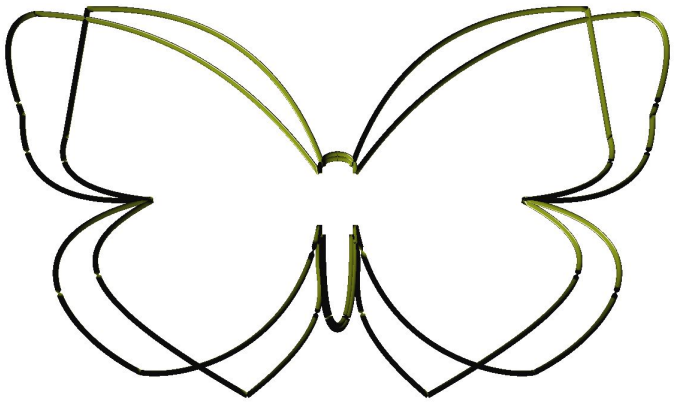


Abbildung 6.7.: Beispielschmetterling und mit einem Hauptkomponente zurück transformierter Schmetterling.

6.3. Generierung

Um die Funktionalität meines Generators zu evaluieren, versuche ich einige Objekte aus realer Welt nachzubilden. Ich habe die Bilder von drei Test-Fahrzeuge und die Bilder von drei Test-Schmetterlinge gefunden. Dann habe ich diese Test-Objekte mit meinem Generator nachgebildet. Die Ergebnisse kann man in den Abbildungen 6.8 bis 6.19 sehen.

Man kann hier sehen, dass der Generator ziemlich gut die Fahrzeuge nachbilden kann. Außer verschiedenen kleineren Details ist die Form des nachgebildeten Fahrzeugs fast identisch mit dem Test-Objekt. Man kann also mit dem Generator nicht nur ßufällige"Fahrzeuge erzeugen, sondern auch ganz bestimmte Objekte nachbilden.

Die Ergebnisse des Schmetterling-Generators sehen nicht so gut aus. Die "primitiveSSchmetterlinge können mit dem Generator ganz gut realisiert werden. Es gibt aber Probleme mit den Schmetterlinge, wessen Flügel die rau Form haben. Besonders schwierig ist es die SSchwänzeäm Flügel nachzubilden. Man muss hier an Model des Schmetterlings weiter arbeiten, zum Beispiel mehr Kontrollpunkten für die unteren Flügel hinzufügen.

Abbildung 6.8.: Test-Fahrzeug 1 (https://pixabay.com, Creative Commons CC0 Lizenz.)

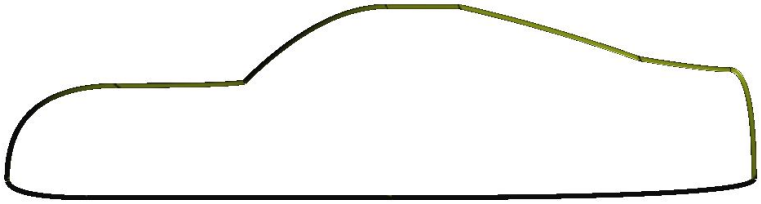


Abbildung 6.9.: Test-Fahrzeug 1, nachgebildet.

Abbildung 6.10.: Test-Fahrzeug 2 (https://pixabay.com, Creative Commons CC0 Lizenz.)

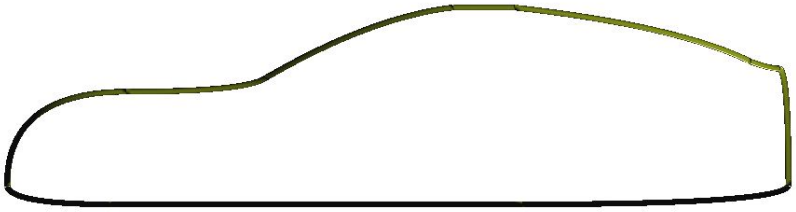


Abbildung 6.11.: Test-Fahrzeug 2, nachgebildet.

Abbildung 6.12.: Test-Fahrzeug 3 (https://pixabay.com, Creative Commons CC0 Lizenz.)

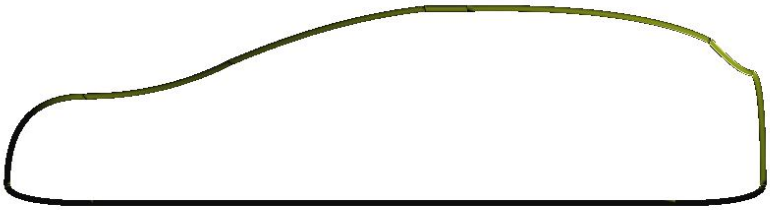


Abbildung 6.13.: Test-Fahrzeug 1, nachgebildet.

Abbildung 6.14.: Test-Schmetterling 1(https://pixabay.com, Creative Commons CC0 Lizenz.)

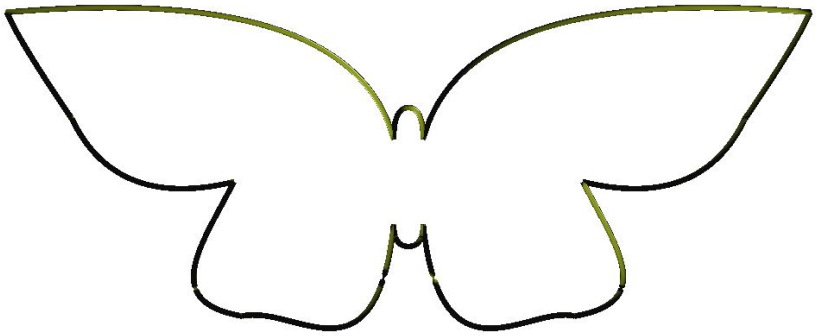


Abbildung 6.15.: Test-Schmetterling 1, nachgebildet.

Abbildung 6.16.: Test-Schmetterling 2 (https://pixabay.com, Creative Commons CC0 Lizenz.)

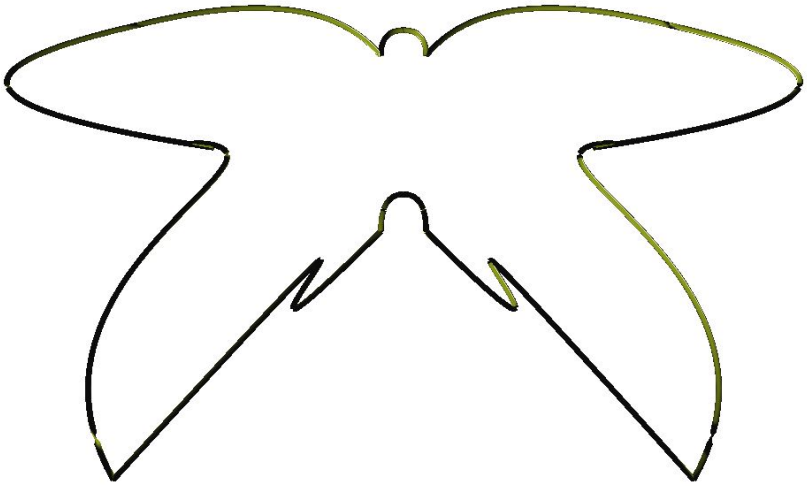


Abbildung 6.17.: Test-Schmetterling 1, nachgebildet.

Abbildung 6.18.: Test-Schmetterling 3 (https://pixabay.com, Creative Commons CC0 Lizenz.)

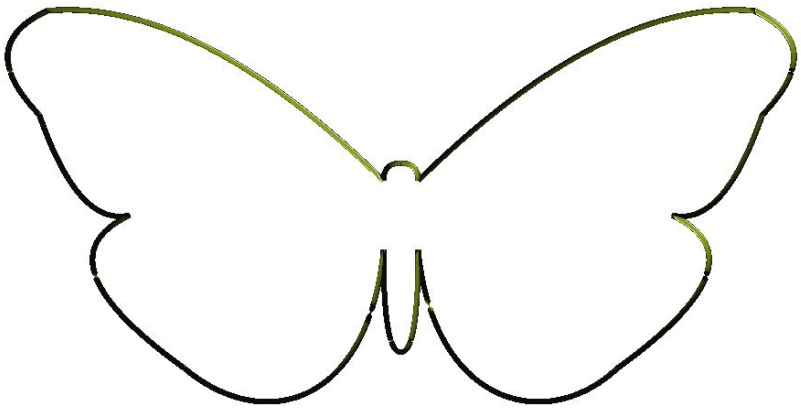


Abbildung 6.19.: Test-Schmetterling 1, nachgebildet.

7. Zusammenfassung und Ausblick

In diesem Kapitel fasse ich alles, was ich bis jetzt erreicht habe, zusammen und sage ein Paar Worte dazu, wie man dieses Projekt weiterentwickeln könnte.

7.1. Zusammenfassung

Die Idee meiner Arbeit war es, einen neuen Ansatz für die prozedurale Modellierung der Silhouetten zu erarbeiten. Es wurde ein Konzept entwickelt und ein Software geschrieben, um das Ziel zu erfüllen.

In Rahmen meiner Arbeit wurde ein formales Modell vorgestellt, welches man für die Analyse und für die Generierung von Objekten verwenden kann. Es wurde ein Ansatz vorgestellt, wie man auf einer Basis von einer Menge analysierter typischer Objekte die neuen Exemplare generieren kann. Dafür wurde die PCA verwendet, um aus den analysierten Daten die Hauptkomponenten zu bestimmen und die Eigenkoeffizienten zu finden, welche man für das Erzeugen von neuen Objekten verwendet.

Außerdem wurde mein Konzept evaluiert, indem man es auf zwei Beispiel-Domäne angewandt hat. Es wurde ein konkretes Modell des Fahrzeugs und ein konkretes Modell des Schmetterlings vorgestellt, sowie eine Menge von typischen Objekten für jede Domäne analysiert. Letztlich wurden die passende Koeffizienten gefunden und diese als GUI (Sliders) dargestellt.

Am Ende habe ich die Ergebnisse von meinem Generator vorgestellt.

7.2. Ausblick

7.2.1. Korrespondenzfindung

Um ein Modell einer beliebigen Domäne für mein Konzept zu entwickeln, muss man zuerst die Korrespondenzen zwischen verschiedenen Exemplaren dieser Domäne finden. Diese Korrespondenzen sind dann die Star-/Endpunkte für die Bézier-Kurven. Ich habe das "manuell"gemacht,

indem ich die Objekte analysiert habe und die Korrespondenzpunkte selber definiert habe. Man könnte die Korrespondenzfindung an dieser Stelle automatisieren.

7.2.2. Automatisierte Modellaufbau

Um das mit Bézier-Kurven beschriebene Modell zu erstellen, wurden die rechteckigen Blöcke verwendet. Diese Blöcke waren die "Behälter"für die Bézier-Kurven und haben außerdem als ein Koordinatensystem für die Korrespondenzpunkte gedient. Hätte man das Korrespondenzproblem gelöst, könnte man den Prozess des Modellaufbaus automatisieren. Der Algorithmus könnte die Angaben (eine Menge von Bildern mit typischen Objekten der Domäne) analysieren, Korrespondenzen finden und diese Punkte sofort mit Bézier-Kurven miteinander verbinden. Das könnte eine Basis von dem Modell sein.

7.2.3. 3D Darstellung

Der nächste große Schritt wäre es ein Modell in drei Dimensionen darzustellen. Die PCA-Komponente analysiert jetzt schon die Daten für z-Achse, die sind aber für die zweidimensionalen Silhouetten überall 0. Deswegen ist es zu überlegen, wie so ein Modell in einem dreidimensionalem Raum aussehen könnte und wie man dieses dann grafisch darstellen könnte.

Literaturverzeichnis

[Blanz u. Vetter] Volker Blanz; Thomas Vetter: A Morphable Model For The Synthesis Of 3D Faces. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA 1999

[Jolliffe] I.T.Jolliffe: Principal Component Analysis,Second Edition. Springer-Verlag, New York, 1986

[Maier] Benjamin Maier: Ein Morphable Model für Fische. Wilhelm Schickard Institut für Informatik, Tübingen, Mai 2007

[Aguilera] A.M. Aguilera; R. Gutierrez; F.A. Ocana and M.J. Valderrama. Computational approaches to estimation in the principal component analysis of a stochastic process. Appl. Stoch. Models Data Anal. 1995

[Aitchison] J. Aitchison. Principal component analysis of compositional data. Biometrika, 70, 1983

[Ali et al.] A. Ali; G.M. Clarke and K. Trustrum. Principal component analysis applied to some data from fruit nutrition experiments. Statistician, 34, 1985

[Al-Kandari] N. Al-Kandari; Variable Selection and Interpretation in Principal Component Analysis. Unpublished Ph.D. thesis, University of Aberdeen, 1998

[Hecker et al.] F. Pighin, J. Hecker, D. Lischinski, Szeliski R, and D. Salesin. Synthesizing realistic facial expressions from photographs. In Computer Graphics Proceedings SIGGRAPH'98, pages 75–84, 1998.

[Prusinkiewicz und Lindenmayer] Prusinkiewicz, Przemyslaw; Lindenmayer, Aristid: The algorithmic beauty of plants. New York: Springer-Verlag, 1990

[Kelly u. McCabe] George Kelly; Hugh McCabe: A Survey of Procedural Techniques for City Generation. ITB Journal, Issue 14, 2006

[Ebert et al.] David S. Ebert; F Kenton Musgrave; Darwyn Peachy; Ken Perlin; Steven Worley; Texturing and Modelling - A Procedural Approach. Morgan Kaufmann 2003.

[Lindenmayer] A. Lindenmayer; Mathematical models for cellular interaction in development, Parts I and II. 1968.

[Schuster et al.] Andreas Schuster; Josef Sattler; Stephan Hoffmann: Bestimmen der aktuellen Abmessungen differenzierter Personen-Bemessungsfahrzeuge. Westsächsische Hochschule Zwickau, Institut für Verkehrssystemtechnik i. G., 2011

[Holldack] Mario Holldack: Bézierkurven. Ausarbeitung im Proseminar Bernsteinpolynome und ihre Anwendungen, 2011

[Dettmers] Sven Dettmers: Prozedurale Generierung von Siedlungen in einer mutable cube world. Fakultät Technik und Informatik, Studiendepartment Informatik. 2014

[Schwarz u. Köchler] Hans Rudolf Schwarz; Norbert Köchler: Numerische Mathematik. 7. Auflage. In: Vieweg+Teubner; GWV Fachverlage GmbH, Wiesbaden 2009

[Handler] Bernhard Handler: Prozedurale Levelgenerierung für 2D Plattformspiele. Fachhochschul-Masterstudiengang Interactive Media ,Hagenberg, Oktober 2012

[Bernstein] S.N. Bernshtein. Collected works, 1 - 2, Moscow (1952 - 1954)

[Laval] Philippe B. Laval. Mathematics for Computer Graphics - Barycentric Coordinates, Kennesaw State University, 2003

[Mardia et al.] Mardia, KV; Kent, JT; Bibby, JM. Multivariate Analysis, New York 1979

[Gene u. Kahan] Gene H. Golub; William Kahan. Calculating the singular values and pseudo-inverse of a matrix. SIAM J. Numer. Anal., Ser. B, vol. 2 1965

[Mandelbrot] Benoit B. Mandelbrot. The Fractal Geometry of Nature. W.H. Freeman & Co. 1982.

[Perlin] Ken Perlin. Making Noise, http://www.noisemachine.com/talk1/index.html. 1999

[Worley] Steven Worley: A cellular texture basis function. Proceeding SIGGRAPH, 1996

[Klass u. Dirig] Carolyn Klass and Robert Dirig: Learning about Butterflies. 4-H Member/Leader Guide 139-M-9

[SpeedTree] Interactive Data Visualization Inc. SpeedTree RT. http://www.speedtree.com, (Dezember 2015).

[Autoteile] http://aceautomotive.biz/car/car-diagram-exterior-parts-names.html, (Dezember 2015).

[Wikipedia] https://en.wikipedia.org/wiki/Three-box_styling, (Dezember 2015).

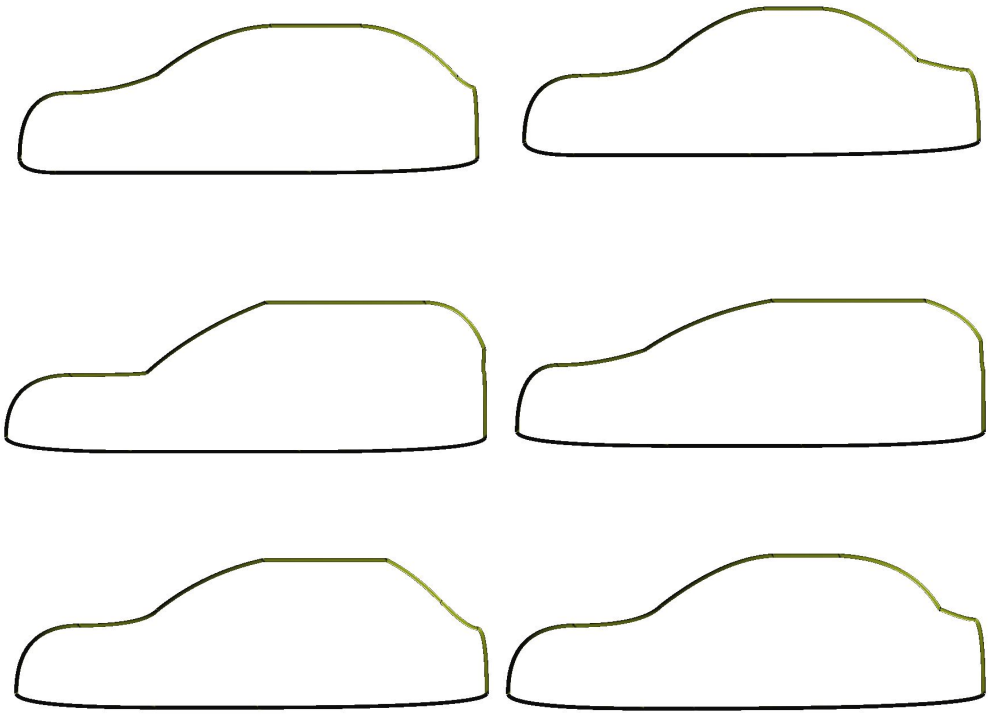
[www.carlook.net] http://carlook.net/de/db/, (Dezember 2015).

[www.babochkiz.narod.ru] http://babochkiz.narod.ru/fon.html, (Februar 2016).

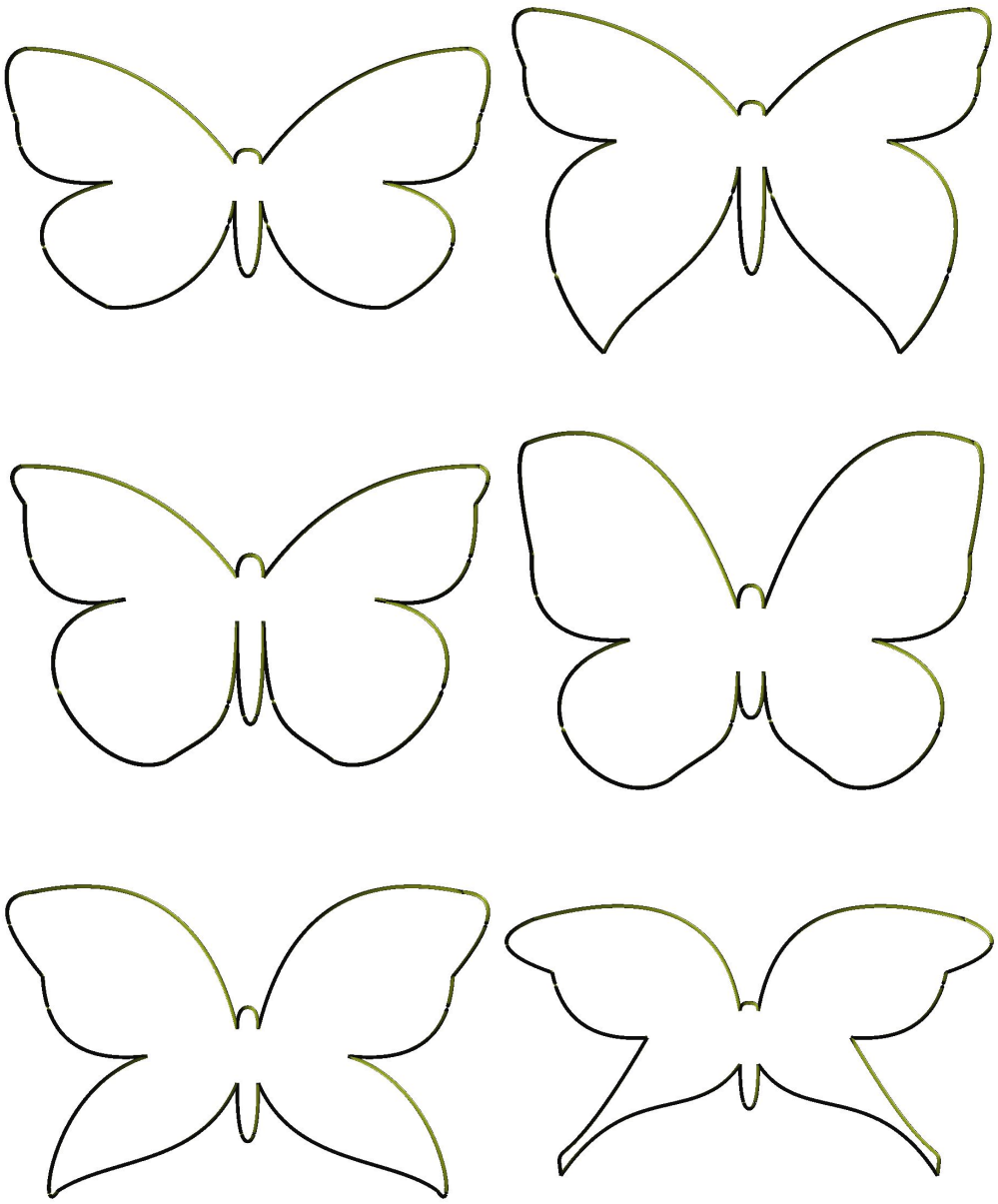
Teil I.

Anhang

.1. Fahrzeug-Generator Ergebnisse



.2. Schmetterling-Generator Ergebnisse



Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 1. Februar 2016 Vitalij Kagadij