Lehrstuhl Digital Additive Production

FAKULTÄT FÜR MASCHINENWESEN

der Rheinisch-Westfälischen Technischen Hochschule Aachen

Entwurf und Implementierung eines Gitter-Struktur-Generators auf Basis der objektorientierten Programmierung*

Praxisbericht 2 im Studiengang Angewandte Mathematik und Informatik

(Prüfungsleistung 950616)

von

Morteza, Montahaee

(Matr.Nr. 3262962)

Aachen, 15.02.2022 Morteza Montahaee Digitally signed by Morteza Montahaee DN: cn=Morteza Montahaee, c=DE, o=RWTH Aachen, email=morteza.montahaee@rwth-aachen.de Date: 2022.02.15 18:40:57 +01'00'

Morteza Montahaee, Auszubildender MATSE, Studierender AMI

Aachen, 15.02.2022 Sebastian Dirks Digital unterschrieben von Sebastian Dirks Datum: 2022.02.15 17:44:47 +01'00'

Sebastian Dirks, Betreuer

Inhalte

1 Praxisbericht

1.1 Motivation

Im Rahmen der additiven Produktion wurde vor einigen Jahren ein Gitter Struktur Generator in der Umgebungsapplikation von Rhinoceros[1] entwickelt (Abbildung 1). Durch den genannten Generator konnte man entlang bestimmter Pfadtypen sowohl im Karteisan als auch im Polar Koordinatensystem einigen Arten von Gitter modellieren. Zeil dieses Projekts war die Rekonstruierung der genannte Generator auf Basis Objekt Orientierter Programmierung (OOP) und gleichzeitig derer Umsetzung als eine Bibliothek für NX-Open.

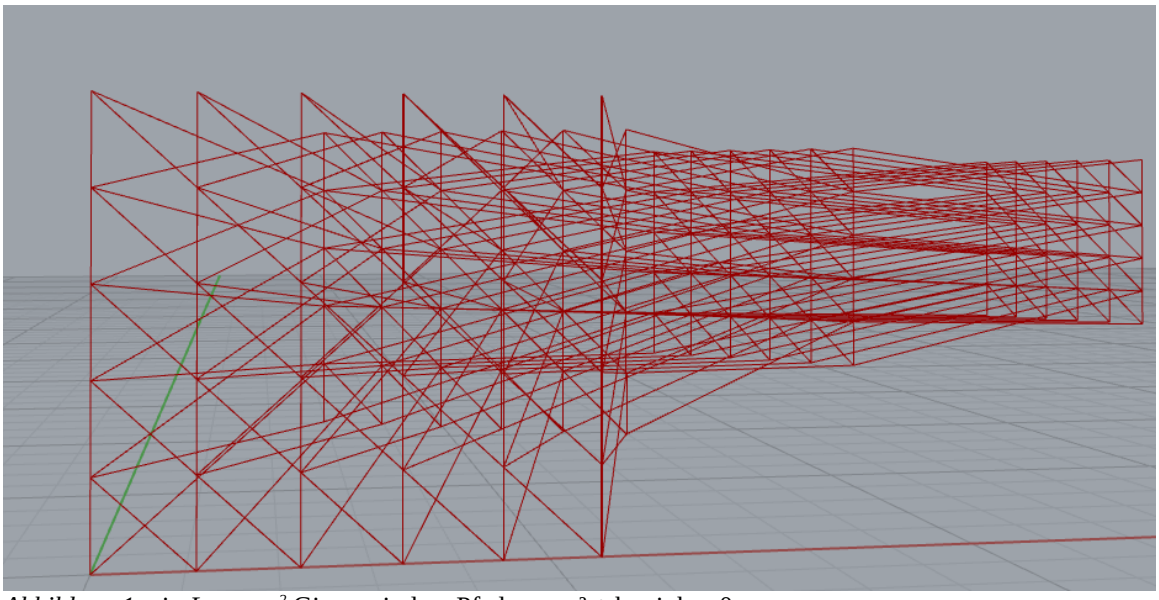


Abbildung 1: ein *Lozenge*[2] Gitter mit dem Pfad $y = ax^2 + b$ mit $b = 0$

1.2 Umsetzung

Im folgenden Abschnitt wird der Softwareentwicklungsprozess für oben beschriebene Problemstellung erörtert. Zusätzlich wird ein Überblick über die dabei verwendeten Werkzeuge und aufgetretene Herausforderungen während des Prozesses gegeben.

1.3 Eingesetzte Werkzeuge

Die Umsetzung des Projekts fand mit der Programmiersprache *C#* 6 statt, erweitert durch manche Open-Source Pakete und Bibliotheken wie z.B. *Net.Core* zur Erstellung des Projekts, *Math.NET* zur Anwendung in Lineare Algebra, *Protobuf* zur Serialisierung der Daten und *NX-Open-Utilitis* zur Anwendung verfügbarer Schnittstelle bzgl. eines geometrischen Dienstprogramms (siehe auch 1-4) . Als Entwicklungsumgebung kam *Visual Studio 2017* zum Einsatz. Die Versionskontrolle und automatische Generierung von Entwicklerdokumentation erfolgten auf dem allgemeinen GitLab der RWTH. Zur Softwaremodellierung wurde *Enterprise Architect 13.4* eingesetzt.

1. Rhinoceros und NX-Open sind Computer-Aided Design (CAD) Applikation Software.
2. Für eine gegebenen Gitterpunktmenge Λ ist ein Lozenge ein quadratisches Polytop wie ein Raute mit allen seiner Knoten in der Λ

1.4 Entwurf

In Bezug auf Anforderungsanalyse für Funktionen der künftigen Bibliothek und Codeanalyse aus der bereits geschrieben Nicht-OOP stellen sich folgenden Merkmale heraus:

- Möglichkeit der Extrusion, der Rotation und der Biegung eines Gitters
- Möglichkeit der Einschränkungen der Kanten eins Gitters im Zusammenhang mit geometrischen und mechanischen Eigenschaften

Der gesamte Entwurf der Klassenstruktur[1] erfolgte erst nach einer Grobdesign von Flussdiagrammen und wurde genau wie das Adaptermuster insofern konzipiert, als zur Visualisierung des Gitters Schnittstellen von *NX-Open* kompatible in unserem Projekt angewandt werden sollten[2]. Da Änderungen der Adapter-Klasse Gesamtarchitektur nicht beeinflussen sollen (Prinzip der losen Kopplung(PLK)) wurde das Modul in einem anderen Paket angelegt.

1.5 Implementierung

Zuerst ist Implementierung der Schnittstelle und Basisklasse festgesetzt, welche folgende Rolle spielen:

- *ILatticeInterface* beschreibt die Schnittstelle zu einem abstrakten Visualisierungsverfahren. Diese besteht aus Methode mit von Protobuf automatisch generierte Gitter als Ausgabesparmeter.
- *Path* besitzt notwendige Methode für Aufbau eines Pfades, auf deren Koordinaten Gitterpunkten (eng Lattice Groupe) angefangen bzw. orientiert werden.

Im Anschluss werden mit Hilfe der obigen Abstraktionen und dem *IDGenarator* in der *LayerGenerator* Klasse zunächst die Schichten erstellt, in welche Gitterpunkte abhängig von Querschnitttypen für den Aufbauraum der Gitter in einem Dictionary indiziert gespeichert werden. Folglich ist in der Methode „*GenereateStruts( )*“ von *StrutGenerator*† zur Erstellung der Kanten zwischen Gitterpunkten instanziiert. Bei beiden Klassen liefert *LatticeParameter* physikalische und geometrische Eigenschaften eines Gitters aus, was wiederum im *BuildRestrectionChecker* zur Anfertigung des gesamten Gitters geprüft werden, bevor in *LatticeGenerator* konkrete Verfahren• zum Gitter-Generator durchgeführt werden können. Für das Verfahren „*revolve( )*“ kommt *Axis* zum Einsatz, welche die Rotationen eines beliebigen Punktes um eine Achse beschreibt[M05].

Abschließend werden zum Prüfen der genannten Verfahren unterschiedliche Testmethoden bzgl. der Gittertypen auf Basis *NXOpenLatticeAdapter* programmiert und getestet. Dabei werden sowohl gemeinsame (unter das Methode „*validateLattice( )*“) als auch spezifische Assertionen in zufälligen und expliziten Gitterräumen eingeführt. Die erwähnte Hilfsklasse enthält im Rahmen der NX-Open-Sessionen mehrere Methoden zur Visualisierung des Gitters und somit tragen die Testmethode zu unserem inkrementellen Softwareentwicklungsprozess vor allem zum Verhindern von Kollisionen von Kanten zwischen jedem eindeutigen Paar bei

1.6 Erweiterbarkeit

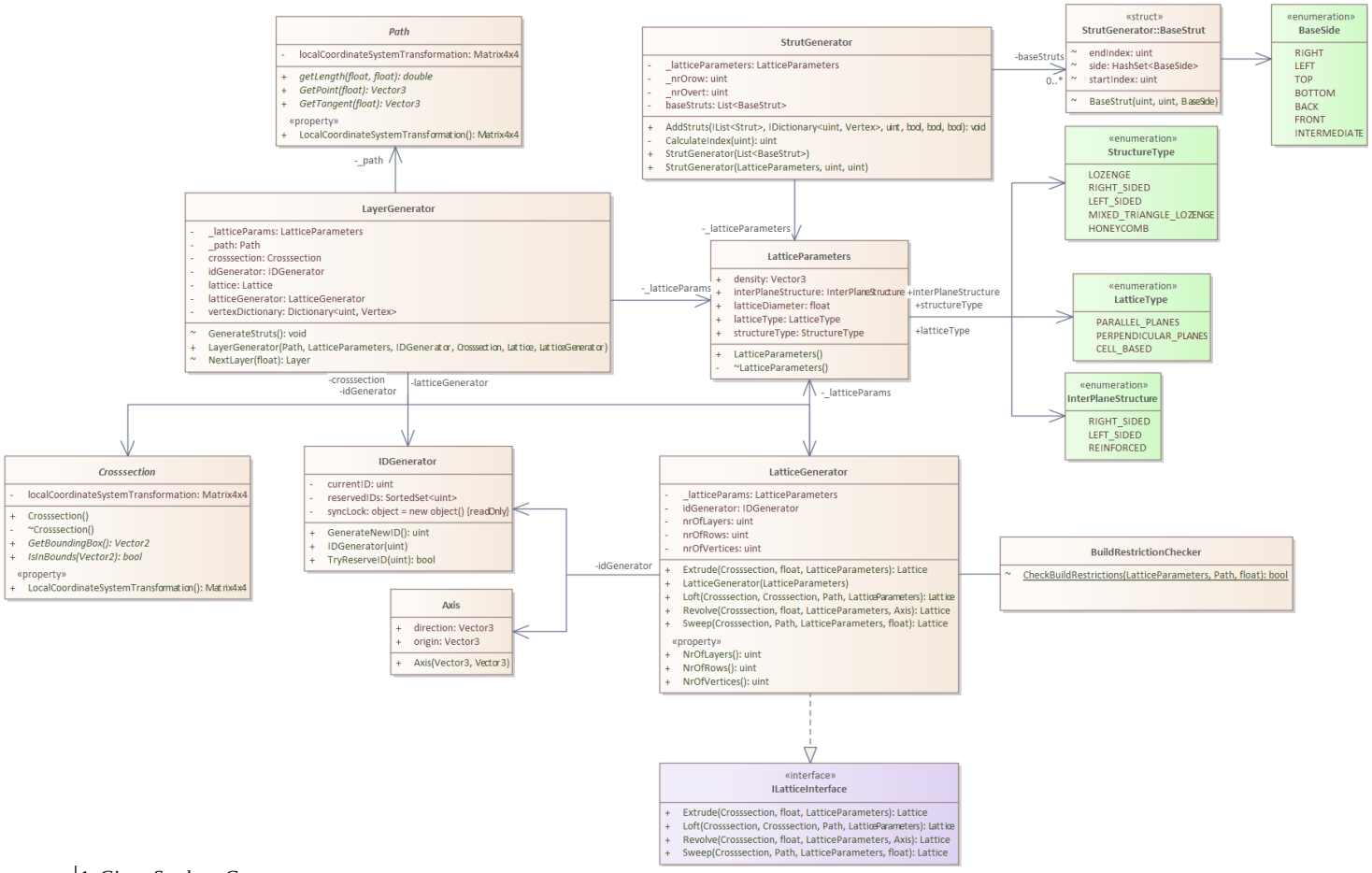
Aufgrund der Anwendung eines Kegelschnittes kann mehrere Pfade unter *CircularPath*[3] rekonstruiert werden. Deshalb kann durch ihre Umwandlung zu einer gut definierten Überklasse möglich viele Gittermodelle ins besonderes bei der Methode „*Revolve( )*“ im Model Paket eingefügt werden[4]. Drüber hinaus ist Ausbau eines polynomial kombinierten Pfads im Paket möglich und allgemein ermöglicht unsere Softwareprodukt wegen die Erfüllung der PLK eine Erweiterung bei anderen Software-Adapter.

1.7 Ergebnis

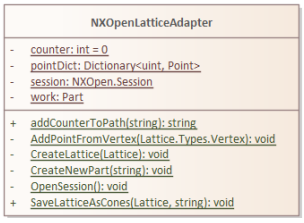
Das Resultat des Projekts ist aktuell eine Applikationsbibliothek für die CAD Software NX mit weiterem Erweiterungsbedarf, was sich mechanisch und geometrisch belangt*. Dennoch ist es in der Lage Gitter auf unterschiedliche Pfade zu konstruieren (siehe Teil 2.3)

2 Anhänge

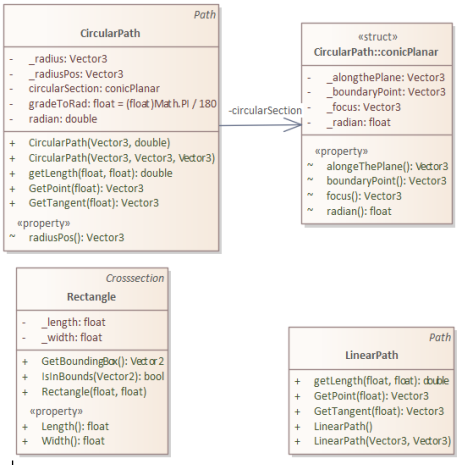
2.1 UML-Klassendiagramm



1. Gitter-Struktur-Generator



2. Adapterklasse zur Visualisierung



3. Aktuelle Pfadmodelle

* Konkret betrifft eine Einschränkunge bei statischen Werten einer Strebe oder gesamten Streben wie z.B.mindestens Normal- und Querkraft und Biegemoment für ein baubar gitterstruktuiertes Objekt, was natürlich geometrische Operationen fordern.

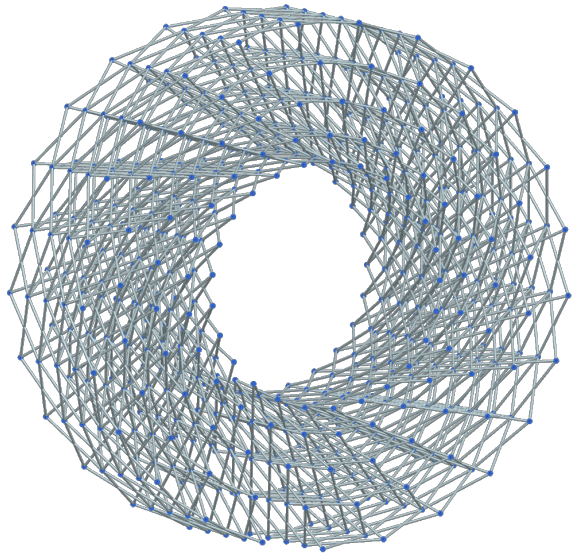
2.2 Code-Schnipsel

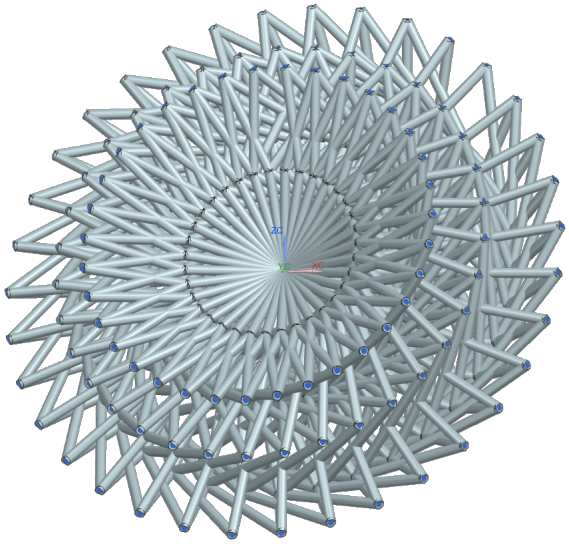
† Der erste Code-Abschnitt veranschaulicht Teile des allgemeinen Basiskantenzustands bzw. ihr Nachbarverhältnis für Gitterpunkte. Zur Reduzierung des Datenfelds (der Attribuierten) der Klasse ist eine Anwendung von Baustruktur durch Enumeration *BaseSide* abgedeckt (keine *BaseSide* Right, …, *BaseSide* Left). Der Zweite illustriert die Testmethode für bestimmte Gitter-Generator.

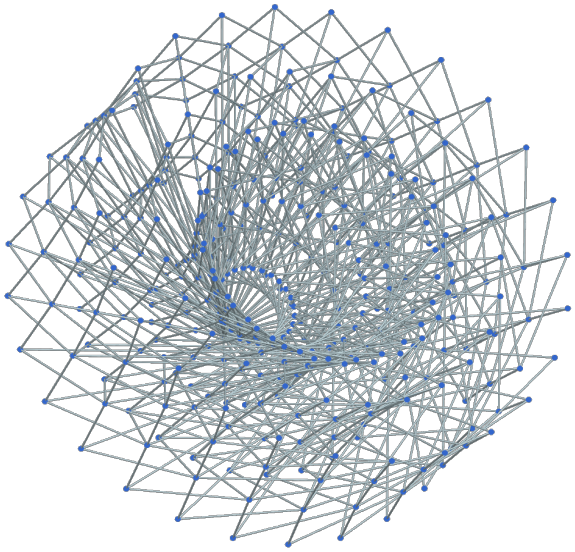
```
class StrutGenerator
{
    internal struct BaseStrut
    {
        internal uint startIndex;
        internal uint endIndex;
        internal HashSet<BaseSide> side;

        internal BaseStrut(uint startIndex, uint endIndex, BaseSide side)
        {
            this.startIndex = startIndex;
            this.endIndex = endIndex;
            this.side = new HashSet<BaseSide> {side };
        }
    }

    //The neighbour relationships
    internal enum BaseSide
    {
        RIGHT,
        LEFT,
        TOP,
        BOTTOM,
        BACK,
        FRONT,
        INTERMEDIATE
    }

    // number of vetrtices per row and rows for each layer
    private uint _nrOrow;
    private uint _nrOvert;
    private List<BaseStrut> baseStruts;
    private LatticeParameters _latticeParameters;
     //the constructor for struts
    public StrutGenerator(List<BaseStrut> baseStruts)
    {
        this.baseStruts = baseStruts ?? throw new ArgumentNullException(nameof(baseStruts));
    }

    public StrutGenerator(LatticeParameters parameter, uint nrOrow, uint nrOvert)
    {
        _latticeParameters = parameter;
        _nrOrow = nrOrow;
        _nrOvert = nrOvert;
        baseStruts = new List<BaseStrut>();
        var newStrut = new BaseStrut();
        switch (parameter.structureType)
        {
            case StructureType.LOZENGE:        // in-plane left/right
                baseStruts.Add(new BaseStrut(1, 7, BaseSide.BACK));
                baseStruts.Add(new BaseStrut(3, 5, BaseSide.BACK));
                baseStruts.Add(new BaseStrut(0, 6, BaseSide.BACK));
                baseStruts.Add(new BaseStrut(2, 4, BaseSide.BACK));
                break;

            case StructureType.RIGHT_SIDED:   // vertical in-plane right

                newStrut = new BaseStrut(0, 2, BaseSide.FRONT);
                newStrut.side.Add(BaseSide.LEFT);
                baseStruts.Add(newStrut);

                baseStruts.Add(new BaseStrut(0, 3, BaseSide.FRONT));

                ⋮
```

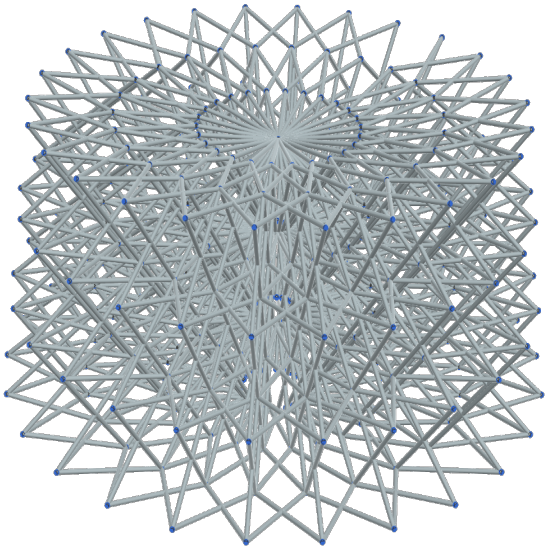
```
[TestMethod]
        // Test of an extrusion for parallel Lozenge
        public void TestExtrudeParallelLozengeRand()
        {
            var rect = new Rectangle((float)rand.NextDouble() * 10 + 1, (float)rand.NextDouble() * 10 +
1);
            parameter.latticeType = LatticeType.PARALLEL_PLANES;
            parameter.interPlaneStructure = InterPlaneStructure.REINFORCED;
            parameter.structureType = StructureType.LOZENGE;
            parameter.density = new System.Numerics.Vector3(1, 1, 1);
            parameter.latticeDiameter = 0.2f;
            var lattice = generator.Extrude(rect, (float) rand.NextDouble() * 10, parameter);
            ValidateLattice(lattice);
            NXOpenAdapter.NXOpenLatticeAdapter.SaveLatticeAsCones(lattice, PathLocation.partFileName);
            var nL = lattice.Layers.Count;
            var nR = lattice.Layers[0].Rows.Count;
            var nV = lattice.Layers[0].Rows[0].Vertices.Count;
            Assert.AreEqual( 2*(nR - 1) * ((2*nV - 1)* nL - nV), lattice.Struts.Count, "");
        }
```

2.3 Visualisierungsbilderbeispiel aus der Rotationsverfahren[4]









2.4 Literaturverzeichnis

[M05] Duncan Marsh: Applied *Geometry for Computer Graphics and CAD* Second Edition in Springer 2005, ISBN 978-1852338015 , pp 45-52