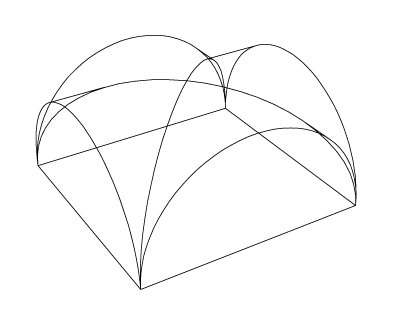
Kreuzgewölbe

Oberflächenberechnung im Detail



Merlin Mabon Worrings, Stephanie Eckert, Sophie Platen, Paul Hahn,  
Lars Hollenbach, Johanna Harder, Jannik Spiering

Inhalt

[2 Einleitung 2](#_Toc485678349)

[3 Ausgangsituation 2](#_Toc485678350)

[3.1 Ausarbeitung eines Lösungsansatzes für die Flächenberechnung eines Kreuzgewölbes 2](#_Toc485678351)

[4 Ressourcen- und Ablaufplanung 2](#_Toc485678352)

[5 Pflichtenheft 2](#_Toc485678353)

[6 Protokolle 3](#_Toc485678354)

[7 Durchführung und Auftragsbearbeitung 3](#_Toc485678355)

[8 Verfahren 4](#_Toc485678356)

[8.1 Ablauf 4](#_Toc485678357)

[9 Approximieren neuer Datenpunkte 5](#_Toc485678358)

[10 Der Ball-Pivoting Algorithmus (BPA) 9](#_Toc485678359)

[11 Diagramme 10](#_Toc485678360)

[12 Projektergebnisse 10](#_Toc485678361)

[12.1 Programm 10](#_Toc485678362)

[13 Fazit 11](#_Toc485678363)

[14 Quellenangaben 11](#_Toc485678364)

[15 Anhang 11](#_Toc485678365)

# Einleitung

Im Rahmen des Projektes „Kreuzgewölbe“ teilten sich die Schüler der Klasse MTS 41 selbständig in Gruppen ein. Ziel der Arbeit war es, die Fläche eines Kreuzgewölbes mit möglichst geringen Fehler zu berechnen. Als Grundlage der Berechnung dienten computergenerierte Messdaten sowie ein virtuelles Kreuzgewölbe welche auch bei der Validierung der entworfenen Algorithmen Verwendung fanden.

Der Fokus des Projektes lag in der Anwendung und Modellierung von mathematischen Modellen sowie im Projektmanagement. Die Schüler teilten das Projekt hierfür in Arbeitspakete innerhalb ihres Teams ein und arbeiteten Zielorientiert an der Umsetzung.

Die in diesem Bericht vorgestellte Gruppe wurde von Merlin Mabon Worrings geleitet. Zu den Mitgliedern des Teams zählten Stephanie Eckert, Sophie Platen, Paul Hahn, Lars Hollenbach, Johanna Harder und Jannik Spiering. Das Team einigte sich zu Projektbeginn, die Umsetzung mithilfe der Programmiersprache C# zu realisieren. Aus Kompatibilitätsgründen mit der bereitgestellten Entwicklungsumgebung wird für die Ausführung das .NET Framework 3.5 vorausgesetzt.

# Ausgangsituation

Die ursprüngliche Aufgabenstellung des Projektes sah vor ein reales Kreuzgewölbe zu vermessen. Bei der Auswahl der verwendeten Algorithmen spielten Bedenken bezüglich der Genauigkeit der Messdatenerfassung eine wichtige Rolle. Grund hierfür war die Tatsache das einige der vorgeschlagenen Verfahren stark von bestimmten Eigenschaften der gemessenen Punkte abhängig sind. Eine fehlerfreie Simulation mit künstlich erzeugten Werten ist mit diesen Algorithmen zwar uneingeschränkt möglich, die Verwendung von realen Messdaten wird hingegen gegebenenfalls aber nur unter Einschränkungen oder unzureichend unterstützt. Diese Bedenken mündeten letztendlich in der Suche einer alternativen Lösung mit hoher durch das Messverfahren bedingten Fehlertoleranz.

Kubische Splines

Lineare Regression

## Ausarbeitung eines Lösungsansatzes für die Flächenberechnung eines Kreuzgewölbes

Das Arbeiten mit abweichungsbehafteten Messdaten resultierte in der Auswahl eines Verfahrens welches sich besonders Robust gegenüber solchen Abweichungen verhält.

# Ressourcen- und Ablaufplanung

# Pflichtenheft

Zur Umsetzung der Anforderung, die Oberfläche eines durch vorher gemessene Punkte definierten Kreuzgewölbes zu berechnen, gedenken wir ein mehrstufiges Verfahren einzusetzen.

Zunächst wollen wir die Messdaten im Raum Triangulieren, um mit möglichst vielseitigen Eingabedaten arbeiten zu können und die Messbedingungen einfach zu halten.  
Damit existiert bereits eine Approximation der gemessenen Oberfläche, die aber, je nach Anzahl der gemessenen Punkte, nicht immer eine hinreichende Genauigkeit hat.  
Deswegen erstellen wir einen Algorithmus, mit dessen Hilfe wir die Genauigkeit steigern können, indem wir die gemessene Fläche an die tatsächliche Fläche (das Kreuzgewölbe) annähern. Dies soll durch Ersetzen jedes Dreiecks der Triangulation durch jeweils vier kleinere und genauere Dreiecke geschehen. Diese wollen wir anhand einer Funktion bestimmen, welche wir über jeder Kante der Triangulation berechnen. Dabei gehen wir davon aus, dass die Messpunkte auf einen Bogen gemessen wurden. Mithilfe der Kantenfunktion wollen wir diesen Bogen ermitteln, sodass wir uns ihm annähern können.

Die Anwendung des Algorithmus soll sich beliebig oft wiederholen lassen. Seine Anwendung soll unter der Vorbedingung, dass das ausgemessene Kreuzgewölbe keine Unregelmäßigkeiten vorweist, die in den Messdaten nicht erfasst wurden, dazu führen dass die berechnete Oberfläche nur noch insignifikant von der tatsächlichen Oberfläche abweicht.

# Protokolle

# Durchführung und Auftragsbearbeitung

# Verfahren

## Ablauf

|  |
| --- |
|  |
|  |
|  |

# Messdatenerzeugung

# Triangulation

## Der Ball-Pivoting Algorithmus (BPA)

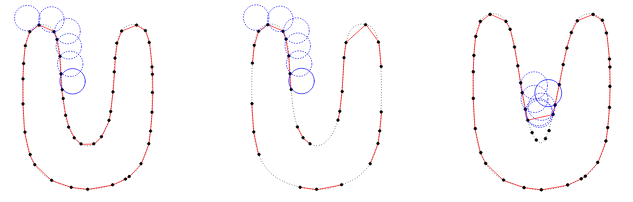
Der Ball-Pivoting Algorithmus (BPA) ist ein Algorithmus zur Oberflächen Rekonstruktion, bei dem man eine Kugel über eine Menge von Punkten rollt, und so die korrekten Faces des Mesh herstellt.

Man beginnt mit einem bekannten Dreieck, in welchem die Kugel liegt. Dabei muss die Kugel alle drei Punkte berühren. Nun wird begonnen die Kugel über eine der Kanten zu rollen. Wenn die Kugel auf einen anderen Punkt trifft, also die Kugel wieder auf allen drei Punkten liegt aber mit keinem anderen Punkt kontakt hat, wurde ein neues Face gefunden. Dieser Vorgang wird für alle Kanten der bekannten Faces wiederholt. Wenn bei einer Kannte kein Punkt gefunden wurde, ist diese eine Rand-Kante und wird entsprechend markiert.

Da das Überprüfen, ob sich ein Punkt mit der Kugel schneidet, bei einem Durchlauf mit besonders vielen Punkten sehr aufwändig wird, werden im BPA die Punkte in Voxel vorsortiert. Da wir in unserem Fall mit einer Menge an Punkten rechnen, die voraussichtlich kleiner als 100 ist, haben wir diesen Teil des Algorithmus nicht implementiert, da der Aufwand nicht gerechtfertigt wird.

Der Vorteil dieses Algorithmus ist, dass er auch komplexere Meshes wiederherstellen kann. Da die Form eines Kreuzgewölbes relativ einfach ist, dürfte das Ergebnis also dementsprechend fehlerfrei ausfallen.

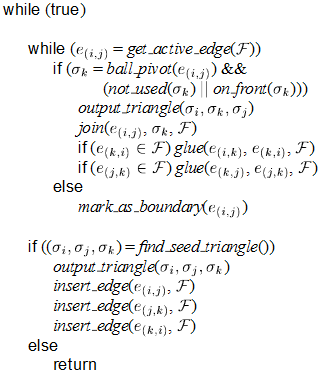
Ein Problem des Algorithmus ist es, den richtigen Radius für die Kugel festzulegen, da wenn dieser zu klein ist, die Kugel mögliche Punkte nicht erreicht. Wenn die Kugel jedoch zu groß ist, kann es sein, dass sie stecken bleibt.



Um Fehler zu vermeiden, wird bei komplexeren Meshes empfohlen, den Algorithmus öfter mit verschiedenen Kugelradien und Startpositionen laufen zu lassen. Was wiederum in unserem Anwendungsfall nicht notwendig sein wird.

Der Ball Pivot Algorithmus ist wie er von IBM erstellt wurde recht umfangreich und undurchsichtig. Zudem gibt es wenige oder nur unzureichende Quellen zu diesem Algorithmus, was eine Implementierung schwierig macht. Aufgrund dessen wurde entschieden, dass der Algorithmus den Umfang des Projektes übersteigt und davon abgesehen diesen zu implementieren. Um den eigentlichen Algorithmus zu testen wurden Testdaten in Form von einer Liste von Dreiecken generiert, was da wir keine realen Testdaten haben sowieso die sinnvollere Methode ist.

Der Pseudocode von IBM:



Quelle: <http://www.research.ibm.com/vistechnology/pdf/bpa_tvcg.pdf>, 22.05.17

# Approximieren neuer Datenpunkte

Durch das Approximieren neuer Datenpunkte soll sich die Dreiecksstruktur der eigentlichen Oberflächenform annähern. Um dies zu realisieren, werden folgende Schritte durchgeführt:

1. Ungeordnete Messpunkte wurden trianguliert. Die betrachtete Oberfläche liegt als Dreieckstruktur vor.

Tabelle 1: Messpunkte

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Punkt** | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** |
| **x** | 1,23 | 0,31 | 2,87 | 0,00 | 5,00 | 0,00 | 1,95 | 3,92 | 2,53 | 4,40 | 0,45 |
| **y** | 2,62 | 3,69 | 2,74 | 5,00 | 0,00 | 0,00 | 4,30 | 2,94 | 0,94 | 1,54 | 1,23 |
| **z** | 4,08 | 3,36 | 3,04 | 0,00 | 0,00 | 5,00 | 1,65 | 0,98 | 4,21 | 1,82 | 4,83 |

|  |  |  |
| --- | --- | --- |
| https://raw.githubusercontent.com/Stephanie91/Kreuzgew-lbe-Projekt/Develope/teil2.png | **Dreiecke** | |
| 1 | ACB |
| 2 | BCG |
| 3 | BGD |
| 4 | CHG |
| 5 | CJH |
| 6 | AIC |
| 7 | CIJ |
| 8 | AKI |
| 9 | ABK |
| 10 | BFK |

Abbildung 1: Triangulation auf einer Kugelteilfläche

1. Für jedes Dreieck werden die Normale der Ebene berechnen, die durch dessen drei Ecken definiert wird. Dabei ist wichtig, dass alle Normalen auf dieselbe Seite der Fläche zeigen.

Tabelle 2: Normalvektoren der Dreiecksflächen

|  |  |  |  |
| --- | --- | --- | --- |
| **ACB** | 1,0264 | **AIC** | 1,7316 |
| 2,1376 | 1,5652 |
| 1,8652 | 2,9112 |
| **BCG** | 1,8197 | **CIJ** | 3,6000 |
| 3,8528 | 1,3753 |
| 3,1196 | 3,1620 |
| **BGD** | 0,1905 | **AKI** | 1,0793 |
| 6,0405 | 1,0764 |
| 2,3375 | 3,1174 |
| **CHG** | 2,9356 | **ABK** | -0,1983 |
| 3,3547 | 1,2516 |
| 1,8220 | 2,1134 |
| **CJH** | 2,7160 | **BFK** | -1,3899 |
| 1,8708 | 0,6853 |
| 1,5660 | 1,2792 |

1. Für jede Dreiecks-Ecke (Vertex) wird der Normalvektoren berechnet.
   * Die Normalvektoren der Dreiecke werden gewichtet. Hierzu wird der Schwerpunkt des jeweiligen Dreiecks berechnet und der Normalvektor durch den Abstand zwischen Vertex und Schwerpunkt ermittelt.
   * Für jeden Vertex wird der Durchschnittsnormalvektor aller gewichteten Normalvektoren der anliegenden Dreiecke wird ermittelt. (im Folgenden wird nur noch beispielhaft die Kante AB betrachtet)

Tabelle 3: Schwerpunkt der Dreiecksflächen und der Abstand zwischen Schwerpunkt und Vertex

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Dreiecke** | **Schwer-punkt** | **Abstand zu A** | **Abstand zu B** | **Dreiecke** | **Schwer-punkt** | **Abstand zu A** | **Abstand zu B** |
| **ACB** | 1,47 | 0,75 | 1,35 | **AIC** | 2,21 | 1,15 | --- |
| 3,02 | 2,10 |
| 3,49 | 3,78 |
| **BCG** | 1,71 | --- | 1,56 | **CIJ** | 3,27 | --- | --- |
| 3,58 | 1,74 |
| 2,68 | 3,02 |
| **BGD** | 0,75 | --- | 1,86 | **AKI** | 1,40 | 1,08 | --- |
| 4,33 | 1,60 |
| 1,67 | 4,37 |
| **CHG** | 2,91 | --- | --- | **ABK** | 0,66 | 0,58 | 1,43 |
| 3,33 | 2,51 |
| 1,89 | 4,09 |
| **CJH** | 3,73 | --- | --- | **BFK** | 0,25 | --- | 2,30 |
| 2,41 | 1,64 |
| 1,95 | 4,40 |

Tabelle 4: Normalvektoren der Vertices

|  |  |
| --- | --- |
| **Punkt A** | **Punkt B** |
| 0,88 | 0,26 |
| 1,85 | 1,70 |
| 2,90 | 1,34 |

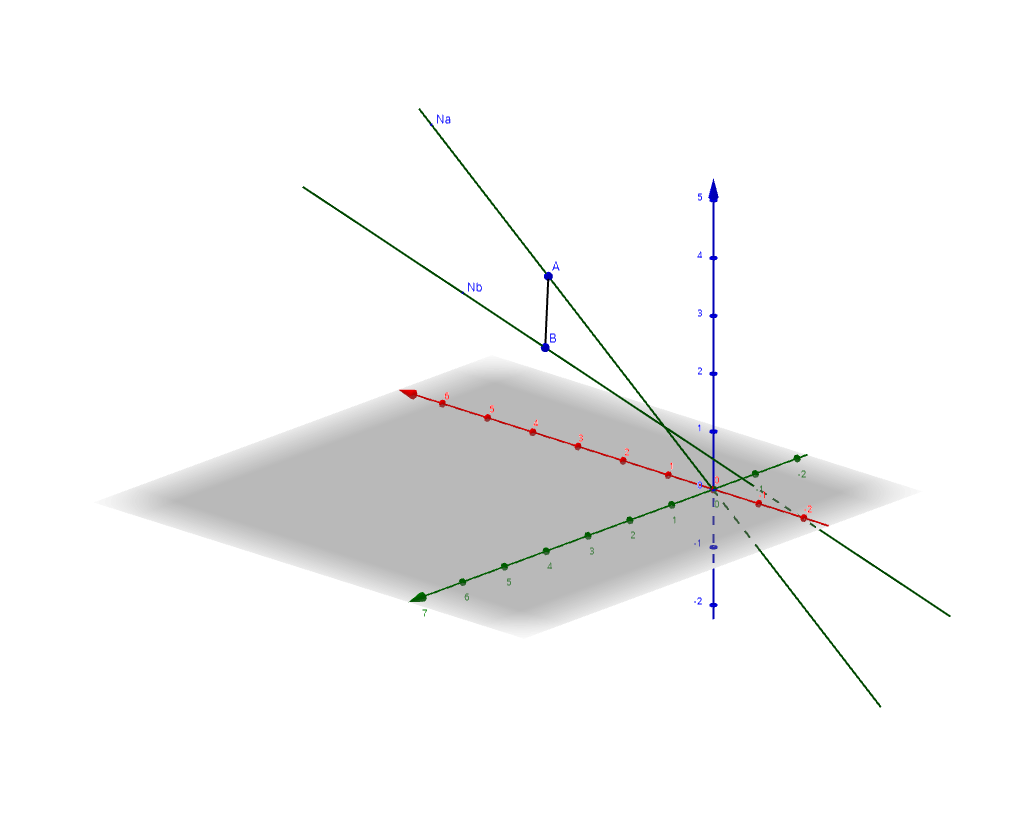


Abbildung 2:Normalvektoren der Vertices (grafische Darstellung)

Jede Ecke enthält nun die Information über die eigene Position und die Normale einer Tangentialebene an diesem Punkt, die an der gemessenen Oberfläche anliegt. Mithilfe dieser Informationen wird nun über jeder Kante eine Funktion gebildet, die der Oberfläche möglichst nahe kommen soll.

* + Die Funktion sollte eine möglichst gleichmäßige Steigung haben um nah an die runde gemessene Oberfläche heranzukommen (was nicht immer perfekt machbar ist).
  + Möglich (und einfach) ist Kubische Funktion, gibt es bessere Möglichkeiten?
  + Schritt für Schritt zur Kubischen Funktion:
* Aus Informationen in den Punkten A und B, die die Kante definieren, die nötigen Informationen für eine Funktion über der Kante bilden:
  1. (Normale in A) x AB für das Lot
  2. (Lot von Na und AB) x Na für Steigungsvektor über der Kante in A
  3. Das gleiche nochmal für die Steigung von B über der Kante
* Parametrisieren der Daten in XY-Ebene um einfaches Verfahren für Kubische Funktion zu erhalten:
  1. Man wählt für die parametrisierte Funktion f(t) folgendes: f(0)=A=0 und f(1)=B=0
  2. Der Vektor Va verläuft zwischen der Kante AB und dem Steigungsvektor in A über der Kante. Er berechnet sich Va = Normalisiert((Lot von Na und AB) x AB)
  3. Für die Parametrisierung der Steigung in A über der Kante werden zwei Geradengleichungen gebildet. G1 = A + r\*(Steigung in A über Kante) und G2 = B + s\*Va. G1 und G2 werden geschnitten. Für den Schnittpunkt ist bekannt, dass er bei t=1 liegt und weil Va normalisiert ist, ist s = f(1). Die parametrisierte Steigung von A über der Kante ist s.
     1. Nach Umstellung der Gleichung für den Schnittpunkt ergibt sich die Formel:

s=(B2\*d1-A2\*d1-B1\*d2+A1\*d2)/(Va1\*d2-Va2\*d1)

d = Steigung in A über der Kante

z.B. d2 = Zweite Komponente des Vektors d

* 1. Das gleiche für die Steigung von B über der Kante, allerdings muss hier –s genommen werden, damit eine negative Steigung entsteht.
  2. Nun sind f‘(0) = g und f‘(1) = h auch bekannt.
  3. Wenn g und h beide 0 sind, wird der Mittelpunkt von AB genommen, statt den Extrempunkt zu berechnen.
* Mit vier Informationen über f(t) lassen sich a,b,c und d in f(t)=at^3+bt^2+ct+d berechnen.
  1. Wir wissen, dass f(0) = 0 und f(1) = 0 und f´(0) = g und f´(1) = h. Daraus folgt für unsere kubische Funktion f(t) dass a = h+g

b = -h-2g

c = g

d = 0

also f(t)= (h+g)t^3+(-h-2g)t^2+gt+0, 0 < t < 1

1. Man nimmt dem höchsten Punkt (der Extremwert der obenstehenden kubischen Funktion) auf der Funktion jeder Kante und bildet aus diesen drei Punkten ein neues Dreieck, das sich nun innerhalb des vorherigen Dreiecks befindet. Da die Punkte auf dem Rand des Dreiecks liegen, wird das Dreieck in vier kleinere Dreiecke geteilt.
   * m = (2(-h-2g) )/(3(h+g))

* Nullstellen: t1/2= (-m/2) +- sqrt((m/2)^2 – g/(3(h+g)))
* Extrempunkt: P(t,f(t))
  + Ein Punkt im Raum(t,y) kann zurückparametrisiert werden in den Raum(x,y,z)
  1. Dafür P(X,Y,Z) = A + t \* (B-A) + y \* Normalisiert(Va\*(1-t)+Vb\*t) rechnen. Va verläuft zwischen der Kante AB und dem Steigungsvektor in A über der Kante. Er berechnet sich Va = Normalisiert((Lot von Na und AB) x AB).
     1. A und B sind die Eckpunkte des Dreiecks im Raum(x,y,z)
* Wenn ein Wendepunkt im Intervall 0 < t < 1 auf f(t) vorliegt, müsste die Kante durch zwei neue Kanten ersetzt werden, die zu diesem Wendepunkt laufen. Die zwei an die Kante anliegenden Dreiecke werden zu vier Dreiecken geteilt. Die anliegenden Normalen müssen neu berechnet werden(Schritt 4 für die vier alten und den neuen Punkt).

1. Man beginnt erneut bei Schritt 3.
   * Die Normalen müssen auch für die Vertices berechnet werden, für die es vorher bereits eine Normale gab, da nun nur noch Teildreiecke der bisherigen anliegenden Dreiecke an den Vertices anliegen.
2. Wiederholen, bis die Teildreiecke so klein sind, dass es keinen signifikanten Unterschied mehr macht, ob man ihre Fläche als Ebene betrachtet.
3. Alle Flächen der winzigen Teildreiecke berechnen und dann aufsummieren.

Das Verfahren löst das Problem, dass es bei einer einfachen Interpolation der Werte gegeben hätte, da ein Punkt in der Mitte des Dreiecks, der einen höheren Abstand zur Dreiecksebene hat, als die Kantenfunktionen nun auch korrekt berechnet werden kann.

# Triangulationsverfeinerung

Beschreibung der Methoden nach Klasse

## TriangulationRefiner

### GetRefinedTriangulation

Erzeugt aus jeder Kante zwischen zwei Punkten eine FunctionEdge, also eine Kante, die anhängig von einer Funktion einen neuen Punkt zur Verfügung stellt. Bildet aus den drei Punkten, die von den FunctionEdge eines Dreiecks und dessen drei Eckpunkten insgesamt vier neue Dreiecke, welche die Verfeinerung des Dreiecks bilden.

## FunctionEdge

Stellt eine Methode zum Abrufen des nächsten Punktes zur Verfügung

## ParameterizedFunctionEdge

Parametrisiert die Positionen und Steigungen in Start- und Endpunkt in den 2D-Raum und übergibt diese an eine abstrakte Methode, um verschiedene parametrisierte Implementierungen zu ermöglichen. Parametrisiert die Ergebniswerte zurück in den 3D-Raum.

## CubicFunctionEdge

Berechnet den nächsten Punkt mithilfe einer Kubischen Funktion.

# Diagramme

# Projektergebnisse

## Programm

# Fazit

Das Projekt Kreuzgewölbe bestimmt die Fläche eines Kreuzgewölbes. Dies kann z.B. bei Malerarbeiten zur Verwendung kommen. Als Eingabe wird eine möglichst große Menge an Punkten des Gewölbes benötigt. Je mehr Punkte es gibt, desto genauer wird das Ergebnis. Diese Punkte werden dann in sinnvolle Dreiecke eingeteilt. Anhand der normalen Vektoren wird wie vorher beschrieben, die wirkliche Form des Kreuzgewölbes, durch weitere Dreiecke, approximiert. Um nun die Fläche des Gewölbes zu bestimmen müssen die Flächeninhalte der Dreiecke errechnet werden. Als Ergebnis wird die Fläche ausgegeben.

Durch diese Vorgehensweise müssen keine Funktionen interpoliert werden. Auch ist sie sehr anschaulich und man kann durch die Oberflächenrekonstruktion eine unsortierte Menge an Punkten in das Programm einwerfen. In der Umsetzung des Programmes erwies sich jedoch gerade diese als eine große Schwierigkeit.

Zudem scheint das Ergebnis von dem tatsächlichen Flächeninhalt abzuweichen. Diese Differenz kann einerseits durch Rundungsfehler, andererseits durch eine unzureichende Genauigkeit beim Erstellen der Dreiecke zustande kommen.

Zur Verbesserung des Programmes muss zuerst die Oberflächenrekonstruktion korrekt implementiert werden. Hier muss die Frage gestellt werden, ob der Ball-Pivot Algorithmus die richtige Wahl ist.

Eine weitere Ergänzung des Programmes, könnte sein den Benutzer die Genauigkeit einstellen zu lassen. Diese Einstellung bestimmt darüber, wie viele „Schichten“ von Dreiecken erstellt werden sollen.

Zudem könnte man die Errechnung der einzelnen Dreiecke in einer Schicht parallelisieren, um auf diese Weise die Laufzeit zu verringern.

# Quellenangaben

<https://de.wikipedia.org/wiki/Delaunay-Triangulierung> (ein paar Algorithmen für 2D)

<http://www.research.ibm.com/vistechnology/pdf/bpa_tvcg.pdf> (Ballpivoting für 3D)

<https://www.kiv.zcu.cz/site/documents/verejne/vyzkum/publikace/technicke-zpravy/2002/tr-2002-02.pdf> (Diskussion versch. Algorithmen für 3D)

# Anhang