Technische Universität München

Ingenieurfakultät Bau Geo Umwelt

Lehrstuhl für Computergestützte Modellierung und Simulation

Parametrische Konstruktion von Systemparkhäusern mit Hilfe von Grasshopper 3D und Tekla Structures

Bachelorthesis

für den Bachelor of Science Studiengang Bauingenieurwesen

Autor: Stephan Oelfe

Matrikelnummer:

1. Betreuer: Prof. Dr.-Ing. André Borrmann

2. Betreuer: Katrin Jahr, M.Sc.

Ausgabedatum: 13. November 2017

Abgabedatum: 13. April 2018

Vorwort

Das Thema dieser Arbeit ist „Die parametrische Konstruktion von Systemparkhäusern mit Hilfe von Grasshopper 3D und Tekla Structures“. Die Arbeit habe ich in Kooperation mit der Firma Max Bögl GmbH verfasst, die mir jede notwendige Software zur Verfügung gestellt hat.

Das Thema entstand dadurch, dass die Abteilung für Systemparkhäuser im Stahlbau einen Weg gesucht hat, ihre Parkhausmodelle auf schnelle Art und Weise zu erstellen, da bereits ein hoher Standardisierungsgrad vorlag. In Absprache mit Timo Engl, der für Anwendung von Tekla Structures in der Firma zuständig ist, kam dabei die Idee auf, die Schnittstelle zu Grasshopper 3D zu nutzen. Da man damit allerdings noch keine Erfahrung hatte, wurde mir dieses Thema als Option für die Bachelor's Thesis angeboten. Da ich das Potential dieser visuellen Programmiersprachen bereits ein Jahr zuvor in dem Kurs Bau- und Umweltinformatik Ergänzungsmodul an der Universität erkannt habe und dadurch auch schon eine kleine Einführung in Dynamo bekam, habe ich genau dieses Thema gewählt.

Vor allen Dingen möchte mich bei der Firma und all denjenigen bedanken, die mich bei der Arbeit unterstützt haben. Allen voran Timo Engl, der mich in organisatorischen und fachlichen Fragen stets beraten und betreut hat. Außerdem möchte ich mich bei den Kollegen der Abteilung für Systemparkhäuser - speziell dem Stahlbau - bedanken, auf die ich immer zurückkommen konnte, um Einzelheiten in der Modellierung oder deren Vorgehensweisen abzustimmen.

Außerdem von Seiten der Technischen Universität München einen ganz großen Dank an Katrin Jahr, die mich hervorragend betreute und mir sowohl für die Organisation als auch für die Rücksprache in fachlichen Angelegenheiten immer zur Seite stand.

Und zu guter Letzt möchte ich noch meiner Familie und Freunden danken, die mich immer wieder motivieren konnten.

Stephan Oelfe

Sulzbürg, 13. April 2018

Abstract

For the effective planning of buildings, Building Information Modeling (BIM) is already being used in practice. However, in some cases it can be very expensive to model large models with very high accuracy. This is where parametric modeling comes into play, which can be very well used, especially when using visual programming languages. There is a trend towards visual programming, especially for students with little programming skills (Preidel et al. 2017, p. 9). This thesis deals with parametric modeling of car parks using Tekla Structures and Grasshopper 3D. At the same time, to create a basic model within a very short time the building system has to be modeled high parameterized. It should be determined based on this model, if and how this interface can be applied in a company. The challange of this thesis is to prepare this interface so far that the user does not need any previous knowledge in visual programming or the interface itself. First, the background to Building Information Modeling and Parametric Modeling will be presented, followed by the structural strengths and weaknesses of visual programming languages. Since this thesis was developed in cooperation with Max Bögl GmbH, the following chapter describes the initial situation, such as the existing structures and procedures for planning car parks. It also analyzes a failed attempt to achieve such rapid model generation and looks for possibilities to avoid the errors contained therein. Next, the interface between Tekla Structures and Grasshopper 3D will be scrutinized to identify and mitigate their weeknesses. To do this, one has to switch to various programming languages, such as Visual Basic for Applications (VBA) or batch files. This enables to use the interface for persons with limited knowledge. However, the developer must be well-versed in programming. In addition, the respective building system must already be high standardized and must have a set of pre-modeled components.

Zusammenfassung

Für die effektive Planung von Bauwerken wird in der Praxis bereits Building Information Modeling (BIM) verwendet. In manchen Fällen kann es jedoch sehr aufwändig sein, große Modelle mit sehr hoher Genauigkeit zu modellieren. Hier tritt die parametrische Modellierung ins Spiel, die gerade bei Verwendung von visuellen Programmiersprachen sehr gut abgedeckt werden kann. Forscher sehen einen Trend hin zur visuellen Programmierung gerade bei Schülern oder Studenten mit wenig Erfahrung im Programmieren (Preidel et al. 2017, S. 9). Diese Arbeit beschäftigt sich mit der parametrischen Modellierung von Systemparkhäusern mithilfe von Tekla Structures und Grasshopper 3D. Dabei soll das Bausystem so weit parametrisiert abgebildet werden, dass die Erstellung eines Basismodells innerhalb kürzester Zeit erfolgen kann. Dabei soll anhand dieser Modellierung festgestellt werden, ob und wie diese Schnittstelle in der Praxis angewendet werden kann. Die Schwierigkeit liegt darin, diese Schnittstelle so weit aufzubereiten, dass der Anwender keine Vorkenntnisse in visueller Programmierung oder der Schnittstelle selbst braucht. Hierzu werden zunächst die Hintergründe zu Building Information Modeling und der parametrischen Modellierung aufgezeigt, um anschließend auf die strukturellen Stärken und Schwächen von visuellen Programmiersprachen überzugehen. Da diese Arbeit in Kooperation mit der Firma Max Bögl GmbH entstanden ist, wird im darauffolgenden Kapitel die Ausgangssituation beschrieben, wie die vorhandenen Strukturen und Abläufe zur Parkhausplanung. Es wird außerdem ein gescheiterter Versuch analysiert, eine solch schnelle Modellerzeugung zu erreichen, und Wege gesucht, die darin enthaltenen Fehler zu vermeiden. Anschließend wird die Schnittstelle von Tekla Structures zu Grasshopper 3D genau untersucht, um auch deren Schwachstellen zu identifizieren und zu umgehen. Dazu muss auf verschiedene weitere Programmiersprachen ausgewichen werden, wie Visual Basic for Applications (VBA) oder Batch-Dateien. Dies macht die Benutzung der Schnittstelle für Personen mit beschränkten Kenntnissen möglich. Dabei muss allerdings der Entwickler fundierte Kenntnisse in Programmierung besitzen. Außerdem muss das jeweilige Bausystem bereits in hohem Maße standardisiert sein und einen Satz an vormodellierten Bauteilen aufweisen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AIA	American Institute of Architects
BAP	BIM Abwicklungsplan
BIM	Building Information Modelling
GaStellV	Garagen- und Stellplatzverordnung
LOD	Level of Development
NURBS	Non-Uniform Rational BSplines
VBA	Visual Basic for Applications

1 Einführung und Ziel der Arbeit

Immer schneller, immer effizienter, immer günstiger! Dieser Grundgedanke findet sich in weiten Teilen der heutigen Gesellschaft wieder. So werden in der Autoindustrie Fahrzeuge maschinell innerhalb eines Bruchteils der Zeit gefertigt, die ein einzelner Handwerker benötigen würde. Möglich macht dies die exakte Vorplanung und Standardisierung der einzelnen Bauteile und Arbeitsschritte. Genau diesen Gedanken griff die Firma Max Bögl GmbH auf während sie bereits weite Teile der in verschiedenen Projekten verwendeten Bauteile standardisierten und über digitale Daten maschinell produzieren. Da aber in der Bauindustrie die Bedürfnisse und Wünsche der Kunden sehr verschieden sind, variiert auch die Form der zu bauenden Projekte. In dieser Arbeit wird nach einem Weg gesucht, ebendiese variierenden Kundenwünsche im Falle des Bausystems Systemparkhäusern als digitale Daten aufzunehmen, um daraus innerhalb kürzester Zeit ein vollwertiges Basismodell in der BIM-Software Tekla Structures zu generieren, das am Ende mit weiteren Informationen manuell ergänzt werden kann. Dafür wird auf die Schnittstelle zu Grasshopper 3D zurückgegriffen, in der Parametrik in Form von visueller Programmierung abgebildet werden kann.

Studien der Technischen Universität München haben einen Trend hin zum visuellen Programmieren feststellen können. So haben sie an einer Reihe von Befragungen und Tests mit Studenten innerhalb ihrer Lehre feststellen können, dass es gerade den Studenten leichter fällt, die wenig bis gar keine Erfahrung in Programmieren haben, solch eine Programmiersprache zu erlernen und zu verstehen, als textbasierte Programmiersprachen (Preidel et al. 2017, S. 9).

1.1 Motivation und Ziel der Arbeit

Die Konstrukteure der Firma Max Bögl GmbH durchlaufen bei der Erstellung eines Parkhausmodells bereits einen festen Basisprozess mit immer wieder kehrenden Arbeitsschritten. Da dieser Prozess bis zu mehreren Wochen dauern kann, wurde nach einem Weg gesucht, diese Schritte automatisch vom Rechner zu vollziehen. Im Unternehmen wurde man bereits vorher auf die Schnittstelle von Tekla Structures zu Grasshopper 3D aufmerksam, woraufhin anschließend die Idee aufkam, diese Schnittstelle für die Erstellung der Parkhausmodelle zu nutzen. Da allerdings keinerlei Erfahrung mit dieser Schnittstelle vorhanden ist, muss zunächst überprüft werden, ob damit

alle nötigen Bauteile genau platziert und deren Eigenschaften gesteuert werden können. Diese Präzision ist die Voraussetzung für das Gelingen des Projekts. Im Anschluss soll überprüft werden, ob Anwender ohne Erfahrung mit dieser Schnittstelle arbeiten können oder ob und welche Vereinfachungen für den Nutzer getroffen werden müssen. Da dieser Generator von ebendiesen Konstrukteuren genutzt werden soll, ist dies ebenfalls wesentlicher Bestandteil dieser Arbeit. Am Ende soll ein fertiges Konstrukt aus Programmen entstehen, das dem Nutzer ermöglicht, ein Basismodell eines Parkhauses innerhalb kürzester Zeit zu generieren, ohne sich mit der Schnittstelle selbst befassen zu müssen. Hierbei soll mit den zuständigen Konstrukteuren zusammengearbeitet werden, um den Generator exakt umzusetzen.

1.2 Struktureller Aufbau der Arbeit

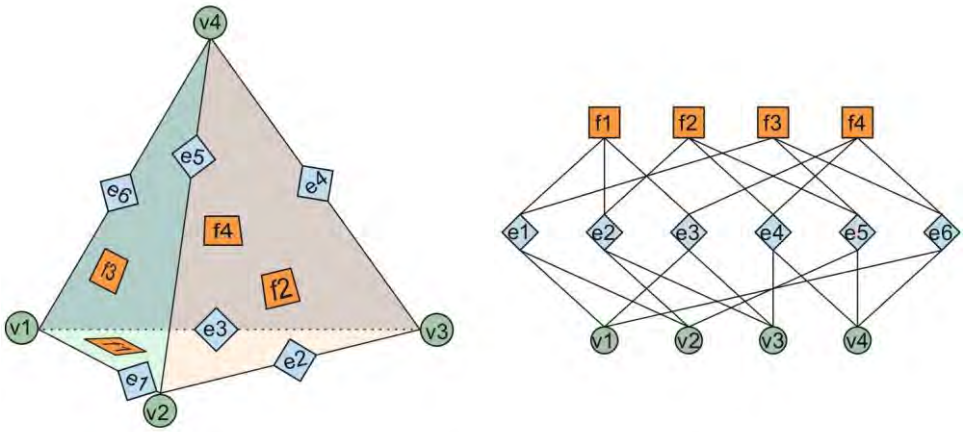
Zunächst wird die Methodik aufgezeigt, auf die in dieser Arbeit zurückgegriffen wird. Dazu zählen die Grundlagen des Building Information Modeling (BIM), sowie die Nutzung von Parametrik innerhalb von BIM-Software. Außerdem soll die Struktur von visuellen Programmiersprachen und deren Stärken und Schwächen allgemein erklärt werden. Hier wird bereits auf die Anwendungen Grasshopper 3D und Dynamo eingegangen, in denen diese Sprachen angewandt werden. Anschließend werden verschiedene Möglichkeiten zur Gestaltung von Parkhäusern und deren Parametern sowie deren Planung mit der BIM-Software Tekla Structures in der Firma Max Bögl GmbH erklärt. Dabei wird bereits auf den Aufbau und die Besonderheiten von Tekla Structures eingegangen. Daraufhin wird zunächst veranschaulicht, auf welche Mittel oder Erfahrungen in dieser Arbeit bereits zurückgegriffen werden kann und in welchem Gesamtkontext der Firmenstruktur dieser Generator ansetzt. Anschließend werden die Stärken und Schwachstellen von Grasshopper 3D und dessen Schnittstelle zu Tekla Structures untersucht, um dieses Wissen bei der Umsetzung des Programmkonstrukts anwenden zu können.

2 Methoden

2.1 Einführung in BIM

Das sogenannte Building Information Modeling (BIM) beschränkt sich nicht nur auf die reine Modellierung von dreidimensionalen Strukturen oder Bauwerken. Es hinterlegt den einzelnen Strukturen auch digitale Informationen, die für bautechnische Analysen oder Planungen wertvoll sind und am Rechner genutzt werden können (Borrmann et al. 2015, S. 0). Da sich diese Arbeit im Wesentlichen auf die reine Erzeugung eines Bauwerksmodells beschränkt, wird hier verstärkt auf die Implementierung von Geometrie am Rechner eingegangen.

Um im Rechner eine bestimmte Geometrie zu implementieren, gibt es nach Borrmann zum einen das explizite und zum anderen das implizite Verfahren. Im expliziten Verfahren werden diese Geometrien über eine Struktur an Daten erzeugt, die durch geometrische Größen wie Punkte, Kanten oder Volumen definiert wird und miteinander in Beziehung stehen (Borrmann et al. 2015, S. 27 f.).



solid	faces
1	1,2,3,4

face	edges
1	1, 2, 3
2	2, 4, 5
3	1, 5, 6
4	3, 4, 6

vertex	coordinates
1	2, 0, 0
2	0, 0, 0
3	3, 0, 0
4	1, 1, 3

edge	vertices
1	1, 2
2	2, 3
3	3, 1
4	3, 4
5	2, 4
6	1, 4

Abbildung 1: Explizites Verfahren (Borrmann et al. 2015, S. 29)

Beim impliziten Verfahren werden einzelne Konstruktionsschritte implementiert, die eine Form definieren sollen. Hier wird auf boolesche Operatoren zurückgegriffen.

Dadurch werden Geometrien in Abhängigkeit von anderen erzeugt (Borrmann et al. 2015, S. 30 ff.).

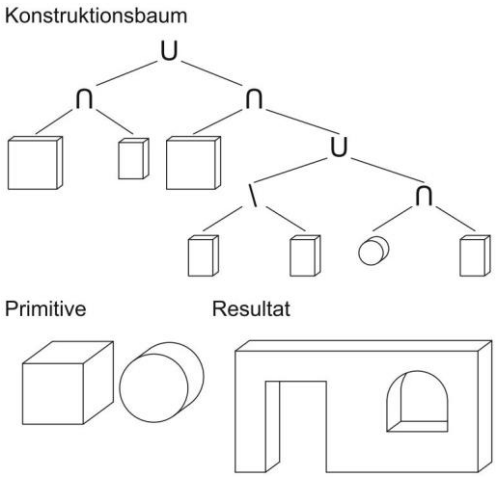


Abbildung 2: Implizites Verfahren (Borrmann et al. 2015, S. 32)

Durch die geometrische Modellierung wird es möglich, das dreidimensionale Modell auf Kollisionen zu überprüfen, um ein korrektes Modell zu erzeugen. Des Weiteren können aus diesem Modell zweidimensionale Pläne direkt abgeleitet werden und somit Unstimmigkeiten zwischen den einzelnen Plänen vermieden werden. Weiter kann damit Mengenermittlung betrieben werden, welche die Kalkulation und somit die Angebotserstellung wesentlich beschleunigen. Außerdem können mit dem sogenannten „Rendering-Verfahren“ verschiedene Ansichten des Modells automatisch erzeugt werden, um dem Bauherrn oder dem Architekten sein gewünschtes Bauwerk veranschaulichen zu können (Borrmann et al. 2015, S. 25 f.).

Um den Grad der Genauigkeiten des Modells allgemein festlegen zu können, wird nach Borrmann auf den sogenannten „Level of Development“ (LOD) zurückgegriffen. Je präziser es modelliert wurde, desto höher ist der LOD. Die von Borrmann beschriebenen Grade wurden nach dem American Institute of Architects (AIA) definiert. Dabei gibt es LODs zwischen 100 und 400. Je höher der LOD, desto genauer ist das Modell abgebildet (Borrmann et al. 2015, S. 141).

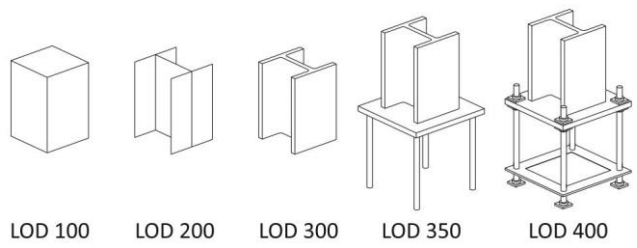


Abbildung 3: Bauteile mit verschiedenen LOD's (Borrmann et al. 2015, S. 142)

Eine Möglichkeit, Kurven mithilfe des Rechners zu beschreiben, ist die Beschreibung mit NURBS, was nach Borrmann eine Abkürzung für „Non-Uniform Rational BSplines“ ist. NURBS sind Kurven, die stark vereinfacht über mathematische Polynome eines bestimmten Grades beschrieben werden, die wiederum über Kontrollpunkte definiert werden. Je nach Grad der Polynome hat die Änderung eines Kontrollpunktes Auswirkung auf den restlichen Verlauf der Kurve. Die Besonderheit der NURBS ist, dass es möglich ist, jedem Kontrollpunkt eine unterschiedliche Auswirkung auf den Gesamtverlauf der Kurve zuweisen zu können (Borrmann et al. 2015, S. 39).

Um nicht nur eine dreidimensionale Geometrie zu erschaffen, wird zusätzlich die objektorientierte Modellierung verwendet. Hierbei werden jedem Bauteil zusätzlich zu seinen geometrischen Abmessungen auch noch Eigenschaften, wie Gewicht oder bestimmte Festigkeiten zugewiesen, um diese Daten für weitere Analysen zu verwenden. Diese Eigenschaften werden hier als „Attribute“ bezeichnet (Borrmann et al. 2015, S. 46). Ein weiterer anzuführender Aspekt ist die Unterteilung von BIM selbst. In dem sogenannten „little bim“ wird zur Kommunikation mit anderen Planern das erstellte Modell selbst nicht verwendet. Genau das Gegenteil des little bims ist das „BIG BIM“, in dem das erstellte Modell selbst und die in ihm enthaltenen Daten zum Informationsaustausch verwendet werden können. Genauso wird unterschieden in „closed BIM“ und „open BIM“. Im open BIM können die Daten über ein bestimmtes Format übertragen werden, das in mehreren Programmen verwendet oder ausgelesen werden kann, im closed BIM nicht (Borrmann et al. 2015, S. 9).

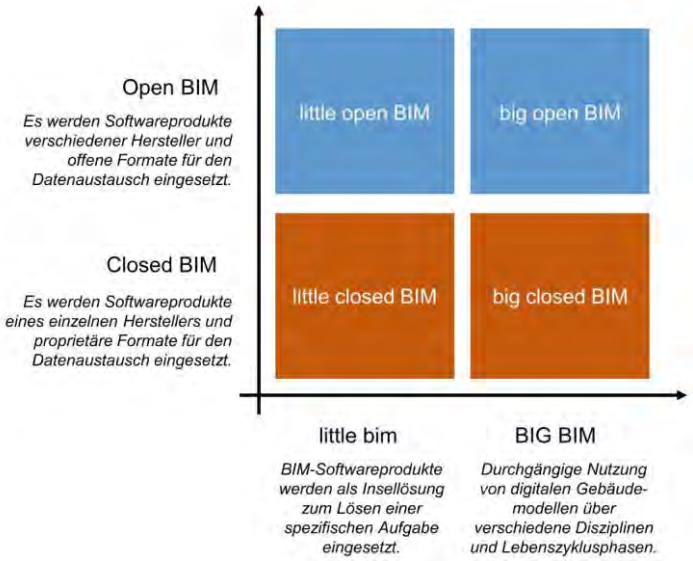


Abbildung 4: Unterscheidungen von BIM (Borrmann et al. 2015, S. 8)

2.2 Parametrik

Da die Parametrik der Grundstein meiner Arbeit ist möchte ich hier auf die wichtigsten Bestandteile der Parametrik eingehen und die Verbindung schaffen von der Parametrik in einzelnen Bauteilen und der in meiner Arbeit verwendeten äußeren Parametrik. Man kann mehrere Bauteile oder die Abmessungen eines Bauteils mithilfe von Parametern miteinander verknüpfen. Die Änderung eines Parameters kann auch andere Parameter beeinflussen. Dadurch lassen sich diese Bauteile sehr leicht je nach Bedarf ändern. Zusammen mit dem impliziten Verfahren der geometrischen Modellierung kann sehr flexibel und genau modelliert werden, was neben der Zeiteinsparung einen bedeutender Faktor darstellt (Borrmann et al. 2015, S. 52).

Je höher die korrekte Parametrisierung, desto weniger Fehler werden am Ende im Modell entstehen. Dabei können nach Borrmann die Positionen der Bauteile oder Teile der Bauteile absolut festgelegt werden und genauso relativ zu diesen Positionen verschoben werden. Borrmann nennt diese Positionen „Referenzebenen bzw. -achsen“ (Borrmann et al. 2015, S. 53). Es treten aber auch Referenzpunkte auf. D.h. diese parametrisierten Objekttypen haben einen relativen Bezug zu bestimmten Kanten o.ä., welche wiederum absolut mit Hilfe von Koordinaten definiert sind. Diese Art der Parametrik innerhalb von BIM-Software ist allerdings begrenzt und bietet sich bei kleineren Modellen oder Bauteilgruppen an. Bei größeren Modellen wird i.d.R. auf visuelle Programmiersprachen zurückgegriffen, die viel flexibler aufgebaut werden und viel mehr Details enthalten können. Gerade bei der Implementierung von Bausystemen mit vielen Fallunterscheidungen sollten diese verwendet werden.

2.3 Visuelle Programmierung

2.3.1 Grundlagen

In diesem Kapitel sollen einige notwendige Begriffe geklärt werden und ein kurzer Überblick über die Thematik der visuellen Programmierung gegeben werden. Dieses Verständnis für den Aufbau einer solchen visuellen Programmiersprache ist essentiell für spätere Erkenntnisse. Nach Schiffer war der Hauptgedanke hinter der Verwendung visueller Programmierung die leichte und schnelle Entwicklung von eigenen Programmen. Eine derartige Programmiersprache könne auch jemand ohne vertiefte Programmierkenntnisse benutzen. Als Vorreiter gelte demnach David Canfield Smith, der das „Pygmalion“ im Zuge seiner Promotion entwickelte (Schiffer 1998, S. 3).

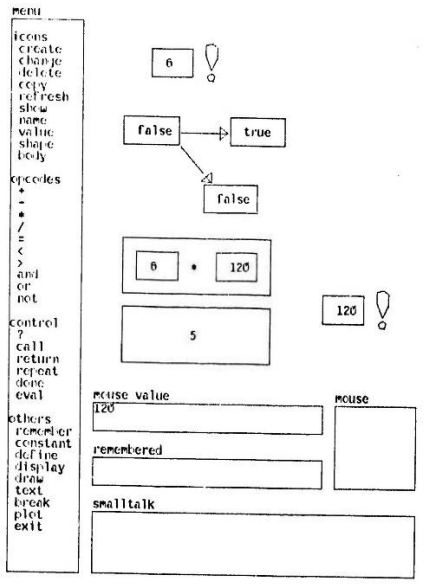


Abbildung 5: Ausschnitt aus dem PYGMALION (Smith 1975, S. 136)

Nach Schiffer sei es schwierig, eine treffende Definition für visuelle Programmierung zu finden. Ein mit visueller Programmiersprache geschriebenes Programm entsteht, indem verschiedene Abbildungen, denen eine bestimmte Operation hinterlegt ist, entsprechend miteinander verknüpft werden. Diese Abbildungen nennt er auch „visuelle Elemente“ (Schiffer 1998, S. 30). Die einzelnen visuellen Elemente oder „Komponenten (Knotenpunkte) können auf der Benutzeroberfläche angeordnet und miteinander verknüpft werden mit gerichteten Kanten (auch bezeichnet als Kabel)“ (Preidel et al. 2017). Durch diese Kabel werden Daten an die nächsten Komponenten übermittelt. So entsteht ein Datenfluss über alle verknüpften Komponenten (Schiffer 1998, S. 99).

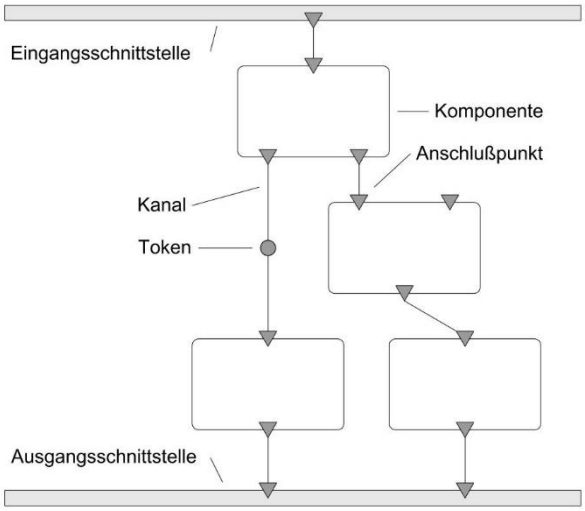


Abbildung 6: Aufbau von visuellen Programmiersprachen (Schiffer 1998, S. 99)

Hierbei werden die Komponenten, die Daten für eine Eingangsstelle einer anderen Komponente produzieren, vor der anderen Komponente ausgeführt. Alle weiteren

Komponenten, die keine direkte Abhängigkeit haben, können parallel ausgeführt werden (Schiffer 1998, S. 106). Schiffer benannte dieses Merkmal als „Implizite Ausführungsreihenfolge und Parallelität“ (Schiffer 1998, S. 106).

Visuelle Programmierung kann sehr sinnvoll eingesetzt werden. Es gibt jedoch einige Schwachstellen in der reinen visuellen Programmierung, weswegen man diese immer auch kritisch betrachten und eventuell durch andere Programmiersprachen ergänzen muss. Beispielsweise könnten größere mathematische Formeln nur mit einer Vielzahl an Komponenten abgebildet werden. Eine weitere Schwachstelle bezeichnete Schiffer als „Spaghetti-Effekt“: Je mehr Komponenten miteinander verknüpft sind, desto mehr Kanäle zur Datenübertragung werden eingesetzt. Sich unter derart vielen Verbindungen zurechtzufinden kann durchaus schwierig sein (Schiffer 1998, S. 57 ff.). Außerdem könnten in datenflussorientierten Programmiersprachen keine Schleifen realisiert werden (Schiffer 1998, S. 120).

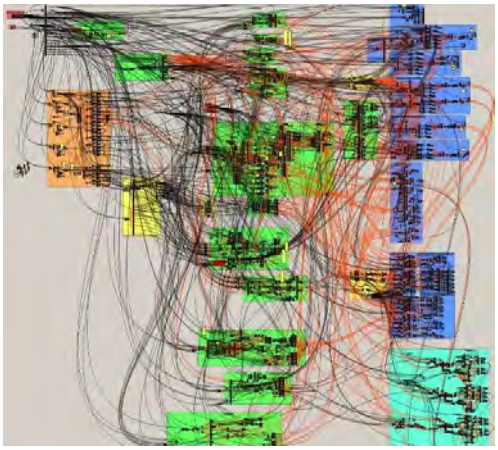


Abbildung 7: Spaghetti-Effekt in Grasshopper 3D

Um die visuelle Programmierung von der textuellen abzugrenzen, definierte Schiffer die verbalen Programmiersprachen, in der visuelle Elemente nur der Veranschaulichung dienen und ansonsten keinen Einfluss haben. Es werden nur der Programmiersprache in Form von Text bekannte Befehle umgesetzt (Schiffer 1998, S. 29).

Hierunter fallen Programmiersprachen, wie C++, Visual Basic for Applications (VBA) oder Python. Beispiele für Anwendungen, die auf visuelle Programmierung basieren, sind Grasshopper 3D und Dynamo (Preidel et al. 2017).

2.3.2 Grasshopper 3D

Grasshopper 3D ist ein Plugin für Rhinoceros 3D, das seit der neuesten Version „Rhino 6“ bereits vorinstalliert ist (Grasshopper - Neu in Rhino 6). Es basiert auf visuelle Programmierung und stellt eine Alternative zu dem „RhinoScript“ dar, das auf textuelle Programmierung basiert. Denn hier lassen sich z.B. Skripte mit Visual Basic erstellen. In Rhinoceros 3D werden Kurven bzw. Flächen hauptsächlich mithilfe von NURBS beschrieben (Lagios et al., S. 3). In Kombination mit Grasshopper 3D macht dies möglich, sehr individuelle parametrisierte Formen in kurzer Zeit zu schaffen, wie geschwungene Bauteile oder aufwändige Fassaden. Deswegen kann es auch sehr gut in der Architektur eingesetzt werden. Durch diese Parametrik lassen sich aber auch Generatoren schnell programmieren, die einen vordefinierten Ablauf mit veränderlichen Variablen abarbeiten können und somit eine gewünschte Struktur schnell generieren können. Auch in Grasshopper 3D werden Komponenten eingesetzt, die durch Kabel-ähnliche gerichtete Kanten oder Kanäle miteinander verbunden werden. Sie haben ihren Eingang auf der linken Seite und ihren Ausgang auf der rechten Seite. Somit ist die Richtung des Datenflusses prinzipiell von links nach rechts. Eine nützliche Komponente in Grasshopper 3D, die aber auch in anderen Programmiersprachen verwendet werden kann, ist ein Schieberegler (engl. „Slider“). Mit diesem wird eine Variable zwischen einem bestimmten Zahlenbereich je nach Position des Schiebereglers erzeugt. Dadurch „bekommt [der Nutzer, Anm. d. Verf.] unmittelbar visuelle Rückmeldung der geometrischen Auswirkung des verändernden Parameters“ (Lagios et al., S. 4).

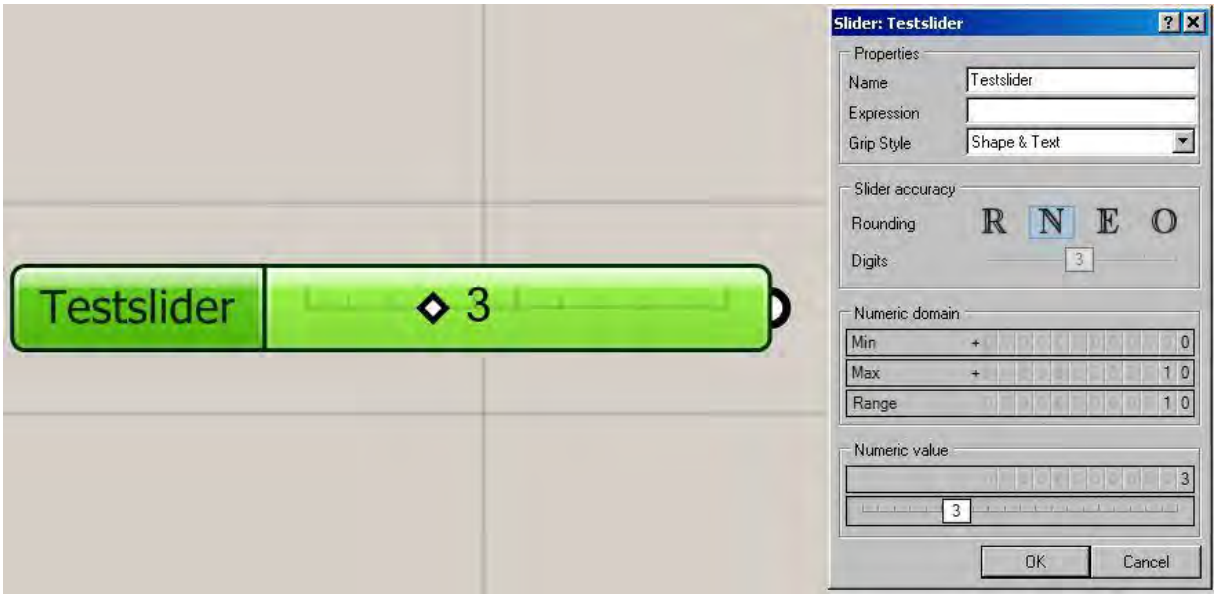


Abbildung 8: Slider mit Einstellung des Wertebereichs

Man hat außerdem die Möglichkeit, eigene Komponenten mithilfe von textueller Programmierung wie wie Python, Visual Basic oder C# zu erstellen. In diesen Komponenten beschreibt man nur den Vorgang, der mit einem einzelnen Eintrag des Inputs gemacht werden soll. Hierbei kann man innerhalb der Komponenten auch Schleifen einbauen, um aus den Eingabeparametern bestimmte Ergebnisse zu erzielen, wie numerische Methoden zur Bestimmung einer Nullstelle zwischen zwei Punkten o.ä. Diese eigens erstellten Komponenten können sinnvoll für komplexe mathematische Zusammenhänge, das Auslesen des Pfades einer Datei oder zum Weiterleiten oder Blockieren von Daten benutzt werden.

Zudem besteht die Möglichkeit, Verknüpfungen zu anderen Programmen wie Excel oder Tekla Structures herzustellen. Dies wird über Plugins realisiert, die noch installiert werden müssen. Diese Plugins sind als gha-Dateien im Internet unter www.food4rhino.com verfügbar und müssen vor Gebrauch in dem Ordner „Components folder“ abgespeichert werden. Um diesen Ordner manuell zu öffnen, kann in Rhinoceros 3D in der Befehlszeile „GrasshopperFolders“ eingegeben und anschließend „Components“ ausgewählt oder eingegeben werden. Danach muss das Programm neu gestartet werden, woraufhin die Plugins installiert werden.

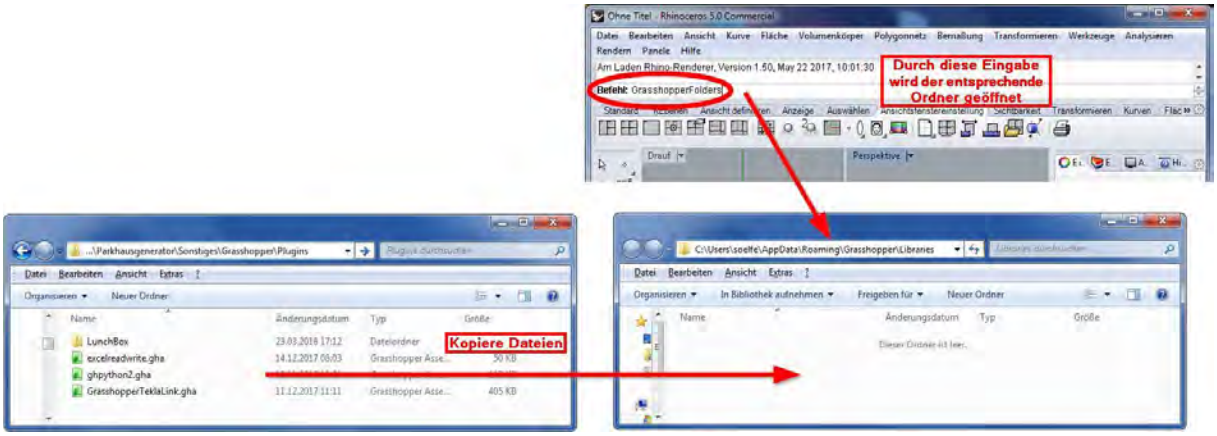


Abbildung 9: Kopieren der Plugin-Dateien von einem beliebigen Ordner in den „Components folder“

2.3.3 Dynamo

Dynamo ist eine ebenfalls auf visuelle Programmierung basierende Anwendung, mit der man mithilfe von parametrischen Beschreibungen Strukturen erzeugen kann. Unter anderem kann man es hervorragend als Plugin für die BIM-Software Revit verwenden, für das viele Elemente bereits zur Verfügung gestellt werden. Die Benutzeroberfläche ist sehr vergleichbar mit der von Grasshopper 3D. So kann man auch einzelne visuelle Elemente mit Kabel miteinander verbinden, um so einen Datenfluss zu generieren. Auch hier erfolgt dieser von links nach rechts. Diese visuellen Elemente werden in Dynamo als „Block“ bezeichnet (Der Dynamo Primer | Der Dynamo Primer 2017).

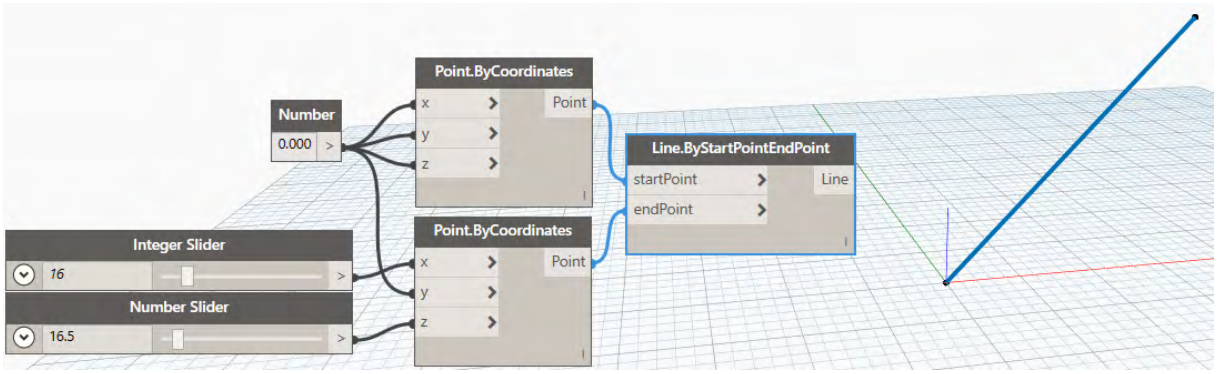


Abbildung 10: Dynamo-Blöcke und deren Verknüpfung

3 Parkhäuser

3.1 Allgemeines

Ein Parkhaus wird dazu verwendet, eine Vielzahl an Fahrzeugen auf einer kleinen Grundstücksfläche abstellen zu können. Grundsätzlich wird bei Parkhäusern unterschieden anhand deren Ausführung der Rampen. Diese definiert die restliche Form des Parkhauses. Die erste Möglichkeit ist eine Wendelrampe bzw. Doppelhelix. Dazu definierte Kleinmanns: „Wendelrampen sind spiralförmig um einen meist offenen Kern steifende Rampen, die häufig außen an das eigentliche Parkhaus angebaut sind, aber auch in den Baukörper integriert sein können“ (Kleinmanns 2011, S. 53).

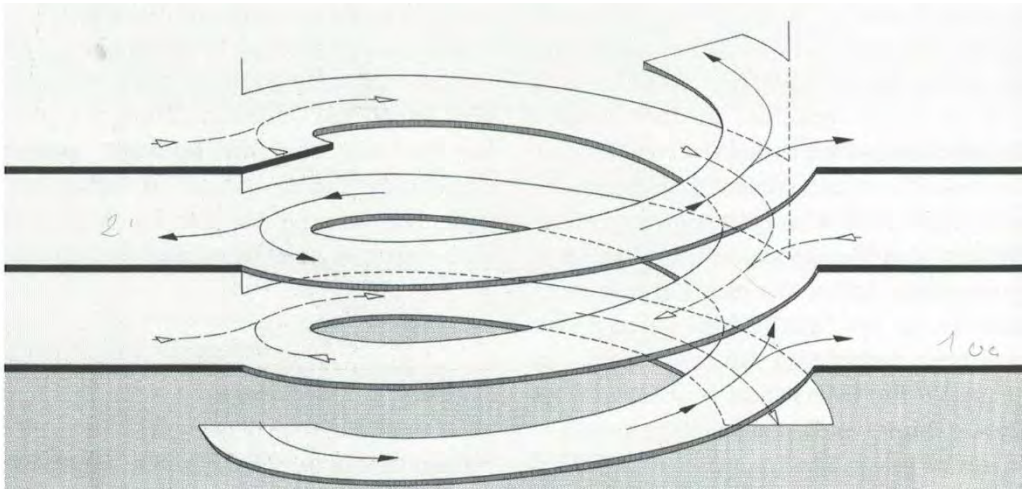


Abbildung 11: Verschachtelte Doppelhelix-Wendelrampe (Kleinmanns (2011), S. 54)

Die Auf- und die Abfahrt können dabei in zwei Spiralen getrennt ausgeführt werden ohne dafür mehr Grundstücksfläche in Anspruch zu nehmen. Nach Kleinmanns wird bei dieser Variante am meisten Platz gespart. Außerdem können durch diese Variante Vollflächengeschosse ohne Versatz zueinander ausgeführt werden (Kleinmanns 2011, S. 53).

Im Gegensatz zu der Doppelhelix werden bei den Split-Level-Garagen die Geschossflächen mit Versatz zueinander ausgeführt. Dies ist das wesentliche Merkmal der Split-Level-Garagen. Durch diesen Versatz verringert sich die Länge der Rampen im Gegensatz zu Parkhäusern mit Vollflächengeschosse, wodurch im Vergleich dazu Platz gespart wird (Kleinmanns 2011, S. 50).

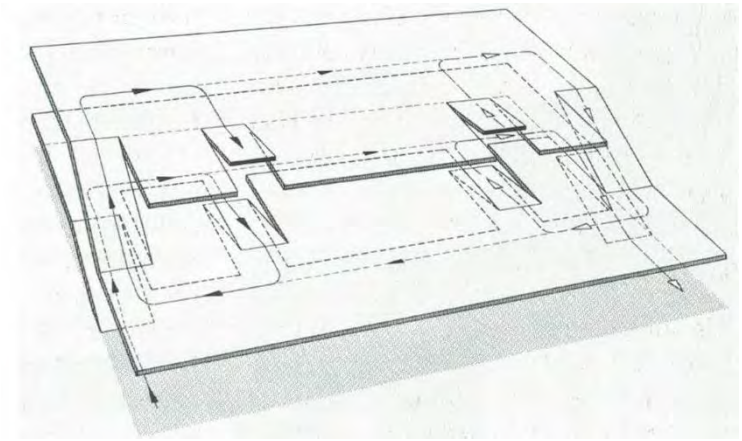


Abbildung 12: Split-Level-Garage mit Rampenkombinationen für Auf- und Abfahrt (Kleinmanns (2011), S. 51)

Die wichtigste Größe, die in der Parkhausplanung beachtet werden muss, ist die Stellplatzbreite. Laut ADAC wurde die Stellplatzbreite von 2,30 m in den Garagenverordnungen seit den 1970er Jahren nicht mehr verändert. Da die Autos mit der Zeit aber immer breiter wurden, hält der ADAC eine Stellplatzbreite von 2,50 m für sinnvoll (Allgemeiner Deutscher Automobil-Club e.V. (2014), S. 12).

Alle weiteren mindestens einzuhaltenden Abstände können aus der Garagen- und Stellplatzverordnung (GaStellV) entnommen werden.

3.2 Parkhäuser mit Tekla Structures

Die Firma Max Bögl plant wie viele andere Projekte auch ihre Systemparkhäuser mithilfe der BIM-Software Tekla Structures. Hierfür werden „[im] Stahlbetonbau [...] ausführungsreife 3D-Modelle mit Bewehrungsführung erstellt [...], wie auch im Stahlbau Schraub- und Schweißverbindungen dreidimensional geplant [...]“ (Borrmann et al. 2015, S. 493). Deswegen kann man diesen Tekla-Modellen einen LOD von ca. 400 oder höher zuweisen. Mit diesen Modellen können nun detaillierte Pläne erstellt werden, die für die Bauausführung benötigt werden. Außerdem werden die digitalen Daten an Maschinen zur Herstellung von Fertigteilen weitergegeben (Borrmann et al. 2015, S. 493). Dies macht eine maschinelle Fertigung von Bauteilen möglich, wodurch zum einen die Präzision der Herstellung erhöht, zum anderen auch noch Zeit gespart wird. Außerdem wird mithilfe dieser Modelle der gesamte Bauablauf geplant und überprüft (Trimble Solutions Germany 2018).

Im Folgenden wird noch auf die Software Tekla Structures eingegangen. Um eine Verwechslung zu vermeiden, werden die Komponenten bzw. die visuellen Elemente von

Grasshopper 3D im Folgenden gemäß der Definition von Dynamo im Kapitel 2.3.3 ab diesem Punkt als „Block“ bezeichnet.

Zunächst kann man mit Tekla Structures sehr einfache Strukturen wie Stützen, Träger, Platten oder auch Fundamente mit einer Reihe an Profilen oder Querschnitten modellieren. Diese kann man aus einem bestehenden Katalog auswählen. In Betonteilen kann man außerdem eine Bewehrung einfügen. Hierfür gibt es bereits vormodellierte Standardbewehrung, die sehr schnell eingefügt aber auch noch bearbeitet werden kann.

In Tekla Structures werden die Bauteile auch mithilfe von Referenzpunkten oder -kanten platziert. An diesen Punkten oder Kanten können die Bauteile ausgerichtet werden oder auch ein Versatz hergestellt werden. Es ist auch möglich, das Objekt mit dem Referenzpunkt als Ursprung zu drehen. Hierbei werden die Referenzkanten gedreht, wodurch sich alle Abstände an die Drehung anpassen. Damit kann die reine Geometrie bereits sehr flexibel gestaltet werden. Es werden aber auch detaillierte Informationen zu den jeweiligen Bauteilen hinterlegt, wie das Material oder Angaben zur Positionierung. Weitere Details, die außerdem mit Tekla Structures geplant werden können, sind Schweißnähte oder Betonierfugen.

Tekla Structures wirbt dafür, mit ihrer Software 3D-Modelle bis in das kleinste Detail modellieren und planen zu können (Tekla Structures). Möglich machen dies die sogenannten benutzerdefinierten Komponenten. Mit diesen Komponenten lässt sich auf der einen Seite ein komplexes Bauteil parametrisch modellieren, das unabhängig von anderen Bauteilen platziert werden kann. Dieses kann mithilfe von Referenzpunkten oder -kanten im freien Raum platziert werden. Auf der anderen Seite kann man ein bestimmtes Bauteil auch abhängig von anderen Bauteilen modellieren, wie z.B. ein Auflager eines Trägers an einer Stütze. Hierzu ist Parametrik zwingend notwendig, wenn das Profil der Stütze oder des Trägers variiert. Diese Komponenten passen sich dem übergebenen Bauteil mithilfe von Parametrik an.

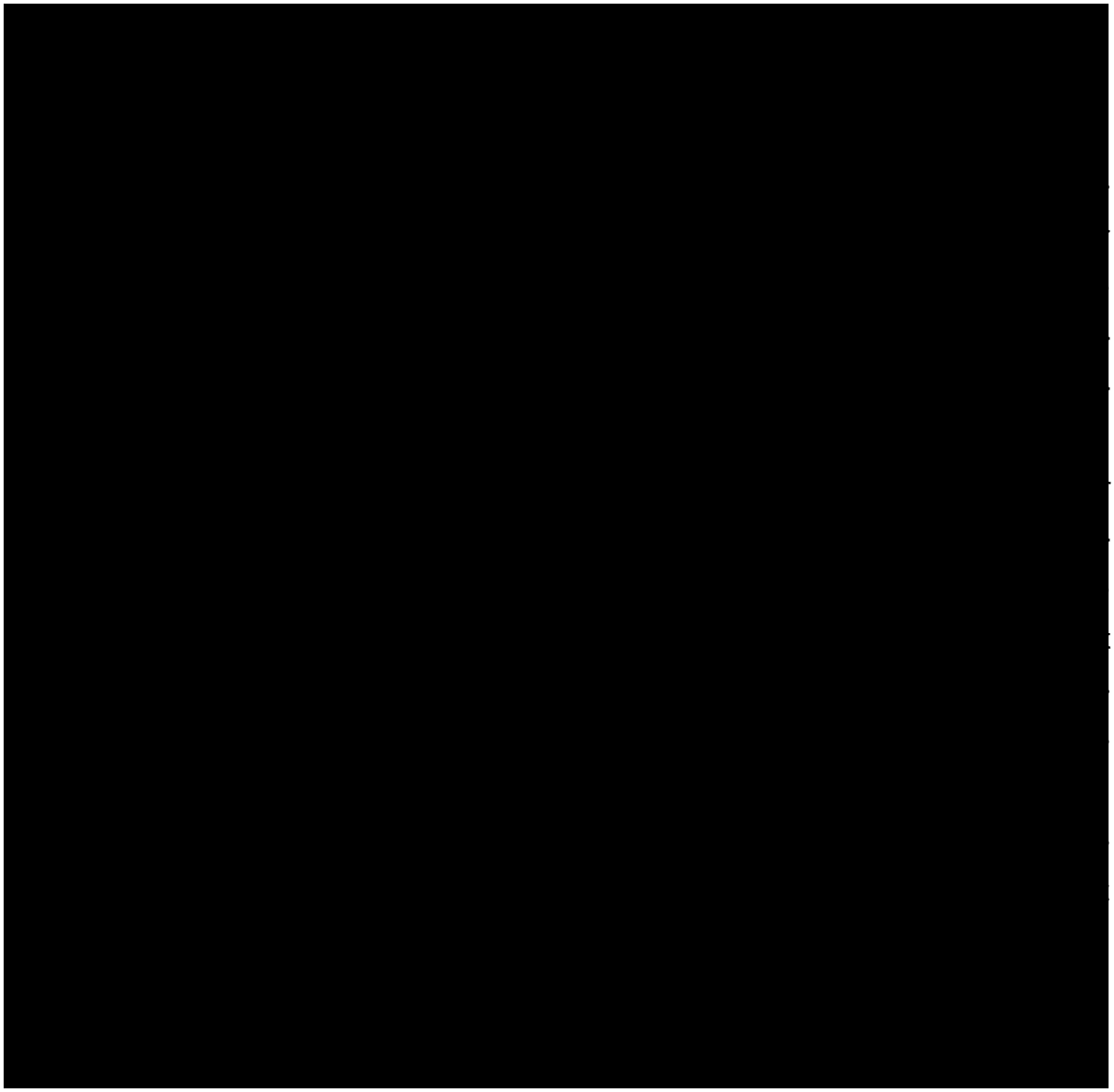
Es gibt bereits Komponenten, die von Tekla selbst in einem Katalog zur Verfügung gestellte werden. Ebenso gibt es benutzerdefinierte Komponenten, die der Benutzer selbst erstellen kann. Für diese Komponenten kann man Vorlagen anlegen, die bestimmte häufig vorkommende Werte bereits enthalten. Somit können Standardlösungen schnell geladen und verwendet werden.

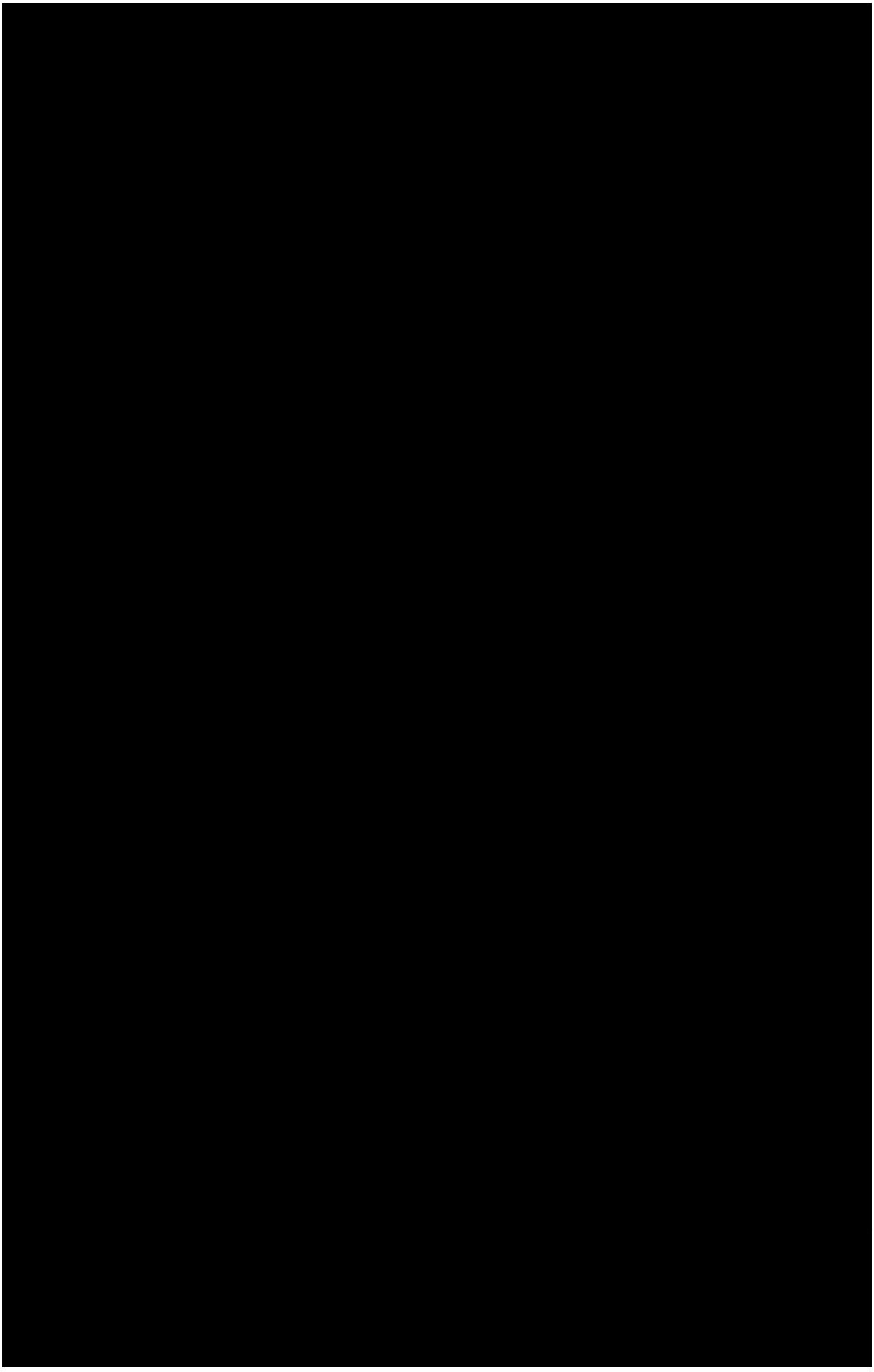
4 Fallstudie

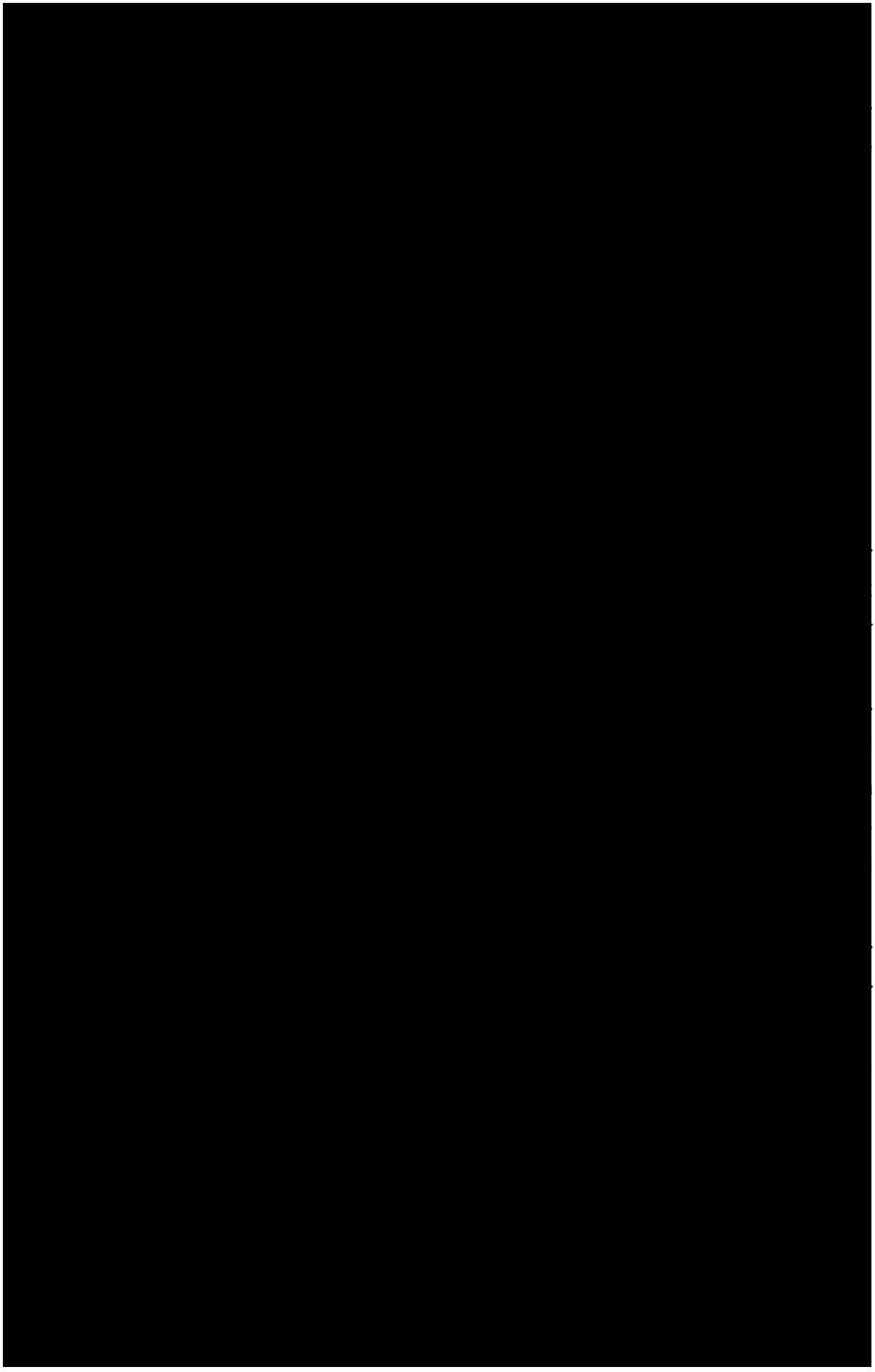
In diesem Kapitel wird zunächst beschrieben, auf welche bereits vorhandenen Firmenstrukturen oder vormodellierte Bauteile zurückgegriffen wird. Anschließend werden die Schwachstellen und Fehler bei der Anwendung eines vorherigen Generators analysiert und zusammen mit den Problemen bei der Benutzung der Schnittstelle zwischen Tekla Structures und Grasshopper 3D eine Lösung gesucht, einen Generator zu erstellen, der für die Anwender leicht zu bedienen ist.

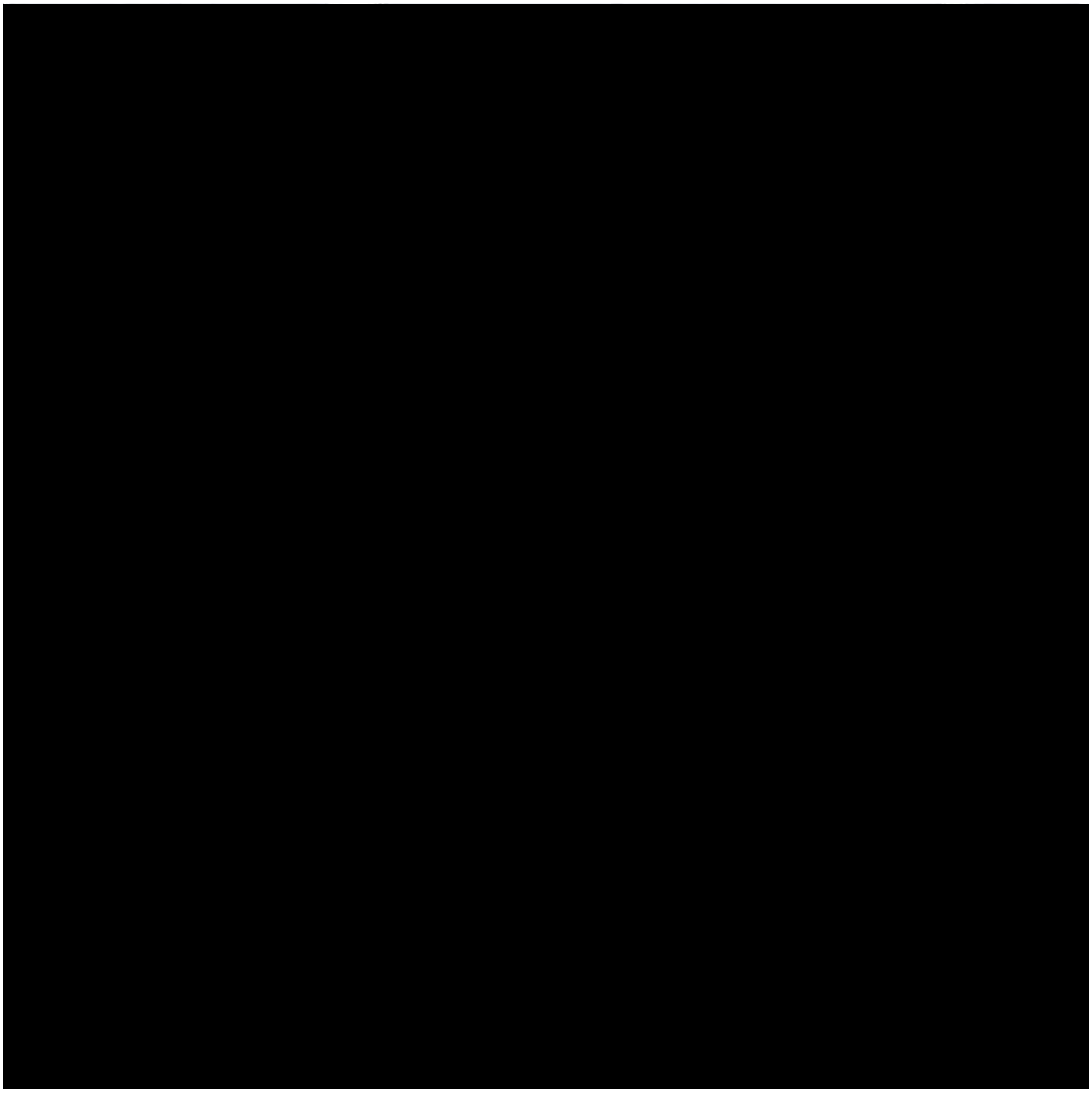
4.1 Ausgangssituation

4.1.1 Systemparkhäuser der Firma Max Bögl









4.1.3 Analyse des Dynamo-Skripts

In diesem Dynamo-Skript liegen noch nicht alle heute festgelegte Parameter vor, da das Bausystem noch nicht so weit entwickelt war. Damals war z.B. die lichte Weite der Deckenträger noch nicht auf einen Wert festgesetzt und die Rampen waren in ihren Abmessungen nicht definiert und variabel. Deswegen wurden beispielsweise für die Rampen noch keine eigenen Stützen aufgenommen und nur Aussparungen vorgesehen, die flexibel in Breite und Länge waren.

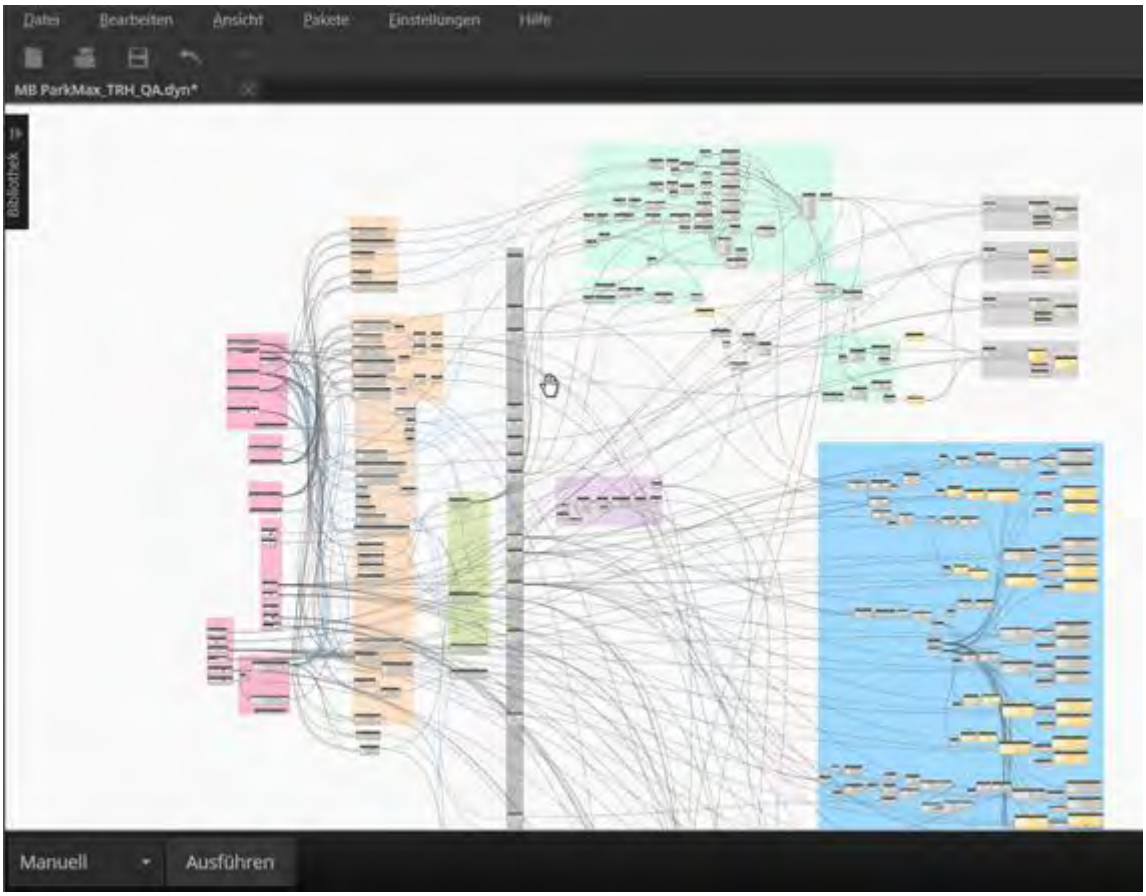


Abbildung 15: Bestehendes Dynamo-Skript

Des Weiteren musste der Nutzer das Skript selbst bedienen, in dem die Slider sehr langsam waren, was es sehr mühsam machte, alle Parameter richtig einzustellen. Außerdem gab es kleinere Fehler in der Modellerstellung, wie falsch positionierte Stützen. Von den Treppenhäusern an der Längsseite konnte man nur eine begrenzte Anzahl einfügen, was die Flexibilität des Generators einschränkt. Manche Details mussten manuell ergänzt oder nachbearbeitet werden, da es unter Umständen nicht enthalten war oder falsch erzeugt wird, wie die Geländer in den Treppenhäusern. Dieser Fehler wurde zwar den Anwendern kommuniziert, allerdings lindert das die Präzision und das Vertrauen des Nutzers in den Generator. Für manche Fälle war es für den Nutzer sogar nötig, Kanäle umzulegen, damit ein bestimmter Parameter richtig umgesetzt wird. Den Benutzer dadurch zu tief in das Dynamo-Skript einzuführen, war der schwerwiegendste Fehler.

Meine Zusammenarbeit mit der zuständigen Abteilung hat gezeigt, dass jemand, der noch nie mit einer visuellen Programmiersprache oder geschweige denn mit einer textuellen gearbeitet hat, zunächst einmal geschockt von dem natürlich auftretenden Spaghetti-Effekt des Skripts ist. Muss der Nutzer dann noch aktiv in diesen eindringen,

weckt das natürlicherweise auch Anwendungsängste. Außerdem musste bei der Inbetriebnahme der Schnittstelle von Revit zu Dynamo so manches beachtet werden. Die zuständige Abteilung hatte zudem nie einen Bedarf für einen Generator gesehen. Damit zu arbeiten wurde auch infolge der schweren Nutzung eher als Mehraufwand empfunden. Deswegen ist es wichtig, von Anfang an mit der zuständigen Abteilung zusammenzuarbeiten und ihnen am Anfang auch das Potential vor Augen zu führen. Dadurch kann eine Motivation erreicht werden, die dem Projekt förderlich ist. Denn somit will die Abteilung auch am möglichen Erfolg mitwirken.

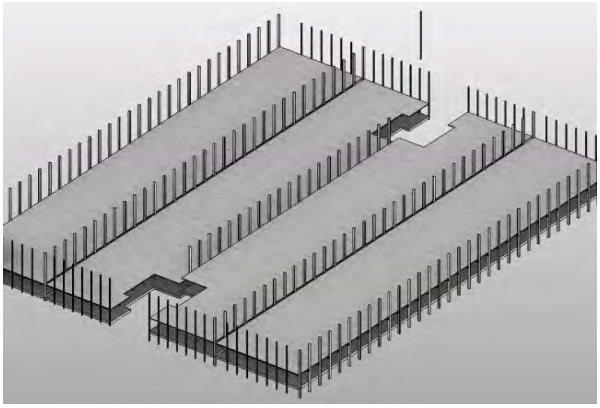


Abbildung 16: Mit dem Dynamo-Skript erstelltes Revit Modell

Zusammenfassend kann man folgende Fehlerquellen lokalisieren:

- Aktiver Eingriff in das Skript
- Fehler in der Modellerstellung
- Nötige manuelle Nachbearbeitung erstellter Bauteile
- Weniger weit ausgereiftes Bausystem
- Geringere Flexibilität der besonderen Bauteile
- Langsame Slider
- Aufwändige Inbetriebnahme

Diese Fehler gilt es zu vermeiden oder zu beheben, um die Anwendbarkeit eines Generators zu erreichen. Denn eine gute Idee setzt sich nicht zwangsläufig durch, sie muss auch gut umgesetzt werden. Hier ist die Zusammenarbeit mit den zuständigen Personen oder Abteilungen notwendig.

4.2 Anwendung von Grasshopper 3D

In diesem Kapitel sollen die Stärken und Schwächen der Anwendung von Grasshopper 3D selbst und dessen Schnittstelle zu Tekla Structures analysiert werden, um anschließend Lösungsansätze aufzuzeigen, die eine Erstellung des Generators überhaupt erst ermöglichen.

4.2.1 Grundlegende Funktionsweise

Ein großer Vorteil von Grasshopper 3D besteht darin, dass in einer Baumstruktur gerechnet wird. Prinzipiell ist die Baumstruktur in Pfade oder Äste aufgebaut, der jeweils einen Eintrag oder eine Liste enthält. Die Pfade werden nummeriert, wobei der Index innerhalb von geschwungenen Klammern steht. Die Einträge in diesen Pfaden bleiben i.d.R. in demselben Pfad enthalten nach durchlaufen eines Blocks, wobei sich die Nummer des Pfades jedoch ändern kann. Diese Einträge sind in Listen angelegt, die genauso durchnummeriert werden. Die Indizes der Listen beginnen allerdings immer bei null. Dies lässt sich auch nicht ändern. Beim Durchlaufen von Blöcken wird jeder einzelne Eintrag in jedem Pfad mit der in dem Block hinterlegten Operation bearbeitet. Hierbei gibt es allerdings Besonderheiten, die bekannt sein müssen. Diese sollen nun anhand eines Beispiels erklärt werden.

Als Eingangsdaten wird eine Baumstruktur bestehend aus zwei Listen übergeben. Diese Listen sind unterschiedlich lang. Als erstes wird ein Zahlenwert hinzuaddiert. Dieser Zahlenwert (im Beispiel die Zahl 2) wird zu jedem Eintrag in jedem Pfad bzw. jeden Ast des Baumes hinzuaddiert. Im zweiten Beispiel wird eine Liste mit nur zwei Einträgen dem bestehenden Input hinzuaddiert. Hier wird der erste Eintrag in jedem Pfad nur zum ersten Eintrag des Inputs am Eingang A hinzuaddiert. Ab dem zweiten Eintrag wird der letzte Eintrag des Inputs am Eingang B verwendet, da die Liste am Eingang B weniger Einträge hat. Hat man am Eingang B nun auch eine Baumstruktur, die der am Eingang A entspricht, wird jedem Eintrag der Baumstruktur am Eingang A auch nur derjenige Eintrag am Eingang B hinzuaddiert, der sowohl den gleichen Pfad als auch den gleichen Listenindex aufweist.

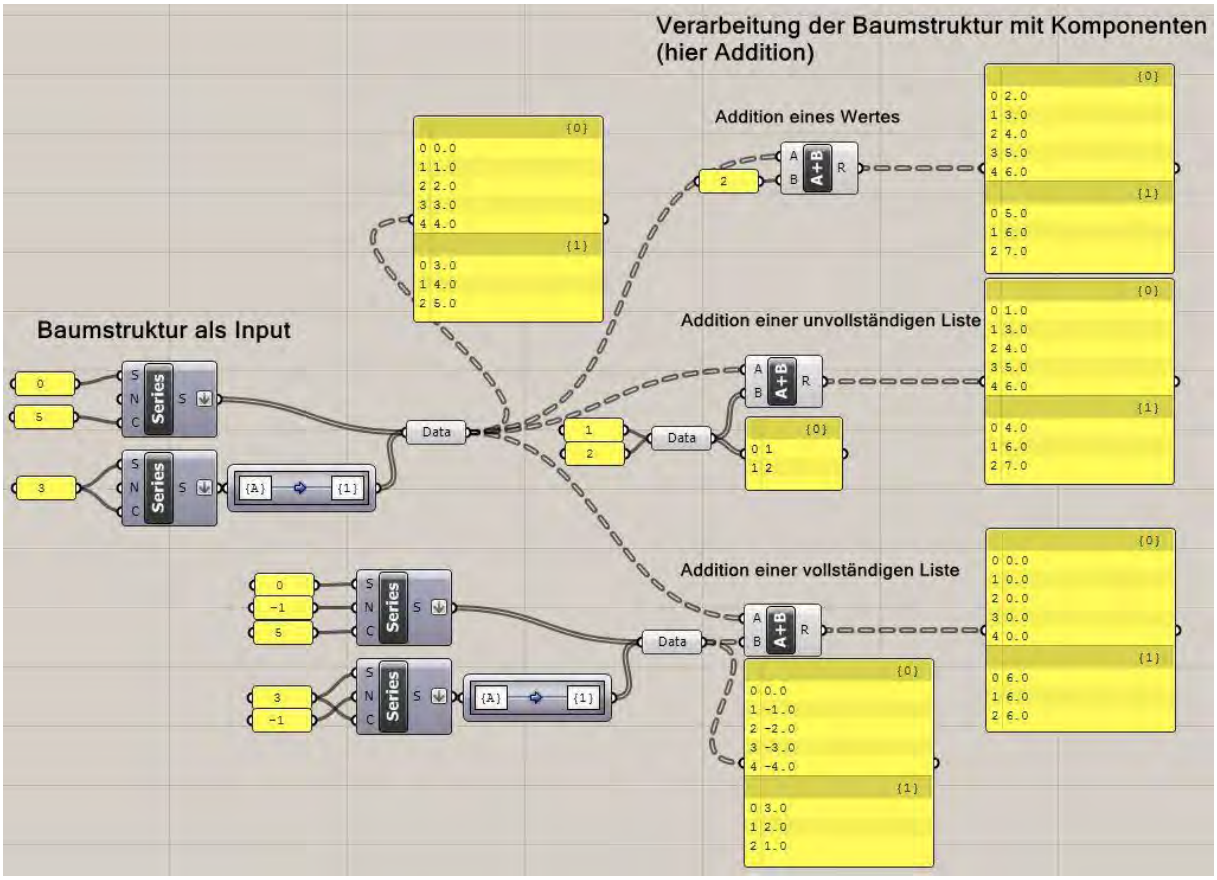


Abbildung 17: Beispiel Baumstruktur

Durch diese Baumstruktur kann eine unbestimmte Anzahl an Eingangsdaten getrennt in Pfaden verarbeitet werden. Außerdem werden deswegen i.d.R. keine Schleifen über mehrere Blöcke mehr benötigt.

Für die Benutzereigabe besteht genau wie in Dynamo die Möglichkeit, Slider zu verwenden. Leider muss man den Wertebereich dieser Slider manuell anpassen, was beispielsweise bei der flexiblen Länge des Parkhauses ein Problem darstellt. Dies kann man umgehen, indem man den Slidern den Wertebereich zwischen 0 und 1 zuordnet und auf den tatsächlichen Wertebereich umrechnet. Dieser umgerechnete Wert kann dann mit einem Fenster angezeigt werden. Allerdings sind dadurch präzise Eingaben mit der Tastatur nicht mehr möglich.

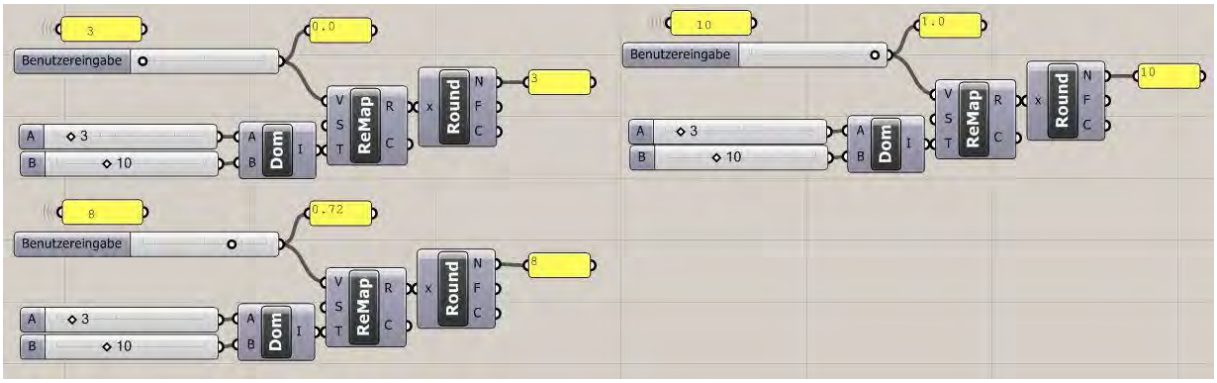


Abbildung 18: Slider mit variablem Wertebereich

Außerdem werden die Slider mit zunehmender Größe des Skripts bzw. der an den Slider anschließenden Blöcken zunehmend langsamer. Bei großen Skripten wie bei der Abbildung von Bausystemen ist es deswegen mühsam, den Regler an eine gewünschte Position zu verschieben. Gerade hier wäre eine Eingabe mit der Tastatur nötig, um exakte Werte einzugeben. Zudem kann je Slider nur ein Wert ausgegeben werden. Wird eine unbestimmte Anzahl an Werten benötigt, müsste für jeden Wert ein neuer Slider angelegt werden und dessen Wertebereich definiert. Natürlich kann eine gewisse Anzahl bereits zur Verfügung gestellt werden, allerdings kostet dies Platz und deckt nicht jeden Fall ab. Eine solch flexible Benutzereingabe könnte über Schleifen realisiert werden. Da in datenflussbasierten Programmiersprachen aber keine Schleifen möglich sind, und von Grasshopper 3D sogar verboten werden, ist eine derartige Realisierung nicht möglich. Grasshopper 3D ist deswegen zwar für die Verarbeitung einer unbestimmten Anzahl an Daten prädestiniert, allerdings ist dies so nicht vereinbar mit der statischen Benutzereingabe.

Außerdem lässt sich der Spaghetti-Effekt auch hier nicht vermeiden. Dennoch kann der Entwickler selbst Überblick über sein Skript behalten, indem bestimmte Bereiche gruppieren werden können. Diese Gruppe bestehend aus mehreren Blöcken und Schriftzügen, wird dann mit einem Rahmen umhüllt wobei man noch die Farbe dieses Bereichs festlegen kann. So wird jeder Prozess für sich betrachtet, auch wenn die einzelnen Eingabeparameter aus anderen Bereichen stammen. Um sich zusätzlichen Überblick zu verschaffen, kann man an beliebiger Stelle einen Text einfügen, um bestimmte Bereiche oder auch Vorgehensweisen innerhalb der Bereiche zu beschreiben. Dadurch kann man zumindest die Struktur gliedern, wobei es dennoch manchmal schwerfällt, diese Verbindungen zu verfolgen, gerade bei sehr vielen sich überschneidenden Verbindungen. Es werden zwar die Kanten markiert, die zu einem ausgewählten Block hin- oder wegführen, allerdings nur sehr schwach. Würde man diese noch

stärker farblich hervorheben, würde sich dies sicherlich positiv auf die Übersichtlichkeit auswirken. Geht man noch eine Ebene tiefer, so kann man auch die einzelnen Blöcke noch zusätzlich strukturieren, indem man sie auf eine bestimmte Art und Weise positioniert und beschriftet.

4.2.2 Schnittstelle zwischen Grasshopper 3D und Tekla Structures

Um die Schnittstelle zwischen Grasshopper 3D und Tekla Structures herzustellen, kann ein Plugin verwendet werden, das in Grasshopper 3D installiert werden muss. In diesem Plugin sind verschiedene Blöcke enthalten, mit denen Strukturen in Tekla Structures erzeugt und deren Attribute gesteuert werden können. Um Bauteile zu platzieren, werden durch in Grasshopper erzeugte Punkte oder Linien bzw. Kurven die Reverenzpunkte oder -kanten der Bauteile erstellt. Um die Bauteile relativ zu diesen Punkten platzieren zu können, kann bei der Verwendung von normalen Profilen eine im Plugin zur Verfügung gestellter Block verwendet werden. Hier kann man auch einen Versatz des Bauteils zu der Kante durch Angabe eines numerischen Wertes herstellen. Bei der Platzierung von Komponenten erfolgt dies textuell über den Eingang für die Eigenschaften der Bauteilattribute.

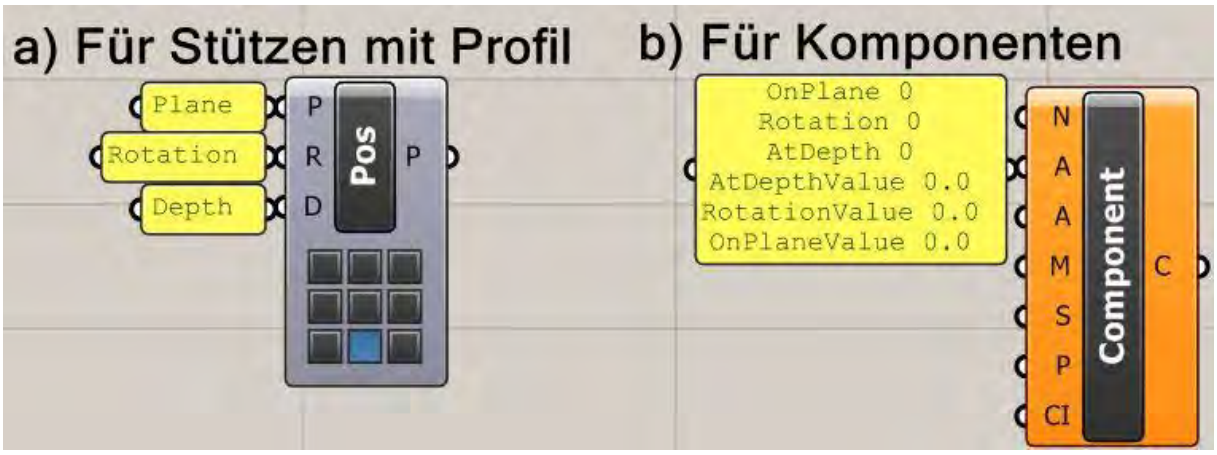


Abbildung 19: Relative Positionierung von Profilen und Komponenten

In Tekla modellierte Komponenten werden entweder mithilfe von Referenzpunkten, Bauteilen oder einer Kombination der beiden platziert. Wie die Komponente platziert wird, wird beim Erstellen der benutzerdefinierten Komponenten festgelegt. Die Übergabe dieser Referenzpunkte und Bauteile muss in einer bestimmten Baumstruktur übergeben werden.

Um die Eigenschaften oder Attribute der erzeugten Bauteile zu steuern, kann der Name und der Wert der Attribute in Textform getrennt durch Leerzeichen an den

Grasshopper-Block übergeben werden. Mehrere Namen und Werte können aufgelistet werden oder durch Leerzeilen bzw. Semikolons voneinander getrennt werden.

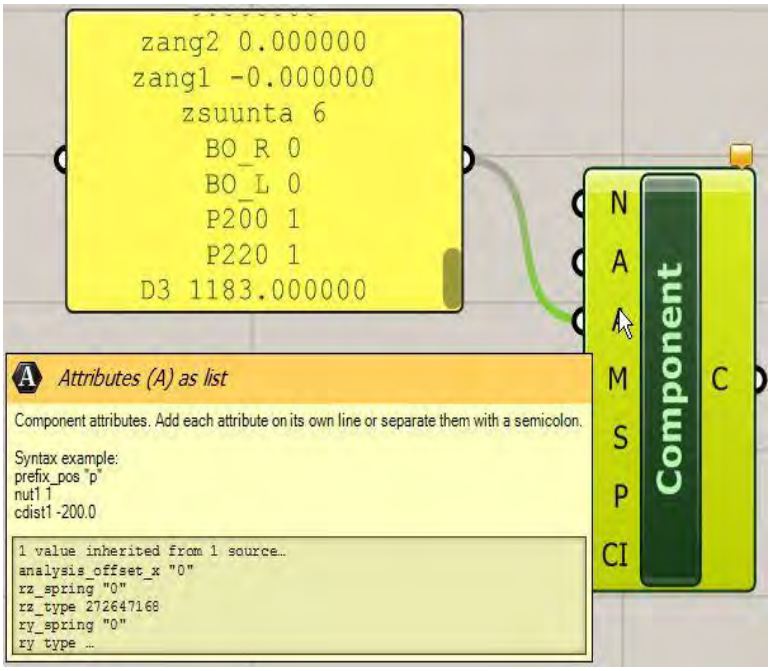


Abbildung 20: Übergabe mehrerer Attribute an die Komponente getrennt durch Leerzeilen

Für viele Komponenten wurden bereits Vorlagen angelegt, um häufig auftretende Bauteileigenschaften schnell laden zu können. Um diese Vorlagen zu verwenden, muss zunächst der Ordner „attributes“ geöffnet werden, der sich im jeweiligen Modellordner befindet. Hier sind alle Vorlagendateien abgespeichert. Diese Vorlagen kann man mit einem beliebigen Textverarbeitungsprogramm auslesen. Der Name und der zugehörige Wert der Attribute befinden sich dabei in jeder Zeile hinter einem Punkt.

```
Winkelanschluss Deckenträger - Stütze_attributes.zang2 0.000000
Winkelanschluss Deckenträger - Stütze_attributes.zang1 -0.000000
Winkelanschluss Deckenträger - Stütze_attributes.zsuunta 7
Winkelanschluss Deckenträger - Stütze_attributes.S10_diameter 16.000000
Winkelanschluss Deckenträger - Stütze_attributes.S1_diameter 16.000000
Winkelanschluss Deckenträger - Stütze_attributes.S3 "2*140"
Winkelanschluss Deckenträger - Stütze_attributes.S12 "220"
Winkelanschluss Deckenträger - Stütze_attributes.S11 70.000000
```

Abbildung 21: Auszug aus einer Vorlagendatei geöffnet mit Word

Hier sind auch finnische Wörter vertreten (z.B. finn. „zsuunta“; Bedeutung: „z-Richtung“), welche schon im Programm für jede Komponente hinterlegt sein können, die man nur mithilfe der Vorlagendateien identifizieren kann. Es ist außerdem sehr mühsam, jeden Namen und Wert der Attribute aus seitenlangen Vorlagendateien herauszukopieren, deswegen ist hier eine Hilfestellung für den Entwickler angebracht.

Ein weiterer wichtiger Punkt ist der Strukturverlust der Ausgangsdaten bei Tekla-Blöcken. Blöcke zur Erzeugung von Bauteilen in Tekla Structures können ebenfalls Ausgangsdaten generieren. Damit können erzeugte Strukturen zur Erzeugung weiterer Bauteile weitergegeben werden. Allerdings wird dabei die erzeugte Baumstruktur zerstört und nur eine Liste ausgegeben, deren Länge gleich der Anzahl der erzeugten Bauteile entspricht. Da diese Struktur aber in den meisten Fällen noch benötigt wird, muss diese erst wiederhergestellt werden.

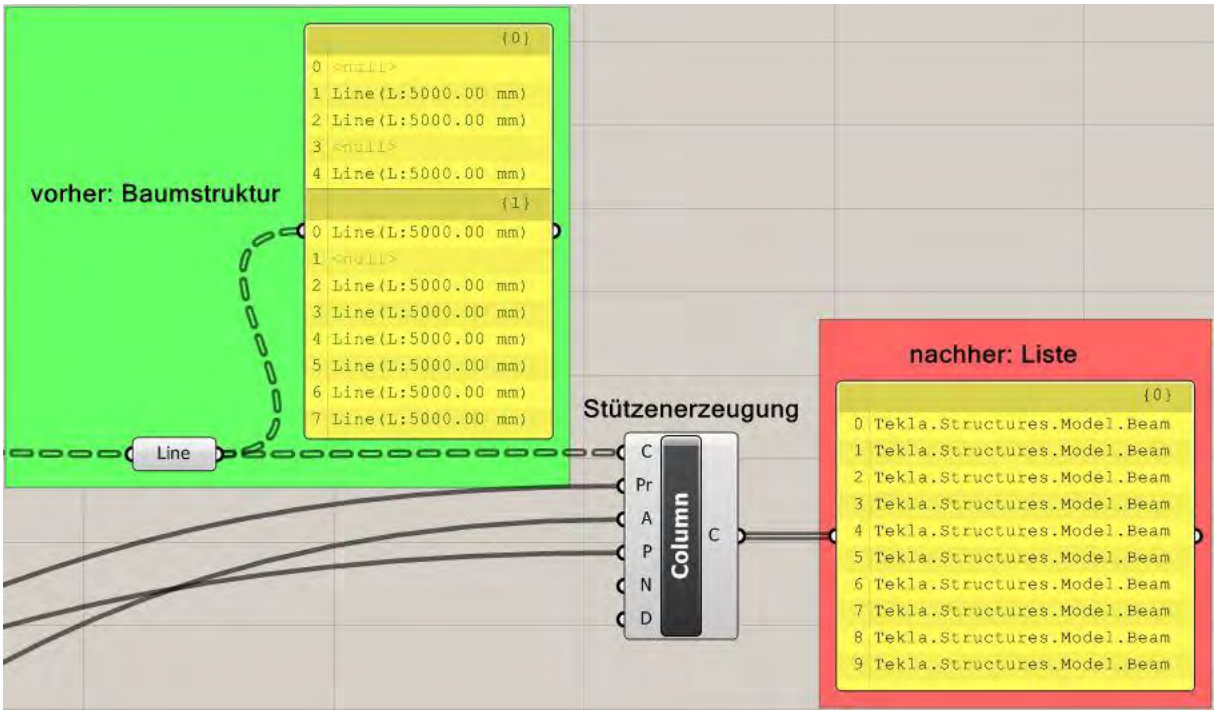


Abbildung 22: Strukturverlust der Ausgangsdaten

Außerdem werden Blöcke, die keine direkte Verbindung miteinander haben, werden parallel ausgeführt, sobald alle benötigten Eingangsinformationen vorliegen. Dies führt zu Problemen bei der Erzeugung von Komponenten in Tekla Structures. So weisen beispielsweise als Komponenten modellierte Stützen willkürliche Höhen oder Tiefen bei paralleler Erzeugung auf.

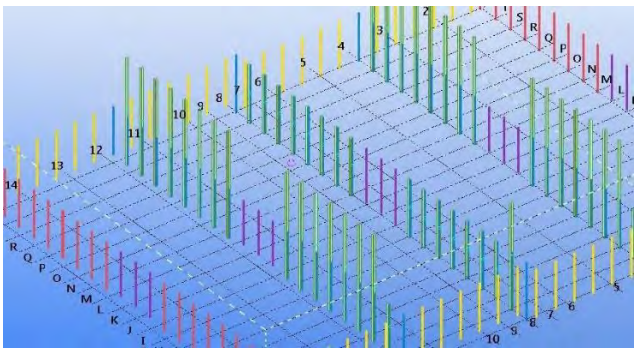


Abbildung 23: Willkürliche Höhen bei paralleler Erzeugung der Stützen

Erstellt man eine Komponente manuell in Tekla Structures, so benötigt sie je nach Komplexität eine bestimmte Zeit. Manche Komponenten werden augenblicklich erstellt, andere benötigen bis zu zwei Sekunden. Genau diese Zeit wird auch benötigt, um aus Grasshopper 3D heraus eine Komponente zu erstellen. Die Rechnung innerhalb von Grasshopper 3D ist sehr schnell und geschieht fast augenblicklich. Nur die Übertragung der Daten und die Erstellung des Modells nimmt etwas mehr Zeit in Anspruch. So kann es sein, dass für die Erstellung eines bestimmten Bauteiltyps bis zu 15 Minuten benötigt werden.

Eine weitere positive Eigenschaft der Schnittstelle ist jedoch die Möglichkeit, Bauteile über Grasshopper 3D zu positionieren (nummerieren) und die Bauteile einem Teilsystem zuzuweisen. Dies ist wichtig für die spätere Planerstellung. Im Betonbau lassen sich sogar Betonierabschnitte u.Ä. definieren.

4.2.3 Lösungsansätze

Zusammenfassend konnten also folgende strukturelle Schwachstellen identifiziert werden:

- Keine Schleifen für die Nutzereingabe in Grasshopper 3D möglich
- Aufwändiges manuelles Auslesen der Bauteilattribute aus langen Vorlagelisten
- Strukturverlust der Ausgangsdaten bei Blöcken, die Strukturen in Tekla Structures erzeugen
- Fehlerhafte Komponentenerstellung bei paralleler Erzeugung

Für die Umgehung dieser Schwachstellen werden nun Lösungsansätze entwickelt.

Keine Schleifen für die Nutzereingabe in Grasshopper 3D möglich

In Grasshopper können keine Schleifen für die Benutzereingabe benutzt werden, wie z.B. für die Eingabe einer unbestimmten Anzahl an Rampen. Hier tritt die Verbindung zu Excel ins Spiel, die mit ihrer zugrunde liegenden Programmiersprache Visual Basic for Applications (VBA) eine hervorragende Basis bietet, beliebig viele Benutzereingaben über Schleifen vom Benutzer abzufragen. Diese Eingaben können dann über die Verbindung von Grasshopper 3D zu Excel abgefragt werden und über die Baumstruktur von Grasshopper 3D weiterverarbeitet werden. Sie ergänzt also die Schwachstelle von Grasshopper, keine Schleifen zuzulassen. Hierfür kann eine Eingabemaske erstellt werden, die zudem falsche Benutzereingaben durch klar definierte Wahlmöglich-

keiten ausschließt. Außerdem kann dem Nutzer somit ein ästhetisches Layout präsentiert und mögliche Auswirkungen von Veränderungen der Parameter bereits visualisiert werden. Außerdem kann direkt aus der Eingabemaske heraus das Skript geladen werden und weitere Operationen durchgeführt werden, wodurch diese Tabelle zu einem zentralen Steuerelement des Generators wird. Der Kontakt zu dem Skript in Grasshopper 3D wird somit weitestgehend vermieden.

Es gibt verschiedene Möglichkeiten, Daten aus Excel in Grasshopper 3D einzulesen. Hierfür gibt es mehrere Plugins, wie das Plugin „excelreadwrite“. Hiermit kann man aus einer Excel-Tabelle mit dem Block „ExcelStaticRead“ oder „ExcelDynamicRead“ den Inhalt von Zellen aus einer Excel-Tabelle auslesen. Es gibt außerdem den Block „ExcelWrite“, mit der man Daten in eine Excel-Tabelle schreiben kann. Hier muss der Pfad der Excel-Tabelle mit angegeben werden, um einen eindeutigen Bezug zu schaffen. Die Auslesung aus den Zellen erfolgt durch Angabe des Bereiches, wie z.B. „A2:A4“ für die zellen A2 bis A4. Allerdings muss die Excel-Tabelle vorher geschlossen sein. Da alle notwendigen Prozesse aus Excel heraus gestartet werden, kann dieser Block nicht verwendet werden, obwohl er einen sicheren Bezug zur Tabelle herstellen würde.

Außerdem gibt es noch ein viel umfassendes Plugin namens „Lunchbox“. Hier sind neben vielen anderen Funktionen auch noch zwei Arten einer Excel-Verbindung enthalten. Der eine Block namens „Excel Reader“ bzw. „Excel Write“ hat den entscheidenden Nachteil, dass sie nur ganze Tabellenblätter auslesen kann. Die zweite Variante namens „Excel Reader LEGACY“ bzw. „Excel Write LEGACY“ (engl. legacy: Hinterlassenschaft, Erbe), liest Daten aus definierten Zellen aus einer beliebigen geöffneten Excel-Tabelle aus. Hier muss allerdings beachtet werden, dass keine weitere Excel-Datei vor dem Start geöffnet ist. Hier kann die jeweilige Zelle durch Angabe der Zeilen- und Spaltenindizes ausgelesen werden., weswegen Zellbezüge flexibel durch Angabe von numerischen Werten hergestellt werden können.

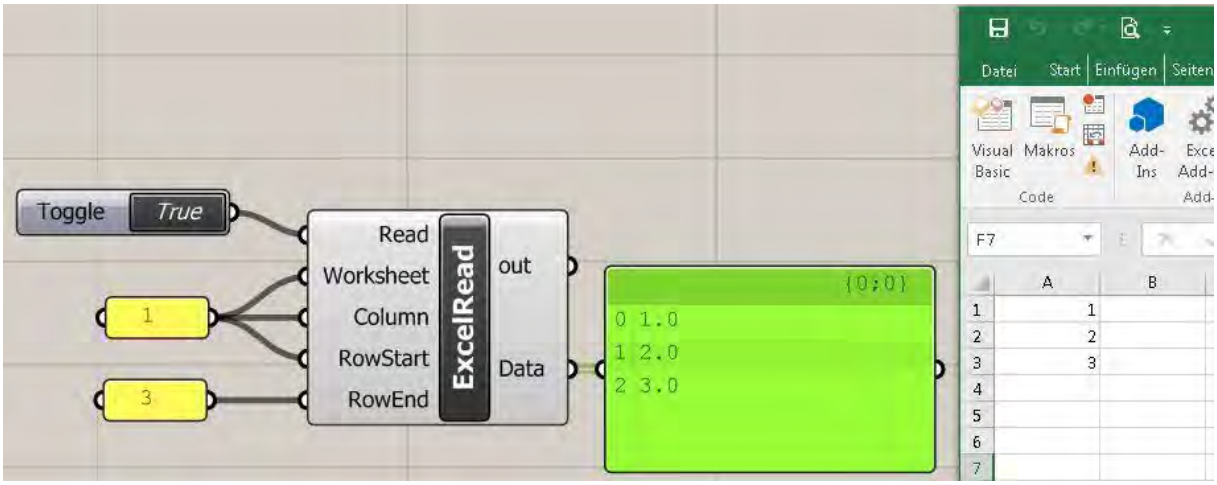


Abbildung 24: Auslesung von Daten durch "Excel Reader LEGACY"

Da das Grasshopper-Skript direkt aus Excel heraus geöffnet wird und flexible Zellbezüge möglich sind, wurde in dieser Arbeit die Variante „Excel Reader LEGACY“ verwendet. Die flexiblen Zellbezüge werden v.a. wegen der unbestimmten Anzahl an Rampen benötigt. Um sicherzustellen, dass keine weitere Excel-Tabelle geöffnet ist, muss der Nutzer vor der Modellerzeugung noch in der Eingabemaske dazu aufgefordert, alle weiteren Excel-Tabellen zu schließen.

Aufwändiges manuelles Auslesen der Bauteilattribute aus langen Vorlagelisten

Um die Attribute schnell aus Vorlagendateien auszulesen, kann eine Word-Datei mit Makros angelegt werden, in der sich eine Schaltfläche in der ersten Zeile befindet. Unterhalb dieser Zeile kopiert man den Text der Vorlage und aktiviert die Schaltfläche. Dabei werden nun alle Zeichen bis nach dem Punkt gelöscht, wobei nur noch die Namen und die Werte der Attribute bestehen. Diese Zeilen können anschließend kopiert, in ein Textfeld in Grasshopper 3D eingefügt und an die entsprechende Komponente angeschlossen werden.

Abbildung 25: Attributname und -wert nach Bearbeitung

Strukturverlust der Ausgangsdaten bei Blöcken, die Strukturen in Tekla Structures erzeugen

Die Ausgangsdaten der Blöcke, die Bauteile in Tekla Structures erzeugen, werden in einer Liste ausgegeben. Die vorherige Baumstruktur wird dabei zerstört und leere Einträge in Listen nicht beibehalten. Diese Struktur ist für manche Prozesse allerdings notwendig, weswegen sie wiederhergestellt werden muss. Um in der ausgegebenen Liste die leeren Einträge an den richtigen Stellen einzufügen, ist es notwendig, die genaue Funktionsweise des verwendeten Blocks „Insert Items“ zu kennen und zu umgehen, der in eine Liste an einem bestimmten Index Werte einfügt. Diese Indizes stimmen nur mit der Liste am Eingang überein. Betrachtet man die Liste nachdem sie den Block durchlaufen hat, befinden sich die Indizes nicht an der gewünschten Stelle, da die Werte zwischen den Indizes einfach direkt nach dem angegebenen Index angeführt werden. Um das gewünschte Ergebnis zu erzielen, müsste man die Liste der Indizes sortieren, und von jedem Eintrag dieser Liste seinen eigenen Listenindex abziehen. Dann würden genau an den gewünschten Stellen ein leerer Eintrag stehen.

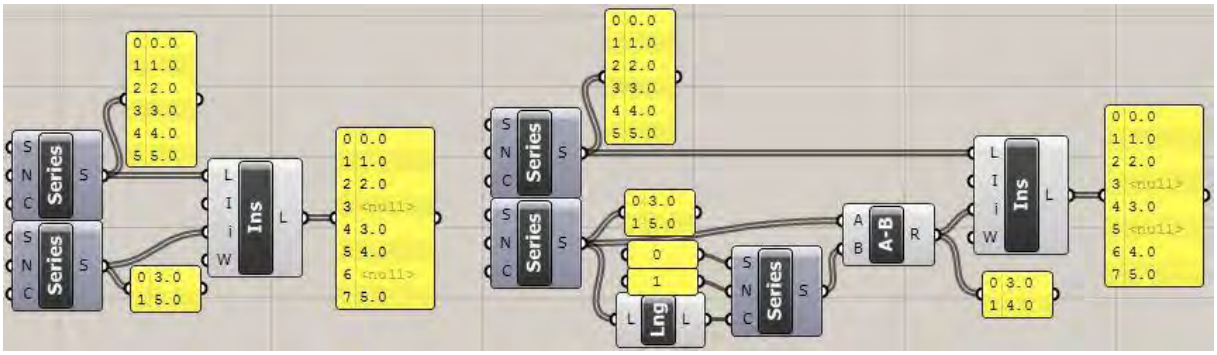


Abbildung 26: Umgang mit dem „Insert Items“-Block

Diese Aufbereitung der Listenindizes ist die Schlüsselfunktion für die Wiederherstellung der Datenstruktur. Das Einfügen von Daten bei bereits bestehender Datenstruktur ist dagegen mit einem einzigen Block namens „Insert Items“ möglich.

Damit kann die vorher bestehende Struktur wiederhergestellt werden und Listen aus verschiedenen Bauteilen zusammengesetzt werden.

Fehlerhafte Komponentenerstellung bei paralleler Erzeugung

Die Fehler in der Komponentenerzeugung, die beim parallelen Ausführen der Blöcke auftreten, werden vermieden, indem man Abhängigkeiten der betroffenen Grasshopper-Blöcke erzeugt, um somit eine implizite Ausführungsweise zu erzwingen. Das wird erreicht, indem man die Eingangsdaten eines Blocks mithilfe einer booleschen Variable mit Wert „False“ blockiert, solange bis die Erzeugung des vorherigen Blocks abgeschlossen wurde. Dann wird der Wert der Variable auf „True“ gesetzt, wodurch Eingangsdaten weitergegeben werden können. Der Block, die diese Variable erzeugt, wurde in einem Python-Skript realisiert, das auf der Abbildung als „Verzögerer“ beschriftet wurde.

Es gibt allerdings noch einen Spezialfall: Manchmal kann und soll ein Block keine Ausgangsdaten generieren, falls dieser Block z.B. gerade nicht benötigt wird. Dieser Fall tritt häufig auf, da z.B. für das Parkhaus eine unterschiedliche Anzahl an Schiffen möglich ist. Wenn sie aber keine Ausgangsdaten generiert, würde sie damit die Kette unterbrechen, da der Verzögerer immer „False“ aussenden würde. Dies wird dadurch umgangen, dass man die Eingangsdaten des Blocks und die boolesche Variable des Vorangehenden berücksichtigt. Gibt es für den Block keine Eingangsdaten und ist der Wert der Variable der vorangehenden Operation schon auf „True“ gesetzt, bedeutet das, dass der Block keine Daten generieren wird. Dann wird die Variable auf „True“

gestellt, und die auszulassende Operation sozusagen übersprungen. Dies betrifft allerdings nur die Grasshopper-Blöcke, die für die Erzeugung zuständig sind. Die vorherigen Operationen, die die Eingangsdaten produzieren, werden ohne eingeführter Unterbrechung berechnet. So wird eine Serienschaltung der Komponentenerzeugung möglich und alle Komponenten können in Tekla Structures richtig erzeugt werden.

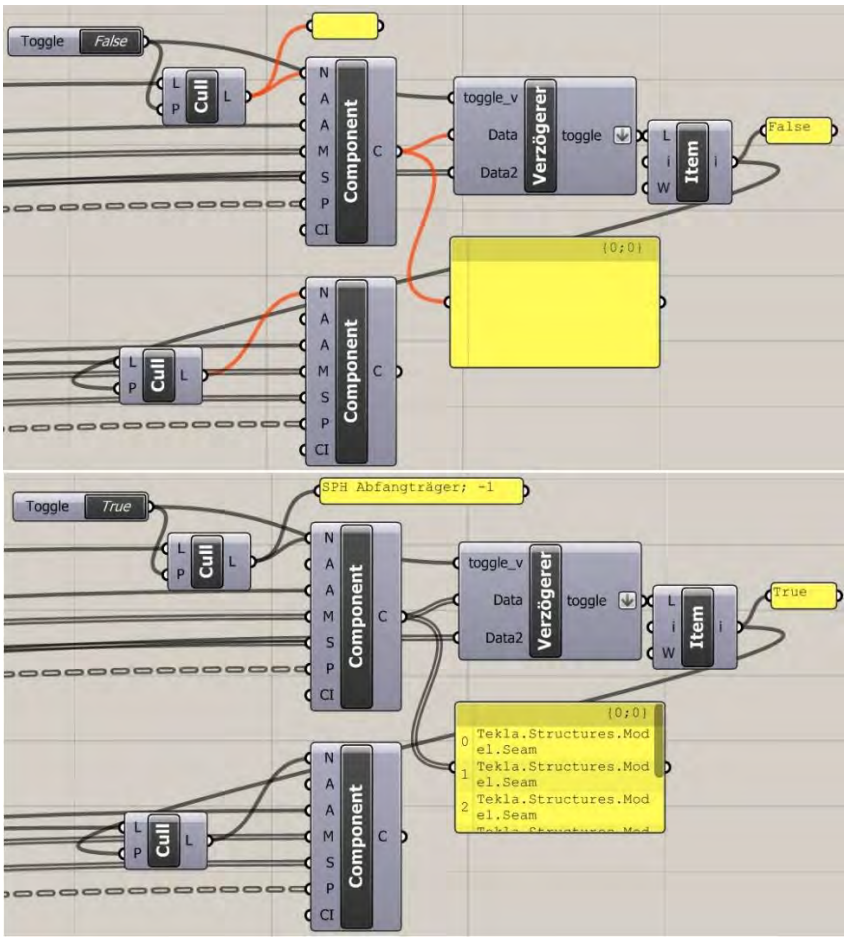


Abbildung 27: Erzeugung einer impliziten Ausführungsweise mit dem Python-Skript „Verzögerer“

Durch diese Lösungsansätze ist eine präzise fehlerlose Modellerstellung mithilfe von Grasshopper 3D möglich.

4.3 Umsetzung des Generators

In diesem Kapitel wird nun die genaue Umsetzung des Modellgenerators und die benötigten Arbeitsschritte des Nutzers beschrieben. Hier wird v.a. auf Aspekte der Benutzerfreundlichkeit eingegangen.

Mithilfe einer gezielt angelegten Ordnerstruktur können Querbeziehungen zwischen den einzelnen Dateien sehr einfach hergestellt werden. Denn der absolute Pfad des obersten Ordners kann von jedem Ort aus ermittelt werden, der sich unterhalb dieses

Ordners befindet. Die relativen Bezüge zwischen den einzelnen Dateien kann dann ebenfalls jederzeit ermittelt werden, da diese untere Ordnerstruktur bestehen bleibt. I.d.R. wird dabei auf die Bearbeitung von Strings zurückgegriffen. Dies macht es möglich, den Ordner „Parkhausgenerator“ überall hin zu verschieben, ohne die Funktionsfähigkeit des Generators zu beeinträchtigen.

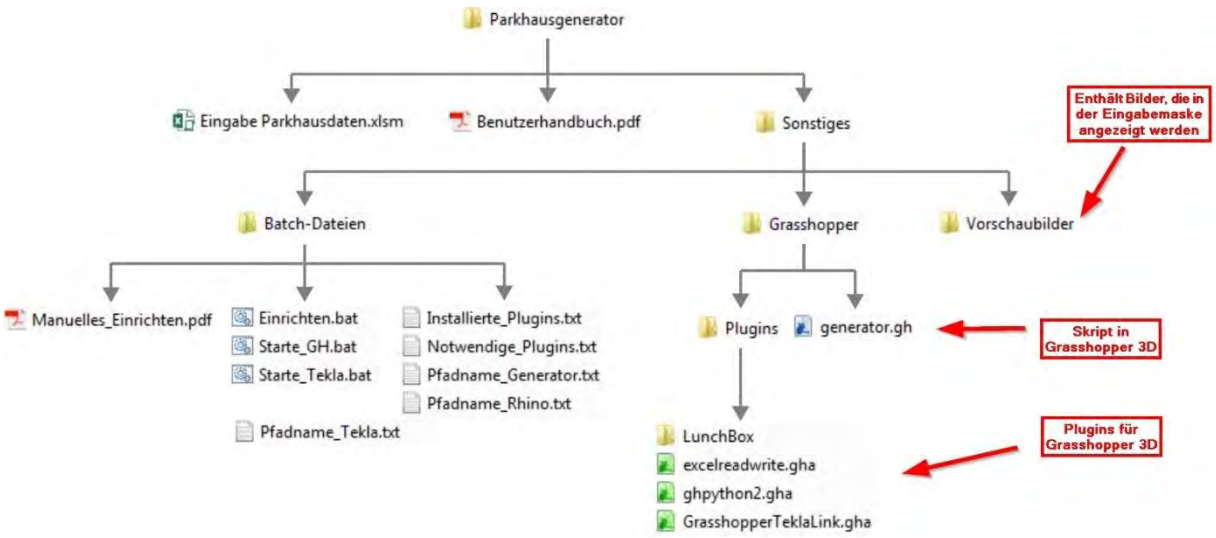


Abbildung 28: Angelegte Ordnerstruktur

Auf die einzelnen Dateien wird in den folgenden Abschnitten genauer eingegangen.

4.3.1 Eingabemaske mithilfe von Visual Basic for Applications (VBA)

Die Eingabemaske, die mithilfe von Visual Basic for Applications (VBA) in Excel entwickelt wurde, wird unmittelbar nach dem Start der Excel-Liste geöffnet. Dadurch kann der Nutzer sofort mit der Dateneingabe starten und wird nicht dazu verleitet, bestimmte Zellbereiche manuell zu manipulieren. Vor dieser Manipulation werden die Tabellenblätter außerdem mit einem Passwort geschützt. Die Maske ist wie folgt aufgebaut: Auf der ersten Seite befinden sich die Basisinformationen, wie Anzahl der Stellplätze und Stockwerksanzahl. Je nach Eingabe von bestimmten Parametern, verändern sich die Vorschaubilder, die direkt in der Eingabemaske eingefügt werden. Hier wird dem Nutzer bereits eine Visualisierung seines entstehenden Modells gegeben und außerdem Missverständnisse vermieden.

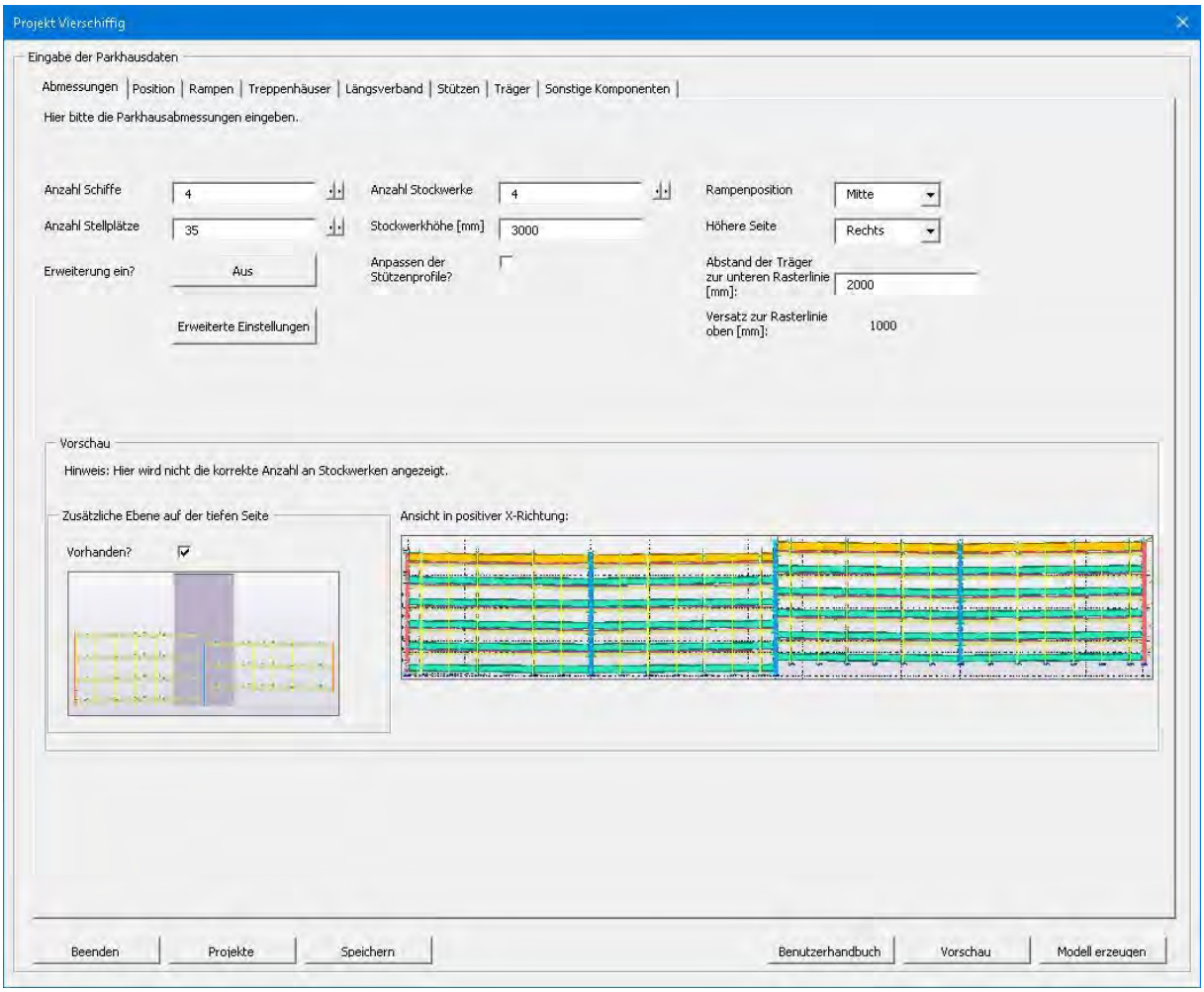


Abbildung 29: Eingabemaske

Manchmal ist es notwendig, das gesamte Modell anhand einer bestimmten Koordinate zu verschieben oder platzieren, da man eventuell bestehende Strukturen im Modell berücksichtigen muss. Deswegen hat man auf der Seite „Position“ die Möglichkeit, das gesamte Modell in x- y- und z-Koordinate zu verschieben. Außerdem kann man wählen, welches Eck am Koordinatensystem anschließen soll. Dies wird je nach Wahl wie auf der ersten Seite visualisiert. Realisiert wird diese Verschiebung dadurch, dass die Länge bzw. die Breite des Parkhauses am Ende zu den Koordinaten dazu gerechnet wird, die vom Nutzer eingegeben werden.

Für die Eingabe der Rampendaten und der Daten für die Treppenhäuser an den Längsseiten wird eine for-Schleife genutzt. Hierbei wird der Benutzer zunächst gefragt, wie viele Bauteile er benötigt. Anschließend kann er die Daten für jedes Bauteil einzeln eingeben. Dazu wird je nach eingegebener Anzahl ein neues Formular mehrfach für jedes Bauteil einzeln geöffnet, nachdem der Nutzer die entsprechende Schaltfläche zur Eingabe der Daten aktiviert hat. Der Benutzer kann dadurch alle Daten der Reihe

nach eingeben und aber auch zurückspringen. Am Ende wird er gefragt, ob er die Eingaben übernehmen will. Gibt der Nutzer diese Daten nicht ein, sind noch keine Daten zu den Bauteilen hinterlegt. Deswegen wird beim Übernehmen der Daten kontrolliert, ob die Anzahl an angegebenen Rampen mit den eingegebenen Daten übereinstimmt. Stimmt sie nicht überein, wird nichts übernommen und der Nutzer wird aufgefordert, die Daten noch einzugeben.

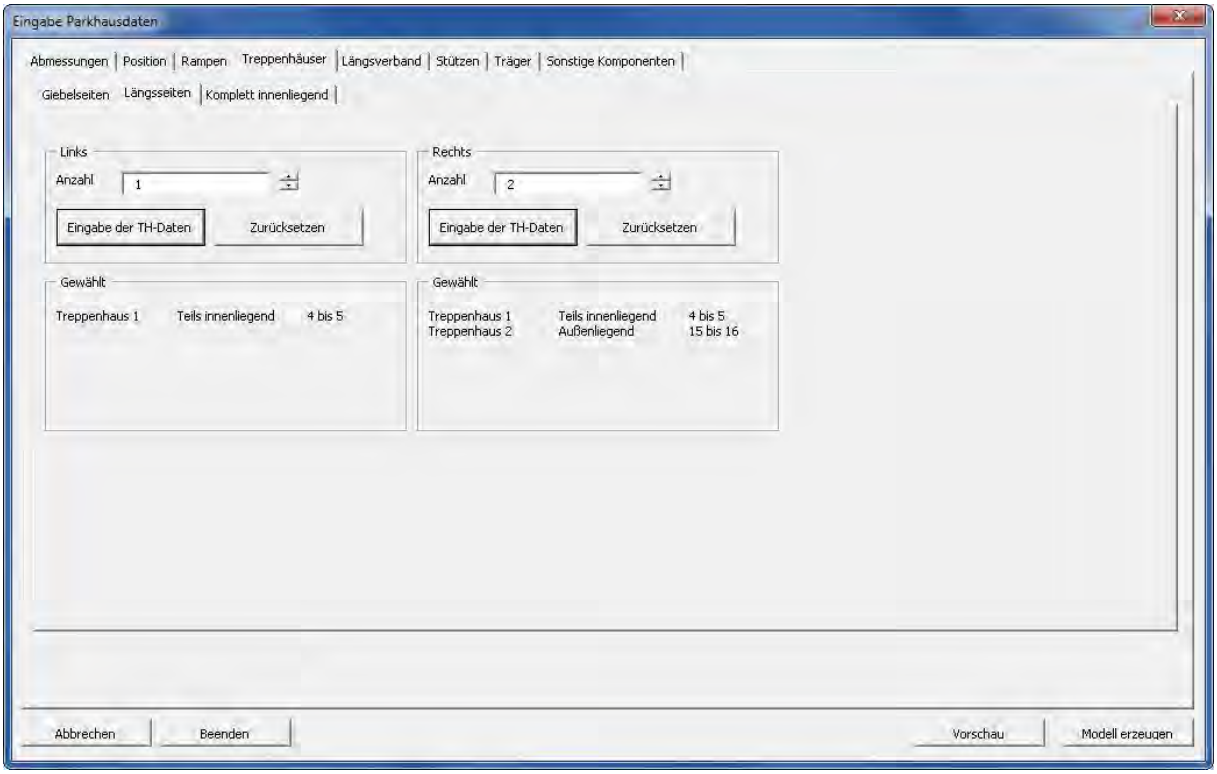


Abbildung 30: Eingabe der Treppenhäuser an der Längsseite

Um in der Eingabemaske einen Überblick über die eingegebenen Positionen der Rampen oder Treppenhäuser zu behalten, werden diese für jede Rampe in einem Textfenster angezeigt, da diese ansonsten nicht sichtbar wären.

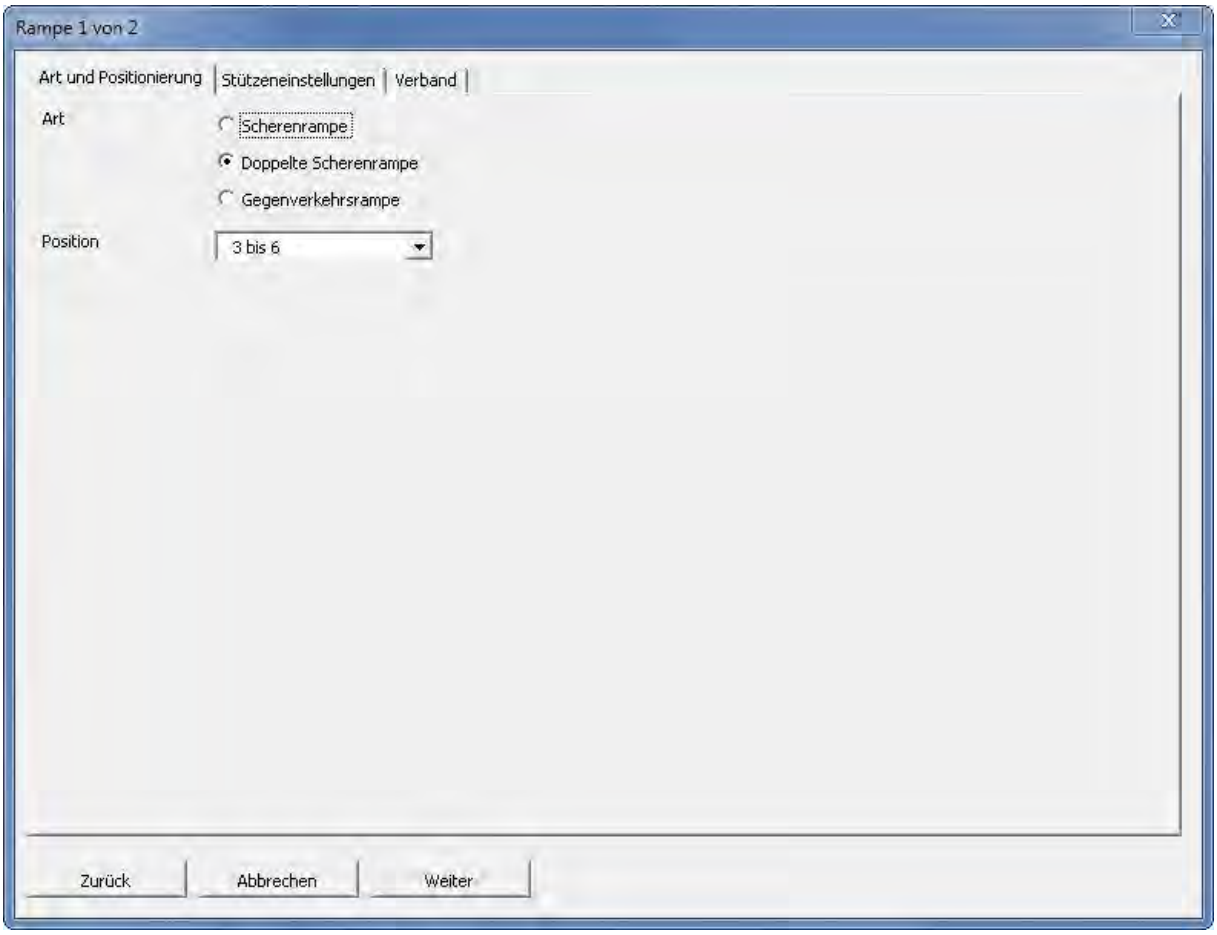


Abbildung 31: Formular zur Eingabe der einzelnen Rampendaten

Beim Starten der Prozedur werden bereits Standardwerte hinterlegt. Dazu wird für jeden einzugebenden Parameter, wie beispielsweise die Position der Rampe, ein String angelegt, der die Informationen für jede Rampe enthält. Die Eingaben der einzelnen Rampen werden durch einen Schrägstrich getrennt. Beim Durchlaufen der for-Schleife wird je nach Wert der Laufvariable der String in mehrere Teilstrings zerlegt, die die Texte zwischen den Schrägstrichen enthalten. Anschließend wird der vorhandene String ausgelesen und in dem Formular angezeigt. Deswegen müssen bereits am Anfang Standardwerte existieren, da sonst kein Wert angezeigt werden kann. Ändert der Nutzer nun diesen Parameter, wird für diesen Teilstring der geänderte Wert eingetragen und der String wieder zusammengesetzt. Somit ist es möglich, dass der Nutzer während der Eingabe jederzeit einen Schritt vor- oder zurückgehen kann, ohne dass die eingegeben Daten verworfen werden.

Bemerkung: Man kann eine Anzahl an Daten auch über Vektoren und Matrizen speichern, die man je nach Anzahl erweitert oder reduziert. Es ist allerdings schwierig, diese als globale Variable zu übergeben. Deswegen wird hier die Methode mit den Strings gewählt, welche sehr gut funktioniert.

Beispielsweise wurden vom Nutzer drei Rampen gewählt. Der Standardstring vor Eingabe für den Parameter „Rampenart“ lautet:

„Scherenrampe/Scherenrampe/Scherenrampe“

Der Nutzer ändert nun die Art der zweiten Rampe in „Gegenverkehrsrampe“. Der String lautet nun:

„Scherenrampe/Gegenverkehrsrampe/Scherenrampe“

Da der Nutzer manchmal Eingaben wieder abbrechen muss, wohingegen die vorherigen Eingaben aber bestehen bleiben sollen, sind die bei der Eingabe benutzten Variablen nur temporäre Variablen. Die Daten, die erhalten bleiben sollen sind in anderen globalen Variablen gespeichert. Will der Nutzer die Daten übernehmen, so werden die temporären Variablen in diesen anderen Variablen übernommen. Diese werden dann am Ende beim endgültigen Übernehmen aller Daten auch genutzt, um die Rampendaten in die Excel-Liste einzutragen. Dazu werden dann hier auch wieder die Strings zerlegt und alle Teilstrings über eine for-Schleife in die Excel-Liste in horizontaler Richtung eingetragen. Dieser Mechanismus wird auch für die Treppenhäuser auf den beiden Längsseiten genutzt.

Die Eingabe der Positionen der besonderen Bauteile wird über sogenannte Kombinationsfelder geregelt, in welchen man nur zwischen bestimmten Eingaben wählen kann. Die Positionen und Breiten werden über die Reihenzahlen definiert, beginnend bei „Reihenzahl 1“. Hierbei werden am Anfang bei der Eingabe der Anzahl an Stellplätzen im Tabellenblatt „Datenüberprüfung“ für Breiten zwischen eins und acht Reihen alle möglichen Positionen innerhalb der eingegebenen Stellplätze aufgelistet und alle weiteren Werte gelöscht. Beim Füllen dieser Kombinationsfelder wird zunächst über die angegebene Breite die jeweilige Spalte gesucht, um dann alle Werte innerhalb der durch die angegebenen Stellplätze gegebenen Grenzen in die Combobox einzutragen. Beispielsweise gibt es bei Angabe von zehn Stellplätzen und einer Breite über zwei Reihen die Möglichkeiten „1 bis 2“, „2 bis 3“ bis „29 bis 30“.

1	2	3	4	5	6	7
1 bis 1	1 bis 2	1 bis 3	1 bis 4	1 bis 5	1 bis 6	1 bis 7
2 bis 2	2 bis 3	2 bis 4	2 bis 5	2 bis 6	2 bis 7	2 bis 8
3 bis 3	3 bis 4	3 bis 5	3 bis 6	3 bis 7	3 bis 8	3 bis 9
4 bis 4	4 bis 5	4 bis 6	4 bis 7	4 bis 8	4 bis 9	4 bis 10
5 bis 5	5 bis 6	5 bis 7	5 bis 8	5 bis 9	5 bis 10	
6 bis 6	6 bis 7	6 bis 8	6 bis 9	6 bis 10		
7 bis 7	7 bis 8	7 bis 9	7 bis 10			
8 bis 8	8 bis 9	8 bis 10				
9 bis 9	9 bis 10					
10 bis 10						

Abbildung 32: Mögliche Positionen abgespeichert in einem Tabellenblatt

Bei der Eingabe der Verbände werden die gleichen Werkzeuge benutzt. Hier werden je nach eingegebener Anzahl der Schiffe bestimmte Bereiche ausgeblendet welche wegen der Wahl der Abmessungen in dem Fall nicht benötigt werden. Z.B. gibt es bei einem zweischiffigen Parkhaus nur drei Stützenreihen in Längsrichtung. Deswegen gibt es hier dann auch nur drei Längsverbände. Die Eingabe ist so gehalten, dass jede Reihe andere Eigenschaften wie Position oder Stützenart aufweisen kann.

Um es dem Nutzer zu ermöglichen, alle eingegebenen Positionen der besonderen Bauteile mit den gegebenen 2D-Plänen zu vergleichen, kann außerdem ein Vorschau-Funktion angelegt werden. Hier kann der Nutzer durch eingefärbte und umrahmte Zellen im Tabellenblatt „Vorschau“ die Struktur des entstehenden Modells einsehen, die je nach Eingaben des Nutzers aktualisiert wird. Auf dieses Tabellenblatt wird der Nutzer bei Aktivierung der Schaltfläche – ebenfalls beschriftet mit „Vorschau“ – weitergeleitet und die Eingabemaske kurzzeitig ausgeblendet. Will der Nutzer wieder zur Eingabe zurückkehren, kann er mit der Schaltfläche „Zurück zur Eingabe“ seine Eingabe fortsetzen.

Jede eingerahmte Zelle symbolisiert eine Stütze. Die Art der Stütze wird durch die Farbe definiert. Die rot gefärbten Zellen symbolisieren Außenstützen, die blau gefärbten die Innenstützen. Grün gefärbte Zellen stehen für das Stützenpaar am Ende der Treppenhausseiten, orange für Rampen- und violett wiederum für Verbandsstützen. Eine weitere Unterscheidung gibt es noch bei den Treppenhäusern an der Giebel- und Längsseite. Hier werden außenliegende Treppenhäuser zusätzlich mit einem „A“ gekennzeichnet. Da es unübersichtlich sein kann, nur die Stützen zu betrachten, werden die Abmessungen der Bauteile in den Zwischenzellen ebenfalls eingefärbt, um zusammengehörige Bauteile zu erkennen. Die Innenreihe, an denen die verschiedenen Ebenen des Parkhauses versetzt sind und an denen sich später die Rampen befinden, ist außerdem mit den Reihenindizes nummeriert.

Abbildung 33: Vorschau der Eingaben

Diese Vorschau dient also zum einen zum Vergleich mit den 2D-Plänen (siehe Anhang A.4), zum anderen aber auch zur automatischen Kontrolle nach Überschneidungen von Bauteilen, die der Nutzer möglicherweise übersieht. Dazu wird in der VBA-Prozedur beim Einfärben einer Zelle überprüft, ob das Feld bereits eine Farbe außer rot oder blau aufweist. Ist die betroffene Zelle bereits entsprechend eingefärbt, wird einer Variable, die die Anzahl an Überschneidungen pro Bauteil zählt, die Zahl eins hinzuaddiert. Am Ende wird bei jeder Kollision ein Nachrichtenfeld geöffnet, in der der Nutzer auf die Art der Kollision und der Anzahl an Überschneidungen aufmerksam gemacht wird.

Abbildung 34: Kollisionsmeldung

Unabhängig davon hat man des Weiteren noch die Möglichkeit, die Profilnamen der Stützen einzugeben sowie zwischen verschiedenen Trägerarten und Komponenten zu wählen.

Bei der Benutzung der Schnittstelle zwischen Tekla Structures und Grasshopper 3D ist eine bestimmte Startreihenfolge der Programme notwendig. Deswegen wird der

Benutzer nach Eingabe seiner Daten anschließend über ein weiteres Benutzerformular dazu aufgefordert, die Programme der Reihe nach zu starten. Dazu wurden Schaltflächen installiert, die beim Ausführen jeweils eine Batch-Datei aufrufen, die das jeweilige Programm startet. Mit der Schaltfläche „Tekla starten“ wird Tekla Structures gestartet und über „Modell erzeugen“ wird das Skript in Grasshopper 3D geladen.

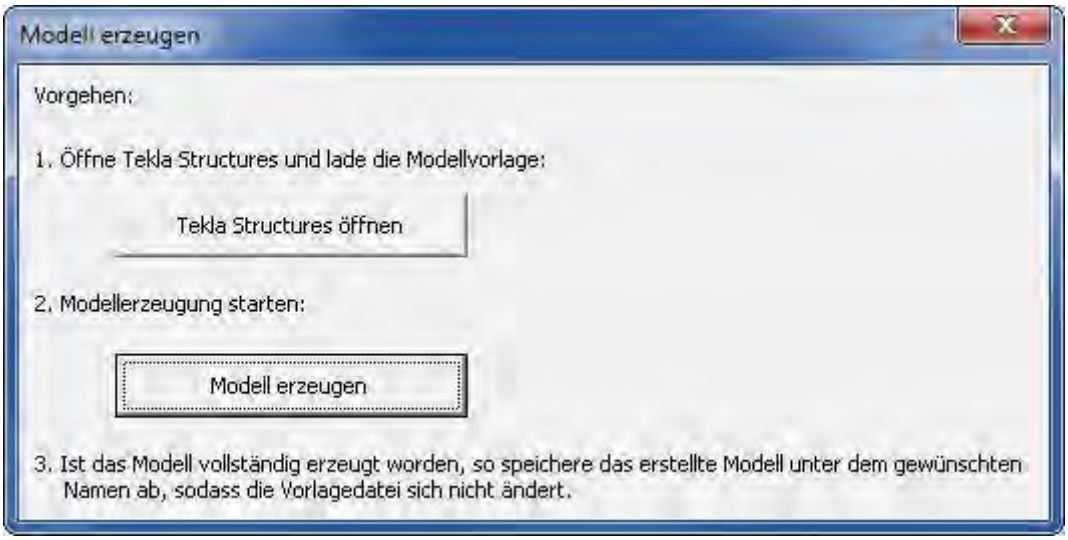


Abbildung 35: Benutzerformular "Modell erzeugen"

Um eine Batch-Datei zu starten, wird in VBA der Befehl „Shell“ verwendet.

Der Nutzer kann außerdem auch bereits vorher eingegebene Daten verwenden. Dazu wird er vor dem Öffnen des vorherigen Formulars dazu aufgefordert, sich für eine der Möglichkeiten zu entscheiden. Bei Entscheidung für die neuen Daten wird noch einmal die Prozedur „Vorschau“ aufgerufen, die die Daten auf Korrektheit überprüft, ohne dass der Nutzer auf das Tabellenblatt weitergeleitet wird. Treten noch Überschneidungen auf, so wird das dem Benutzer mitgeteilt, wonach er die Daten aktualisieren muss. Falls die Daten korrekt eingegeben wurden, werden die Daten auf das Tabellenblatt „Datenauslesung“ in definierten Zellen übertragen. Bei Verwendung der alten Daten erfolgt keine Überprüfung, da die Daten beim vorherigen Eintragen bereits überprüft worden sind.

Position

X-Koordinate [mm]	0
Y-Koordinate [mm]	0
Z-Koordinate [mm]	0
Position	RV
Fehlerfrei?	WAHR

Längsverband

Außen links	
Breite	3
Startposition	17
Endposition	19
Profil	HEM300
Innen links mehrschiffig	
Breite	3
Startposition	17
Endposition	19
Profil	HEM300
Innen	
Breite	3
Startposition	17
Endposition	19

Abmessungen

Anzahl Schiffe	4
Anzahl Stellplätze	35
Anzahl Stockwerke	5
Stockwerkhöhe [mm]	3000
Erweiterung ein?	FALSCH
Innen überhöht?	FALSCH
Höhere Seite?	Links
Zusätzliche Ebene?	FALSCH
Versatz Träger	1000

Erweiterte Einstellungen

Stellplatzgröße	2500
Lichte Tragweite	17000
Deckenneigung	2,0
Abstand Giebelstützen	3000

Rampen

Anzahl Rampen	3
Position	Mitte

	Rampe 1	Rampe 2
Art	Scherenrampe	Scherenrampe
Startposition	4	30
Endposition	6	32
Höhenversatz unten		
Stützenreihe 1	1000	1000
Stützenreihe 2	1000	1000
Stützenreihe 3	1000	1000
Stützenreihe 4	1000	1000
Stützenreihe 5	1000	1000
Höhenversatz oben		
Stützenreihe 1	1000	1000
Stützenreihe 2	1000	1000
Stützenreihe 3	1000	1000
Stützenreihe 4	1000	1000
Stützenreihe 5	1000	1000
Position Verband	1	1
Profil	HEM260	HEM260
Profil Verband	HEM300	HEM300

Abbildung 36: Auszug aus dem Tabellenblatt „Datenauslesung“

Außerdem wird überprüft, ob die angegebene Anzahl an Rampen oder Treppenhäuser auf der Längsseite mit den eingegebenen Rampen- oder Treppenhausdaten übereinstimmen. Mit diesen Kontrollmechanismen kann der Nutzer Überschneidungen oder Unstimmigkeiten in den eingegebenen Daten frühzeitig erkennen und ändern.

4.3.2 Batch-Dateien

Um dem Nutzer verschiedene Operationen und Vorgehensweisen abzunehmen, kann auf Batch-Dateien zurückgegriffen werden. Batch-Dateien (engl. „batch“: stapeln) sind Dateien, mithilfe deren man über textuelle Programmierung verschiedene Windows-Operationen stapelweise durchführen kann.

Um die Startreihenfolge der Programme sicherzustellen, werden aus der Eingabemaske heraus über Schaltflächen ebendiese Batch-Dateien gestartet, die ein sicheres Starten der Programme gewährleisten sollen. Um ein Programm zu starten, muss eine exe-Datei aktiviert werden. Beispielsweise wird beim Öffnen des Programms Word die Datei „WINWORD.exe“ ausgeführt. Diese Dateien sind i.d.R. an einem bestimmten Ordner in Windows abgespeichert, dessen Pfad bereits in einer Textdatei in demselben Ordner, in dem die Batch-Dateien gespeichert wurden, hinterlegt wurde. Um sicherzustellen, dass der richtige Pfad verwendet wird, wird dieser zunächst auf seine Existenz überprüft. Stimmt der Pfad, wird dieser Pfad für das Starten des jeweiligen Programms herangezogen. Falls nicht, wird der Pfad der exe-Datei erst gesucht anhand des Namens der Datei, im Beispiel Word mit dem Suchbegriff „WINWORD.exe“. Anschließend wird der gerade ermittelte Pfad in dieselbe Textdatei abgespeichert, in der der alte Pfad abgespeichert wurde. So wird sichergestellt, dass immer der richtige

Pfad zum Starten der Programme verwendet wird. Nach dem Start von Tekla Structures aus der Eingabemaske heraus muss der Nutzer noch eine Modellvorlage laden. Dies wird nicht automatisch über die Batch-Datei geregelt, da sich die Vorlagen regelmäßig ändern. Diese Freiheit wird im Zuge dessen noch dem Nutzer überlassen.

Um Grasshopper 3D zu starten und das Skript zu laden müsste zunächst Rhinoceros 3D gestartet und in dessen Kommandozeile „Grasshopper“ eingegeben werden, um danach die entsprechende Datei mit dem Modell-erzeugenden Skript als Inhalt zu laden. Diese Schritte können ebenfalls über eine Batch-Datei getätigt werden, die aus der Excel-Eingabemaske heraus gestartet werden und alle benötigten Pfade wie beim Starten von Tekla Structures vorher ausliest bzw. ermittelt. Um den Pfad der Grasshopper-Datei mit dem enthaltenen Skript zu ermitteln, wird auf die angelegte Ordnerstruktur zurückgegriffen.

Noch vor der ersten Benutzung des Generators müssen die Plugins für Grasshopper 3D installiert werden. Dazu zählen „ghpython2“, welches erlaubt, eigene Blöcke mit der Programmiersprache Python zu programmieren, „LunchBox“, welches für die Verbindung zu Excel benötigt wird sowie „GrasshopperTeklaLink“ für die Schnittstelle zu Tekla Structures. Für diesen Prozess wurde ebenfalls eine Batch-Datei angelegt, welche zunächst überprüft, ob diese Plugins bereits installiert sind. Hierfür wurden die Dateien der Plugins bereits in einem Unterordner des Ordners des Parkhausgenerators abgespeichert und deren Namen mit den Namen der Dateien in dem Grasshopper-Ordner auf Übereinstimmung verglichen. Sind diese installiert, wird die Prozedur hier bereits beendet. Falls nicht, wird dem Nutzer direkt mitgeteilt, dass der Generator gerade noch eingerichtet wird. Anschließend werden die Plugin-Dateien von dem in der Ordnerstruktur angelegten Unterordner „Plugins“ in den entsprechenden Grasshopper-Ordner kopiert. Der Pfad dieses Ordners ist %APPDATA%\Grasshopper\Libraries. „%APPDATA%“ ist eine Umgebungsvariable, die für jeden Benutzer unterschiedlich definiert und somit unabhängig vom benutzten Rechner ist. Für den Kopiervorgang wird der Befehl „xcopy“ verwendet. Anschließend wird Rhinoceros 3D gestartet und durch automatische Eingaben in der Befehlszeile des Programms das Plugin Grasshopper 3D gestartet. Anschließend werden ebenfalls durch Eingabe in der Befehlszeile die beiden Programme wieder geschlossen, wodurch die Plugins erfolgreich installiert wurden.

Diese Batch-Datei wird bei jedem Start der Excel-Tabelle ausgeführt, weshalb bei jeder Benutzung des Generators sichergestellt wird, dass alle benötigten Plugins installiert sind. Der Prozess des Kopierens der Dateien und des Installierens der Plugins wird allerdings nur ausgeführt, wenn nicht alle Plugins installiert sind. Die installierten Plugins werden innerhalb eines Bruchteils einer Sekunde überprüft, wodurch nur eine minimale Zeitverzögerung beim Starten von Excel auftritt.

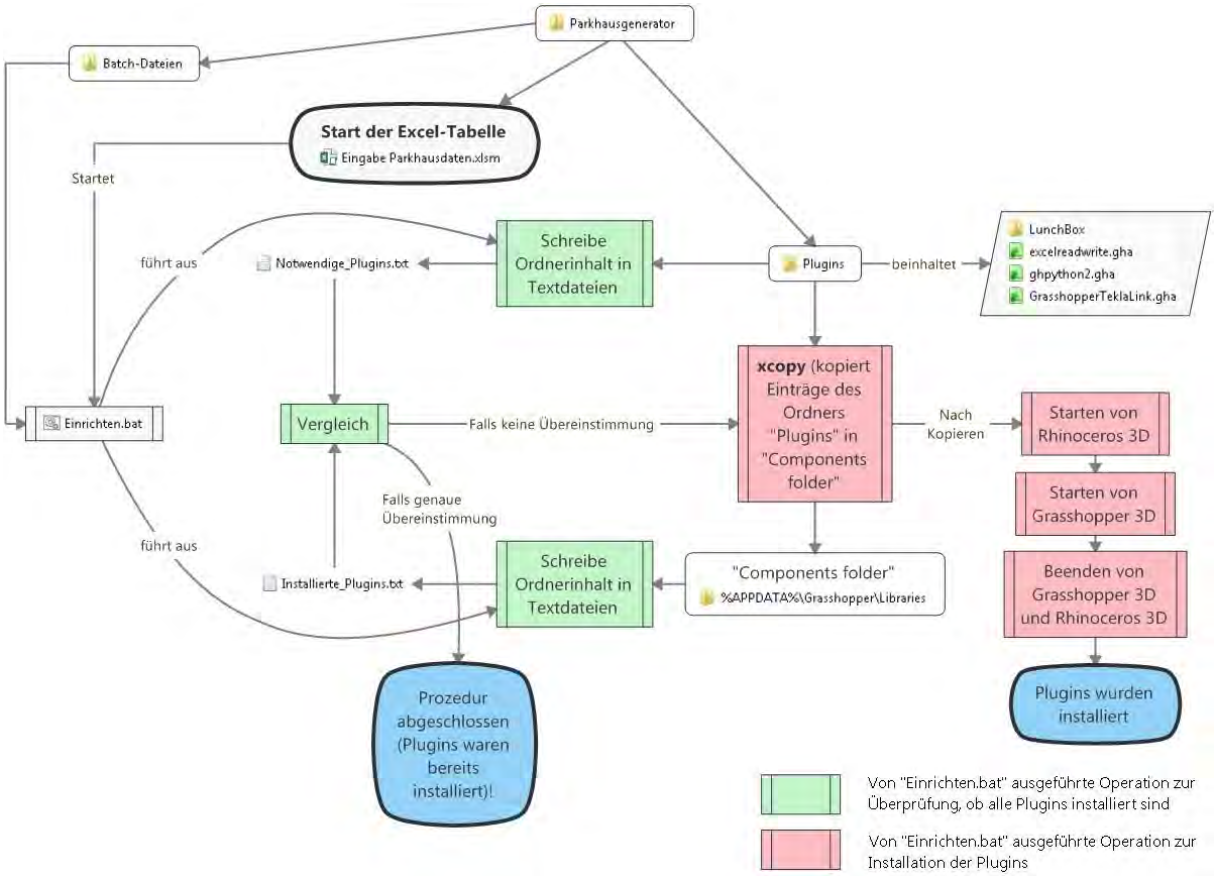


Abbildung 37: Vorgang des Einrichtens

Zusammenfassend wird durch diese Batch-Dateien sichergestellt, dass zum einen die notwendige Startreihenfolge beachtet wird, zum anderen die im Skript in Grasshopper 3D verwendeten Plugins installiert sind. Dadurch kann die Benutzerfreundlichkeit deutlich gesteigert werden, da alle notwendigen Schritte automatisch vollzogen werden. In Kombination mit der Eingabemaske ist demnach erreicht worden, dass die Dateneingabe, die Einrichtung von Grasshopper 3D und das Starten der Programme in richtiger Reihenfolge von einem zentralen Ort aus erfolgt, der Excel-Tabelle.

4.3.3 Skript in Grasshopper 3D

Das Skript in Grasshopper 3D wird dazu verwendet, das Parkhausmodell mit den in Excel eingegebenen Daten automatisch zu erstellen. Um einen Überblick über das gesamte Skript zu geben, wird das Skript in einzelne Bereiche unterteilt.

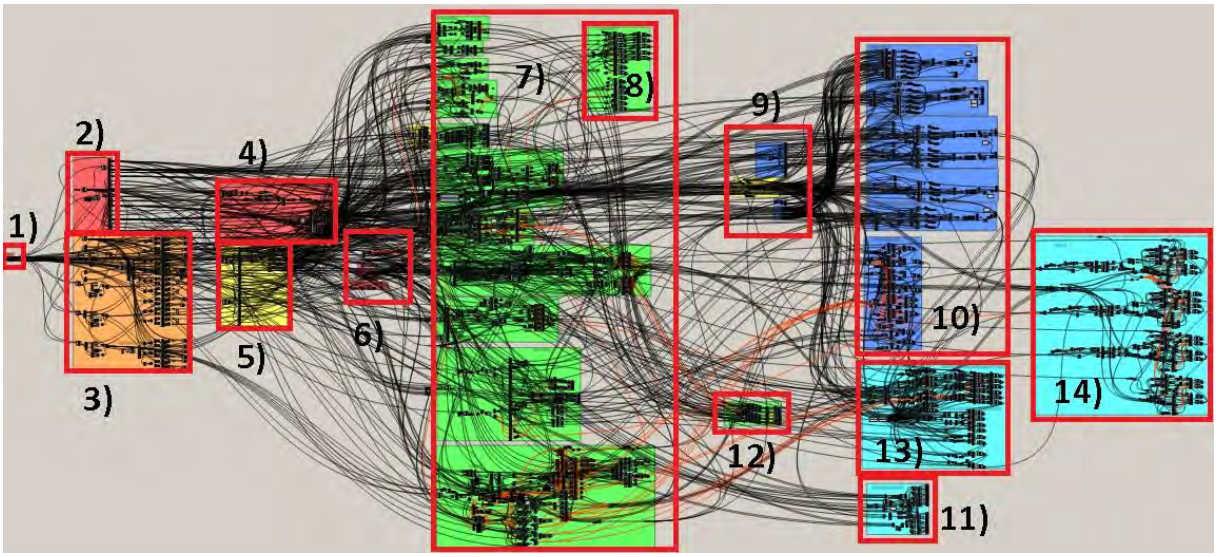


Abbildung 38: Strukturierung des Skriptes in Grasshopper 3D

Hier wird nun erklärt, wie das Parkhaus mithilfe des Skriptes entsteht.

4.3.3.1 Einlesen der Daten (Bereich 1)

Zu Beginn wird im ersten Bereich der Datenstrom über einige Schalter aktiviert. Hier werden einzelne Bereiche noch getrennt voneinander geführt, die aber noch zusammengeführt werden können, um eine Modellerstellung auf Knopfdruck zu ermöglichen. Dabei kann die Anzahl der Schalter auf einen einzigen reduziert werden, der zu Beginn an aktiviert ist. Somit würde die Erstellung genau bei der Öffnung des Skripts starten.

Abbildung 39: Startschalter

Dadurch werden u.a. Schalter aktiviert, die zunächst Daten aus der Excel-Tabelle auslesen und an die folgenden Blöcke weitergeben.

4.3.3.2 Berechnung der Basiskoordinaten (Bereiche 2 bis 6)

Im zweiten Bereich werden nun aus der noch geöffneten Excel-Datei die Hauptparameter ausgelesen, wie die Anzahl der Schiffe oder der Stellplätze, also Parameter, die die Form des Parkhauses definieren. Parallel dazu werden im dritten Bereich die Namen der Stützenprofile ausgelesen, um damit die Abmessungen der jeweiligen Stützen ermitteln zu können. Dazu wird ein Block verwendet, der mithilfe des Profilnamens eine Polylinie erstellt, die genau dem Profilquerschnitt entspricht. Mit einem Block, der NURBS-Kurven in mehrere Bereiche anhand seiner Kontrollpunkte unterteilt und diese Kontrollpunkte als Output generiert, kann man die Koordinaten der Eckpunkte des Querschnitts erhalten. Sortiert man diese Koordinaten in x- und y-Richtung, hat man am einen Ende der Liste den ersten und am anderen den letzten Punkt, womit man dann die Profilhöhe und die Profilbreite ermitteln kann.

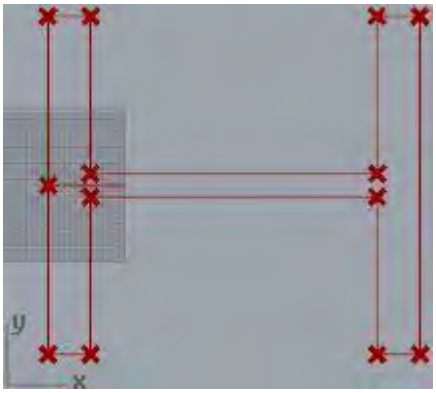


Abbildung 40: Kontrollpunkte der Polylinie eines HEM-Profils in Rhino

Mit den vorher ermittelten oder ausgelesenen Daten kann nun ein Netz aus Basiskoordinaten erstellt werden, aus denen alle späteren Referenzpunkte abgeleitet werden. Dabei gibt es zwei verschiedene Bereiche: Im Bereich 4) wird ein Netz erzeugt, das in horizontaler Ebene auf der z-Koordinate „0“ liegt und also nur durch x- und y-Koordinaten definiert wird. Diese Koordinaten werden in Abhängigkeit der gegebenen Daten ermittelt, wobei die lichte Weite und die Stützenabmessungen die tragende Rolle innehaben. Demnach wird zunächst eine Querreihe definiert, die dann in Längsrichtung je nach Anzahl und Größe der Stellplätze verschoben wird. Im Bereich 5) werden dann noch die z-Koordinaten über zwei Listen jeweils für die linke und die rechte Seite des Parkhauses definiert, da in einem Split-Level-Parkhaus die zwei Seiten versetzt zueinander sind.

Diese Basiskoordinaten werden im Bereich 6) zusammengeführt, wodurch eine dreidimensionale Punktewolke entsteht. Diese Punktewolke wird verwendet, um das Achsraster in Tekla Structures zu erzeugen. Leider ist es über Grasshopper nicht möglich, die Nummerierung bzw. die Beschriftung der Achsen vorzunehmen. Dies müsste entweder per Hand im Nachhinein getan werden oder wenn möglich über ein externes Skript.

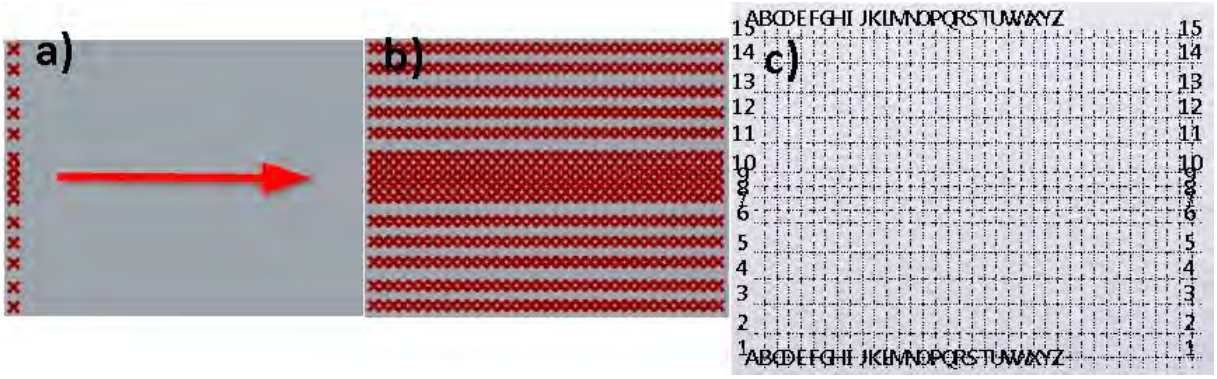


Abbildung 41: Von der Querreihe a) zum Achsraster c)

An jeder Querreihe werden in nachfolgenden Schritten Strukturen erzeugt. Dies macht es möglich, dass alle Strukturen, die sich auf die Querreihen beziehen, wie die Stützen oder die Träger, mit variablen Abständen erzeugt werden können. Angewandt wird dies z.B. für erweiterte Stellplätze an den Längsrändern. Allerdings muss man diese Unregelmäßigkeit bei Bauteilen, die in Längsrichtung erzeugt werden und von der Stellplatzgröße abhängig sind, durch Listen berücksichtigen, die diese unterschiedlichen Abstände enthalten. Ein Beispiel, in der die Methode der Unterbrechung angewandt wird, ist die Erzeugung der Absturzsicherung, die im Kapitel „4.3.3.5 Weitere Bauteile“ noch beschrieben wird.

4.3.3.3 Stützenerstellung (Bereich 7 und 8)

Im nächsten Bereich 7) werden die Basiskoordinaten verwendet, um die ersten Stützen zu erzeugen. Für diesen Prozess ist es zweckmäßig, zu unterscheiden zwischen Basisstützen und besonderen Stützen. Basisstützen sind Stützen, die das unerschlossene Parkhaus in seiner reinen Form definieren. Dazu zählen die normalen Außen- oder Innenstützen. Stützenarten, die Hohlräume in dem unerschlossenen Parkhaus erzeugen, sind die besonderen Stützen, wie die Stützen an Treppenhäuser, Rampen oder die Verbandsstützen.

Dadurch, dass das erzeugte Punktenetz in Pfaden angelegt wurde, deren Punkte der Längsreihen sich jeweils innerhalb eines Pfades befinden, ist bereits jeder Punkt auf

dieser Längsreihe mit einem Index versehen. Diese Indizes werden verwendet, um die Position der Treppenhäuser und Rampen festzulegen. Um Aussparungen der Basisstützen umzusetzen, bzw. um die Referenzpunkte der besonderen Stützen abzuleiten, werden die aus der Excel-Tabelle ausgelesenen Indizes verwendet. Dies erfolgt in Punktelisten, in denen entweder die betroffenen Punkte mit den entsprechenden Indizes der besonderen Bauteile ausgestrichen oder ausgelesen werden. Hier ist allerdings noch eine Übersetzung nötig, da in der Benutzereingabe Querreihen mit dem Index 1 beginnen. Hierzu muss einfach diese eins von den ausgelesenen Indizes abgezogen werden.

Für die Erstellung der Giebel- und Treppenhausstützen können nicht direkt die Basispunkte verwendet werden. Hier müssen die Referenzpunkte erst entsprechend erstellt werden, wobei allerdings die Basispunkte genutzt werden. Hier werden allerdings keine Treppenhäuser eingefügt, da diese Teil des Betonbaus sind. Deswegen werden nur Betonkörper erstellt, die die Form des Treppenhauses widerspiegeln, um das Modell übersichtlicher zu gestalten. Eine Bedeutung haben diese nicht und werden nach der Modellierung wieder gelöscht.

Diese Referenzpunkte werden anschließend an die untere Höhe der Stütze verschoben, die sich aus den Basishöhen und der Einbindetiefe ergibt. Danach wird der Referenzpunkt erzeugt, der die Höhe definiert. Aus diesen zwei Punkten wird dann eine Linie gebildet, die an die Tekla-Komponente übergeben wird, um die Stützen letztendlich zu erzeugen.

Um eine unbestimmte Anzahl an besonderen Bauteilen zu erstellen, werden bei der Auslesung aus der Excel-Liste die Daten eines Bauteils in Pfade unterteilt. Jeder Pfad steht für ein Bauteil eines bestimmten Typs. Somit können die Daten verschiedener Varianten eines Bauteils aufgeteilt und an den entsprechenden Bereich weitergegeben werden.

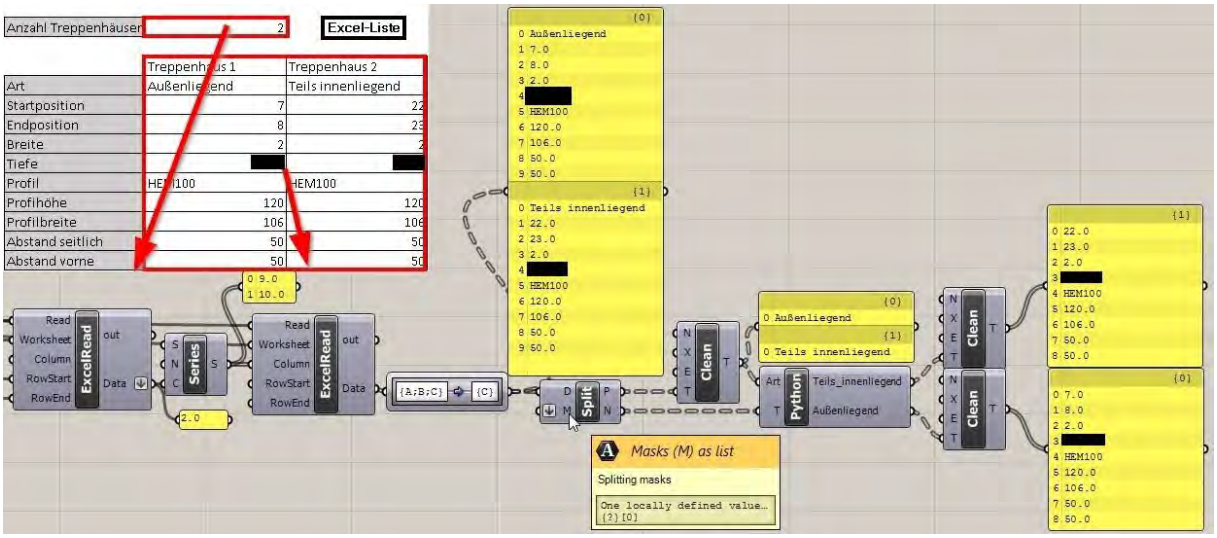


Abbildung 42: Aufteilung der Daten nach der Treppenhausart

Das gleiche gilt für die Rampenstützen, wobei hier unterschieden wird zwischen Gegenverkehrsrampe, Scherenrampe und doppelter Scherenrampe.

Ein weiterer Aspekt, indem sich der Vorteil der Baumstruktur zeigt, ist die Erstellung des Rampenverbandes, der für die Aufnahme aller horizontalen Lasten in Querrichtung zuständig ist. Hierfür werden Stützen mit höherer Steifigkeit benötigt, weswegen hierfür andere Profile verwendet werden. Nutzt man die von Grasshopper 3D zur Verfügung gestellte Baumstruktur, kann man für jede Rampe eine unterschiedliche Position des Rampenverbandes erstellen. Hierfür werden Indizes der Rampenstützenreihen verwendet, um den Verband definieren bzw. identifizieren zu können. Zur besseren Erkennbarkeit werden die Stützen des Rampenverbandes braun eingefärbt und die normalen Rampenstützen orange.

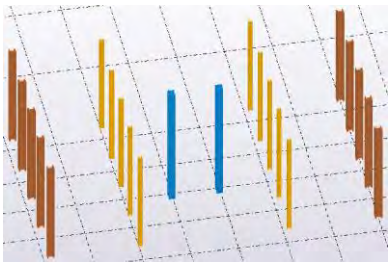


Abbildung 43: Zwei Scherenrampen mit unterschiedlichem Rampenverbandsindex

Diese Stützen werden nun an den Bereich 8) übergeben, um die Stützenfußplatte, mit der die Stützen mit dem Stahlbetonfundament verbunden sind, zu erzeugen. Dabei muss nur unterschieden werden zwischen den einzelnen Stützenarten und den Stüt-

zenhöhen. Alle gleichen Stützenarten mit derselben Höhe können mit einem Block erstellt werden, alle weiteren müssen in Serie geschaltet werden, um Fehler in der Erzeugung zu vermeiden.

4.3.3.4 Trägererstellung (Bereich 9 und 10)

Im Bereich 10) werden die Träger erstellt. Diese wurden als Komponente modelliert, da sie eine Überhöhung in Trägermitte aufweisen und noch einige Schraubenanschlüsse oder Durchlässe enthalten. Einige für die Trägererstellung benötigten Daten sind bereits im Bereich 9) zusammengefasst, um die Streuung der Kabel etwas zu verringern. Die Träger werden mithilfe von zwei Referenzpunkten platziert, die aus den Basispunkten abgeleitet werden. Auch hier wird unterschieden zwischen den Basisträgern und den besonderen Trägern, die an Orten mit Störkörpern eingesetzt werden. Diese Träger unterscheiden sich in ihrer Farbe. Um die Indizes der Störkörper zu identifizieren, müssen hier die der zwei benachbarten Reihen gemeinsam betrachtet werden. Außerdem muss eine bestimmte Baumstruktur als Eingangsdaten beachtet werden, in der jeweils genau zwei Referenzpunkte in einem Zweig enthalten sind. Ansonsten kann die Erstellung nicht erfolgen. Deswegen müssen die erstellten Punkte anschließend sorgfältig sortiert werden und die Baumstruktur von leeren Zweigen bereinigt werden, da bei Übergabe von leeren Zweigen immer eine Fehlermeldung auftritt und keine Komponenten erstellt werden.

Außerdem werden die Träger mit einem Höhenversatz ausgebildet, sodass Wasser abfließen kann. Die Erstellung wurde so definiert, dass der tiefere Punkt als erstes übergeben werden muss. Deswegen müssen die Zweige anschließend noch sortiert werden.

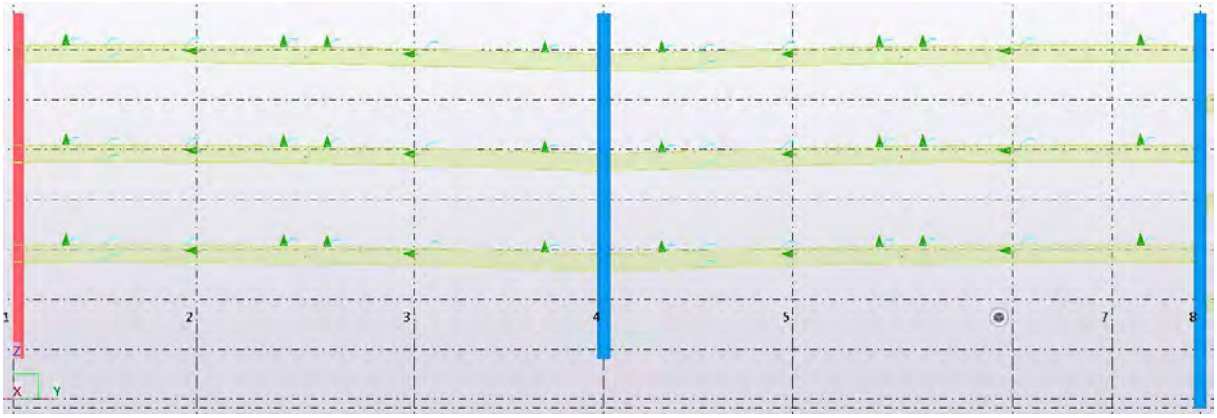


Abbildung 44: Neigung der Träger veranschaulicht durch die Basisträger

Eine weitere besondere Trägerart ist beispielsweise der Träger im Überfahrtsbereich, der die Träger an den Rampen auffangen und deren Lasten an die Randstützen weitergeben soll. Für deren Erstellung müssen zunächst die beiden Stützen, zwischen die der Träger eingefügt werden soll, als Haupt- und Nebenteil übergeben werden. Anschließend müssen noch zwei Referenzpunkte in derselben Baumstruktur wie bei den anderen Trägern übergeben werden. Außerdem müssen die betroffenen Stützen je nach Anzahl der Stockwerke dupliziert werden, um ein passendes Verhältnis von Referenzpunkten zu Bauteilen herzustellen. Dadurch werden allerdings nur die Daten im Skript vervielfacht und nicht die tatsächlich erstellte Anzahl an Stützen im Modell.

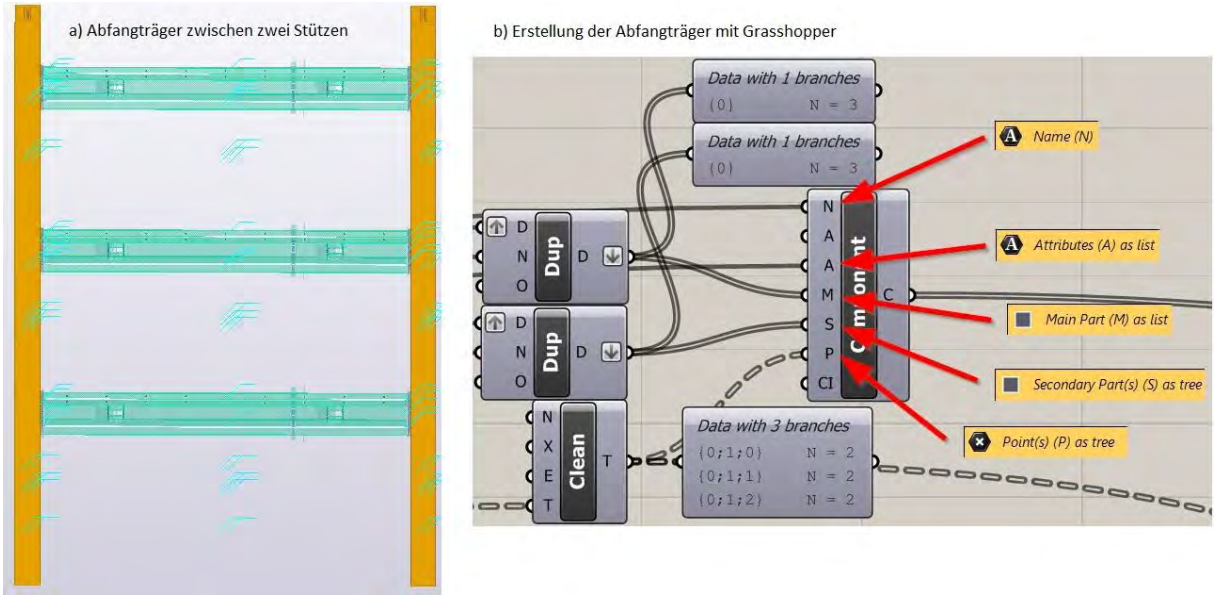


Abbildung 45: Abfangträger

4.3.3.5 Weitere Bauteile (Bereich 12 bis 14)

Alle weiteren Bauteile wurden in Tekla Structures ebenfalls als Komponente modelliert. Die in diesem Skript bereits enthaltenen Bauteile sind die Lampentrapezprofile, die Winkelanschlüsse und die Absturzsicherungen. Auf die Bauteile selbst und deren Erstellung mithilfe von Grasshopper 3D wird nun im Folgenden eingegangen.

Um die Beleuchtung des Parkhauses sicherzustellen, werden sogenannte „Lampentrapezprofile“ eingebaut, auf denen die Beleuchtung installiert wird. Diese Profile verlaufen in Längsrichtung und lagern auf Auflagerschuhen auf, die an den Trägern befestigt sind. Dafür sind auch bereits die Durchlässe im Träger in der Komponente modelliert. In das Modell eingefügt werden sie im Bereich 11 über zwei Referenzpunkte, die sich – wie bei den Trägern – in einem Zweig befinden müssen. Um jeweils den ersten Punkt mit dem zweiten und dem zweiten mit dem dritten und so fort in einem

Zweig unterzubringen, werden zwei um einen Eintrag versetzte Listen zu einem Datenbaum umstrukturiert und anschließend vermengt.

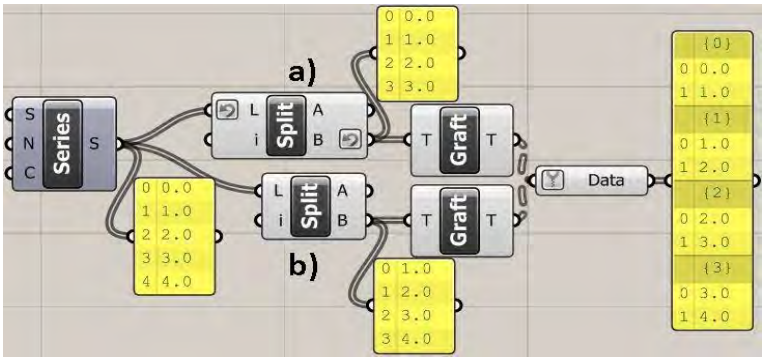


Abbildung 46: Sortierung der Referenzpunkte

Diese Komponente ist diejenige, die für die Erstellung am meisten Zeit benötigt. Teilweise benötigt dieser Vorgang je nach Größe des Modells bis zu 30 Minuten.

Wegen der Zerstörung der Baumstruktur beim Ausgang der Tekla-Blöcke wird im Bereich 12 die Listenstruktur wiederhergestellt. Hier werden die Basisstützen jeder Hauptreihe verwendet und die Aussparungen als leere Einträge in die Liste eingeschoben. So erhält die Liste ihre ursprüngliche Länge und Form. Anschließend können Stützen, die in den meisten Fällen als normale Stützen behandelt werden, wie die Verbandsstützen oder die Innenstützen an den Giebelachsen, an den leeren Stellen in der Liste an den richtigen Stellen eingefügt werden. Es werden außerdem Listen angefertigt, die die einzelnen Stützenabmessungen in Längsrichtung enthalten, um die Bauteile je nach Stützenprofil richtig positionieren zu können.

Für die Erstellung der Winkelanschlüsse im Bereich 13 ist ein Hauptteil und ein Referenzpunkt notwendig. Hierbei müssen die Hauptteile wieder dupliziert werden je nach Anzahl der Stockwerke. Zu den Basishöhen der jeweiligen Seite wird noch sowohl ein bestimmter Versatz hinzuaddiert als auch die Überhöhung auf der betroffenen Seite, die durch die Neigung der Deckenträger resultiert. Hierbei werden die Listen mit wiederhergestellter Struktur verwendet, um anschließend bestimmte Aussparungen aussortieren zu können mithilfe der Indizes der besonderen Bauteile. Eine Stütze, die in den Winkelanschlüssen nicht berücksichtigt wird, ist z.B. die Außenstütze direkt neben einem Treppenhaus auf der Längsseite. Diese ist zwar in der wiederhergestellten Liste vorhanden, wird aber mit keinem Träger verbunden, der eine solche Auflagerung benötigt. Deswegen muss diese Stütze aussortiert werden. Hier wird noch unterteilt zwi-

schen den Winkelanschlüssen in den Hauptreihen mit Aussparungen der Treppenhäuser und der Rampen (in den Innenreihen) Die Rampen erhalten eigene Winkelanschlussdetails, die man in den Attributen bearbeiten kann.

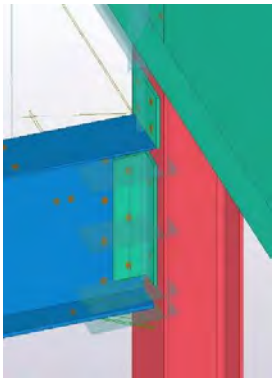


Abbildung 47: Winkelanschluss des Träger an eine Stütze

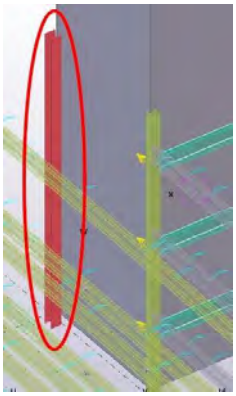


Abbildung 48: Außenstütze direkt neben Treppenhaus

Die Absturzsicherung ist die komplexeste bisher verwendete Komponente. Diese Sicherungen werden i.d.R. über zwei Stellplätze eingebaut. Die Komponente wird erstellt wie die der Überfahrtsträger, es werden also hier auch je Bauteil ein Haupt- und ein Nebenteil sowie zwei Referenzpunkte benötigt. In manchen Fällen kann es aber auch vorkommen, dass einzelne Felder verbaut werden müssen, wenn die Anzahl an Stellplätzen ungerade ist, oder eine Unterbrechung aufgrund eines Treppenhauses oder eine Rampe eine ungerade Zahl der Stellplätze in den vorderen oder hinteren Bereich auslösen.

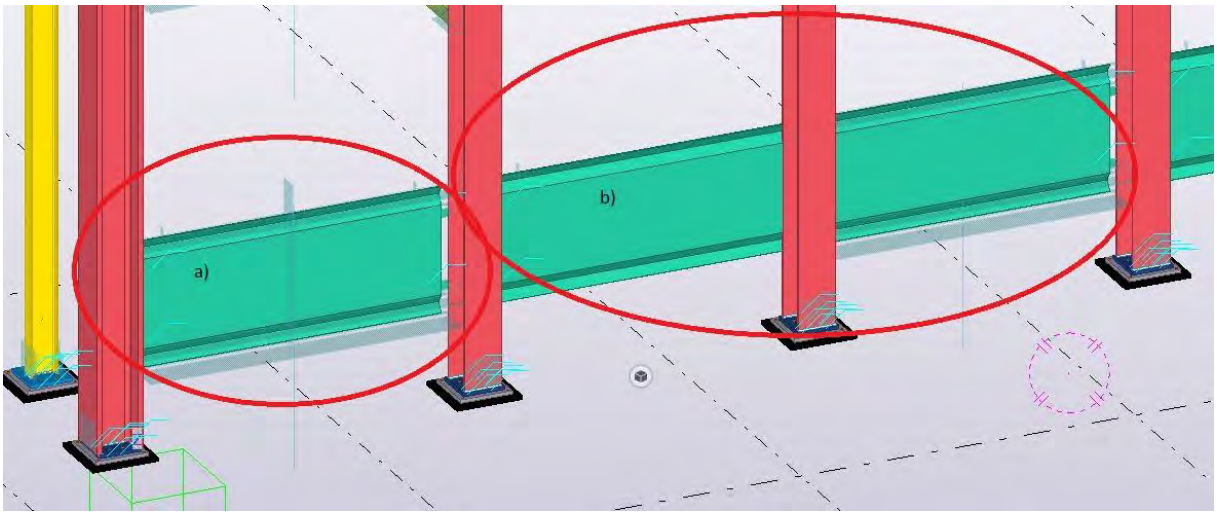


Abbildung 49: Absturzsicherung über ein Feld in a) und über zwei Felder in b)

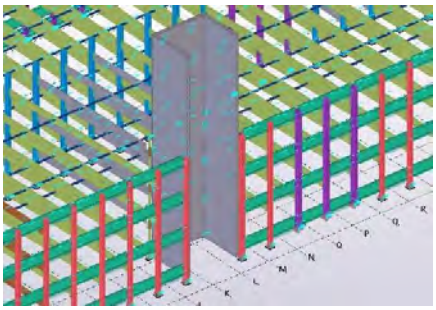


Abbildung 50: Aussparung infolge eines Treppenhauses an der Längsseite

Um Unterbrechungen infolge von Treppenhäusern oder Rampen berücksichtigen zu können, wird die eine wiederhergestellte Liste der betroffenen Stützenreihe, deren Struktur im Bereich 12 erfolgreich wiederhergestellt wurde, auf leere Einträge untersucht in mehrere Listen unterteilt, deren Bereich durch diese leeren Einträge festgelegt werden. Damit kann jeder Bereich für sich betrachtet in einer Baumstruktur verarbeitet werden. Die Indizes der leeren Einträge der Stützenliste werden außerdem dafür verwendet, eine Liste mit Punkten an der betroffenen Längsreihe mit denselben Aussparungen zu erstellen und zu unterteilen. Damit besteht immer die gleiche Anzahl an Stützen wie Punkte. Jeder dieser Bereiche wird nun entsprechend sortiert und in eine für die Komponenten verträgliche Datenstruktur überführt. Diese entspricht der der Überfahrtsträger. Außerdem wird überprüft, ob die Anzahl an Stützen in einem Bereich gerade oder ungerade ist. Ist die Anzahl ungerade, so sind keine einzelnen Elemente nötig. Falls doch, wird die letzte und die vorletzte Stütze und der letzte und der vorletzte Punkt verwendet, um ein einzelnes Abschlusselement zu erzeugen. Der Prozess der Komponentenerzeugung ist zweigeteilt, da ab einer bestimmten Höhe eine andere Art von Absturzsicherung benötigt wird.

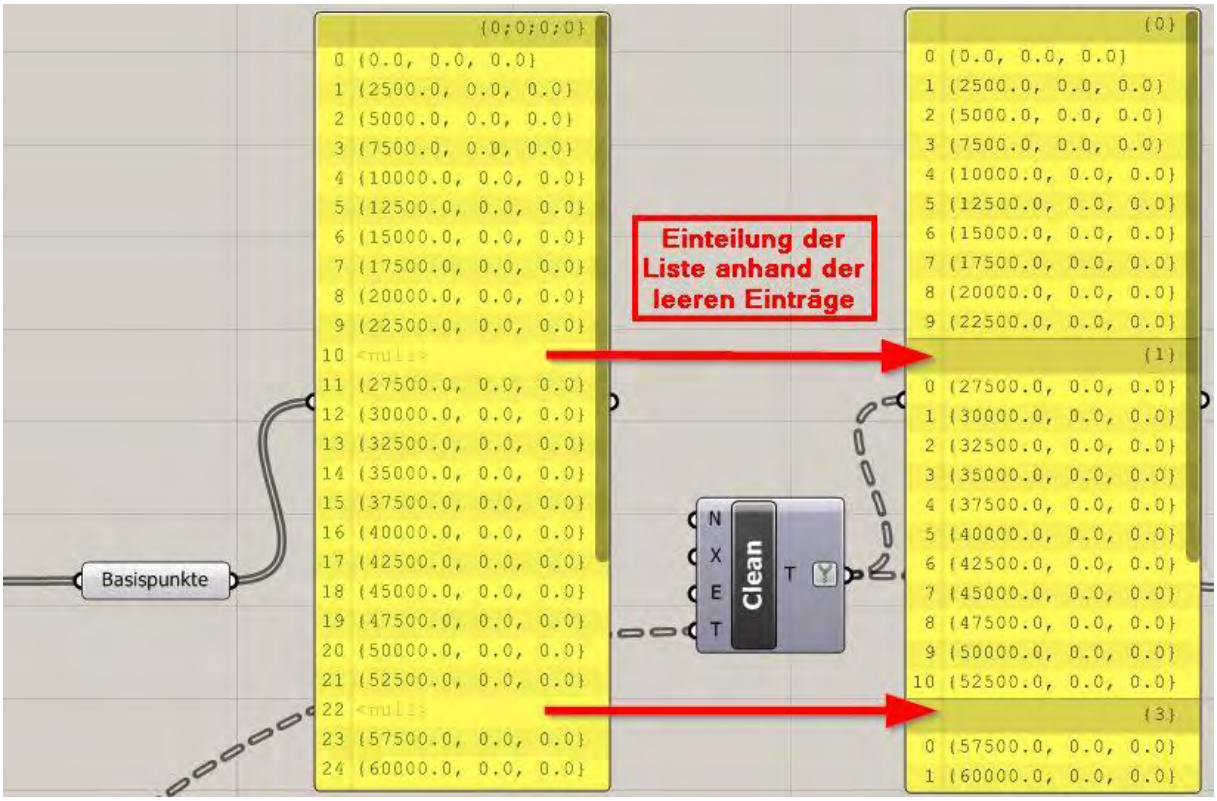


Abbildung 51: Einteilung in einen Baum mehrerer Listen

An der Giebelachse gilt es außerdem noch die Neigung mit zu berücksichtigen und die Abmessungen des Parkhauses. Hierbei sind viele Fallunterscheidungen nötig. Um die Neigung einzupflegen, werden zunächst die absoluten Koordinaten aller Referenzpunkte verwendet, um anschließend die relativen Abstände der Stützen zueinander zu bestimmen, mithilfe derer die Höhe infolge der gegebenen Neigung berechnet wird. So werden die Absturzsicherungen in einer Linie mit einer konstanten Neigung genau positioniert.

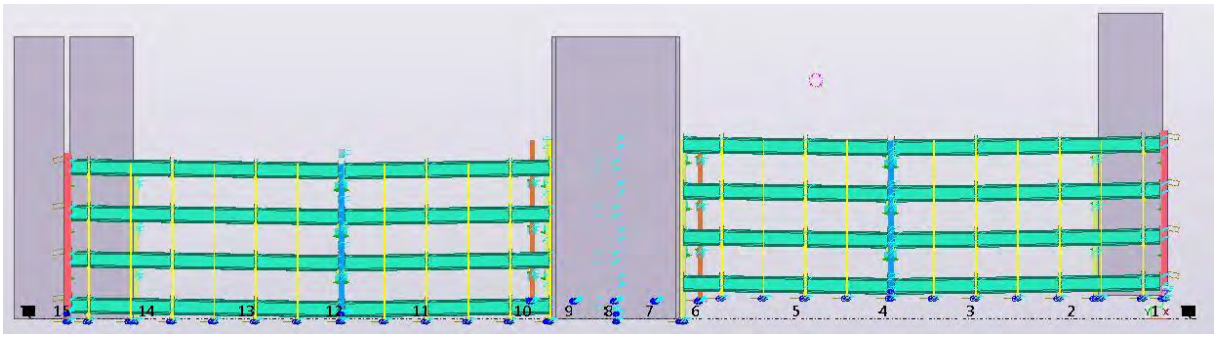


Abbildung 52: Neigung an den Giebelachsen (hier innen überhöht)

In weiteren Schritten könnte man noch weitere bereits als Komponenten implementierte Bauteile mit in das Skript aufnehmen, wodurch das entstehende Basismodell noch mehr Bauteile enthielte. Die meisten dieser Bauteile werden auf gleiche Art und Weise platziert, wie die bereits beschriebenen. Es gibt aber auch kleinere Details, die

entweder momentan noch nicht standardisiert sind und deswegen hier auch noch nicht mit aufgenommen werden können. Generell wird jedoch mehr Zeit eingespart, je mehr Bauteile integriert sind.

4.3.4 Überblick

Um einen Überblick über die verwendeten Abläufe und Programme zu geben, wird hier zusammenfassend die Vorgehensweise und die Ordnerstruktur des Generators dargestellt.

Der Nutzer bekommt einen Ordner namens „Parkhausgenerator“ zur Verfügung gestellt, der eine Excel-Tabelle, ein Benutzerhandbuch und einen weiteren Unterordner enthält. Für den Nutzer sind nur die Excel-Tabelle und das Benutzerhandbuch relevant, während in dem Unterordner für die Benutzung wichtige Dateien enthalten sind. Aus der Excel-Tabelle heraus wird jeder notwendige Prozess automatisch gestartet oder kommuniziert. Beim Öffnen der Tabelle wird zunächst automatisch überprüft, ob die verwendeten Plugins für Grasshopper 3D bereits installiert sind. Falls nicht, wird dies automatisch vollzogen und der Benutzer anschließend auf die Eingabemaske weitergeführt. Falls sie schon installiert sind, wird der Einrichtvorgang übersprungen und unmittelbar danach die Eingabemaske gestartet. Hierin kann der Nutzer nun die Parkhausdaten eingeben. Hat der Nutzer alle Eingaben getätigt, so kann er für die Modellerzeugung auf die Schaltfläche „Modell erzeugen“ klicken. Hier gibt es die Möglichkeit, bereits in der Excel-Tabelle hinterlegte Daten zu verwenden. Das ist v.a. sinnvoll, wenn die Daten von einem anderen Mitarbeiter eingepflegt wurden. Hierzu kann der Benutzer wählen zwischen den bereits hinterlegten und den neu in der Eingabemaske eingegebenen Daten. Je nach Wahl werden die eingegebenen Daten noch auf Überschneidungen überprüft. Als nächstes wird dem Benutzer mitgeteilt, alle weiteren Excel-Tabellen zu schließen, das Programm Tekla Structures zu öffnen und eine Modellvorlage zu laden. Das Programm selbst lässt sich über eine Schaltfläche starten, die Modellvorlage muss allerdings noch manuell geladen werden. Anschließend kann er über die Schaltfläche „Modell erzeugen“ Grasshopper 3D starten und das Skript laden, in dem das Parkhaus in parametrisierter Form hinterlegt ist. Anschließend wird das Parkhausmodell in Tekla Structures erzeugt (siehe Anhang A.1 bis A.3). Dieser Vorgang dauert ca. eine halbe Stunde. Nachdem das Modell erzeugt wurde, muss der Generator noch manuell vom Nutzer wieder geschlossen werden.

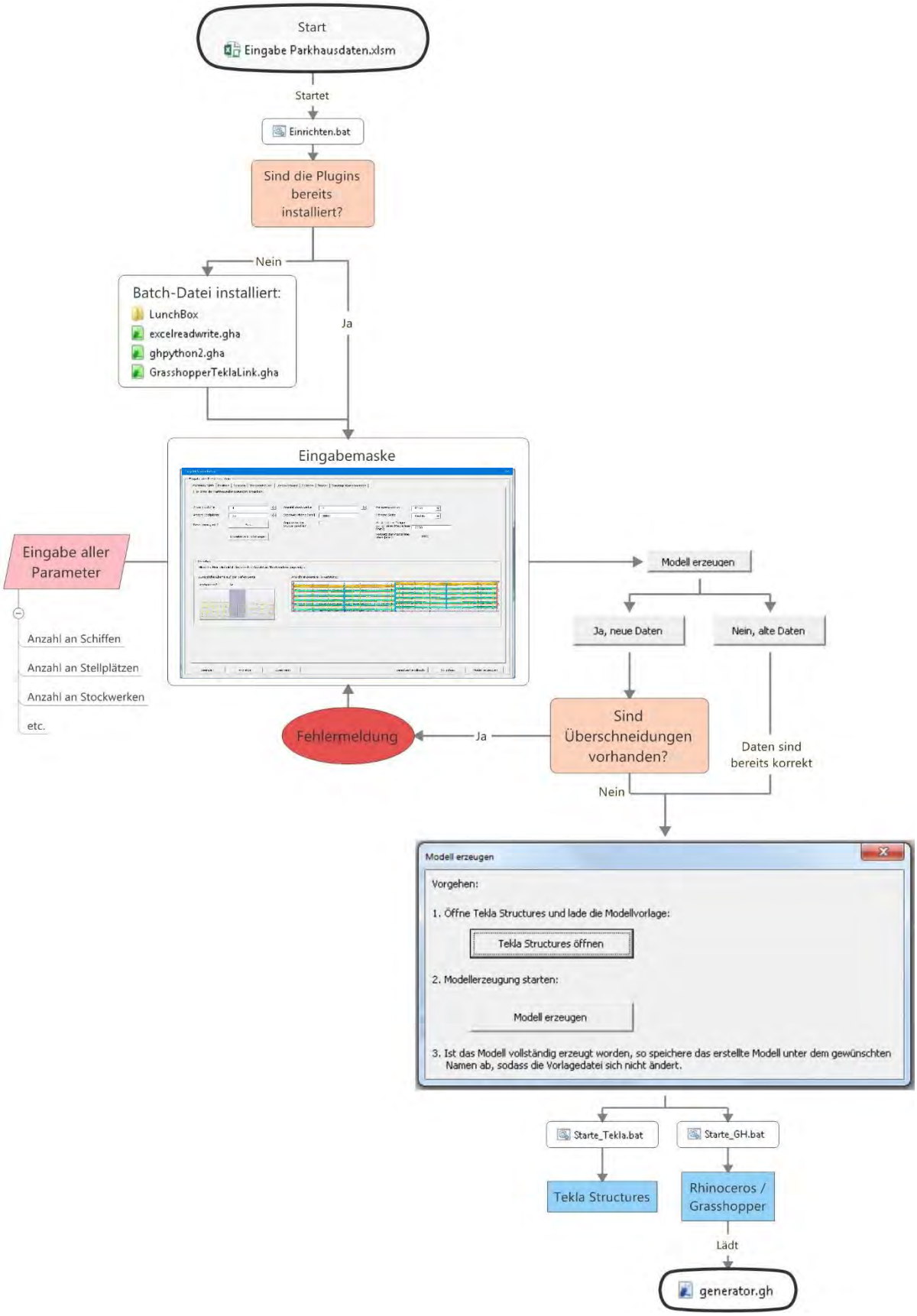


Abbildung 53: Vorgehensweise zur Erzeugung eines Basismodells

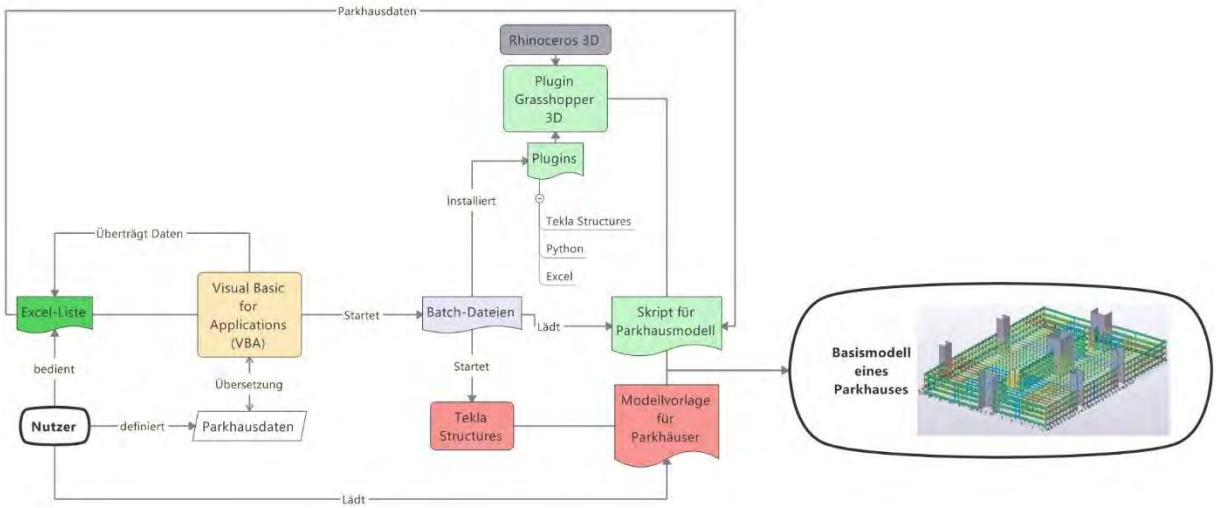


Abbildung 54: Verknüpfung aller beteiligten Programme, Daten und des Nutzers

Der unter 2.3.1 erwähnte Ansatz der absoluten Parametrik kann hier auch noch verdeutlicht werden. Hier stellt Grasshopper 3D sozusagen die absolute Instanz dar, die sowohl die äußeren Parameter steuert, die für die Abmessungen des Parkhauses zuständig sind, als auch auf die Parametrik in Tekla Structures zugreift.

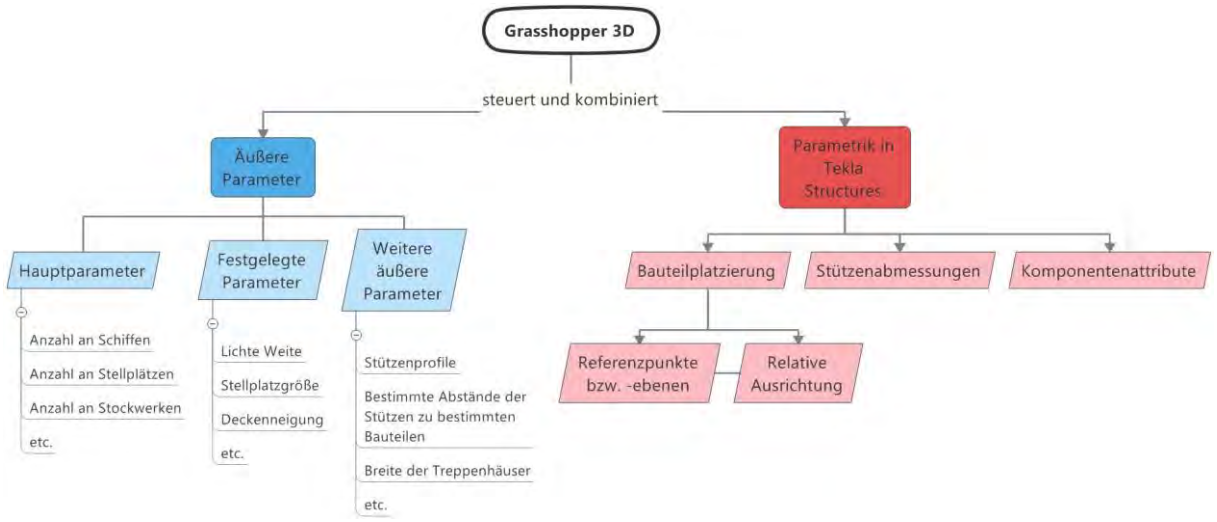


Abbildung 55: Übersicht über die verwendete Parametrik

5 Fazit und Ausblick

Zusammenfassend ist es möglich, einen Parkhausgenerator eines Modelles mit LOD 450 mithilfe von Grasshopper 3D in Tekla Structures zu schaffen. Allerdings müssen dafür ein weit entwickeltes Bausystem und ein definierter Satz an Bauteilen bereits vorhanden sein. Die Serienschaltung der Komponenten macht es möglich, das gesamte Skript mithilfe einer Schaltfläche zu aktivieren, wodurch das gesamte Basismodell vom Benutzer mit nur einem Klick erstellt werden kann. Da der Benutzer i.d.R. keine Kenntnisse über die Schnittstelle von Tekla Structures und Grasshopper 3D hat, muss auf weitere Programmiersprachen ausgewichen werden, die strukturelle Schwächen von Grasshopper 3D bzw. dessen Schnittstelle zu Tekla Structures ausgleichen oder notwendige Prozesse automatisch erledigen. Voraussetzung dafür ist die Beherrschung einiger weiterer Programmiersprachen, was im Widerspruch zu der leichten Bedienung von visuellen Programmiersprachen steht. Diese „leichte“ Programmierung wäre gegeben, wenn der Nutzer selbst mit Grasshopper 3D vertraut wäre oder sogar das Skript selbst schriebe. Ein weiterer anzuführender Aspekt ist, dass der Generator der Entwicklung des Bausystems in gewissem Maße hinterherläuft. Gerade bei kurzfristigen Ideen oder Strategien muss die Neuigkeit erst in den Generator eingepflegt werden. Deswegen muss die Entwicklung des Generators Hand in Hand mit der Entwicklung des Bausystems zusammenarbeiten und aktuell gehalten werden. Sicherlich wird er jedoch mit seinen Aufgaben wachsen.

Alles in allem lässt sich festhalten, dass durch den Einsatz dieses Plugins und die schnelle Modellerzeugung wertvolle Zeit und damit Geld in der Planung eingespart werden kann.

Zu guter Letzt soll noch einen Ausblick über eine mögliche weitere Vorgehensweise gegeben werden. Für die Planung der Parkhäuser werden für jeweils drei Modelle angefertigt: eins in Dynamo zur Angebotserstellung und zur Mengenkalkulation, zwei in Tekla Structures für einmal den Betonbau und einmal den Stahlbau. Allen drei Modellen liegen die gleichen grundlegenden Abmessungen zugrunde. Allerdings sind nicht alle Informationen für alle Abteilungen relevant, da sich die Prozesse nach der Modellerstellung in den Abteilungen unterscheiden. Wird nun jedes Modell einzeln per

Hand erstellt, können Fehler und Unstimmigkeiten zu den anderen Modellen entstehen. Dadurch wird ständiger Abgleich und Kontrolle essentiell notwendig. Mit folgender Idee könnten drei übereinstimmende Basismodelle für die drei Abteilungen innerhalb kürzester Zeit erstellt und die Grundstruktur der Parkhausplanung beibehalten werden. Man könnte für die Angebotserstellung einen neuen Generator basierend auf Dynamo einsetzen, der auch von einer Excel-Datei angesteuert wird. Die Benutzereingabe erfolgt also auch über eine Eingabemaske. Hier gibt der Kollege die für ihn relevanten Daten ein, um damit sein Revit-Modell zu erzeugen. Somit kann er dem Kunden zeitnah oder sogar direkt in der Besprechung erste Visualisierungen des Parkhauses präsentieren. Nachdem der Auftrag erteilt wurde, würde diese Excel-Liste mit den am Ende abgestimmten Daten an den Betonbau weitergegeben, welcher wieder die für sich relevanten Daten eingibt, um zusammen mit den Basisinformationen, die bereits von den Kollegen der Angebotserstellung eingegeben worden sind, und einem weiteren Grasshopper-Skript ein Modell für den Betonbau zu erzeugen. Diese Informationen werden wiederum digital über die Excel-Datei an den Stahlbau weitergegeben, der auf diese Informationen aus dem Betonbau aufbaut. Auf diese Weise würden alle Informationen in einer Datei abgespeichert sein, welche von den einzelnen Modellerzeugungsskripten ausgelesen und weiterverarbeitet werden. Hierbei würden drei Basismodelle entstehen, denen alle die gleiche Basis an Informationen zu Grunde liegt. Alle weiteren Informationen, welche nicht in den Generatoren abgebildet werden, müssten allerdings weiterhin wie vorher noch manuell modelliert und abgesprochen werden. Dazu ist eine Abstimmung notwendig. Die Informationen, die noch über 2D-Pläne übermittelt werden könnten so digital übergeben werden, was einen weiteren Schritt in Richtung „BIG BIM“ darstellen könnte.

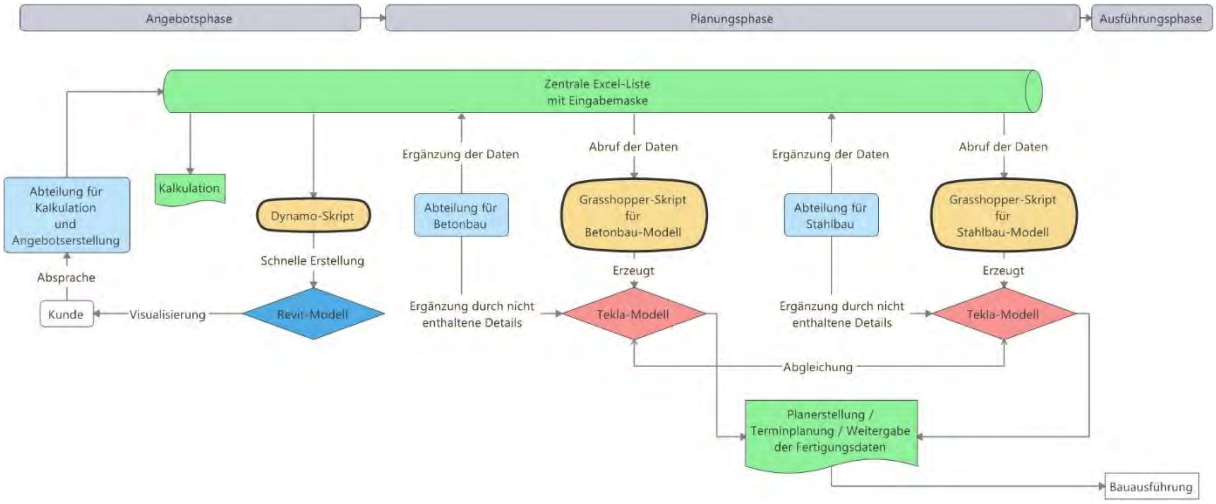


Abbildung 56: Ablauf mit einer zentralen Excel-Liste

Um die Abteilungen nur die für sie relevanten Daten eingeben zu lassen bzw. um die anderen Bereiche zu schützen, könnte man die Eingabemaske in drei Bereiche unterteilen. Bei Benutzung der Eingabemaske könnte zunächst abgefragt werden, zu welcher Abteilung der Bearbeiter gehört. Je nachdem, welchen Bereich dieser wählt, kann er die jeweiligen Daten bearbeiten. Diese Bereiche könnte man zusätzlich mit Passwörtern schützen, um zu gewährleisten, dass nur zuständige Personen den jeweiligen Bereich bearbeiten können.

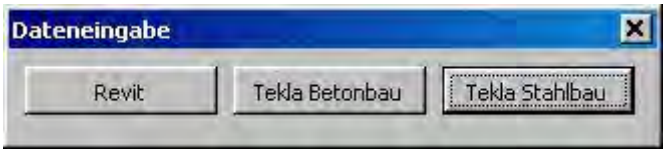
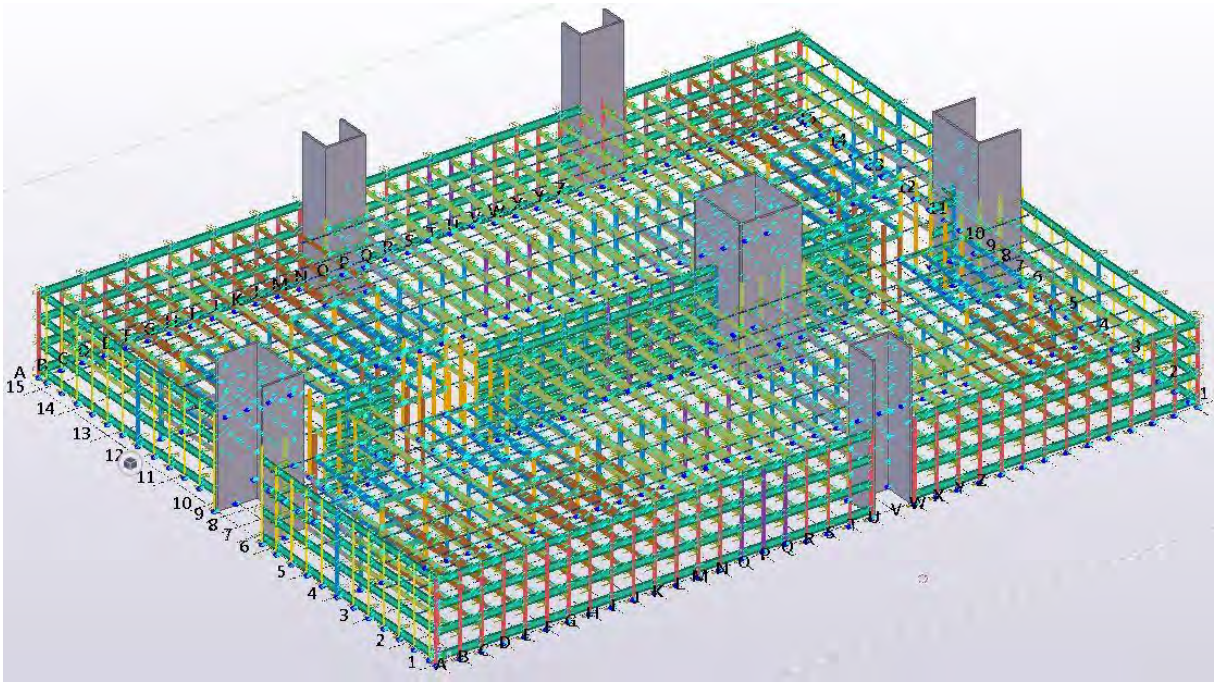


Abbildung 57: Abteilungswahl

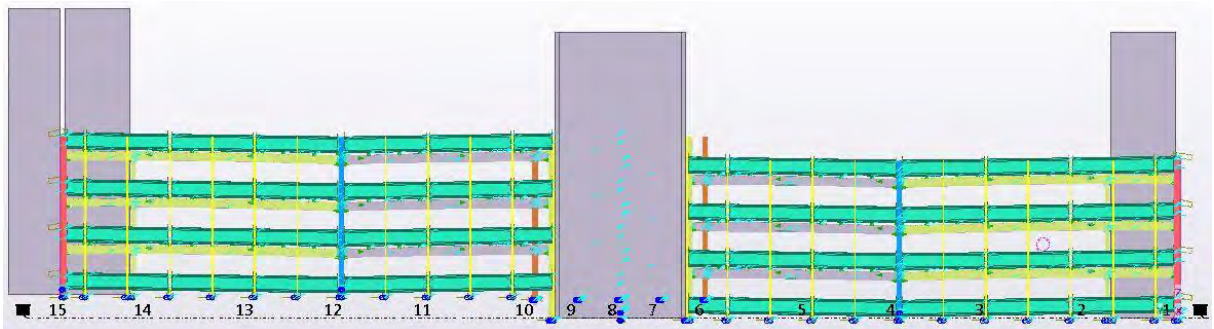
Die Voraussetzungen für diese Idee sind gegeben, zumal bereits eine Verbindung zwischen Excel und Dynamo besteht. Die frühe Planungsphase würde bei einer Erstellung eines Modells durch den Einsatz eines solchen Generators noch einmal stark verkürzt werden und somit die Kosten für die aufwändige manuelle Modellerstellung sparen.

Anhang A

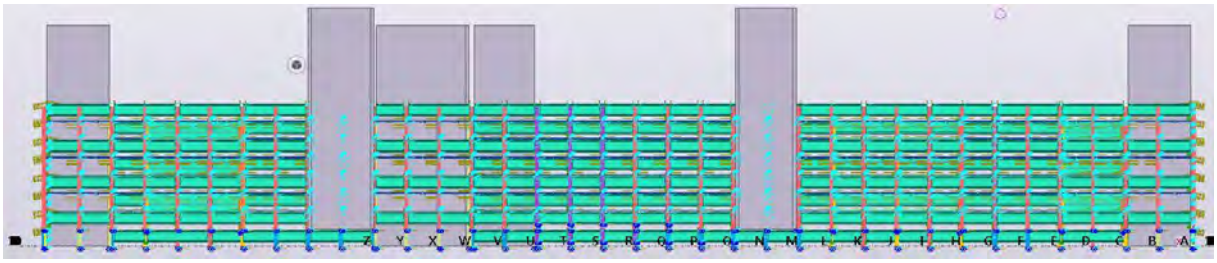
A.1 3D-Ansicht eines automatisch generierten Modells mit verschiedenen Treppenhaus- und Rampenvariationen



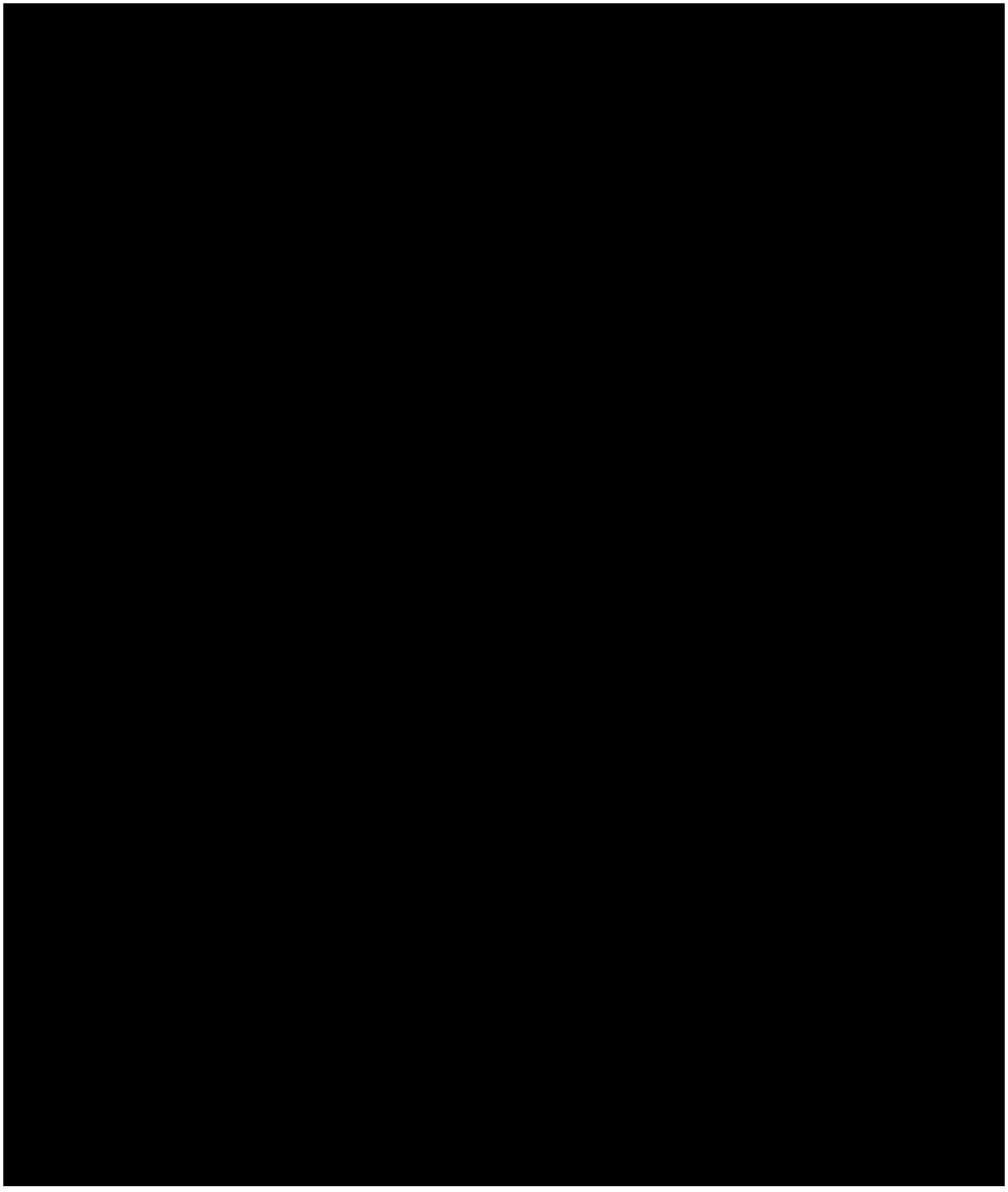
A.2 2D-Ansicht der Giebelseite eines automatisch generierten Modells mit verschiedenen Treppenhaus- und Rampenvariationen



A.3 2D-Ansicht der Längsseite eines automatisch generierten Modells mit verschiedenen Treppenhaus- und Rampenvariationen



A.4 Vergleich der Excel-Vorschau mit dem generierten Modell



Anhang B

Auf dem beigefügten Datenträger befindet sich folgender Inhalt:

- Der schriftliche Teil der Arbeit als Worddokument
- Videos zur Veranschaulichung der Eingabemaske, des Vorgangs des Einrichtens und der Modellerzeugung

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelor-Thesis selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ich versichere außerdem, dass die vorliegende Arbeit noch nicht einem anderen Prüfungsverfahren zugrunde gelegen hat.

München, 13. April 2018 Stephan, Oelfe

Vorname Nachname

Stephan Oelfe

Hinterer Berg 24

D-92360 Mühlhausen

stephan.oelfe@tum.de

Literaturverzeichnis

Allgemeiner Deutscher Automobil-Club e.V., Ressort Verkehr (Hg.) (2014): Benutzerfreundliche Parkhäuser. Online verfügbar unter https://www.adac.de/_mmm/pdf/fi_benutzerreundliche_parkhauser_0114_238764.pdf, zuletzt geprüft am 13.01.2018.

Borrmann, André; König, Markus; Koch, Christian; Beetz, Jakob (Hg.) (2015): Building Information Modeling. Technologische Grundlagen und industrielle Praxis. Wiesbaden: Springer Vieweg (VDI-Buch).

Dammeyer, Klaus (2014): Systemparkhaus in Beton-Stahlverbund-Bauweise. In: *Stahlbau* 83 (10), S. 759–760. DOI: 10.1002/stab.201420208.

Der Dynamo Primer | Der Dynamo Primer (2017). Online verfügbar unter http://dynamoprimer.com/de/index.html, zuletzt aktualisiert am 10.07.2017, zuletzt geprüft am 05.04.2018.

Grasshopper - Neu in Rhino 6. Online verfügbar unter https://www.rhino3d.com/de/6/new/grasshopper, zuletzt geprüft am 05.04.2018.

Kleinmanns, Joachim (2011): Parkhäuser. Architekturgeschichte einer ungeliebten Notwendigkeit. 1. Aufl. Marburg: Jonas-Verl.

Lagios, Kera; Niemasz, Jeff; Reinhart Christoph F.: ANIMATED BUILDING PERFORMANCE SIMULATION (ABPS) – LINKING RHINOCEROS/GRASSHOPPER WITH RADIANCE/DAYSIM. Online verfügbar unter https://web.archive.org/web/20100910201013/http://www.gsd.harvard.edu/research/gsdsquare/Publications/DaylightingAnalysisInRhinoAndGrasshopper.pdf, zuletzt geprüft am 13.01.2018.

Max Bögl GmbH: PSB – Parkhaus Systeme Bögl. Online verfügbar unter https://www.max-boegl.de/downloads/656-psb-parkhaus-systeme-boegl/file.html, zuletzt geprüft am 10.04.2018.

Preidel, Cornelius; Daum, Simon; Borrmann, André (2017): Data retrieval from building information models based on visual programming. In: *Vis. in Eng.* 5 (1), S. 89. DOI: 10.1186/s40327-017-0055-0.

Schiffer, Stefan (1998): Visuelle Programmierung - Grundlagen und Einsatzmöglichkeiten. Online verfügbar unter http://www.schiffer.at/vp/Stefan%20Schiffer%20-%20Visuelle%20Programmierung.pdf, zuletzt geprüft am 13.01.2018.

Smith, David Canfield (1975): PYGMALION: A Creative Programming Environment. Online verfügbar unter http://worrydream.com/refs/Smith%20-%20Pygmalion.pdf, zuletzt geprüft am 13.01.2018.

Tekla Structures. Online verfügbar unter https://www.tekla.com/de/produkte/tekla-structures, zuletzt geprüft am 06.04.2018.

Trimble Solutions Germany (2018): Firmengruppe Max Bögl: Ausgefeilte Prozesse mit BIM . Online verfügbar unter https://www.tekla.com/de/referenzen/max-boegl-vorfertigung-mit-bim, zuletzt aktualisiert am 12.01.2018, zuletzt geprüft am 13.01.2018.