Technische Universität München

Ingenieurpraxisbericht

JavaScript STL-Library for Online Microfluidics Programming Platform

.

Betreuer/-in Yushen Zhang
Lehrstuhl für Entwurfsautomatisierung

Eingereicht von Antonius Edwin Adisoemarta
0370815
Guerickestr. 19
80805 München
+49 176 8820 4486

Eingereicht am München, den 14.05.2023

Inhaltsverzeichnis

1. Einleitung ……………………………………………………………………………1

2.1. Motivation ………………………………………………………………..1

2.2. Problemstellung …………………………………………………………..2

2.2. Ziel des Projektes …………………………………………………………3

2. Technischer Hintergrund……………………………………………………………..4

2.1.Microfluidischer Chips……………………………………………………..4

2.2. Mathematischer Hintergrund…………………………………………….6

2.3. JavaScript…………………………………………………………………7

2.4. STL-FIle……………………………………………………………………..8

3. Hauptteil…………………………………………………………………………10

3.1. Konzeption………………………………………………………………10

3.2. Implementierung…………………………………………………………..11

3.3. Bibliothek Verwendung…………………………………………………..16

4. Ergebnis……………………………………………………………………………..23

4.1. Test………………………………………………………………………...23

4.2. Probleme…………………………………………………………………24

4.3. Vergleich zu alten Java Code…………………………………………24

5. Zusammenfassung……………………………………………………………..26

6. Literaturverzeichnis……………………………………………………………….27

1. Einleitung

1.1. Motivation

Die „Microfluidic Large Scale Integration” (MLSI) bezieht sich auf die Entwicklung von Mikrofluidik Chips mit Tausenden von integrierten mikromechanischen Ventilen und Steuerkomponenten. Diese Technologie wird in vielen Bereichen der Biologie und Chemie eingesetzt und ist ein Kandidat für die Ablösung des heutigen herkömmlichen Automatisierungsparadigmas zu ersetzen, das aus folgenden Komponenten besteht Roboter für die Handhabung von Flüssigkeiten.

Um eine Mikrofluidikchip zu herstellen, ist die Kenntnis einer 3D-Computergrafiksoftware unerlässlich. Obwohl nicht jeder Forscher, insbesondere Biologen oder Chemiker, über Programmierkenntnisse verfügt, wollen die Lehrstuhl für Entwurfsautomatisierung eine interaktive, aber intuitive Programmierplattform für Mikrofluidik-Forscher einführen, die keine Vorkenntnisse im Programmieren erfordern, um dieses Problem zu überwinden.

Die grundlegende Motivation, an den insgesamt 9 Wochen Praktikumszeit gearbeitet werden soll, liegt in der Entwicklung JavaScript-Bibliothek zur Herstellung und Kombinierung verschiedener 3D Objekte, die zu Erstellung des 3D Modell von einer microfluidischen Chip dient.

1.2. Problemstellung

Die Lehrstuhl Entwurfsautomatisierung hat bereits erfolgreich eine temporäre Website zur Erstellung von Online-Mikrofluidikchips eingerichtet. Diese Website heißt Flui3d. Das Hauptproblem dieser Website ist jedoch, dass die von der Website eingegebenen Daten zur Erstellung der Mikrofluidikchips zunächst an den Backend-Java-Server gesendet werden müssen, um das 3D-Modell zu erstellen. Dies wiederum macht die Erstellung eines Mikrofluidikchipmodells langsam und es würde manchmal sogar mehr als 5 Minuten dauern, wenn ein sehr komplexer Mikrofluidikchip gerendert wird.

Das derzeitige Java Backend Logik Programm für die Erstellung des 3D Modells hat auch einige andere Probleme, wie zum Beispiel, dass es nicht in der Lage ist, konvexe Polygone zu erstellen.

Die Größe der STL-Datei, die erstellt wird, ist ebenfalls sehr groß und wird nur noch größer, je komplexer der Chip ist, den man erstellen möchte.

1.3. Ziel des Projektes

Ausgehend von der Motivation und den zuvor genannten Problemen ist das Ziel dieses Projekts die Erstellung einer JavaScript-Bibliothek, die 3D-Formen in Form einer STL-File erzeugen kann. Diese Bibliothek würde dann helfen, mikrofluidischer Chips zu erstellen.

Damit das Java-Backend-Skript ersetzt werden kann, muss die neue JavaScript-Bibliothek auch diese Ziele erreichen:

- Reduzierung der Renderzeit in Vergleich zu dem alten Java-Backend Codes des 3D-Modells zur Erstellung von Mikrofluidikchips.

- Reduzierung der Größe der STL-Datei in Vergleich zu dem alten Java-Backend Codes, die erstellt wird.

- Erfolgreiche Herstellung des konvexes Polygon

2. Technischer Hintergrund

Dieses Projekt besteht aus 3 technischen Hintergründen, die man zuerst verstehen muss, um das gesamte Projekt zu verstehen:

a. Microfluidischer-Chips

b. Notwendige Mathematische Grundlagen

c. JavaScript Grundlagen

d. STL-File

2.1. Microfluidischer-Chips

2.1.1. Definition

Ein Microfluidischer-Chip ist ein kleines Gerät, das zur Steuerung und Analyse von Flüssigkeiten auf der Mikroskala verwendet wird. Er besteht aus einem winzigen Kanalnetzwerk und winzigen Kammerstrukturen, die in ein kleines Stück Silizium oder Glas eingraviert sind.

Der Chip kann dazu verwendet werden, winzige Mengen von Flüssigkeiten zu manipulieren, zu trennen und zu analysieren, indem sie durch winzige Kanäle geleitet werden. Da die Kanäle und Kammerstrukturen so klein sind, können sie Flüssigkeiten auf einer Skala von nur wenigen Mikrometern oder Nanometern genau steuern.

Microfluidische Chips finden Anwendung in der Biotechnologie, Medizin, Chemie und anderen Bereichen, in denen die Kontrolle und Analyse von winzigen Flüssigkeitsmengen erforderlich ist. Zum Beispiel können sie zur Herstellung von künstlichem Gewebe, zur schnellen Diagnose von Krankheiten oder zur Analyse von Proteinproben verwendet werden. [1]

2.1.2. Struktur einer Microfluidischer-Chip

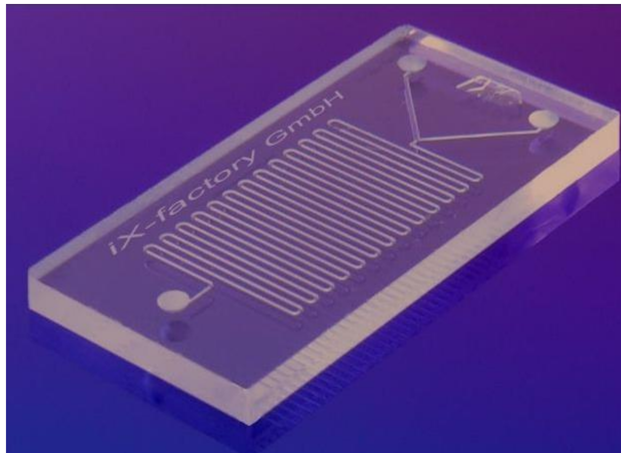


Abbildung 1

Die typische Form oder Struktur eines microfluidischen Chips kann je nach Anwendung variieren, aber es gibt einige gemeinsame Merkmale. In der Regel besteht ein microfluidischer Chip aus einem flachen Stück Silizium oder Glas mit winzigen Kanälen und Kammerstrukturen, die mithilfe von lithographischen Verfahren hergestellt werden. Die Kanäle können in verschiedenen Formen und Größen hergestellt werden und können gerade, gewunden oder verzweigt sein, je nachdem, welche Funktion der Chip erfüllen soll.

Die Kanäle und Kammerstrukturen sind oft mit einer Deckplatte aus Glas oder Polymer bedeckt, um sie vor Verunreinigungen und Verschmutzungen zu schützen. Die Strukturen auf dem Chip können mithilfe von Pumpen, Ventilen oder elektrischen Feldern gesteuert werden, um die Flüssigkeit in bestimmte Bereiche des Chips zu bewegen oder zu stoppen.

Kurz gesagt, die mikrofluidischen Chips können in viele andere kleine Teile wie den Kanal und die Filter zerlegt werden. Diese Teile sind oft einfache primitive Formen wie Zylinder, Quader oder Ringe. Eine Kombination der richtigen primitiven Formen würde ein 3D-Modell eines mikrofluidischen Chips ergeben.

2.2. Mathematischer Hintergrund

2.2.1. Linear Algebra

Linear Algebra ist ein Zweig der Mathematik, der sich mit der Studie von Vektoren, Matrizen und linearen Gleichungssystemen befasst. Dieses Konzept ist der Schlüssel zum Rendern von 3D-Objekten und auch in diesem Projekt.[2]

Lineare Algebra ist für fast alle Bereiche der Mathematik von zentraler Bedeutung. So ist die lineare Algebra in modernen Darstellungen der Geometrie von grundlegender Bedeutung, auch für die Definition grundlegender Objekte wie Linien, Ebenen und Drehungen. Auch die Funktionalanalysis, ein Teilgebiet der mathematischen Analyse, kann als Anwendung der linearen Algebra auf Funktionsräume betrachtet werden. Dieses Konzept ist für dieses Projekt sehr wichtig

eine geometrische Form kann mit Hilfe der linearen Algebra durch eine Matrix dargestellt werden.

ein Punkt im dreidimensionalen System kann durch einen Satz von 3 Punkten beschrieben werden, die jeweils die x-, y- und z-Achse darstellen

Beispiel: Ein Punkt mit x=0,y=1,z=0 wäre (0,1,0)

2.3. JavaScript

2.3.1. Definition

Javascript ist eine Skriptsprache, die hauptsächlich in Webanwendungen eingesetzt wird, um Interaktivität auf Webseiten zu ermöglichen. Das bedeutet, dass mit Javascript dynamische Änderungen an einer Webseite vorgenommen werden können, ohne dass die Seite neu geladen werden muss. Das wird durch den Einsatz von sogenannten "Event-Handlern" erreicht, die auf Benutzeraktionen wie Mausklicks oder Tastaturanschläge reagieren und dann bestimmte Aktionen ausführen, wie zum Beispiel das Ändern von Text oder das Anzeigen von Bildern.[3]

Wir werden in diesem in diesem Projekt mit JavaScript arbeiten.

2.3.2. JavaScript Bibliotehken

Es würde für diese Projekt folgende Bibliotheken benutzt:

2.3.2.1. Earcut.js

earcut.js ist eine JavaScript-Bibliothek, die für das sogenannte "Ear Clipping Triangulation" verwendet wird. Diese Technik wird in der Computergrafik eingesetzt, um eine konvexe Polylinie (eine Figur, die aus mehreren geraden Segmenten besteht und nicht gekrümmt ist) in ein Dreiecksnetz zu zerlegen.

Die earcut.js-Bibliothek bietet eine effiziente Möglichkeit, dieses Verfahren auf einer Vielzahl von Plattformen auszuführen, einschließlich im Webbrowser. Es ist in der Lage, komplexe Polylinien mit Tausenden von Punkten in Bruchteilen einer Sekunde zu triangulieren, was es zu einer wertvollen Ressource für Entwickler macht, die mit 2D-Grafiken arbeiten.

Die Bibliothek ist in der Lage, die polygoneffizient in Dreiecke zu unterteilen, indem sie die "Ohren" des Polygons identifiziert und entfernt, wobei jedes "Ohr" ein konkaves Dreieck aufweist, das sich nicht innerhalb des Polygons befindet. Der Algorithmus wählt das nächstliegende Ohr aus, entfernt es und fügt die resultierenden Dreiecke in das endgültige Dreiecksnetz ein. Dies wird wiederholt, bis alle Ohren entfernt wurden und das gesamte Polygon in Dreiecke unterteilt wurde. [4]

2.4. STL-File

2.4.1. Definition

Ein STL-Dateiformat ist ein 3D-Modellformat, das verwendet wird, um die Geometrie von dreidimensionalen Objekten darzustellen. Die Abkürzung STL steht für "STereoLithography" und wurde von der Firma 3D Systems entwickelt. [5]

Ein STL-File besteht aus einer Sammlung von dreieckigen Flächen, die zusammen eine 3D-Geometrie ergeben. Diese Flächen werden auch als Facetten bezeichnet und werden durch eine Gruppe von drei Punkten definiert, die die Eckpunkte jedes Dreiecks darstellen. Das Format enthält normalerweise keine Farbinformationen oder Texturkoordinaten und ist daher in erster Linie auf die Darstellung der Geometrie beschränkt.

STL-Dateien werden häufig in der Computer Aided Design (CAD)-Software verwendet und sind ein Standardformat für den 3D-Druck. Sie können jedoch auch in anderen Anwendungen wie der Simulation von physikalischen Phänomenen, der virtuellen Realität oder der Computerspieleentwicklung eingesetzt werden.

Es gibt zwei Arten von STL-Dateiformaten: binäre (binary) und ASCII-Dateien. Die beiden Formate unterscheiden sich in der Art und Weise, wie sie die Daten darstellen. Für unsere Library wird es so gemacht dass es sowohl binäre als auch ASCII STL files generieren kann.

2.4.2. ASCII-STL

Eine ASCII STL-Datei ist eine Textdatei, die die Geometrie-Informationen als lesbaren Text enthält. ASCII-Dateien sind größer als binäre Dateien, da sie mehr Speicherplatz benötigen, um die gleichen Informationen darzustellen. Jedes Dreieck wird in ASCII-Format durch eine Beschreibung mit Koordinaten und Normalen dargestellt. Ein ASCII STL-File enthält außerdem Header-Informationen am Anfang und Ende der Datei.

Die ASCII-STL-Datei beginnt mit der obligatorischen Zeile:

```
solid <name>
```

wobei <name> der Name des 3D-Modells ist. Dieses Feld kann leer gelassen werden, aber in diesem Fall muss nach dem Wort „fest“ ein Leerzeichen stehen.

Die Datei wird mit Informationen zu den Abdeckdreiecken fortgesetzt. Informationen über die Scheitelpunkte und den Normalenvektor werden wie folgt dargestellt:

```
facet normal nₓ nᵧ n_z
    outer loop
        vertex v1ₓ v1ᵧ v1_z
        vertex v2ₓ v2ᵧ v2_z
        vertex v3ₓ v3ᵧ v3_z
    endloop
endfacet
```

Hier ist n die Normale des Dreiecks und v1, v2 und v3 sind die Eckpunkte des Dreiecks. Koordinatenwerte werden als Fließkommazahl im Format sign-mantisse-esign-exponent dargestellt – zum Beispiel „3.245000e-002“.

Die Datei endet mit der obligatorischen Zeile:

```
endsolid <name>
```

2.4.3. Binary STL

Da ASCII-STL-Dateien sehr groß sein können, existiert eine binäre Version von STL. Eine binäre STL-Datei hat einen 80-Zeichen-Header, der im Allgemeinen ignoriert wird, aber niemals mit der ASCII-Darstellung der Zeichenfolge solid beginnen sollte, da dies dazu führen kann, dass manche Software sie mit einer ASCII-STL-Datei verwechselt. Dem Header folgt eine 4-Byte-Little-Endian-Ganzzahl ohne Vorzeichen, die die Anzahl der dreieckigen Facetten in der Datei angibt. Darauf folgen Daten, die der Reihe nach jedes Dreieck beschreiben. Die Datei endet einfach nach dem letzten Dreieck.

Jedes Dreieck wird durch zwölf 32-Bit-Gleitkommazahlen beschrieben: drei für die normale und dann drei für die X/Y/Z-Koordinate jedes Scheitelpunkts – genau wie bei der ASCII-Version von STL. Danach folgt eine 2-Byte („short“) unsigned Integer, das ist der „Attribute Byte Count“ – im Standardformat sollte dieser null sein, da die meisten Programme nichts anderes verstehen.[8]

Gleitkommazahlen werden als IEEE-Gleitkommazahlen dargestellt und als Little-Endian angenommen, obwohl dies in der Dokumentation nicht angegeben ist.

```
UINT8[80]    - Header                    -     80 bytes
UINT32       - Number of triangles       -      4 bytes
foreach triangle                         - 50 bytes:
    REAL32[3] - Normal vector                - 12 bytes
    REAL32[3] - Vertex 1                     - 12 bytes
    REAL32[3] - Vertex 2                     - 12 bytes
    REAL32[3] - Vertex 3                     - 12 bytes
    UINT16    - Attribute byte count         -  2 bytes
end
```

3. Hauptteil

3.1. Konzeption

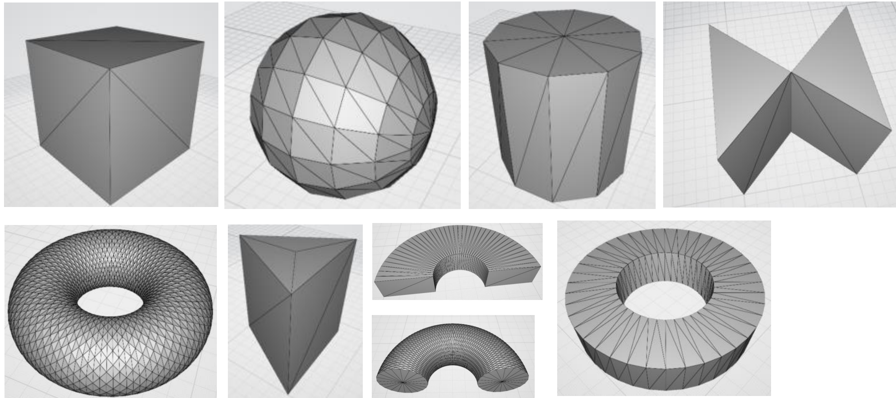
Das Konzept der Bibliothek besteht darin, zunächst einfache 3D-Primitivformen wie Würfel, Quader, Zylinder, Pyramide, Ring, Torus, Polyeder, Kugel und andere Primitivformen zu erstellen. Die erstellten primitiven Formen werden als ein Array von Punkten erstellt, die die Lage ihrer Punkte definieren. Der zweite Schritt (falls erforderlich) besteht dann darin, die erstellten primitiven Formen mit Hilfe boolescher Operationen (Vereinigung, Schnittpunkt und Differenz) zu kombinieren. Mit den booleschen Operationen können wir komplexere Formen erstellen und auch Formen kombinieren, so dass wir am Ende ein 3D-Modell eines Mikrofluidik-Chips erstellen können. Wenn der Benutzer mit dem Endergebnis zufrieden ist, verfügt die Bibliothek über eine Funktion zur Erstellung der STL-Datei (binär oder ASCII).

Zusammengefasst hat die Library 3 Hauptfitur

1. Herstellung der primitiven Formen (Quader, Würfel, Zylinder,usw)
2. Kombinierung der primitiven Formen durch Boolean Operationen (Union, Intersection oder Difference)
3. Herstellung der STL-File

3.2. Implementierung

3.2.1. Primitive Shapes



Das Bild oben zeigt alle primitiven Formen, die mit dieser Bibliothek erstellt werden können.

```
STL.Cuboid()
STL.Sphere()
STL.Prism()
STL.Cone()
STL.Polyhedron()
STL.Torus()
STL.partTorus()
STL.Ring()
STL.partRing()
```

3.2.1.1. Struktur des Codes

Die gesamte Erzeugung von primitiven Formen in dieser Bibliothek folgt demselben Code Struktur nämlich:

- Erzeugung von Flächen/Punkten
- Rotation von Flächen
- Verschiebung

Beispiel:

```
STL.cuboid = function (startPoint = [0, 0, 0], length = 1, width = 1, height = 1, angleX = 0, angleY = 0, angleZ = 0) {

  var facets = [
    // Bottom facing facets
    [[length, 0, 0, 1], [0, 0, 0, 1], [0, width, 0, 1], [length, width, 0, 1]],
    // Upper facing facets
    [[0, width, height, 1], [0, 0, height, 1], [length, 0, height, 1], [length, width, height, 1],],
    // Back facing facets
    [[length, width, 0, 1], [0, width, 0, 1], [0, width, height, 1], [length, width, height, 1]],
    // Front facing facets
    [[length, 0, 0, 1], [length, 0, height, 1], [0, 0, height, 1], [0, 0, 0, 1]],
    // Side facing facets 1
    [[length, 0, height, 1], [length, 0, 0, 1], [length, width, 0, 1], [length, width, height, 1]],
    // Side facing facets 2
    [[0, width, 0, 1], [0, 0, 0, 1], [0, 0, height, 1], [0, width, height, 1]],
  ]
  if (angleX != 0) {
    facets = rotateX(facets, angleX)
  }
  if (angleY != 0) {
    facets = rotateY(facets, angleY)
  }
  if (angleZ != 0) {
    facets = rotateZ(facets, angleZ)
  }
  return solidBuilder(translate(facets, startPoint[0], startPoint[1], startPoint[2]))
}
```

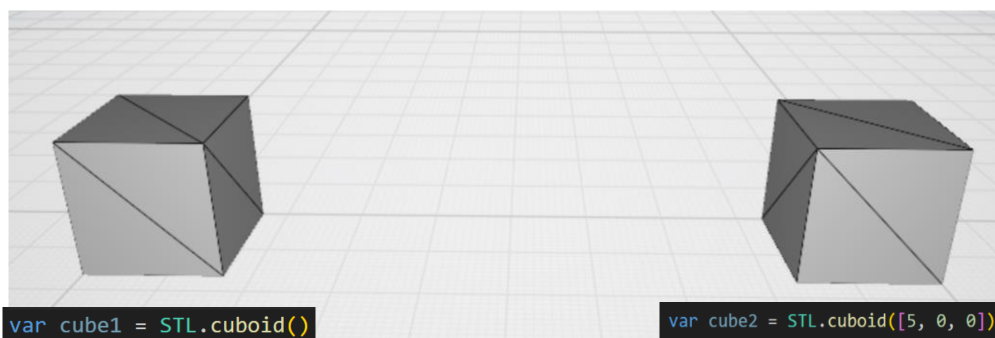
1. Faces Generation

2. Rotation

3. Translation

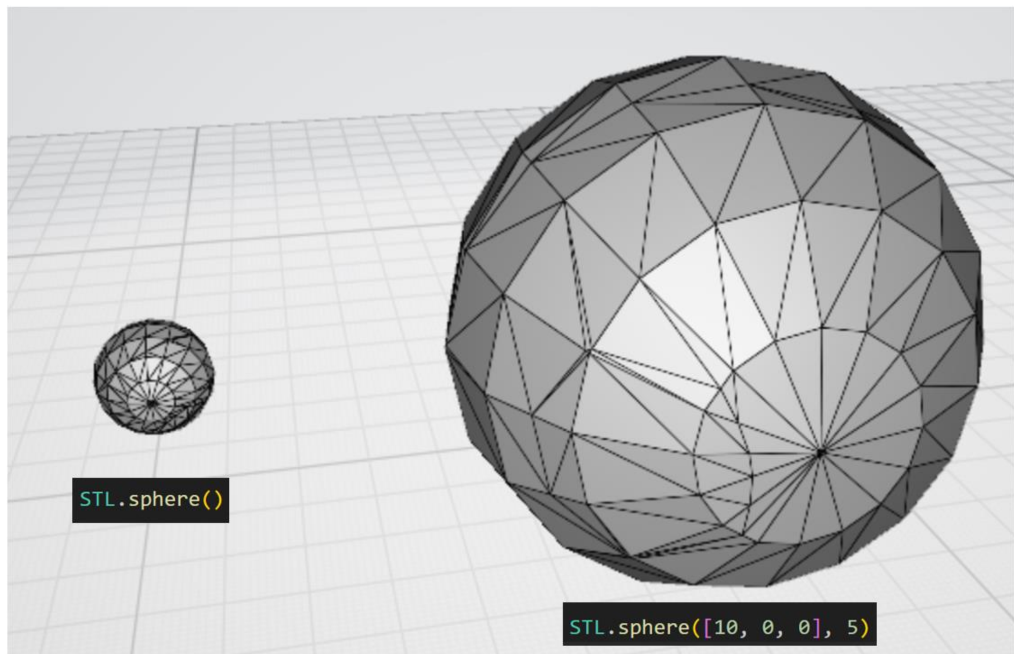
3.2.1.2. Standardeingabeparameter

- Startposition (Beispiel [5,0,0])



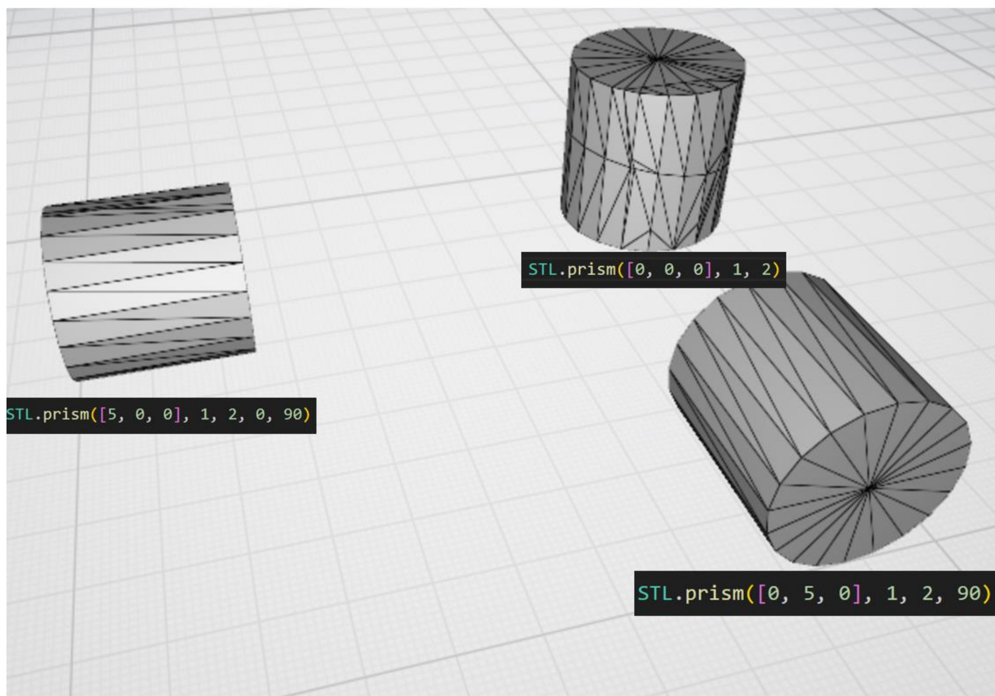
Beispiel einer Verschiebung der Startposition

- Größe (Radius/Länge/Höhe/ usw.)



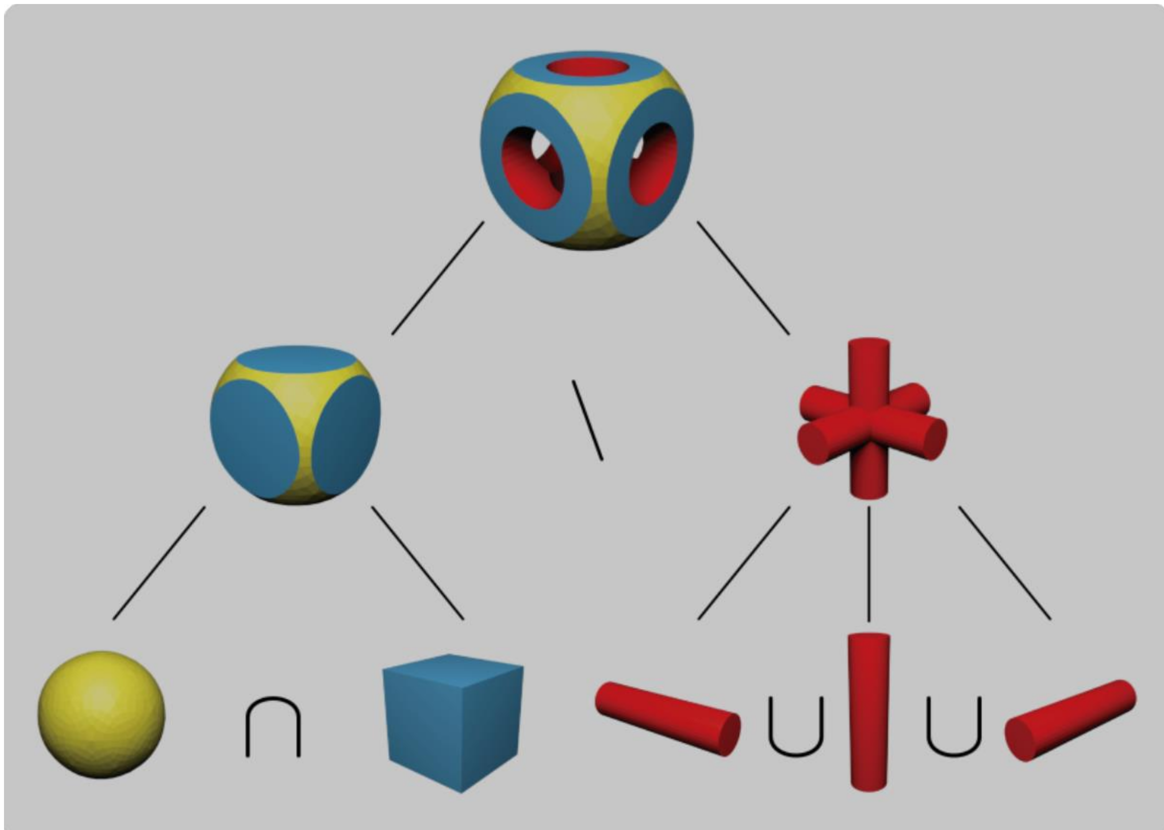
Beispiel einer Größeänderung

- Drehung (in Bezug auf die x,y,z-Achse)



Beispiel einer Drehung

3.2.2. Kombinierung der primitiven Formen



Um primitiven Formen zu Kombinieren benutzt dieses JavaScript Bibliothek die Konzept der „Constructive Solid Geometry“ (CSG).

CSG steht für Constructive Solid Geometry, was auf Deutsch konstruktive Festkörpergeometrie bedeutet. Es handelt sich um eine Technik zur Erstellung von 3D-Modellen, bei der solide Objekte durch die Kombination von einfachen geometrischen Formen erstellt werden.

CSG bietet eine Vielzahl von Funktionen, mit denen Sie 3D-Modelle erstellen können, einschließlich boolescher Operationen wie Vereinigung, Schnitt und Differenz.

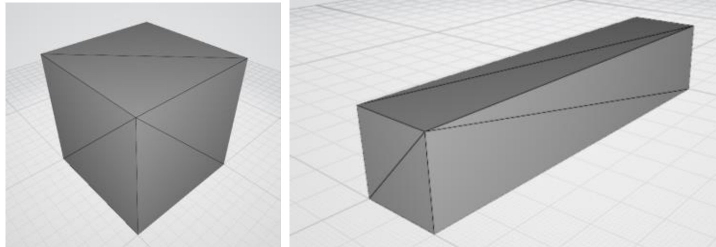
Zur Durchführung dieser Methoden werden folgende Schritte unternommen:

1. Iterieren Sie durch alle Polygone in beiden Formen.
2. Prüfen Sie für jedes Polygon, ob es sich mit einem der Polygone in der anderen Form schneidet.
3. Wenn es einen Schnittpunkt gibt, teilen Sie die Polygone am Schnittpunkt in kleinere auf.
4. Je nach Operation Innen- oder Außenpolygone zum endgültigen Polygon entfernen oder hinzufügen
5. Verschmelzen Sie die verbleibenden Polygone aus beiden Formen zu einer einzigen Form.

3.3. Bibilothek Verwendung

3.3.1. Erzeugung der primitiven Formen

3.3.1.1. Cuboid



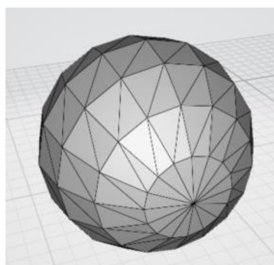
Eingabeparametern:

- Startpunkt
- Länge
- Breite
- Höhe
- Drehung (x,y,z)

Beispiel Verwendung:

```
STL.Cuboid()
```

3.3.1.2. Sphere



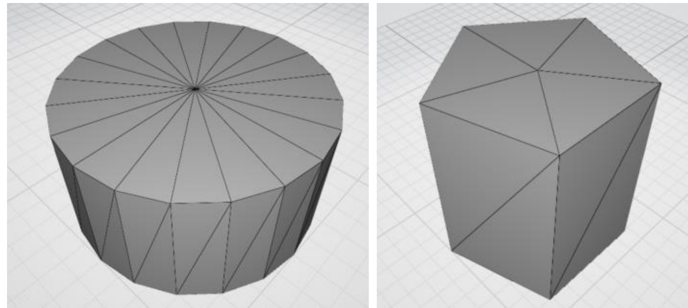
Eingabeparametern:

- Startpunkt
- Radius

Beispiel Verwendung:

```
STL.Sphere()
```

3.3.1.3. Prism



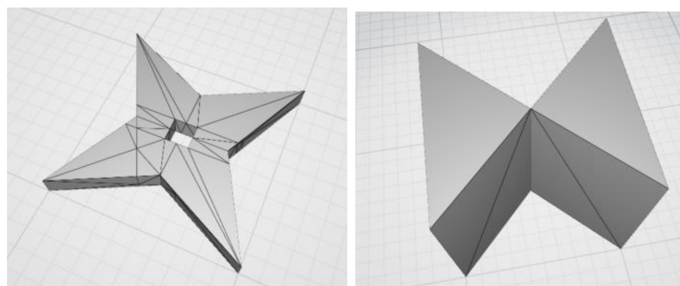
Eingabeparametern:

- Startpunkt
- Radius
- Höhe
- Drehung (x,y,z)
- Resolutions
  - Wird Benutzt um einzugeben wie viel Punkte der ober oder untere Polygon generiert wird. Zum Besipiel bei 100 wird es ein Cylinder generiert

Beispiel Verwendung:

```
STL.Prism()
```

3.3.1.4. Polyhedron



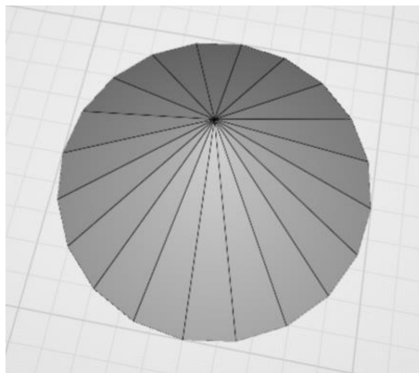
Eingabeparametern:

- Startpunkt
- **Points (verplichtend)**
- Höhe
- Drehung (x,y,z)

Beispiel Verwendung:

```
STL.Polyhedron(points)
```

3.3.1.5. Cone



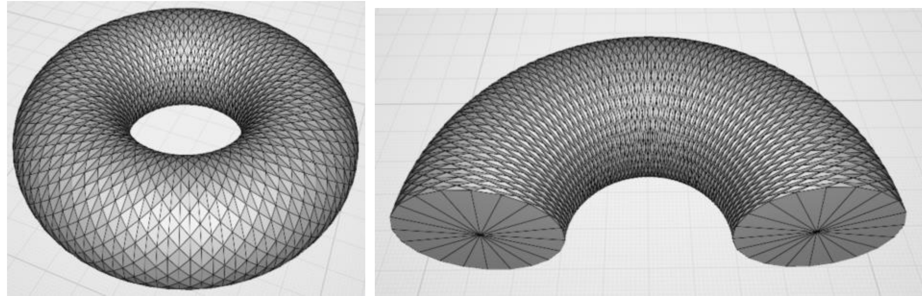
Eingabeparametern:

- Startpunkt
- Radius
- Höhe
- Drehung (x,y,z)

Beispiel Verwendung:

```
STL.Cone()
```

3.3.1.6. Torus und partTorus



Eingabeparametern:

- **Starting Angle (verplichtend)**
- **End Angle (verplichtend)**
- Starting Point
- Inner Radius
- Thickness Radius
- Height
- Rotation (x,y,z)
- Resolution

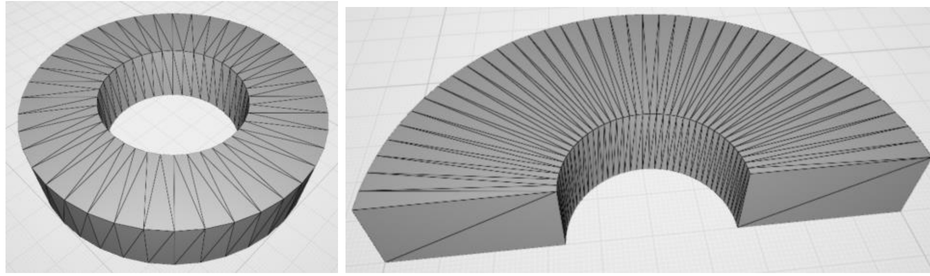
Beispiel Verwendung:

```
Torus
STL.Torus()

partTorus

STL.partTorus(0,Math.PI)
```

3.3.1.7. Ring und partRing



Eingabeparametern:

- **Starting Angle (verplichtend)**
- **End Angle (verplichtend)**
- Starting Point
- Inner Radius
- Thickness Radius
- Height
- Rotation (x,y,z)
- Resolution

Beispiel Verwendung:

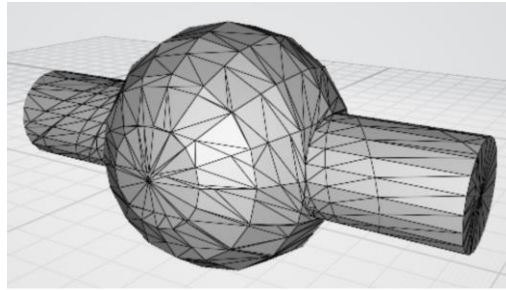
```
Ring
STL.Ring()

partRing

STL.partRing(0,Math.PI)
```

3.3.2. Kombinierung der Primitiven Formen

3.3.2.1. Union

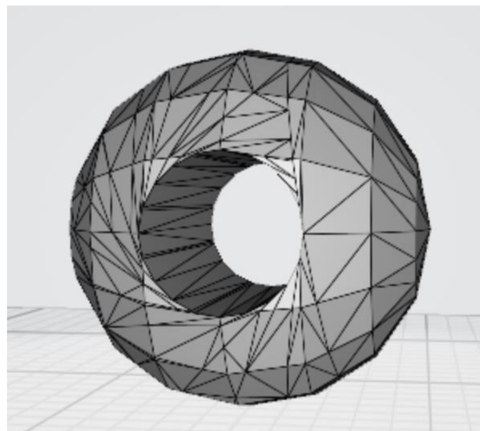


```
var c = a.union(b)
```

Beispiel Verwendung:

```
var cylinder = STL.prism([-5, 0, 5], 1, 10, 0, 90)
var sphere = STL.sphere([0, 0, 5], 2.5)
var result = cylinder.union(sphere)
```

3.3.2.2. Subtraction

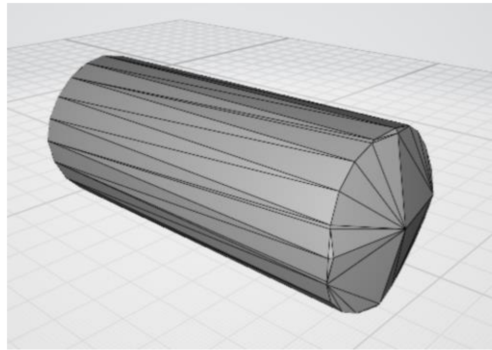


```
var c = a.union(b)
```

Beispiel Verwendung:

```
var cylinder = STL.prism([-5, 0, 5], 1, 10, 0, 90)
var sphere = STL.sphere([0, 0, 5], 2.5)
var result = sphere.subtract(cylinder)
```

3.3.2.3. Intersection



```
var c = a.union(b)
```

Beispiel Verwendung:

```
var cylinder = STL.prism([-5, 0, 5], 1, 10, 0, 90)
var sphere = STL.sphere([0, 0, 5], 2.5)
var result = sphere.intersect(cylinder)
```

3.3.3. Generierung der STL-Files

Befehl, um STL-Files zu erstellen:

```
STL.generateASCIISTL()
STL.generateBinarySTL()
```

Eingabeparametern:

- STL Object
- Namen des Files

Beispiel Verwendung:

```
var cylinder = STL.prism([-5, 0, 5], 1, 10, 0, 90)
var sphere = STL.sphere([0, 0, 5], 2.5)
var result = sphere.intersect(cylinder)

STL.genereteASCIISTL(result, "Object")
```

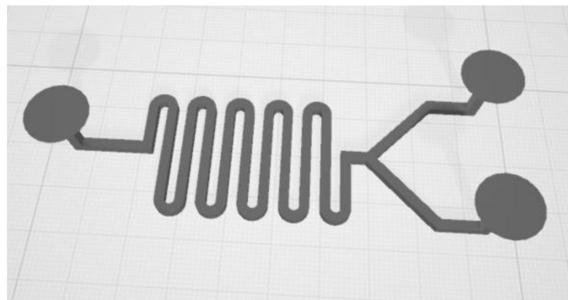
4. Ergebnis

4.1. Test

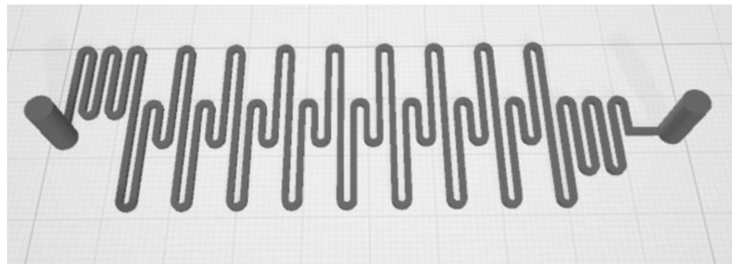
Diese Bibliothek wurde entwickelt, um 3D-Modelle eines mikrofluidischen Chips zu erstellen. Aus diesem Grund würden wir zum Testen der Leistung versuchen, mithilfe dieser Bibliothek Objekte nachzubilden, die Mikrofluidik-Chips darstellen.

Ein typischer mikrofluidischer Chip besteht aus vielen Ventilen und Filtern, daher möchten wir sowas ähnliches:

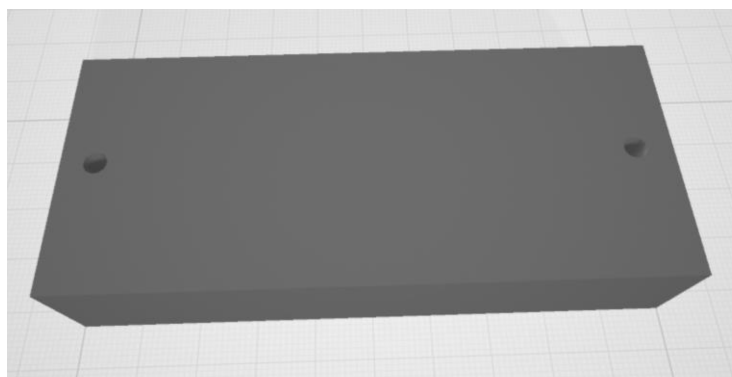
Zu erzeugende Beispiel Ventilen:



Form 1



Form 2



Form 3

Hinweis: Form 3 ist die endgültige Form einer microfluidischen Chip. Es kann durch Subtraction einer Cuboid (Quader) mit entsprechenden Maßen mit den Ventilen erzeugt werden.

4.1.1. Leistungstest

Nach mehreren Durchführungen benötigt die JavaScript Bibliothek etwa +- 3 Sekunde um die erste Form zu erzeugen. Die Zweite Form ist eine deutlich komplizierter Form und dafür braucht die Bibliothek etwa +- 5.5 Sekunde. Zuletzt wenn wir dazu bei Form 1 und Form2 einen Cuboid hinzufügt, braucht die Bibliothek eine extra Sekunde , um die Form zu gestalten.

4.1.2. Dateigröße

Die Dateigröße für das erste Formular beträgt etwa 2,5 MB und für das zweite Formular 5 MB. Mit der Box oder der 3. Form ändert sich die Dateigröße nicht so sehr, was bedeutet, dass die erstellten Dateidaten recht klein sind

4.2. Probleme

Dies sind einige Probleme, die in dieser Javascript-Bibliothek immer noch auftreten und möglicherweise in einer zukünftigen Forschung behoben werden:

- Viele Fehler bei Gleitkommazahlen
- Es entstehen immer noch viele unnötige Dreiecke.

4.3. Vergleich mit altem Java-Code

Um zu beweisen, dass dieser Code besser ist, vergleiche ich den vorherigen Test mit dem alten Java-Backend-Code, um zu sehen, wie er tatsächlich abschneidet

Ergebnis:

- Erzeugungsgeschwindigkeit : Leichte Verringerung (-+ 1 Sekunde Unterschied)
- Dateigröße: Verringerung der Dateigröße, insbesondere bei komplexen Designs
- Fähigkeit, konvexe Polygone und Polygone mit sich überschneidenden Kanten zu erzeugen

5. Zusammenfassung

Alles in allem lässt sich aus diesem Bericht schließen, dass dieses Projekt ein Erfolg ist, da es gelungen ist, eine JavaScript-Bibliothek zu erstellen, die in der Lage ist, 3D-Objekte zu erstellen, die beim Rendern von Mikrofluidik-Chips helfen würden. Die Bibliothek ist auch besser als der alte Java-Code, nämlich weil sie eine bessere Leistung, eine kleinere Dateigröße und die Möglichkeit hat, konvexe Polygone zu erstellen. Mir persönlich hat die Erstellung dieser Bibliothek viel Spaß gemacht und ich habe das Gefühl, dass ich bei diesem Projekt so viel gelernt habe

6. Literaturverzeichnis

[1] Microfluidischer Chips

https://www.elveflow.com/microfluidic-reviews/general-microfluidics/a-general-overview-of-microfluidics/

[2] Linear Algebra

https://en.wikipedia.org/wiki/Linear_algebra

[3] Javascript

https://en.wikipedia.org/wiki/JavaScript

[4] Earcut.js

https://github.com/mapbox/earcut

[5] STL-File

https://en.wikipedia.org/wiki/STL_(file_format)