GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Uploaddatum: 17.09.2020
Uploadzeit: 15:44

Dies ist ein von FlexNow automatisch beim Upload generiertes Deckblatt. Es dient dazu, die Arbeit automatisiert der Prüfungsakte zuordnen zu können.

**This is a machine generated frontpage added by FlexNow.
Its purpose is to link your upload to your examination file.**

Matrikelnummer: 21662292

Bachelorarbeit im Studiengang „Angewandte Informatik“

Entwurf und Implementierung eines Konstruktions-Werkzeugs für NURBS-Kurven und -Flächen

Elsa Rommerskirchen

Institut für Informatik

Bachelor- und Masterarbeiten des Zentrums für angewandte Informatik an der Georg-August-Universität

Georg-August-Universität
Institut für Informatik
Goldschmidtstr. 7,
37077 Goettingen

Erstbetreuer: Prof. Dr. Kurth
Zweitbetreuer: Prof. Dr. Damm

Inhaltsverzeichnis

Abkürzungsverzeichnis

CAD – Computer-Aided Design
GLenum - Graphics Library enumerator value
GroIMP - Growth Grammar-related Interactive Modelling Platform
JOGL - Java Binding for OpenGL
OpenGL - Open Graphics Library
NURBS - Non-Uniform Rational B-Spline
XL - Programmiersprache: eXtended L-Systems

1. Einführung

1.1 Motivation

Konstruktionswerkzeuge in Softwaresystemen sind unverzichtbare Werkzeuge, mit denen Administratoren, Entwickler und Nutzer laufender Systeme die Anpassung einfacher Modelle an spezielle Bedürfnisse verbessern können. Eine Anforderung an Modellierung ist es, die Realität vereinfacht und trotzdem mathematisch genau darzustellen. Gerade im Bereich der Forstwissenschaft, wo enge Verknüpfungen von Form und Funktionalität herrschen, kann die realitätsnahe Darstellung von Objekten oder Formen eine große Rolle bei der Beschreibung und Beurteilung von Zusammenhängen spielen. Einige Beispiele dafür sind die Oberfläche von Blättern bei der Beurteilung von Photosyntheseleistung, die Wuchsform von Baumkronen für die Wiedergabe von Strukturparametern oder die Bewertung des Konkurrenzverhaltens von Baumarten anhand von Baumwachstumsformen. Dafür kann es notwendig sein, in einem Modell nicht nur einfache geometrische Formen, sondern komplexere Objekte zu nutzen. Eine Möglichkeit solche Objekte mathematisch genau auszudrücken bieten Non-uniform rational BSplines (NURBS). NURBS sind Kurven oder Flächen, die durch einzelne Kontrollpunkte definiert werden.
Das Ziel dieser Arbeit ist es, GroIMP, eine an der Forstlichen Fakultät der Universität Göttingen entwickelte Software, zur regelbasierten dreidimensionalen Modellierung dynamischer Systeme, insbesondere von Wachstumsprozessen, um einen Editor zu erweitern, mit dem NURBS-Kurven und -Flächen interaktiv erstellt werden. GroIMP bietet bereits die Möglichkeit NURBS-Objekte darzustellen, jedoch muss dazu die Definition des Objekts, inklusive Angabe aller Kontrollpunkte durch Koordinaten, in einem Texteditor aufgerufen werden. Das ist umständlich und durch die unmittelbare Eingabe über den Texteditor ist die Angabe der Koordinaten unanschaulich. Bei einer großen Anzahl an Kontrollpunkten führt diese Art der Eingabe leicht zu Fehlern oder unerwarteten Ergebnissen. Daher hat das hier entwickelte Plug-in die Aufgabe die Eingabe über einen graphischen Editor zu ermöglichen, um Kurven und Flächen schnell, einfach und präzise definieren zu können. Es soll die Möglichkeit bestehen, die erstellten Kurven und Flächen in einer Szene in GroIMP zu nutzen. Beispiele für spätere Anwendungen können die Darstellung von einzelnen Pflanzenteilen wie Blüten oder Blätter und von Bodenflächen sein.

1.2 Struktur der Arbeit

In der vorliegenden Arbeit werden zunächst die softwaretechnischen Voraussetzungen und die mathematischen Grundlagen vorgestellt, insbesondere das zugrundeliegende Programm GroIMP, die externe Bibliothek JOGL und die mathematischen Definitionen für NURBS-Kurven und -Flächen. Anschließend wird die Struktur des Plug-ins entsprechend eines modell-view-controller Architektur- und Entwurfsmusters (MVC) erläutert, bevor die Implementierung der zwei- und dreidimensionalen Kurven und Flächen beschrieben, sowie die Manipulationsmöglichkeiten gezeigt werden. Die Funktionsfähigkeit des Plug-ins soll dann mithilfe von Ergebnissen gezeigt werden und einige Beispiele veranschaulichen den Funktionsumfang und einzelne Anwendungen. Zum Schluss folgen eine Zusammenfassung der Arbeit und ein Ausblick auf Möglichkeiten zur Erweiterung der Funktionalität.

2. Grundlagen

Im Folgenden werden die Grundlagen beschrieben, die die Voraussetzungen für die Konzeption und Einbindung des Plug-ins darstellen.

2.1 GroIMP

Die Software GroIMP ist eine graphbasierte Modellierungsplattform. Sie verwendet eine eigene regelbasierte Programmiersprache, genannt XL, die sich als eine Kombination aus Graphgrammatiken und Java definieren lässt (Kniemeyer 2008). Durch die regelbasierten Ersetzungsschritte ergibt sich die Möglichkeit eine zeitliche Entwicklung, wie Wachstum, von Objekten darzustellen. Dabei können unterschiedliche Einflussfaktoren in die Berechnung der Entwicklung einbezogen werden, um zum Beispiel die Wirkung von Licht, Konkurrenz, o.ä. zu simulieren. Die zugrundeliegende Datenstruktur der Szenenobjekte und Transformationen ist eine Graphstruktur (Kniemeyer 2008). Zu den darstellbaren Szenenobjekten gehören Primitivobjekte wie unter anderem Kugel, Quader oder Zylinder, aber auch Freiformkurven und -flächen (NURBS). Diese Objekte bilden zusammen mit Transformationsoperationen, wie Translation und Rotation, die Menge der Knoten der Graphstruktur. Die Transformationen wirken sich auf den gesamten Subgraphen aus, also auf alle nachfolgenden Knoten. Ein Szenengraph kann mithilfe der Knoten durchlaufen werden und es kann nach einzelnen Knoten oder Mustern aus Knoten gesucht werden. Auf diese Weise können Ersetzungsregeln automatisch auf eine gesamte Szene angewendet werden (Kniemeyer 2008).
Die Software GroIMP ist als Plug-in-Struktur aufgebaut, wobei ausgehend von einer Plattform-Core-Einheit alle weiteren verfügbaren Packages als Plug-ins geladen werden. Für diese Schnittstelle sind bestimmte Anforderungen definiert. Zum Erstellen eines neuen Plug-ins ist es nötig, die Implementation einer vorgegebenen Ordnerstruktur anzupassen und eine plugin.xml-Datei als Verbindungsstelle zwischen Plug-in und Plattform-Core zu definieren. Dies wird in Kapitel 3 Konzept erläutert.

2.2 OpenGL

OpenGL ist eine 1992 definierte Programmierschnittstelle und gibt als Standard Befehle zur graphischen Darstellung komplexer zwei- und dreidimensionaler Szenen vor (Shreiner et al. 2013).
Es ist als Zustandsautomat konstruiert, es werden also nicht in jedem Schritt alle Parameter neu angegeben oder definiert, sondern Parameter werden solange gespeichert, bis sie geändert werden. Beispiele dafür sind die Funktion `setColor()`, welche die Farbe für alle nachfolgenden Objekte setzt, oder die Funktion `Translate()`, welche die gesamte Szene verschiebt und nicht für jedes Teilobjekt einzeln aufgerufen werden muss.
Die Basis, um dreidimensional erscheinende Graphiken auf dem Bildschirm zu erzeugen, bilden Transformationen.

2.2.1 Räumliche Transformation

Für das Erzeugen von dreidimensionalen Computergrafiken werden die darzustellenden Objekte durch dreidimensionale Koordinaten gespeichert. OpenGL definiert einen dreidimensionalen Raum (Abb. 1) durch eine horizontale x-Achse, eine vertikale y-Achse und eine z-Achse, die auf den Betrachter zugeht (Eck 2018).

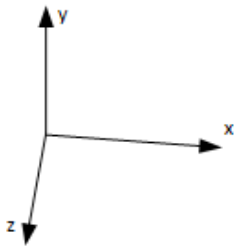

Abb. 1: Orientierung eines dreidimensionalen Koordinatensystems

Um auf einer zweidimensionalen Ebene wie dem Computerbildschirm den Eindruck eines dreidimensionalen Raumes zu erzeugen, müssen Umformungen durchgeführt werden. Es muss festgelegt werden, welchen Bildbereich der Bildschirm in x-, y- und z-Richtung umfasst und welche Art der Projektion genutzt wird. OpenGL bietet dafür zwei Projektionen: orthografisch (`glOrtho()`) und perspektivisch (`glPerspective()`). Die orthografische Projektion erzeugt als Sichtfeld einen Quader (Abb. 2), was dem Nutzer eine zweidimensionale Ansicht bietet, da die Objekte unabhängig von der Entfernung zum Blickpunkt (in Abb. 2 gegeben durch „Eye at z = 0“) gleich groß erscheinen. Die Methode `glOrtho()` wird mit Parametern, die den linken, rechten, oberen, unteren, vorderen und hinteren Rand des Quaders bestimmen, aufgerufen.

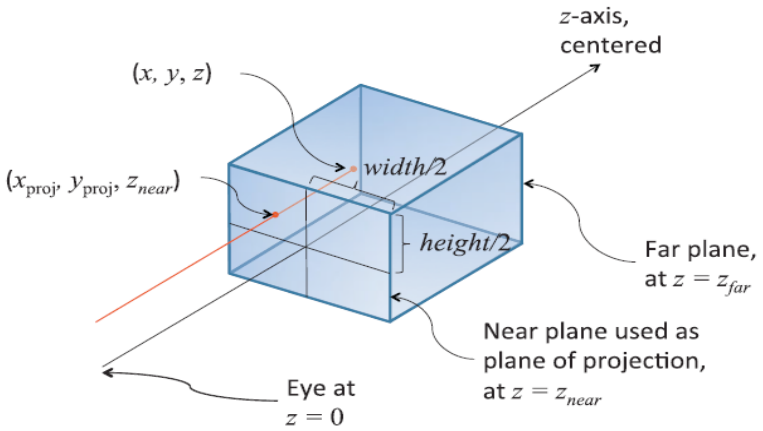

Abb. 2: Sichtbarer Bereich der orthografischen Ansicht (Shreiner et al. 2013, S. 230)

Im Vergleich dazu wird mit der perspektivischen Projektion ein Sichtbereich in Form eines Frustums (Pyramidenstumpf) definiert (Abb.3). Dieses ermöglicht die realitätsnahe Darstellung von Objekten in einem dreidimensionalen Raum, da die Größe der Objekte abhängig von der Entfernung zum Blickpunkt variiert.

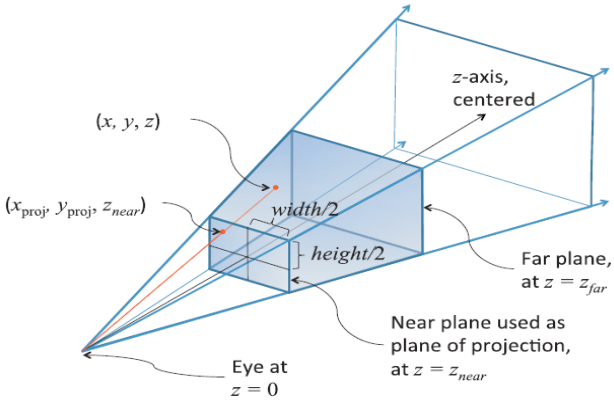

Abb. 3: Sichtbarer Bereich einer perspektivischen Projektion (Shreiner et al. 2013, S. 228)

Um eine perspektivische Projektion zu erzeugen, muss der Winkel in y-Richtung und der in x-Richtung, sowie der vordere und hintere Rand des Frustums angegeben werden.

2.2.2 Geometrische Transformationen

OpenGL bietet vordefinierte Funktionen zur Transformation von Objekten innerhalb der Szene. Zu den Operationen gehören Translation, Skalierung und Rotation.

- Die Translation bewegt jeden Punkt des Objekts um einen konstanten Wert in die angegebene Richtung (Abb.4).

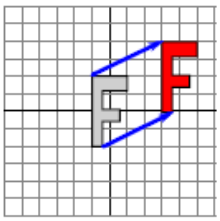

Abb. 4: Verschieben eines Objektes (Eck 2018, S. 30)

Diese Operation kann durch die Methoden `glTranslatef()` mit float-Werten oder `glTranslated()` mit double-Werten durchgeführt werden. Als Parameter wird die Verschiebung in x-, y- und z-Richtung angegeben. Eine Verschiebung eines Objektes entsteht durch Addition des Verschiebungsvektors mit den Koordinatenvektoren.

- Bei der Skalierung wird das Objekt in einem unveränderten Koordinatensystem entweder gestaucht oder gestreckt (Abb. 5).

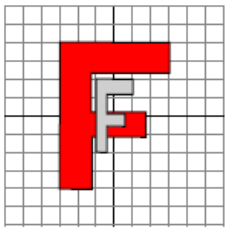

Abb. 5: Skalieren eines Objektes (Eck 2018, S. 32)

Die Skalierung kann durch die Methoden `glscalef()` und `glscaled()` durchgeführt werden, welche grundsätzlich eine Vektormultiplikation umsetzen.

- Die Rotation dreht das Objekt von einem gegebenen Punkt aus um einen bestimmten Winkel (Abb.6).

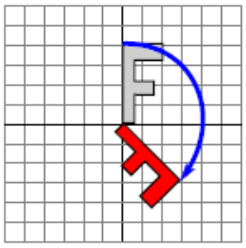

Abb. 6: Rotation eines Objekte (Eck 2018, S. 31)

Bei einer Rotation wird die Methode `glrotatef()` oder `glrotated()` genutzt, um den Koordinatenvektor mit einer Rotationsmatrix zu multiplizieren. Die Formel (2.1) stellt die zweidimensionale Rotationsmatrix für die Rotation um den Winkel α dar. Die Einträge der Rotationsmatrix sind abhängig von dem Winkel α der Rotation und im dreidimensionalen Fall auch von der Rotationsachse.

$$R_\alpha = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{2.1}$$

Durch die Struktur des Zustandsautomaten werden die entsprechenden Transformationen auf alle nachfolgend definierten Objekte angewendet.

2.2.3 Primitivobjekte

Als Unterstützung für das Programmieren von graphischen Ausgaben gibt es in OpenGL vordefinierte Primitive, also simple geometrische Objekte wie Punkt, Linie oder Polygon, mithilfe derer komplexere Objekte zusammengesetzt werden können. Die Definition der zu rendernden Objekte folgt einer bestimmten Struktur. Die Befehle `glBegin()` und `glEnd()` umschließen die durch `glVertex()` gegebenen Eckpunkte eines Primitivobjekts (Eck 2018).
Die Definition eines Primitivobjekts hat folgenden grundsätzlichen Aufbau:

```
glBegin(GLenum mode);
     glVertex();
     ...
     glVertex();
glEnd();
```

Für die Methode `glVertex()` gibt es mehrere Umsetzungen, abhängig vom Datentyp für short, int, float, double mit jeweils zwei, drei oder vier Dimensionen. Eine vierte Dimension wird in der Computergrafik häufig genutzt, um im dreidimensionalen Raum homogene Koordinaten zu erstellen, mit denen Translationen durch lineare Abbildungen und damit durch Matrizen berechnet werden (Shreiner et al. 2013).

Durch das `GLenum mode` wird festgelegt, auf welche Art die Punkte beziehungsweise Vertices miteinander verbunden werden. Mögliche Argumente für mode sind:

Tab. 1: Übersicht über Parameter der Methode glBegin

Parameter	Bedeutung
GL_POINTS	Jeder Vertex wird als einzelner Punkt betrachtet; ohne Verbindungslinie
GL_LINES	Je zwei nachfolgende Punkte bilden ein Paar und werden durch Linien verbunden; es entstehen mehrere kurze Linien
GL_LINE_STRIP	Vom ersten bis zum letzten Vertex werden alle Vertices, der Reihenfolge nach, durch eine durchgehende Linie verbunden
GL_LINE_LOOP	Vom ersten bis zum letzten Vertex werden alle Vertices durch eine zusammenhängende Linie verbunden und der letzte Vertex direkt mit dem ersten Vertex, sodass eine geschlossene Figur entsteht
GL_TRIANGLE	Verbindet jeweils drei Vertices zu einem Dreieck
GL_QUAD	Verbindet jeweils vier Vertices zu einem Viereck

2.2.4 JOGL

JOGL (Java Bindings for OpenGL) ist eine Bibliothek, die eine Java-Unterstützung für den OpenGL Standard darstellt. Sie ermöglicht den Zugriff auf die meist in der Programmiersprache C implementierten Methoden des Grafikchip, indem in Java programmierte Wrapperklassen zu den entsprechenden nativen Funktionen definiert wurden. Ein Beispiel ist das Objekt `GL gl`, über das Methoden zur graphischen Darstellung aufgerufen werden, ähnlich zu dem Objekt `Graphics g` in Java. Dieses Objekt wird genutzt um Methoden wie `glTranslate()`, `glRotate()` und `glVertex()` aufzurufen. Für die Anwendung mit OpenGL muss JOGL als eine externe Bibliothek heruntergeladen und explizit ins Package einbezogen werden.

2.3 Non-Uniform Rational B-Splines (NURBS)

Non-Uniform Rational B-Splines sind Kurven oder Flächen, die mithilfe von gewichteten Kontrollpunkten erstellt werden und im Bereich der Modellierung und Computergrafik zur Darstellung beliebiger Freiformkurven oder -flächen genutzt werden. Sie bieten den Vorteil flexibel veränderbar zu sein und können trotzdem mathematisch genau definiert werden (Rogers 2001).

2.3.1 Kurven

Die Grundlage einer Kurve bildet eine B-Spline, eine stückweise definierte Polynomialfunktion, die durch *Kontrollpunkte* P_i bestimmt wird. Aus der *Anzahl der Kontrollpunkte n* ergibt sich die Ordnung der Kurve als *n* - 1. Jeder Punkt auf der Kurve wird durch den Einfluss

der einzelnen Kontrollpunkte berechnet. Im Vergleich zu B-Splines werden bei NURBS die Kontrollpunkte zusätzlich mit einer *Gewichtung* w_i versehen. Die Kurve beginnt im ersten und endet im letzten Punkt, jeder Kontrollpunkt dazwischen zieht sozusagen die Kurve abhängig vom Gewicht unterschiedlich stark in Richtung des Kontrollpunktes. Da die Kurve stückweise betrachtet wird, wird für jeden Punkt der Kurve nur eine begrenzte Menge an Kontrollpunkten genutzt. Diese werden durch einen *Knotenvektor* gegeben. Definition der NURBS-Kurve mit *n* Kontrollpunkten P_i und Grad *p* (Piegl & Tiller 1995):

$$\mathbf{C}(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i \mathbf{P}_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i} \qquad a \leq u \leq b \tag{2.2}$$

Diese Definition lässt sich umformen zu:

$$\mathbf{C}(u) = \sum_{i=0}^{n} R_{i,p}(u) \mathbf{P}_i \tag{2.3}$$

$$R_{i,p}(u) = \frac{N_{i,p}(u) w_i}{\sum_{j=0}^{n} N_{j,p}(u) w_j} \tag{2.4}$$

Die Formeln (2.2), (2.3) und (2.4) basieren auf der *Basisfunktion* $N_{i,p}$ beziehungsweise der rationalen Basisfunktion $R_{i,p}$, dem Gewicht w_i und den Koordinaten eines Kontrollpunktes P_i. Die Basisfunktion $N_{i,p}$ bestimmt den Einfluss, den ein Kontrollpunkt auf die Kurve hat, und ist rekursiv definiert. Die Basisfunktion für eine Kurve mit Grad *p* am Kontrollpunkt *i* wird durch die folgenden Formeln beschrieben:

$$\begin{aligned} N_{i,0}(u) &= \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ N_{i,p}(u) &= \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \end{aligned} \tag{2.5}$$

Diese Basisfunktion, Formel (2.5), wird mithilfe der Werte des *Knotenvektors U* berechnet. Der Knotenvektor besteht aus Einträgen u_i, für die gilt: $0 \leq u_i \leq 1$ und $u_i \leq u_{i+1}$. Dieser hat folgende Form, mit *a* gleich null und *b* gleich eins:

$$U = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}\}$$

Anhand eines Beispiels von Piegl und Tiller (1995) kann der Einfluss der Gewichtung eines Punktes auf den Verlauf der Kurve und der Basisfunktionen verdeutlicht werden (Abb. 7). Die dargestellten NURBS-Kurven mit den Kontrollpunkte P_0 bis P_6 unterscheiden sich in der Gewichtung w_3 des Punktes P_3, alle anderen Punkte haben jeweils Gewicht gleich eins.

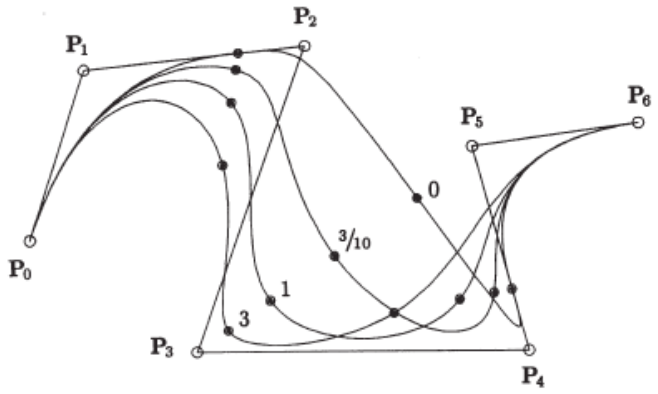

Abb. 7: NURBS-Kurven abhängig von w_3 (Piegl & Tiller 1995, S. 120)

Abhängig von der Gewichtung ändern sich die Basisfunktionen. Die drei Diagramme (Abb.8) zeigen die rationalen Basisfunktionen $R_{i,p}$ nach Definition (2.4) zu den oben gezeigten NURBS-Kurven (Abb. 7) mit Gewicht $w_3 = 1$ (a), $w_3 = 3/10$ (b) und $w_3 = 0$ (c) in Abhängigkeit von u.

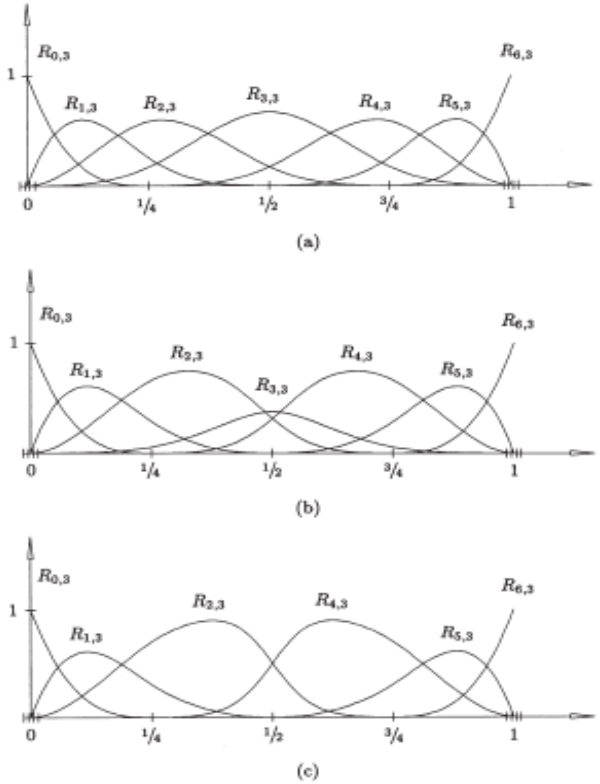

Abb. 8: Basisfunktionen zu NURBS-Kurven abhängig von w_3 (Piegl & Tiller 1995, S.121)

Der Wertebereich der rationalen Basisfunktionen umfasst Werte im Bereich von null bis eins. Bei einem Wert gleich null hat dieser Punkt keinen Einfluss auf die Kurve an der Stelle u, entsprechend hängt bei einem Wert von eins die Kurve nur von diesem Punkt ab. Die obere

Grafik (Abb. 8 (a)) zeigt die Basisfunktionen bei einem Gewicht von eins für jeden Punkt, nur die Basisfunktionen $R_{0,3}$, $R_{6,3}$ der Randpunkte P_0 und P_6 haben am Anfang beziehungsweise Ende einen auffallend hohen Einfluss. Die Diagramme zeigen deutlich, wie der Einfluss der Basisfunktion $R_{3,3}$ abhängig vom Gewicht w_3 abnimmt und der Einfluss der benachbarten Punkte, gegeben durch $R_{2,3}$ und $R_{4,3}$, steigt. Die Basisfunktionen $R_{0,3}$, $R_{1,3}$, $R_{5,3}$ und $R_{6,3}$ der am Rand liegenden Punkte P_0, P_1, P_5 und P_6 werden dadurch nicht beeinflusst.

2.3.2 Flächen

Die Definition und Berechnung einer NURBS-Fläche entsprechen weitgehend denen der Kurve. Ausgehend von gewichteten Kontrollpunkten werden stückweise rationale Funktionen definiert. Bei einer NURBS-Fläche wird durch zwei *Richtungsvektoren u* und *v* ein zweidimensionales Kontrollnetz aufgespannt. Auch die Gewichtung $w_{i,j}$ ist zweidimensional.
Eine NURBS-Fläche mit Kontrollpunkten $P_{i,j}$ und Gewichtung $w_{i,j}$ ist wie folgt definiert (Piegl & Tiller 1995):

$$\mathbf{S}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}\mathbf{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}} \qquad 0 \le u,v \le 1 \tag{2.6}$$

Die Definitionen der beiden Knotenvektoren sind:

$$U = \{\underbrace{0,\ldots,0}_{p+1}, u_{p+1},\ldots,u_{r-p-1}, \underbrace{1,\ldots,1}_{p+1}\}$$

$$V = \{\underbrace{0,\ldots,0}_{q+1}, v_{q+1},\ldots,v_{s-q-1}, \underbrace{1,\ldots,1}_{q+1}\}$$

Nach diesen Definitionen lässt sich aus den Koordinaten der Kontrollpunkte ein Kontrollnetz aufspannen (Abb.9), aus welchem sich mithilfe des Knotenvektors und der Gewichtung eine Fläche erzeugen lässt.

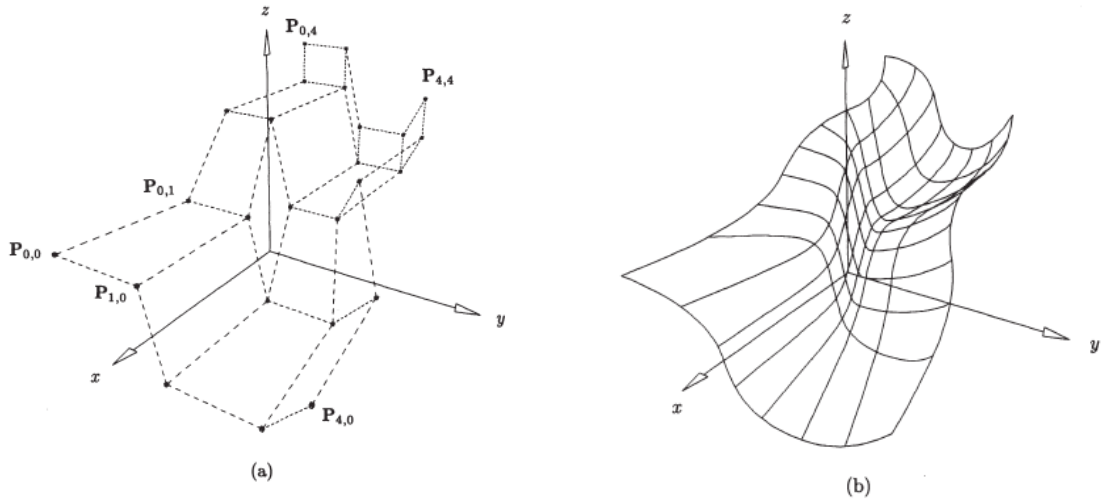

Abb. 9: NURBS-Fläche gegeben durch Kontrollpunkte (a) und als Fläche (b) (Piegl &Tiller 1995, S. 129)

2.4 Interaktive Geometrie-Modellierung

Die Anwendungsgebiete von Computergrafik sind sehr unterschiedlich, zum Beispiel wird sie genutzt im Bereich von Filmen, Videospielen und Animation mithilfe von Software wie Blender, Maya oder Cinema4D. Ein anderer Anwendungsbereich für Computergrafik ist in der Entwicklung von Produkten wie Autos, Flugzeugen oder Bauwerken. Dafür wird CAD-Software verwendet wie AutoCAD. Aber auch bei der Modellierung von Pflanzen durch Software wie beispielsweise LStudio, OpenAlea oder AMAPStudio kommt Computergrafik zum Einsatz. Bei der Darstellung von Computergrafiken bilden Freiform-Kurven und -Flächen die Grundlage vieler Modelle. Die Objekte können typischerweise durch geometrische Transformationen spezifiziert und editiert und in einer dreidimensionalen Ansicht dargestellt werden.

3. Konzept

Die bestimmenden Teile des Konzeptes sind die verbindende Schnittstelle zur Software GroIMP, die Unterscheidung zwischen der Darstellung von zwei- und dreidimensionalen Objekten und das MVC-Architekturmodell zur Entwicklung der Anwendung.

3.1 Schnittstelle zu GroIMP

GroIMP bietet durch die Plug-in Struktur eine vordefinierte Schnittstelle, die leicht genutzt werden kann. Dabei muss berücksichtigt werden, dass, wie in der Arbeit von Leppich (2017) dargestellt, die Anforderungen an die Ordnerstruktur und die Konfigurationsdateien eingehalten werden. Es sind drei Konfigurationsdateien notwendig: build.xml, plugin.xml und plugin.properties. Die Datei build.xml ist erforderlich, damit das Plug-in von dem von GroIMP genutzten Build-Management-Werkzeug Ant einbezogen wird. Es wird definiert, an welcher Stelle der Benutzeroberfläche von GoIMP das Plug-in aufgelistet ist und von wo aus es aufgerufen wird. Die plugin.xml-Datei legt fest, welche weiteren Packages für das Plug-in benötigt werden, und es ist eine Datei plugin.properties erforderlich, die die Namen der Menüeinträge festlegt. Die Klasse NURBSEditor dient als Verbindungsstelle zu GroIMP.

3.2 Plug-in Struktur

Die Darstellung von Objekten in 2D und 3D wird separat betrachtet und ist in zwei Packages getrennt, da sich die Berechnungen bei unterschiedlichen Dimensionen und Objekten nicht verallgemeinern lassen. Trotzdem sind für beide Editoren die gleichen Funktionen gegeben.
Die Struktur innerhalb eines Packages lässt sich als MVC-Architekturmodell beschreiben, bestehend aus den drei voneinander abhängigen Komponenten Model, View und Controller (Abb. 10).

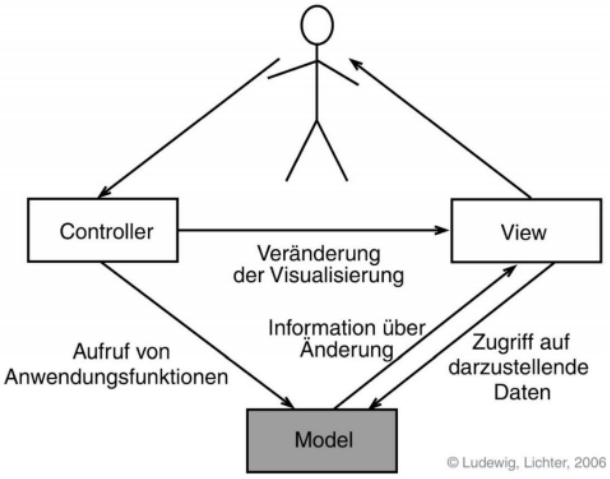

Abb. 10: MVC-Architekturmodell (Ludewig & Lichter 2006)

Die Struktur teilt sich auf in drei Programmteile, von denen einer, der View, sich ausschließlich mit der Darstellung und dem Einlesen der Daten befasst, einem Model, das die Programmlogik oder Datenbank enthält, und einem Controller, der zwischen Model und View vermittelt.
Das Architekturmodell soll das Projekt inhaltlich übersichtlich machen und eine Kapselung unterstützen. Entsprechend sind die MVC-Komponenten in unterschiedliche Java-Klassen aufgeteilt (Tab. 2).

Tab. 2: Aufteilung der Java-Klassen des Plug-ins in Model, View und Controller

Model	View	Controller
• ObjectGeometry2D/3D	• NURBSDisplay2D/3D	• NURBSEditor • Curve2D/3DPanel • ActionListener • MouseListener • MouseWheelListener

4. Implementierung

Die Implementation des Plug-ins erfolgt in Java und nutzt Methoden der API OpenGL durch die externe Bibliothek JOGL der Version 1.1. Diese Version wird bereits in GroIMP im Package IMP-3D zur Darstellung dreidimensionaler Szenen verwendet.

4.1 Grundlagen der Benutzeroberfläche

Um die in JOGL definierten Methoden zum Erzeugen von graphischen Ausgaben zu nutzen, ist es notwendig, das Interface GLEventListener zu implementieren. Dieses umfasst die folgenden Methoden:

```
public void init(GLAutoDrawable drawable)
public void display(GLAutoDrawable drawable)
public void displayChanged(GLAutoDrawable drawable)
public void reshape(GLAutoDrawable drawable,int x,
                    int y, int width, int height)
```

Der Parameter `drawable` gibt an, welche Oberfläche zum Rendern genutzt wird. Das Interface GLAutoDrawable wird zum Beispiel von dem Java Objekt GLJPanel implementiert (Eck 2018).
Die Methode `init()` wird üblicherweise nur einmal bei Erstellen der Klasse aufgerufen.
Die Methode `reshape()` wird auch beim Erzeugen des Objektes aufgerufen und jedes Mal, wenn die Größe des GLJPanels verändert wird. In dieser Methode wird definiert, ob die Ansicht orthographisch oder perspektivisch dargstellt wird. Für beide Editoren (zweidimensional und dreidimensional) wird eine orthographische Ansicht gewählt, da sich perspektivische Ansichten nicht für Bildschirmeingaben mittels der Maus eignen.
Die Methode `display()` übernimmt die eigentliche Aufgabe der graphischen Ausgabe. Von dort aus wird in jedem Schritt die gesamte Szene gelöscht und neu gerendert oder es können Methoden zum Rendern aller aktuellen Objekte aufgerufen werden.

Um die Eingaben von der Maus (Maustaste, Mausrad, Drag&Drop) auf dem Bildschirmbereich als Koordinaten auswerten zu können, muss eine Umrechnung von Pixelwerten zu Koordinaten erfolgen. Diese Umrechnung muss abhängig von der Größe des Ausgabefensters, genauer des GLJPanels, und auf Grundlage des späteren Verwendungszweckes stattfinden. Aus den Pixelkoordinaten einer Mauseingabe auf dem Bildschirm, hier (oldX, oldY) genannt, können mithilfe der nachfolgenden Formel die Koordinaten der OpenGL Ansicht berechnet werden (Abb. 11).

```
newX = newLeft + oldX / oldRight * (newRight – newLeft)
newY = newTop + oldY / oldBottom * (newBottom - newTop)
```

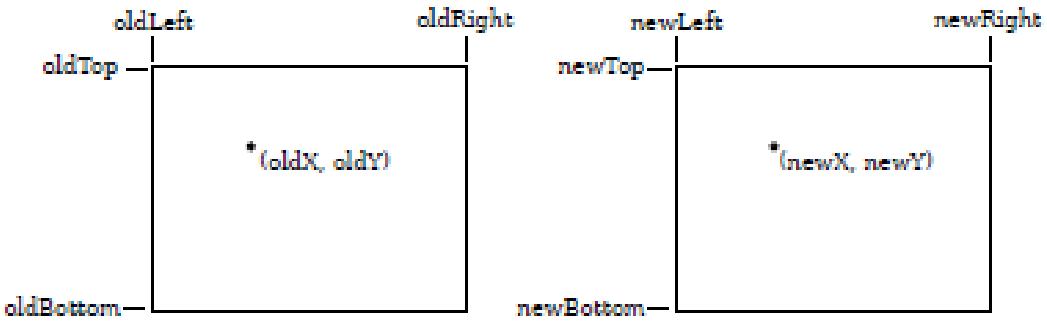

Abb. 11: Umrechnung von Koordinatensystemen (Eck 2018)

4.2 NURBS

Für die Implementation der NURBS-Objekte werden die Klassen `NURBSCurve` und `NURBSSurface` aus GroIMP verwendet. Um mehrere NURBS-Objekte gleichzeitig in einer Szene zu rendern, wird ein Array von Objekten gespeichert.
Die Objekte `NURBSCurve` und `NURBSSurface` werden mit dem Parameter `BSplineCurve curve` beziehungsweise `BSplineSurface surface` im Konstruktor erstellt, wobei BSplineCurve und BSplineSurface Interfaces sind.
Das Interface `BSplineCurve` wird von mehreren Objekten implementiert, darunter die Klassen `BezierCurve, Circular, Helix` oder `VertexSequence.` Es wird an dieser Stelle die Klasse `BSPlineOfVertices` verwendet, welche die Kontrollpunkte durch eine `VertexList` als Parameter speichert. Eine `VertexList` kann durch die Klasse `VertexListImpl` erstellt werden.

```
BSplineOfVertices(VertexList vertices, int degree,
                boolean periodic, boolean bezier)
VertexListImpl(float[] data, int dimension)
```

Damit die Gewichtung der Kontrollpunkte bei der Berechnung der Kurve beachtet wird, muss das Attribut `rational` der `BSplineOfVertices` auf *true* gesetzt werden und in der `VertexListImpl` wird zu den Koordinaten jedes Kontrollpunktes eine Koordinate hinzugefügt, die die Gewichtung des Punktes darstellt. Dadurch wird auch die Dimension der `VertexListImpl` um eins erhöht. Die Parameter `periodic` und `bezier` werden in diesem Fall durchgehend als *false* gesetzt.

Mögliche Klassen, die das Interface `BSplineSurface` implementieren, sind zum Beispiel die Klassen `BezierSurface`, `SkinnedSurface` oder `BSplineSurfaceImpl`. In dem Plug-in wird ein Objekt der Klasse `BSplineSurfaceImpl` verwendet, von dem mithilfe der Methode `create()` eine darstellbare NURBS-Fläche erzeugt wird.

```
BSplineSurfaceImpl()
create(float[] controlPoints, int uCount, int
          dimension, int uDegree, boolean uClamp,
          boolean uPeriodic, int vDegree, boolean
          vClamp, boolean vPeriodic)
```

Die Methode teilt mithilfe der Parameter die Menge der Kontrollpunkte `controlPoints` in zwei Gruppen, entsprechend der Richtungsvektoren *u* und *v*, ein, aus denen ein Kontrollnetz erzeugt wird. Die Parameter `uCount`, `uDegree`, `uClamp` und `uPeriodic` beziehen sich auf den ersten Richtungsvektor *u*, die anderen Parameter auf den zweiten Vektor *v*. Die Werte für `controlPoints, uCount, dimension` und `uDegree` ergeben sich durch die graphische Eingabe der NURBS-Fläche und die Paramter `uPeriodic` und `vPeriodic` werden auf *false gesetzt.* Die Parameter `uClamp` und `vClamp` werden auf

true gesetzt, damit der äußere Rand der erzeugten NURBS-Fläche an den Kontrollpunkten abschließt (Abb. 12 links).

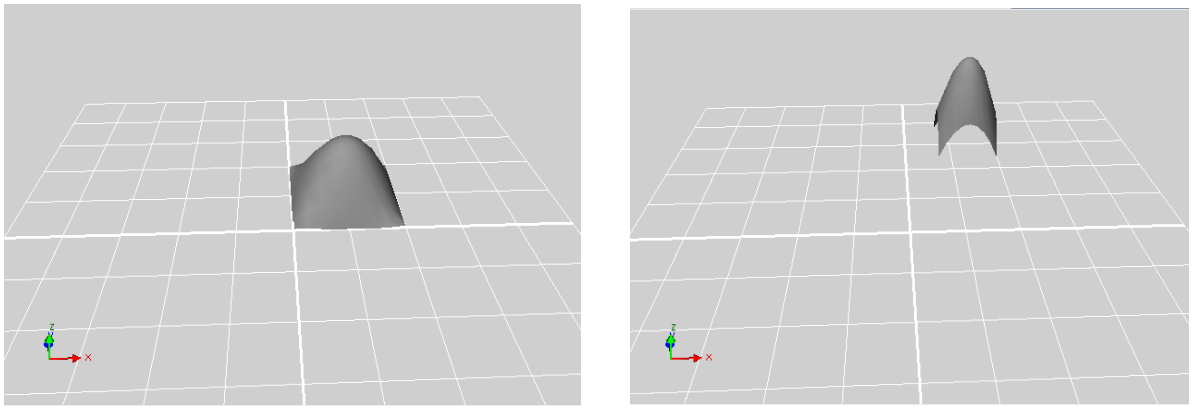

Abb. 12: NURBS-Fläche mit uClamp, vClamp gleich true (links) und uClamp, vClamp gleich false (rechts)

4.3 Manipulationsmöglichkeiten

Obwohl der OpenGL-Standard Methoden zum Verschieben, Rotieren und Skalieren graphischer Objekte definiert, können diese im Plug-in nicht ohne Probleme genutzt werden, da diese Methoden nur die graphische Ausgabe verändern würden. Sie speichern aber nicht die neuen, aus einer Transformation entstehenden Koordinaten der Kontrollpunkte. Dadurch würden erstellte Szenen nicht nach GroIMP exportiert werden können und die Eingaben mithilfe der Maus ließen sich nicht graphischen Objekten zuordnen. Daher werden diese Operationen eigenständig implementiert und sollen nochmal einzeln betrachtet werden.
Das Plug-in ermöglicht folgende Funktionalität zum Verändern von Kurven und Flächen in GroIMP: Verschieben, Skalieren, Rotieren und Kopieren ganzer Objekte, sowie das Verschieben und Gewichten einzelner Kontrollpunkte.
Für die Kurven und Flächen wird zusätzlich der Mittelpunkt des Objekts durch den geometrischen Schwerpunkt berechnet und gerendert, um die Eingabe von Transformationen, die alle Kontrollpunkte des Objekts gleich beeinflussen, steuern zu können.

4.3.1 Verschieben

Zum Verschieben eines NURBS-Objektes oder eines einzelnen Kontrollpunktes wird mit der Maus der Mittelpunkt oder ein einzelner Kontrollpunkt verschoben. Die Klasse `Mouselistener` wird aufgerufen; es wird die Distanz und Richtung zwischen dem Anfangs- und Endpunkt der Verschiebung berechnet. Dieser Vektor wird zu allen Kontrollpunkten der Kurve oder Fläche addiert, die neuen Koordinaten werden gespeichert und zum weiteren Rendern der Szene genutzt.

4.3.2 Rotieren

Das Rotieren eines Objektes basiert darauf, dass alle Kontrollpunkte des Objektes um einen gewählten Winkel um den Mittelpunkt gedreht werden. Dieser Winkel wird im Editor abhängig vom Mittelpunkt der Kurve oder Fläche berechnet. Für das Rotieren eines Objekts wird zusätzlich zusätzlich zum Mauszeiger das Mausrad genutzt. Wenn der Mauszeiger auf

dem Mittelpunkt eines Objektes zeigt, kann durch Drehen des Mausrades dieses Objekt mit oder gegen den Uhrzeigersinn rotiert werden.

4.3.3 Skalieren

Zum Skalieren eines NURBS-Objekts wird ebenfalls das Mausrad verwendet. Wenn sich der Mauszeiger über dem Mittelpunkt des Objekts befindet, kann durch gleichzeitiges Drücken der linken Maustaste und Drehen des Mausrades die Kurve oder Fläche vergrößert beziehungsweise verkleinert werden. Zum Skalieren wird für jeden Kontrollpunkt der Vektor zum Objektmittelpunkt berechnet und dieser um einen Faktor, abhängig von der Mausradeingabe, verlängert oder verkürzt. Daraus ergeben sich neue Kontrollpunkte, über die das Objekt neu berechnet wird. Der Mittelpunkt der Kurve oder Fläche bleibt dabei unverändert.

4.3.4 Kopieren

Das Kopieren eines Objektes wird mithilfe eines Buttons im oberen Bereich des Fensters realisiert. Durch Klicken des Buttons wird eine neue Kurve oder Fläche hinzugefügt, die die um eine Einheit in x-Richtung beziehungsweise y-Richtung verschobenen Koordinaten des alten Objekts besitzt. Auch die anderen Parameter wie die Gewichtung der Kontrollpunkte oder der uCount bei Flächen werden identisch übernommen.

4.3.5 Löschen

Auch das Löschen von Objekten wird durch einen Button im oberen Bereich des Plug-in Fensters durchgeführt. Abhängig von dem ausgewählten RadioButton, wird das Objekt gelöscht, was als letztes erstellt wurde. Beim Löschen werden die Koordinaten der Kontrollpunkte auf *null* gesetzt und damit entfernt. Dadurch wird sichergestellt, dass es durch Überschreiben von alten Werten nicht zu Fehlern kommen kann und neue Koordinaten problemlos initialisiert werden können.

4.3.6 Exportieren

Auch für das Exportieren der Szene befindet sich ein Button im oberen Bereich des Fensters. Beim Exportieren wird eine RGG-Datei erstellt, die aus zum Teil vordefiniertem XL-Code besteht und durch die Parameter der aktuellen Objekte ergänzt wird, um eine ausführbare Datei zu erstellen. Nach dem Erstellen der Datei wird diese automatisch im GroIMP Texteditor geöffnet und kann in diesem Format als Projekt bearbeitet und gespeichert werden.

4.4 Primitivobjekte

Als Primitivobjekte lassen sich im zweidimensionalen Fall Rechtecke, Dreiecke und Kreise darstellen. Mit dem dreidimensionalen Editor können Quader, Kegel, Kugeln, Zylinder und Frustuma dargestellt werden. Dazu wird im Plug-in jeweils ein Array vom Typ des entsprechenden Objekts und jeweils ein Array mit Koordinaten eines Stützpunktes

gespeichert. Es werden die GroIMP-eigenen Objekte `Polygon` für Rechteck und Dreieck, `Circle`, `Box`, `Cone`, `Sphere`, `Cylinder` und `Frustum` genutzt.

5. Ergebnisse

Durch die Benutzeroberfläche eines GroIMP-Projektes lässt sich das Plug-in über den Reiter „Panels“ und die Auswahl „NURBSEditor“ aufrufen (Abb. 13). Dort lässt sich auswählen, ob zwei- oder dreidimensionale Objekte erstellt werden.

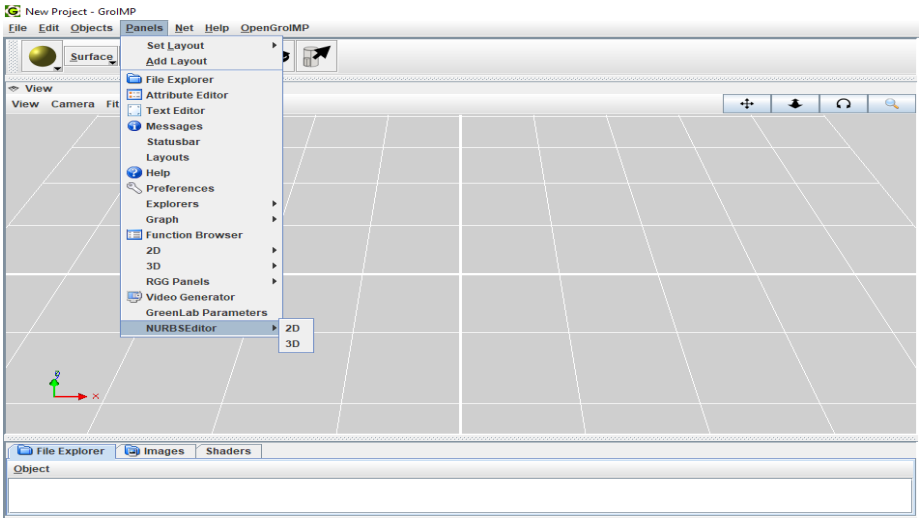

Abb. 13: Ansicht der Benutzeroberfläche

5.1 Der 2D-Editor

Wurde die zweidimensionale Ansicht gewählt, öffnet sich ein neues Fenster im Vordergrund, das im oberen Bereich eine Tableiste mit Buttons besitzt und im unteren Bereich ein Koordinatensystem zeigt (Abb. 14).

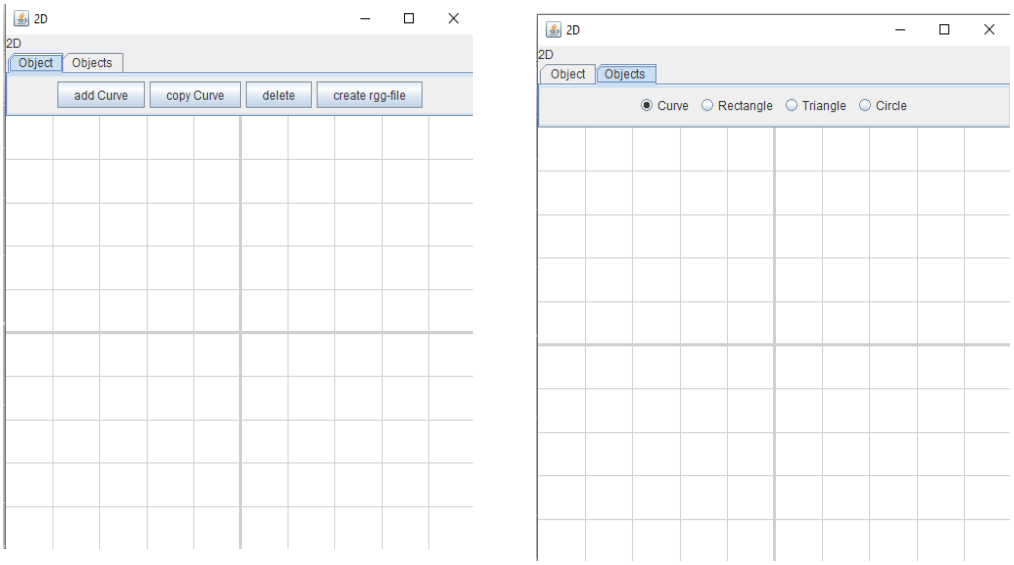

Abb. 14: Benutzeroberfläche des 2D-Editors

Der obere Bereich des Fensters ist in die zwei Tabs *Object* und *Objects* aufgeteilt, die die Möglichkeit bieten Operationen durchzuführen oder andere Objekte zu erzeugen. Über den ersten Tab *Object* können durch Buttons Funktionen aufgerufen werden, wie

- Hinzufügen einer neuen Kurve,
- Kopieren der zuletzt erstellten Kurve,
- Exportieren der gesamte Szene in den GroIMP-Texteditor in Form einer rgg-Datei und
- Löschen der zuletzt erstellten Kurve.

Über den Tab *Objects* lässt sich mithilfe von RadioButtons auswählen, welches Objekt als nächstes hinzugefügt wird. Es besteht eine Auswahl zwischen NURBS-Kurve, Rechteck, Dreieck und Kreis.

Im unteren Bereich des Fensters befindet sich das GLJPanel zum Erstellen der Objekte mit einem zweidimensionalen Koordinatensystem als Hintergrund. Durch Klicken der linken Maustaste lassen sich Primitivobjekte oder Kontrollpunkte einer NURBS-Kurve hinzufügen. Welches Objekt hinzugefügt wird, hängt von der Auswahl in dem Tab *Objects* ab. Beim Erstellen einer NURBS-Kurve werden alle Mauseingaben der Reihe nach als Kontrollpunkte markiert und zu einer durchgehenden NURBS-Kurve verbunden. Es lassen sich mehrere Kurven in einer Szene modellartig anordnen (Abb. 15) oder unterschiedlichste Formen erstellen. Einige Beispiele für Kurven, die durch den 2D-Editor erstellt wurden, beinhalten Abb. 15 und Abb. 16.

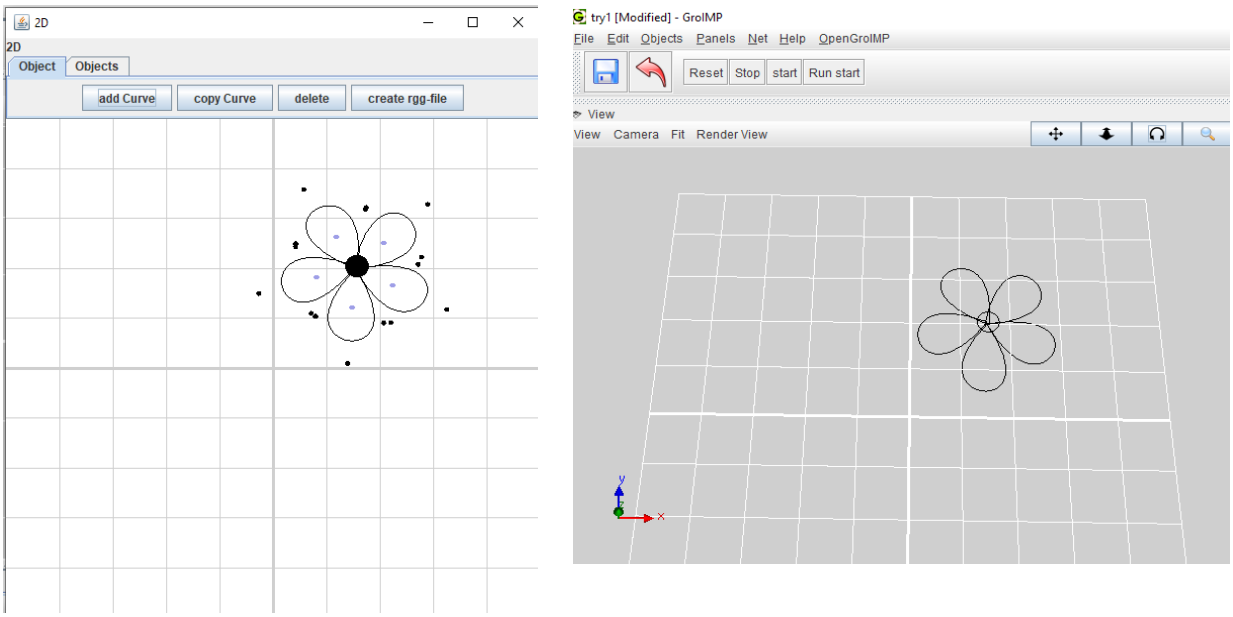

Abb. 15: zweidimensionales Blütenmodel im Editor und als rgg-Datei erzeugt

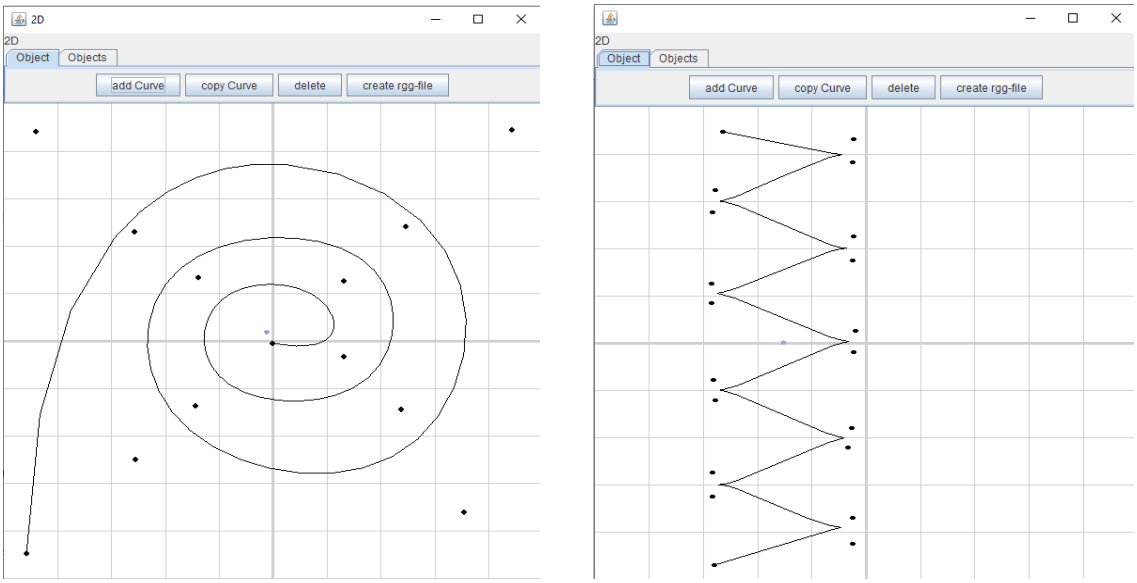

Abb. 16: Beispiele für NURBS-Kurven

5.2 Der 3D-Editor

Für den dreidimensionalen Editor gibt es im oberen Bereich des Fensters die gleiche Aufteilung in Tabs wie bei dem zweidimensionalen Editor (Abb. 17). Der Tab *Object* bietet, neben den bekannten Auswahlmöglichkeiten für NURBS-Kurven, zusätzlich auch die gleichen Funktionen für NURBS-Flächen. Zu diesen Funktionen zählen das Hinzufügen, Kopieren und Löschen einer Fläche. Unter dem Tab *Objects* gibt es zusätzlich zu den Objekten NURBS-Kurve und -Fläche die Primitivobjekte. Diese werden durch Klicken auf den Szenenbereich eingefügt. Sie werden in einem ersten Schritt nur mit Standardparametern initialisiert. Die NURBS-Flächen können, genauso wie die NURBS-Kurven, durch Eingaben der Maus rotiert, skaliert und verschoben werden.

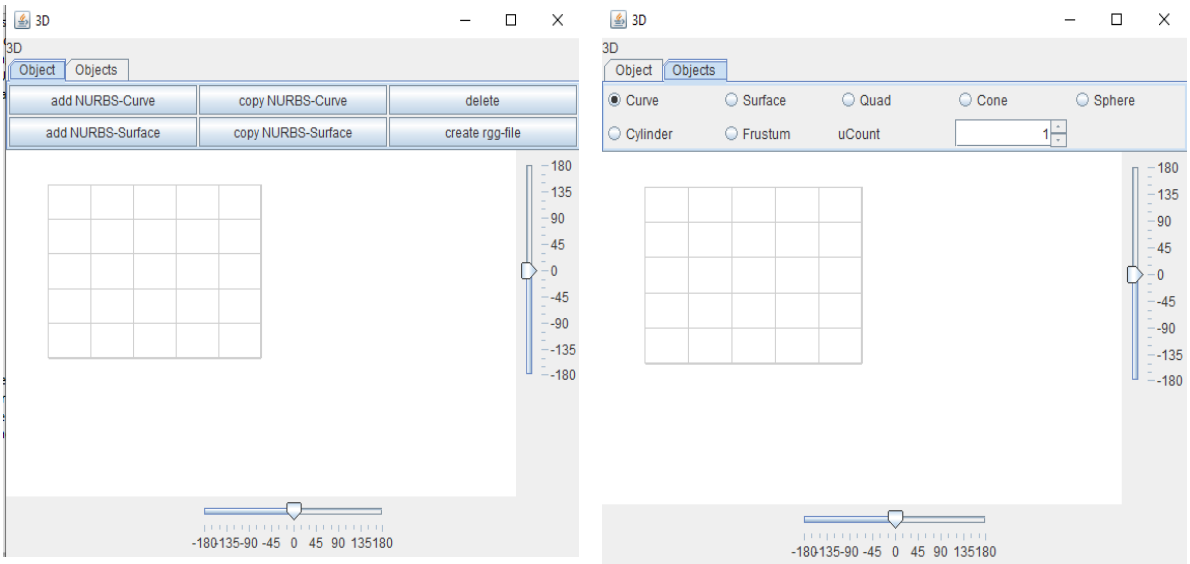

Abb. 17: Benutzeroberfläche des 3D-Editors

Die Ausrichtung des dreidimensionalen Koordinatensystems lässt sich mithilfe von Slidern verändern. Durch den Slider unterhalb des GLJPanels lässt sich das Koordinatensystem um die y-Achse rotieren und der Slider rechts neben dem Panel dreht die Szene um die z-Achse. Eine Eingabe über die Maus ist nur möglich, wenn die Slider auf 0, 90, -90, 180 oder -180 stehen und damit eine Ebene des Koordinatensystems parallel zur Bildschirmoberfläche ist. Bei der Eingabe eines neuen Kontrollpunktes wird eine Koordinate abhängig von der Ansicht auf null gesetzt, sodass der Punkt auf der Ebene liegt, die parallel zur Bildschirmoberfläche ist.
Zum Erstellen von NURBS-Flächen ist es notwendig den Parameter uCount anzugeben. Dafür ist unter dem Tab *Objects* ein Eingabefeld. Der Wert von `uCount` bestimmt die Anzahl der Knoten, die zu dem ersten der beiden Vektoren gehören. Die Menge der Kontrollpunkte des anderen Vektors und der Grad der Vektoren wird abhängig von `uCount` automatisch berechnet. Nicht für jeden möglichen Eingabewert `uCount` lässt sich eine NURBS-Fläche berechnen. Einige Beispiele für Flächen, die durch den 3D-Editor erstellt wurden, beinhalten Abb. 18 und Abb. 19.

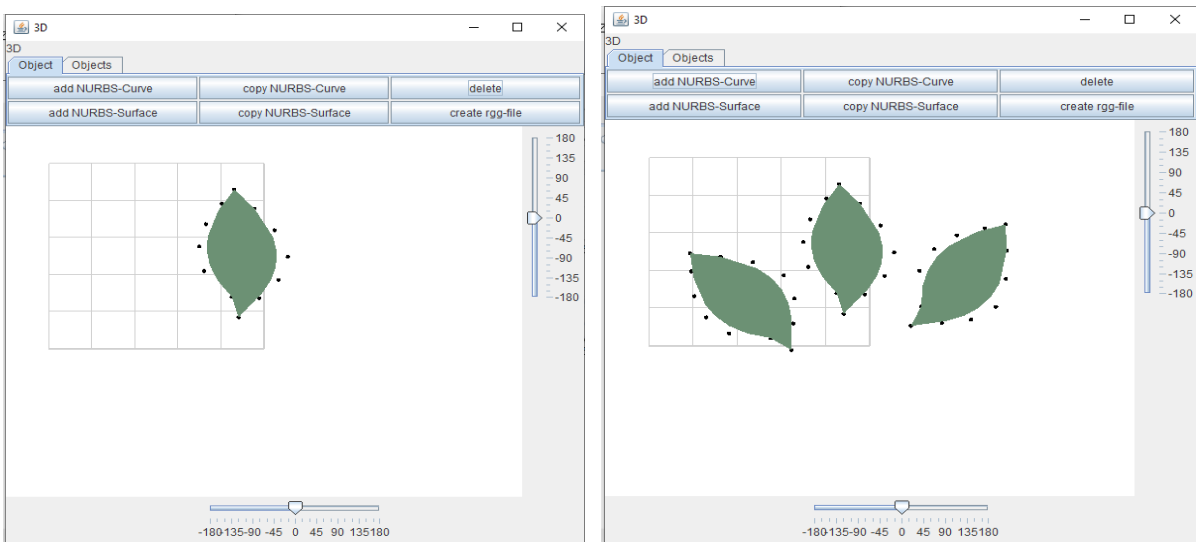

Abb. 18: Beispiel einer einzelnen Fläche und zwei rotiert und verschobenen Klonen

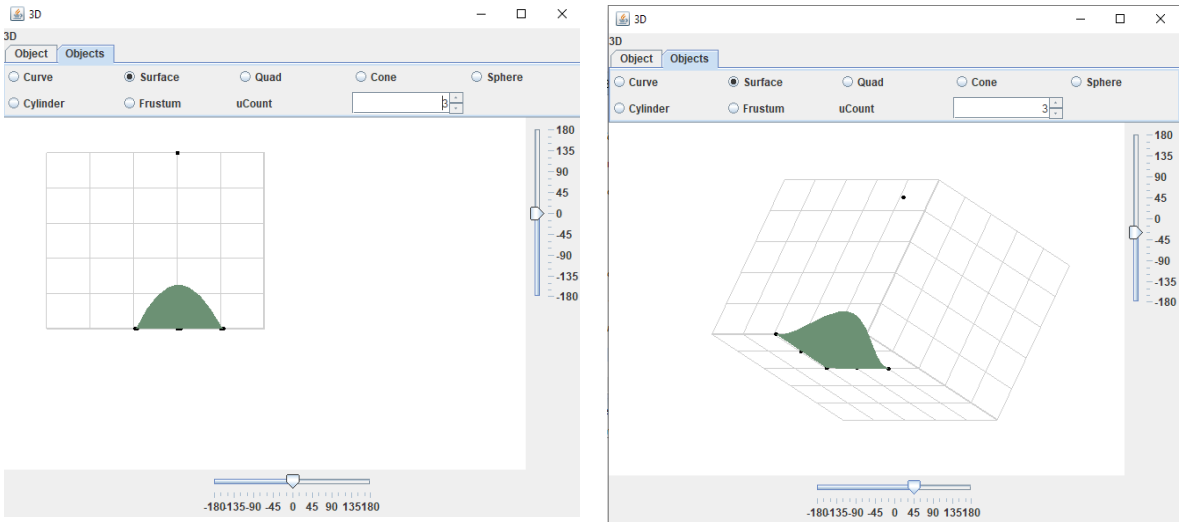

Abb. 19: Beispiel für eine dreidimensionale Fläche

5.3 Anwendungsbeispiel

Anhand der beiden Beispiele (Abb. 15 und Abb. 19) soll ein Konstruktionsablauf schrittweise gezeigt werden.
Wurde, wie in Abbildung 13 gezeigt, der zweidimensionale NURBS-Editor aufgerufen, lässt sich durch Klicken der linken Maustaste in dem Koordinatenraster ein Kontrollpunkt einer NURBS-Kurve hinzufügen (Abb. 20). Jeder neu hinzugefügte Kontrollpunkt gehört automatisch zum aktuellen Objekt. Mithilfe des Buttons „add Curve“ kann eine neue Kurve angefangen werden.

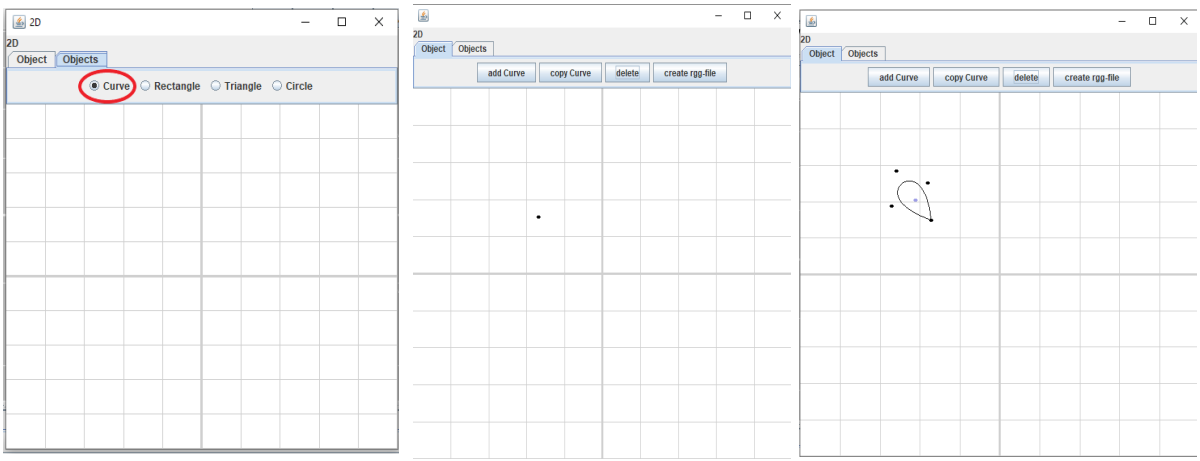

Abb. 20: Hinzufügen von Kontrollpunkten bis Ellipsenform entsteht

Die Kontrollpunkte (schwarz) lassen sich durch Drag&Drop verschieben und mithilfe des Mausrads kann die Gewichtung eines Kontrollpunktes verändert werden. Der Mittelpunkt der Kurve (violett) kann genutzt werden, um die gesamte Kurve zu verschieben, rotieren oder skalieren.
Zum Kopieren der zuletzt hinzugefügten Kurve wird auf den Button „copy Curve“ geklickt und die Kopie wird leicht verschoben eingefügt (Abb 21).

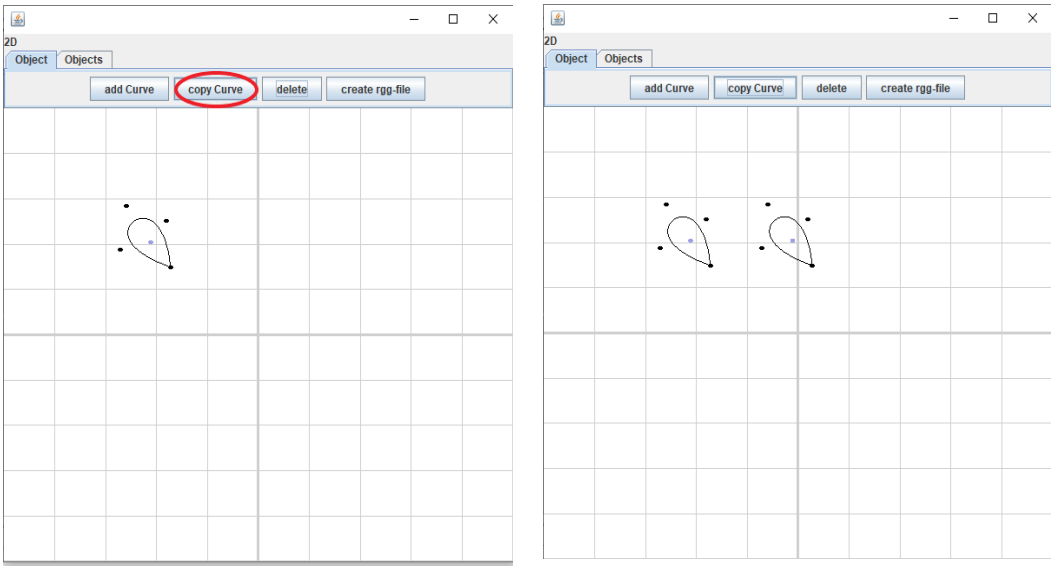

Abb. 21: Kopieren einer Kurve

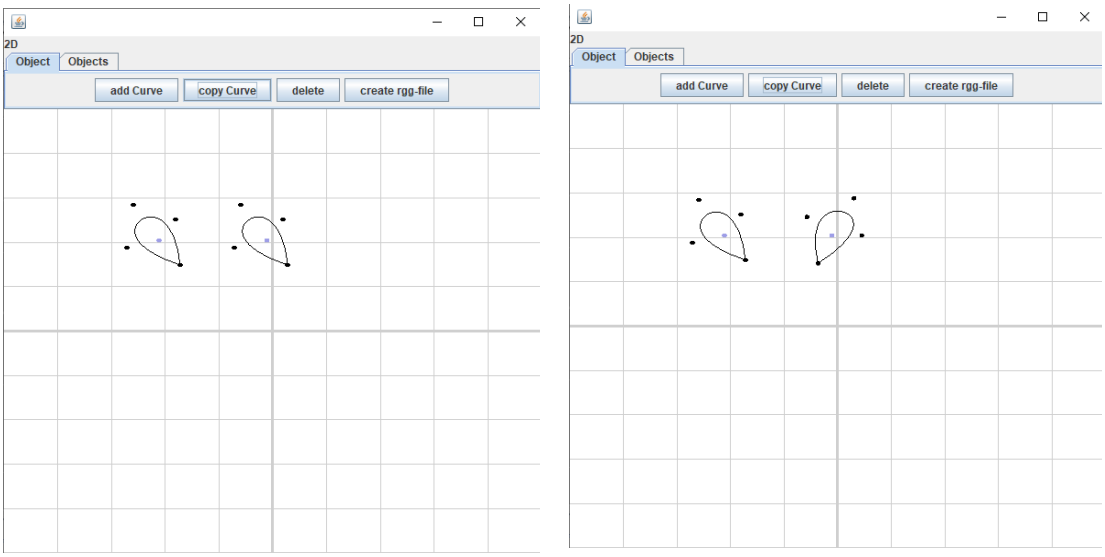

Abb. 22: Rotieren der Kurve

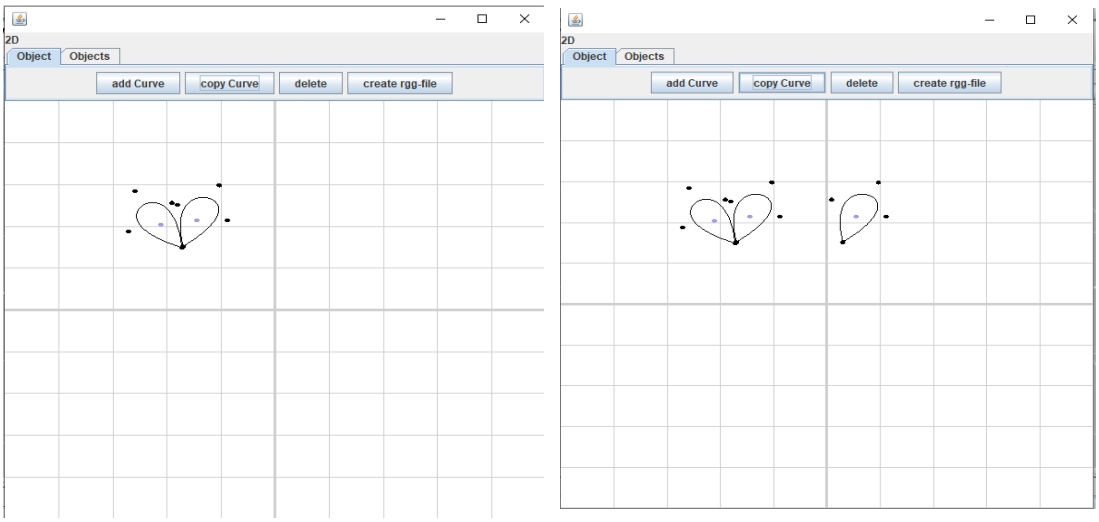

Abb. 23: Kopieren und Rotieren

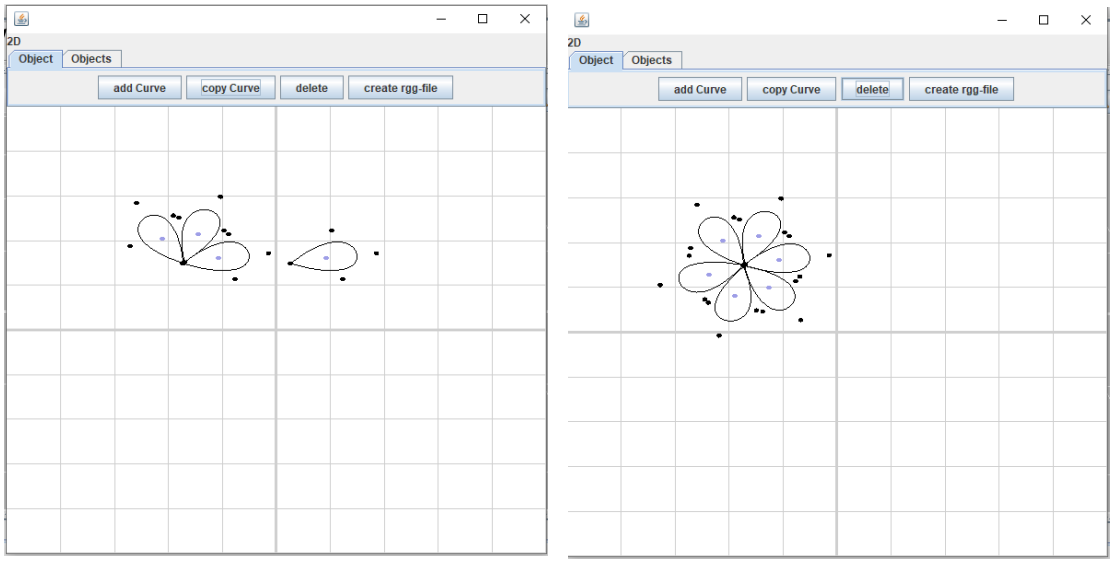

Abb. 24: Wiederholen von Kopieren, Rotieren und Verschieben

Soll ein anderes Objekt als eine Kurve erzeugt werden, kann in dem Tab *Objects* der entsprechende RadioButton ausgewählt werden. Die Primitivobjekte Rechteck und Dreieck werden durch ihre Eckpunkte, ein Kreis durch den Mittelpunkt und einen Punkt als Radius definiert. Die Eckpunkte werden nicht explizit gerendert und ein Objekt erscheint erst nachdem alle Eckpunkte eingegeben wurden. So muss also, um ein Rechteck zu erzeugen, der Button „Rectangle“ ausgewählt werden und an vier Stellen auf dem Bildschirm geklickt werden (Abb. 25). Diese vier Punkte bestimmen dann das Rechteck.

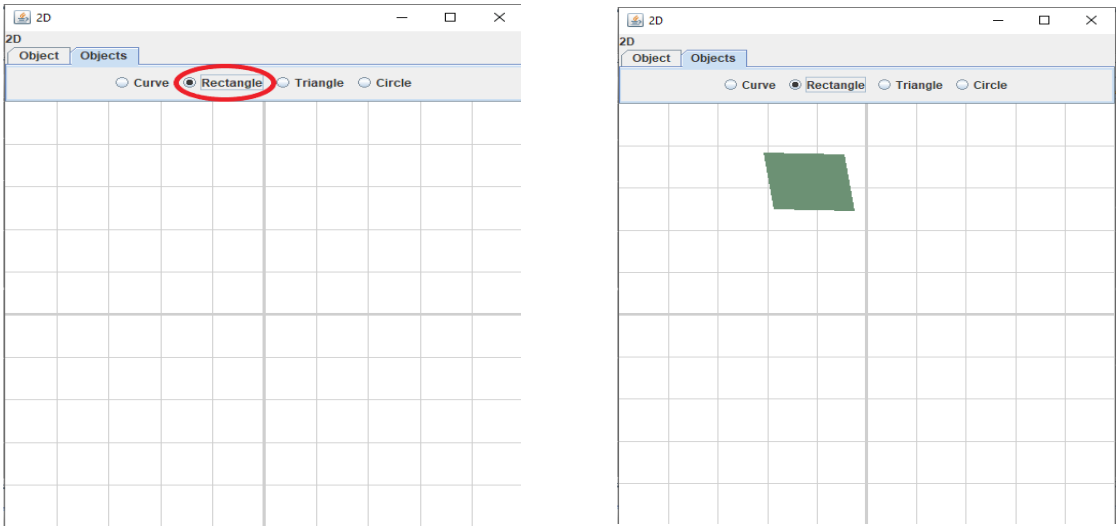

Abb. 25: Erstellen eines Rechtecks

Wenn die dreidimensionale Ansicht gewählt wird, lassen sich neben Kurven und Primitivobjekten auch NURBS-Flächen erstellen. Wie bei dem zweidimensionalen Editor wird über dan Tab „*Objects*“ Die Auswahl des Objekts getroffen, in diesem Fall NURBSSurface. Da der Großteil der Kontrollpunkte auf der x-z-Ebene liegen soll, wird die Ansicht für die Eingabe um -90 Grad um die z-Achse rotiert (Abb. 26).

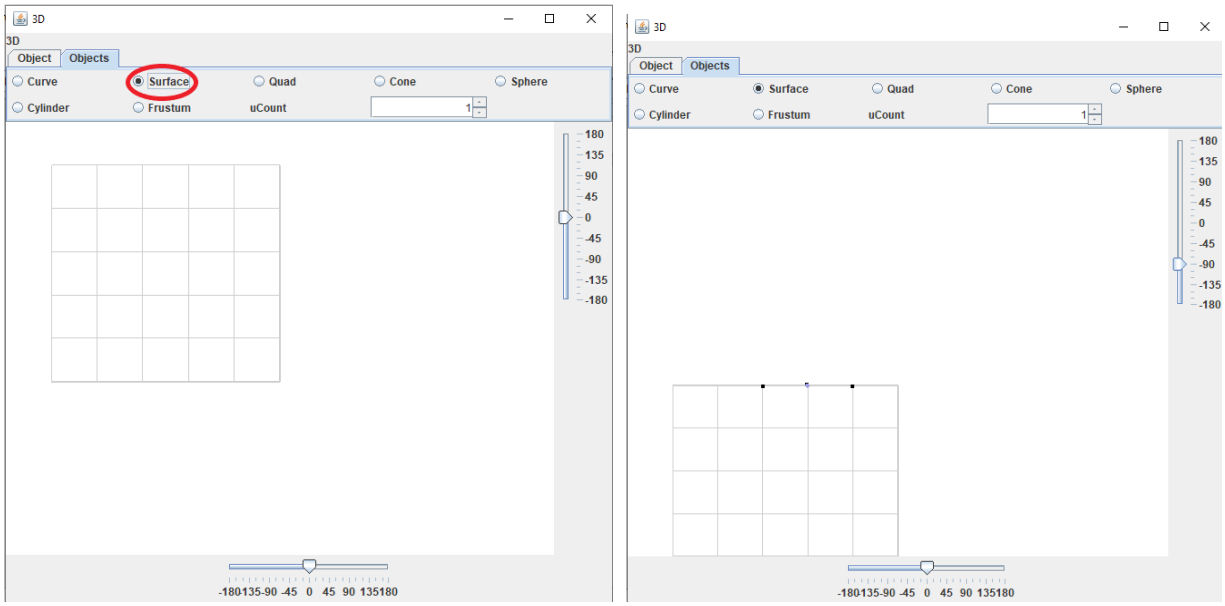

Abb. 26: Auswahl der NURBSSurface und Rotation der Ansicht

Bei der Eingabe der Kontrollpunkte einer NURBSSurface ist darauf zu achten, dass die Punkte in der Reihenfolge entsprechend der Richtungsvektoren eingegeben werden. Abhängig von der Verteilung der Punkte kann es sinnvoll sein, Punkte anfangs versetzt zu platzieren, um sie nachträglich leichter manipulieren zu können (Abb. 27).

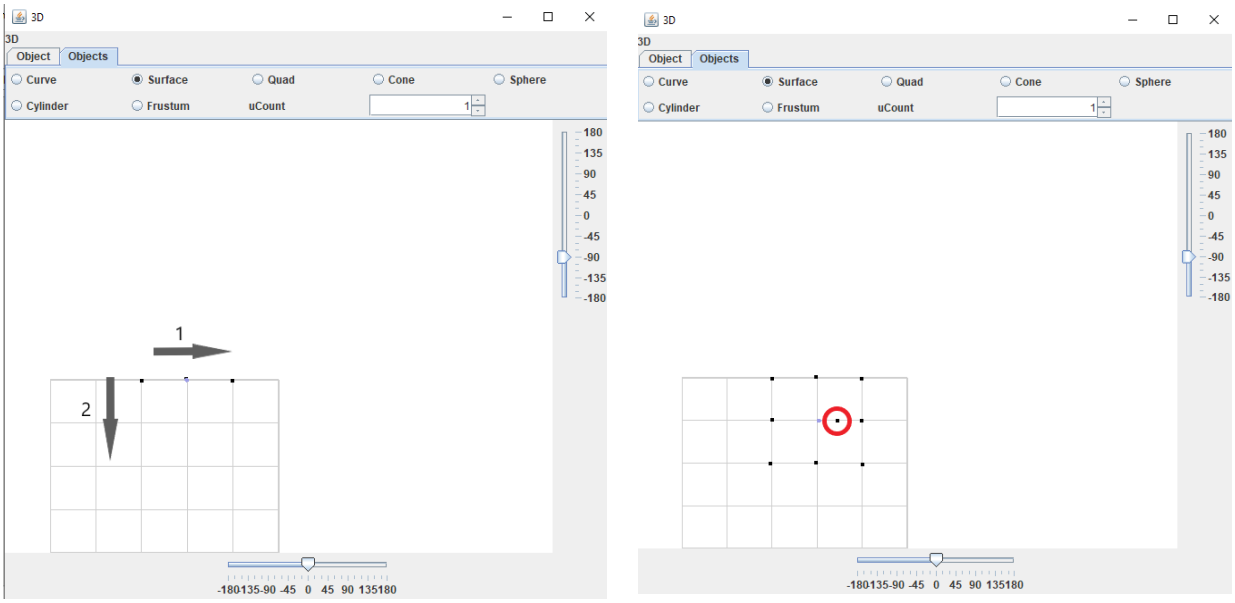

Abb. 27: Definition der Kontrollpunkte

Nach einer erneuten Rotation um die z-Achse können die Punke nun in y-Richtung verschoben werden (Abb. 28).

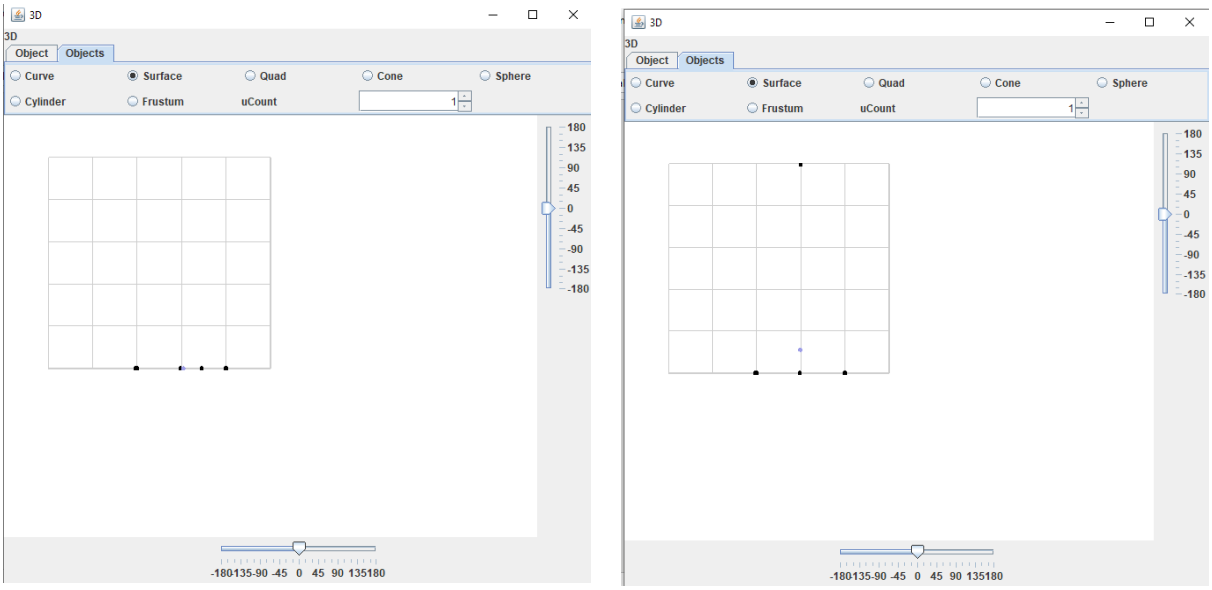

Abb. 28: Rotation der Ansicht und Verschieben eines Punktes

Über das Eingabefeld im oberen Bereich des Fensters kann der Parameter uCount gesetzt werden, um eine NURBSSurface zu erzeugen (Abb. 29).

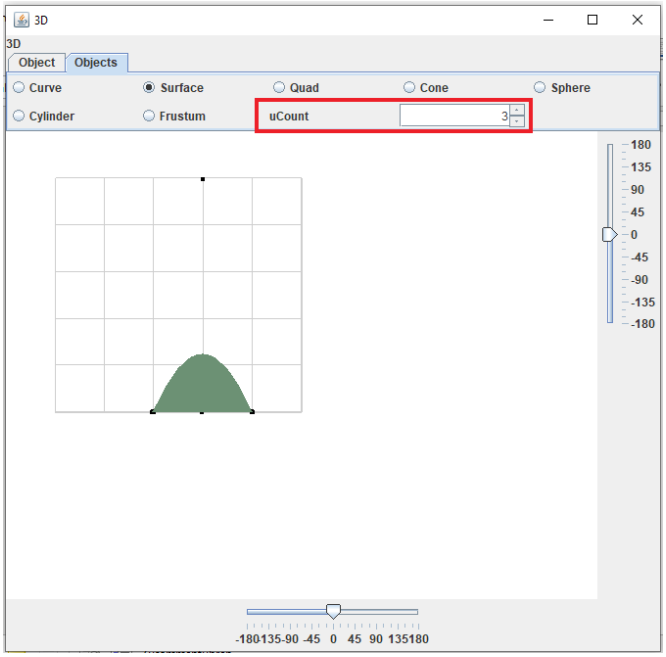

Abb. 29: Wert für uCount eintragen

6. Diskussion und Ausblick

Das Plug-in ergänzt GroIMP um einen graphischen Editor zur Spezifikation und zum Editieren von NURBS-Kurven (zweidimensional und dreidimensional) und -Flächen. Neben dem Erstellen der mathematisch genauen Objekte können diese auch in GroIMP-Szenen genutzt werden, da die Objekte auf in GroIMP definierten Objekten basieren.
Die graphische Darstellung wird durch die Bibliothek JOGL unterstützt, welche die Methoden der Programmierschnittstelle OpenGL in Java zugänglich macht. Durch die Einbindung von JOGL ist es möglich realisitische dreidimensionale Szenen mit einer großen Menge von Objekten in Echtzeit darzustellen. Bisher ist die Anzahl der Objekte auf 50 begrenzt, wobei die Kurven und Flächen selber eine unbestimmt große Anzahl an Knoten haben können.
Für die Nutzer von GroIMP besteht durch das Plug-in erstmals die Möglichkeit, dass vielfältige Objekte in Form von Freiform-Kurven und -Flächen interaktiv entworfen und editiert werden. Dadurch werden Modellierungen verschiedener Pflanzenteile, wie Blätter, Blüten unmittelbar deutlich verbessert. Vorteile des entworfenen Plug-ins sind die Einfachheit der Umsetzung, die Visualisierung des Vorgehens und die Vielfalt der Variationsmöglichkeiten. Analysen und Auswertungen unterschiedlicher Waldstruktur und -zusammensetzung können deutlich präziser und sicherer vorgenommen werden. Die Möglichkeiten, die das Modellierungswerkzeug bietet, sind bei weitem noch nicht ausgeschöpft
Es gab einige Punkte, die während der Implementierung des Plug-ins Schwierigkeiten darstellten. Ein weit verbreitetes Problem bei der Softwareentwicklung und auch bei GroIMP ist, dass es wenig oder zum Teil gar keine Dokumentation zu den Klassen, Methoden oder Attributen gibt. Das erschwert die Arbeit mit den vorhanden Klassen deutlich, da Funktionen nicht nachvollziehbar sind oder Abhängigkeiten und Parameter unklar bleiben. Besonders für ein Plug-in ist es zwingend Kompatibilität zu der bestehenden Software zu haben und daher auch zu den vorhanden Klassen und Methoden.

Ein anderes Hindernis ist die unterschiedliche Ausrichtung des Koordinatensystems. Da die Orientierung der Koordinatensysteme in GroIMP und der in OpenGL definierten Methoden sich unterscheiden, wäre eigentlich eine Rotation von 90 Grad um die x-Achse nötig, um das gleiche Bild zu erhalten, jedoch ändert sich damit nicht die Orientierung bei Primitivobjekten.
Eine größere Problematik bestand darin, die Eingaben auf der zweidimensionalen Oberfläche des Bildschirms zu dreidimensionalen Koordinaten umzuwandeln, um zu prüfen, ob eine Mausaktion ein Objekt der Szene betrifft. Diese wurde gelöst, indem die Möglichkeit der Eingabe abhängig von der Ausrichtung des Koordinatensystems beschränkt wurde. Alternativ wäre es möglich gewesen, ein feinmaschiges dreidimensionales Raster als Hilfslinien in der Szene zu rendern und die Koordinaten der Mauseingabe mit den Achsen des Koordinatensystems zu vergleichen, um bei Punkte an den Stellen im Raum zu erzeugen wo Mausinput und Raster sich schneiden. Man würde also ein zweites Objekt in der Szene nutzen um die dreidimensionalen Koordinaten einer Mauseingabe zu bestimmen. Ein so umfangreiches Koordinatensystem würde jedoch die Ansicht sehr unübersichtlich machen und andere Objekte der Szene verdecken.

Ausblick

Es bleiben noch einige Möglichkeiten, wie die Funktionalität des Plug-ins in Zukunft erweitert werden kann. Ein Punkt, der bisher aus zeitlichen Gründen nicht beachtet werden konnte, ist das Gruppieren von Objekten und die Repräsentation des gruppierten Objekts als ein Knoten auf einer höheren Skalenebene im GroIMP-Szenengraph.
Eine weitere sinnvolle Erweiterungsmöglichkeit ist es, innerhalb des Editors einzelne Aktionen, wie das Verschieben oder Skalieren eines Kontrollpunktes, wieder rückgängig machen zu können und abseits des zuletzt bearbeiteten Objekts weiter auswählen zu können, welches der Objekte in der Szene gelöscht oder kopiert wird. Spezieller könnte man auch innerhalb bestehender Objekte an variabler Stelle Knoten hinzufügen oder löschen, um die Form nachträglich zu manipulieren.
Außerdem ließe sich das Rotieren erweitern, indem Objekte um beliebige Kontrollpunkte rotiert werden könnten und nicht nur um den eigenen Mittelpunkt.
Beim Erzeugen der Objekte können noch mehr Parameter oder Objekteigenschaften von der Benutzereingabe abhängig gesetzt werden. Es ließen sich zum Beispiel für die Oberflächen der Objekte individuelle Farben oder Texturen festlegen. Dies ist besonders nützlich, wenn die erstellten Objekte in GroIMP weiter genutzt werden soll, um den Einfluss von Licht zu simulieren und analysieren.
Zusätzlich könnte man ausnutzen, dass GroIMP mehrere andere Möglichkeiten bietet NURBS-Kurven und NURBS-Flächen zu definieren. Dazu gehört die Darstellung von NURBS-Flächen durch eine zylinder- beziehungsweise röhrenartige Form, wie im Beispiel nurbs.gsz auf grogra.de zu sehen ist.
Ebenfalls gibt es noch andere, denkbare Erweiterungen wie die Option der automatischen Ausrichtung von eingefügten Objekten an einem Kontrollraster oder "Snapping" der Position von Objekten bei Anfügen an den Rand existierender Objekte. Wie bereits in GroIMP möglich, könnten die Objekte in Layer klassifiziert werden, die in der Ansicht einzeln ausgeschaltet werden koennen.

Abbildungsverzeichnis

Literaturverzeichnis

[Eck 2018] Eck, D., (2018): Introduction to Computer Graphics. Version 1.2. Hobart and William Smith Colleges, New York.

[Kniemeyer 2008] Kniemeyer, O., (2008): Design and Implementation of a Graph Grammar Based Language for Functional-Structural Plant Modelling. Dissertation, Technische Universität Cottbus.

[Leppich 2017] Leppich, D. (2017): Entwicklung eines Plugins für die 3D-Modellierungsplatform GroIMP. Forschungsbezogenes Praktikum im Studiengang "Angewandte Informatik", Georg-August-Universität Göttingen.

[Lindenmayer, Prusinkiewicz 1990] Aristid Lindenmayer, Przemyslaw Prusinkiewicz, et al., (1990): The algorithmic beauty of plants. New York: Springer-Verlag.

[Ludewig, Lichter 2006] Ludewig, J., Lichter, H., (2006): Software Engineering – Grundlagen, Menschen, Prozesse, Techniken. dpunkt.verlag Heidelberg .

[Piegl, Tiller 1995] Piegl, L. A., Tiller, W., (1995): The NURBS Book. Monographs in Visual Communications. Berlin Heidelberg New York: Springer-Verlag.

[Shreiner et al. 2013] Shreiner, D., Sellers, G., Kessenich, J., Licea-Kane, B., (2013): OpenGL Programming Guide. The Official Guide to Learning OpenGL. Version 4.3, The Khronos OpenGL ARB Working Group, Addison-Wesley.

[Reiners 2004] Reiners, D., (2004): Herausforderungen an moderne Szenengraphsysteme am Beispiel OpenSG. Informatik-Spektrum: Vol. 27, No. 6. Berlin Heidelberg: Springer-Verlag.(S. 531-541).

[Rogers 2001] Rogers, D. F., (2001): An Introduction to NURBS with Historical Perspective. Morgan Kaufmann Publishers, San Francisco

Verwendete Software:
Modellierungsplattform GroIMP (www.grogra.de)
Programmierschnittstelle OpenGL (https://www.opengl.org)
Bibliothek JOGL (www.jogamp.org/jogl)