Georg-August-Universität Göttingen

Implementierung von Hüllkörpern in der Plattform GroIMP für Pflanzenmodellierung

---

Praktikant
Jan Tristan Koch
Matrikelnummer: 21565495

Unter Anleitung von
Prof. Dr. Winfried Kurth

Modul
B.Inf.1808: Anwendungsorientierte Systementwicklung im forschungsbezogenen Praktikum

Studiengang
Angewandte Informatik

24. September 2021

Inhaltsverzeichnis

1 Einleitung

Hüllkörper sind einfache dreidimensionale Körper, die ein oder mehrere komplexe, dreidimensionale Körper umschließen. Ein Anwendungsgebiet ist die Kollisionsabfrage mittels Hierarchien von Hüllkörpern[6][1][4], mit dem Hintergrund, dass Kollisionen von einfachen Hüllkörpern sehr viel einfacher und schneller zu berechnen sind als Kollisionen von komplexen, dreidimensionalen Objekten. Aus dem gleichen Grund werden Hüllkörper beispielsweise auch zur Steigerung der Effizienz beim Ray-Tracing verwendet[2][8]. Durch Verbünde von Objekten, die von einzelnen Hüllkörpern umfasst werden, kann vor der ressourcenaufwändigen Kollisionsberechnung von komplexen, dreidimensionalen Objekten eine Vorauswahl von potentiellen kollidierenden Objekten mithilfe der Hüllkörper getroffen werden.

Im Laufe dieses Praktikums sollen mehrere Arten von Hüllkörpern für die Software GroIMP[5] implementiert werden. Diese bietet eine Plattform für die funktionelle, strukturelle Modellierung von Pflanzenwachstum. Die Hüllkörper könnten dort in Zukunft für Erkenntnisgewinne durch eine Gliederung und eine gröbere Darstellung von Verbünden von Objekten sorgen. Weiterhin könnten sie eine Basis zur effizienten Implementierung von Kollisionsermittlungen und Ray-Tracing bieten.

Es folgt eine kurze Erläuterung der genutzten Methoden, der Implementierung, eine Darstellung der Ergebnisse anhand einiger Beispiele und zuletzt ein Ausblick in die mögliche Zukunft und Weiterverwendung der implementierten Hüllkörper.

2 Auswahl der Arten von Hüllkörpern und deren Berechnungsmethoden

Bevor die Implementierung der Hüllkörper beginnen kann, muss eine Auswahl an Arten von Körpern getroffen werden.

Achsenparallele Quader (kurz, engl.: AABB) sind rechteckige Hüllkörper, die nach den globalen Koordinatenachsen einer Szene ausgerichtet sind und die schnell berechnet werden können, indem globale Extrempunkte auf den Rändern einzelner oder mehrerer Objekte gesucht werden. Sie bieten eine Möglichkeit zur sehr schnellen Kollisionsberechnung, zeigen jedoch große Schwächen in der Genauigkeit der Überlappung, falls das umhüllte Objekt eine längliche Form hat und nicht nach den globalen Koordinatenachsen ausgerichtet ist.

Orientierte Quader (kurz, engl.: OBB) sind, genauso wie die AABB, rechteckige Hüllkörper, die allerdings nicht auf eine Ausrichtung nach den globalen Koordinatenachsen beschränkt sind. Stattdessen können sie eine beliebige Orientierung in der Szene einnehmen. Dies führt zu einer besseren Überlappung mit dem umhüllten Objekt, falls dieses für einen achsenparallelen Quader ungünstig im Raum liegt. Die Berechnung ist für einfache Objekte wie Zylinder oder Kegel sehr einfach, da sie durch wenige Attribute, wie Radius und Länge, beschrieben werden können. Für Mengen von Objekten oder komplexe Objekte, wie NURBS-

Flächen, die in GroIMP[5] zur Darstellung von komplexen Körpern genutzt werden, wird jedoch eine kompliziertere Methode zur Berechnung des Hüllquaders benötigt, da zuerst eine gute Orientierung des Quaders gefunden werden muss, mit der eine enge Umschließung des Objekts möglich ist. Hierzu werden die Objekte, oder das einzelne Objekt, in eine Menge von Punkten übersetzt und auf dieser Menge eine Hauptkomponentenanalyse[7] (kurz, engl.: PCA) durchgeführt. Der orientierte Hüllquader wird nach den berechneten Hauptachsen ausgerichtet. Die Überlappung mit annähernd kugelförmigen Objekten, wie sie in der Natur recht häufig vorkommen, stellt gleichermaßen eine Schwäche von AABBs und OBBs dar.

Hüllellipsoide mit minimalem Volumen (kurz, engl.: MVEE) sollen diese Lücke schließen. Für einzelne Objekttypen, wie Zylinder und Parallelogramme, ist die Berechnung eines Hüllellipsoids recht einfach und schnell möglich. Für alle anderen Objekte wird es jedoch schnell kompliziert ein optimales MVEE zu finden, wodurch nur eine Annäherung an das Hüllellipsoid mit minimalem Volumen mithilfe des Ellipsoid-Algorithmus von Khachiyan[3] berechnet wird. Dieser Algorithmus berechnet eine Folge von Ellipsoiden, bis eines einen vorher festgelegten Bereich der Fehlertoleranz zu einer zuvor berechneten Kugelwolke trifft. Hüllellipsoide stellen daher für größere Punktwolken einen deutlich erhöhten Rechenaufwand im Vergleich zu den beiden anderen Hüllobjekten dar.

Weitere mögliche Arten von Hüllobjekten wären beispielsweise Hüllkugeln, die sehr speicher- und recheneffizient sind, jedoch bei vielen Objekten eine unzureichende Überlappung bieten oder Hüllpolyeder, die Schwächen der Hüllquader bei annähernd kugelförmigen Objekten ausgleichen können, jedoch bei erhöhter Anzahl von Kanten auch einen erhöhten Rechenaufwand bedeuten.

3 Implementierung

3.1 Erstellung von Punktwolken auf den Rändern verschiedener Objekttypen

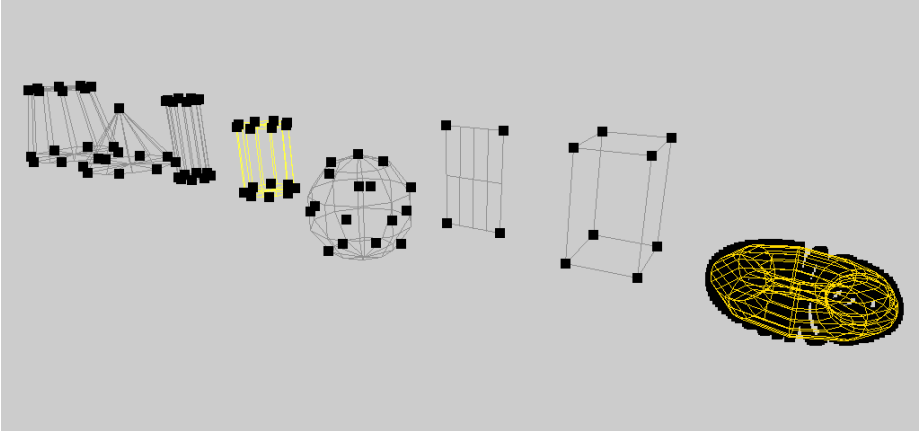


Abbildung 1: Punktwolken auf Objekten von links nach rechts vom Typ *Frustum*, *Cone*, *Cylinder*, *F*, *Sphere*, *Parallelogram*, *Box* und $NURBSSurface$ in Form eines Torus. Parameter $divisions = 4$ und $flatness = 10$.

Einige der implementierten Methoden zur Berechnung von Hüllobjekten verwenden eine Menge von Punkten, die auf den Rändern der zu umhüllenden Objekte liegen. Diese Mengen werden Punktwolken genannt. Für die Objekttypen *Box* und *Parallelogram* enthält die Punktwolke alle Eckpunkte des Objekts. Für die Berechnung der Punktwolken von Objekten mit Kreisflächen, wie *Frustum*, *Cone*, *Cylinder* oder *F* (ein weiterer zylindrischer Objekttyp), wird der ganzzahlige Parameter *divisions* übergeben. Auf der Kante jeder Kreisfläche werden dann $2^{divisions-1}$ Punkte berechnet, wobei benachbarte Punkte immer den gleichen Abstand haben. Punktwolken für Objekte vom Typ *Sphere* werden mithilfe des Fibonacci-Kugel-Algorithmus erstellt, der ein Fibonacci-Gitter spiralförmig auf eine Kugel projeziert, sodass eine gleichmäßige Punkteverteilung entsteht. Für Objekte vom Typ $NURBSSurface$ wird deren Methode *getPolygonization* genutzt, um an eine Punktwolke zu gelangen. Diese Methode bekommt das Argument $flatness$ übergeben, worüber die Anzahl der Punkte gesteuert werden kann. Je höher der Wert für $flatness$, desto weniger Punkte enthält die Punktwolke.

3.2 Implementierung von achsenparallelen Hüllquadern (AABB)

Zur Berechnung von achsenparallelen Hüllquadern (AABBs) zu einem Objekt wird zuerst eine Menge von Punkten im globalen Koordinatensystem berechnet,

die auf dem Rand des Objekts liegen. Wenn ein Hüllquader für eine Menge von Objekten berechnet werden soll, werden die Punktemengen der einzelnen Objekte vereint. Danach werden die Maximal- und Minimalwerte für alle drei Koordinatenachsen bestimmt und daraus die Länge, Breite und Höhe des Hüllquaders berechnet. Bei Parallelogrammen wird die Höhe auf 0 gesetzt. Bei der Berechnung der globalen Punktemenge wird zwischen einzelnen, einfachen Objekten vom Typ $Sphere$, Box, $Frustum$, $Cylinder$, F, $Cone$ oder $Parallelogram$ und Objekten vom Typ $NURBSSurface$ unterschieden, die zur Modellierung beliebiger Formen verwendet werden können.

3.2.1 Einfache Objekte

Mithilfe der globalen Ausrichtung und Attributen wie Länge, Breite, Höhe oder Radius eines einfachen Objekts werden Punkte berechnet, die auf dem Rand des Objekts liegen und die globale Extrempunkte des Objekts sein könnten.

3.2.2 NURBSSurfaces

Für kompliziertere Objekte, wie NURBS-Flächen wird zur Berechnung einer AABB zuerst eine Punktwolke erstellt und mithilfe der globalen Transformationsmatrix des Objekts zu Punkten im globalen Koordinatensystem der Szene übersetzt.

3.3 Implementierung von orientierten Hüllquadern (OBB)

Die Länge, Breite und Höhe der OBB wird direkt aus den Attributen Länge, Breite, Höhe und Radius eines einfachen Objekts vom Typ $Sphere$, Box, $Frustum$, $Cylinder$, F, $Cone$ oder $Parallelogram$ berechnet. Als globale Transformation wird die Transformationsmatrix des umhüllten Objekts übernommen.

Für Objekte vom Typ $NURBSSurface$ oder für Mengen von Objekten wird eine Punktwolke erstellt. Auf diesen Punktwolken wird eine Hauptkomponentenanalyse (PCA)[7] durchgeführt. Mit der PCA werden die drei Hauptachsen der Punktwolke bestimmt und diese als Position und Orientierung der OBB verwendet. Die Ausmaße der OBB werden durch die Extrempunkte der Punktwolke auf den Hauptachsen bestimmt. Durch Verwendung der PCA ist die OBB für $NURBSSurfaces$ und Objektmengen möglicherweise nicht optimal.

3.4 Implementierung von Hüllellipsoiden mit minimalem Volumen (MVEE)

Zur Berechnung eines MVEE müssen die Radien auf dessen drei Hauptachsen bestimmt werden. Für Objekte der Typen $Parallelogram$ (nur Rechtecke), $Cylinder$ und F werden dazu die halbe Länge und, gegebenenfalls, die halbe Breite oder Radius jeweils mit $\sqrt{2}$ multipliziert. Der Radius auf der Höhenachse des Rechtecks wird auf 0 gesetzt, wodurch es sich bei dem MVEE um eine Hüllellipse handelt.

Der Faktor $\sqrt{2}$ kommt dadurch zustande, dass ein Rechteck ein, um Länge und Breite gestrecktes, Einheitsquadrat darstellt. Da ein Einheitsquadrat eine Diagonale der Länge $\sqrt{2}$ hat, muss ein umschließender Kreis den Durchmesser $\sqrt{2}$ haben, was einer Streckung der Länge oder Höhe um $\sqrt{2}$ entspricht. Für die Hüllellipse des Rechtecks wird der umschließende Kreis des Quadrats an den Achsen um die Länge und Breite des Rechtecks gestreckt. Die Achsenrotation eines Rechtecks um Längen- oder Breitenachse ergibt einen Zylinder, wodurch die Nutzung des Faktors $\sqrt{2}$ auch für die Objekttypen $Cylinder$ und F gerechtfertigt wird.

Die Position und Orientierung des MVEE wird direkt vom umhüllten Objekt übernommen.

Für alle anderen Objekttypen und Objektmengen wird eine Punktwolke erstellt und von dieser Punktwolke die konvexe Hülle gebildet, um die Punktanzahl und somit die Laufzeit des Ellipsoid-Algorithmus zu verringern. Auf die Konvexe Hülle wird dann der Khachiyan-Algorithmus[3] angewendet. Der Khachiyan-Algorithmus liefert nur eine Annäherung an das Hüllellipsoid mit minimalem Volumen, wodurch das Ergebnis nicht optimal ist. Die Genauigkeit kann mithilfe des *tolerance*-Parameters für den Khachiyan-Algorithmus und des $flatness$-Parameters zur Erstellung der Punktwolken von $NURBSSurfaces$ gesteuert werden.

3.5 Skalierungsobjekte

Skalierungsobjekte sind unsichtbare Objekte, die mit ausgehenden $refinement$-Kanten zu mehreren Objekten eine Zusammenfassung dieser darstellen. Die dadurch entstehenden Skalenebenen können zur Erstellung von Hüllkörpern um Objektverbünde genutzt werden.

3.6 Baumstruktur der Hüllobjekte

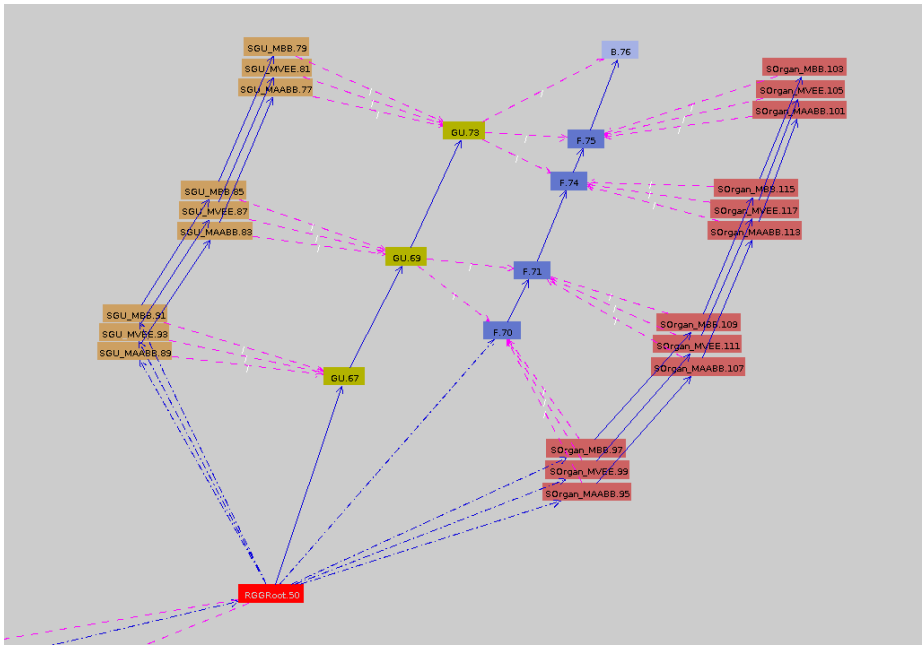


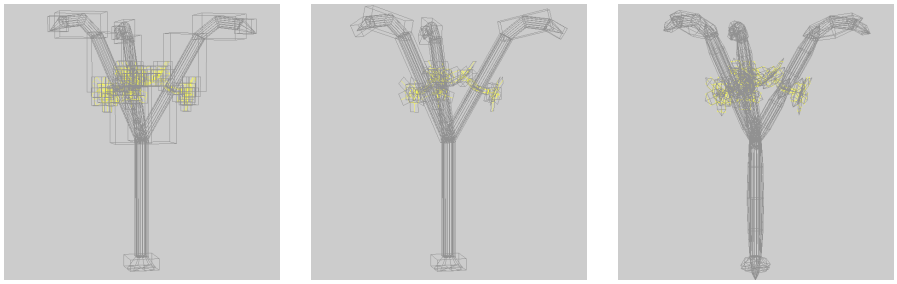
Abbildung 2: Baumstruktur einer Szene mit zwei Skalenebenen. Die blauen Pfeile stellen *successor*- und *branch*-Kanten dar. Die $refinement$-Kanten sind pink dargestellt. In der ersten Skalenebene befinden sich die sichtbaren Objekte der Szene. Diese sind hier als dunkelblaue Knoten dargestellt. Der hellblaue Knoten ist ein Hilfsknoten in der ersten Skalenebene, der nicht graphisch repräsentiert wird und der daher auch keinen Hüllkörper benötigt. Die dunkelroten Knoten auf der rechten Seite des Graphen repräsentieren die Hüllkörper der ersten Skalenebene. $refinement$-Kanten von den dunkelroten zu den dunkelblauen Knoten symbolisieren eine Verfeinerung der Hüllobjekte durch die umhüllten Objekte. Die zweite Skalenebene enthält Skalierungsobjekte, die in gelb dargestellt sind. Der Verbund von Objekten, der von einem Skalierungsobjekt zusammengefasst wird, ist durch $refinement$-Kanten von den gelben zu den blauen Knoten dargestellt und symbolisiert eine Verfeinerung der Skalierungsobjekte durch mehrere einzelne Objekte der ersten Skalenebene. Die Hüllkörper der Skalierungsobjekte in der zweiten Skalenebene sind orange dargestellt und durch $refinement$-Kanten mit diesen verbunden. Der hellrote Knoten zeigt den Wurzelknoten der Szene.

Die Baumstruktur der Originalszene sollte auf allen Skalenebenen, und daher auch auf den Ebenen der Hüllobjekte, konserviert sein. Daher müssen Kanten zwischen Hüllobjekten des gleichen Typs erstellt werden, die den Kanten zwischen den umhüllten Objekten entsprechen. Zuerst wird dazu eine $refinement$-

Kante von jedem Hüllobjekt zum zugehörigen umhüllten Objekt erstellt, um eine Verfeinerung in diese Richtung zu signalisieren. Dann werden *successor*- und *branch*-Kanten Richtung Wurzelknoten zurückverfolgt, bis ein Knoten erreicht wird, der eine eingehende $refinement$-Kante von einem Hüllobjekt der selben Skalenebene besitzt. Falls auf dem Weg zwischen den umhüllten Objekten eine *branch*-Kante zurückverfolgt wurde, wird auch eine *branch*-Kante zwischen den Hüllobjekten erstellt. Andernfalls wurden nur *successor*-Kanten zurückverfolgt, wodurch auch zwischen den Hüllobjekten eine *successor*-Kante erstellt wird. Falls bis zum Wurzelknoten kein weiteres umhülltes Objekt gefunden wurde, wird eine *branch*-Kante vom Wurzelknoten zum Hüllobjekt erstellt. In Abbildung 2 ist die Baumstruktur mit zwei Skalenebenen in einer Beispielszene veranschaulicht.

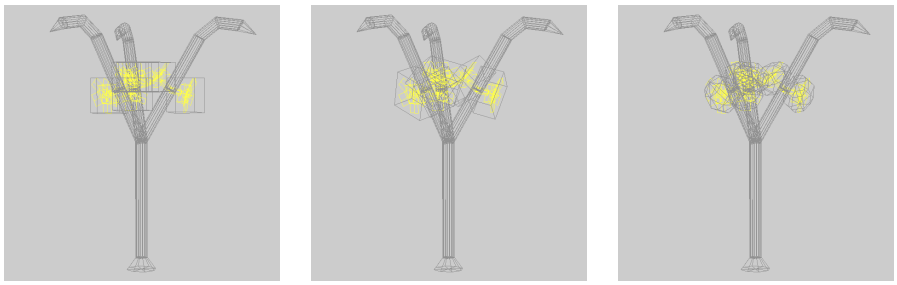
4 Ergebnisse

4.1 Darstellung der Hüllkörper in verschiedenen Skalenebenen anhand einer Beispielszene



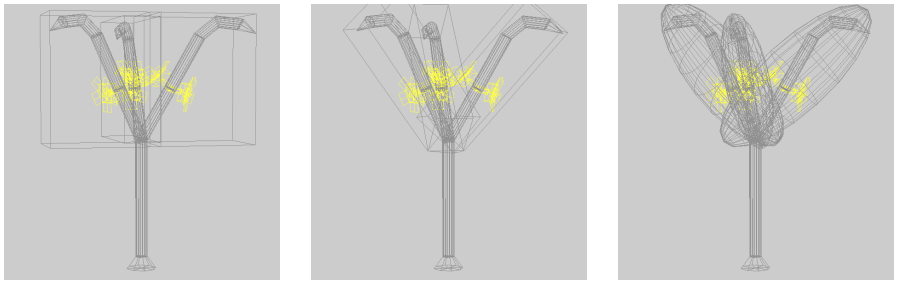
(a) Achsenparallele Hüllquader (b) Orientierte Hüllquader (c) Hüllellipsoide

Abbildung 3: Beispielszene mit Hüllkörpern auf der feinsten Skalenebene mit allen Objekten.



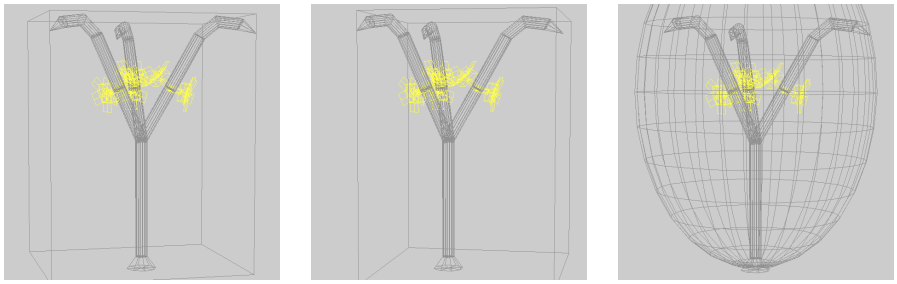
(a) Achsenparallele Hüllquader (b) Orientierte Hüllquader (c) Hüllellipsoide

Abbildung 4: Beispielszene mit Hüllkörpern auf der Skalenebene ganzer Blüten.



(a) Achsenparallele Hüllquader (b) Orientierte Hüllquader (c) Hüllellipsoide

Abbildung 5: Beispielszene mit Hüllkörpern auf der Skalenebene ganzer Äste.



(a) Achsenparallele Hüllquader (b) Orientierte Hüllquader (c) Hüllellipsoide

Abbildung 6: Beispielszene mit Hüllkörpern auf der Skalenebene des ganzen Baumes.

Die Methoden zur Erstellung von Hüllobjekten wurden auf mehrere Szenen angewandt. Die erste Szene stellt einen Baum bestehend aus einem Stamm und drei

Ästen mit jeweils zwei gelben Blüten dar. In dieser Szene sind alle Objekttypen enthalten, für die die Berechnung der Hüllkörper implementiert wurde. In der Szene existieren vier Skalenebenen, die in den nächsten Abbildungen mithilfe der zugehörigen Hüllkörper dargestellt werden. In jeder der folgenden Abbildungen mit drei Bildern zeigt das linke Bild die achsenorientierten Hüllquader, das Bild in der Mitte die orientierten Hüllquader und das rechte Bild die Hüllellipsoide. Skalenebene 1 enthält alle sichtbaren Objekte und ist in Abbildung 3 zu sehen. In Skalenebene 2 wird jedes zu einer Blüte gehörige Objekt mit einem Skalierungsobjekt zu einem Verbund zusammengefasst. Dies ist in Abbildung 4 zu sehen. Abbildung 5 zeigt die Zusammenfassung ganzer Äste, inklusive zugehöriger Blüten, zu einem Verbund mittels eines Skalierungsobjekts. Zuletzt, in Abbildung 6, wird der ganze Baum, zu einem einzigen Verbund zusammengefasst, mit Hüllkörpern dargestellt.

4.2 Experimentelle Messungen der Laufzeiten des Ellipsoid-Algorithmus für verschiedene Werte des *tolerance*-Arguments und verschiedene Größen von Punktwolken.

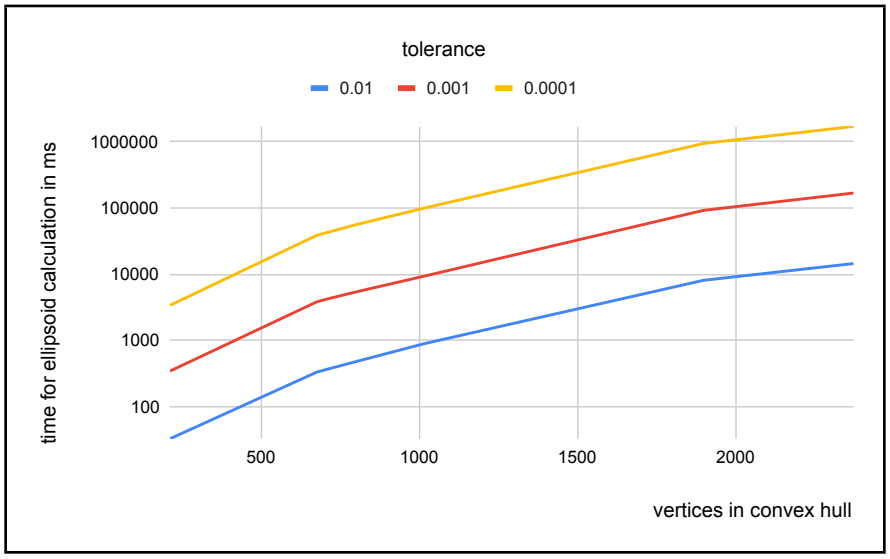


Abbildung 7: Gemessene Zeiten in Millisekunden zur Berechnung eines Hüll-Ellipsoid für die konvexe Hülle einer gegebenen Punktwolke mithilfe des Khachiyan-Algorithmus. Die Zeiten sind für eine bessere Sichtbarkeit in logarithmischer Skalierung eingezeichnet. Für die Messungen wurde ein Kern des AMD Ryzen 7 3700X mit einem Prozessortakt von 3.60 und 1024MB RAM verwendet.

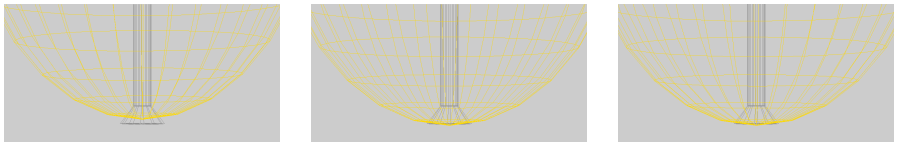
Um die Laufzeit des genutzten Algorithmus zur Berechnung eines Hüllellipsoids für eine gegebene Punktwolke zu bestimmen, wurden einige Messungen durch-

geführt, die in Abbildung 7 dargestellt werden. Das Objekt, für das die Zeiten zur Berechnung des Hüllkörpers gemessen wurden, ist vom Typ $NURBSSurface$. Die verschiedenen Größen der Punktwolke, die als Parameter für den Ellipsoid-Algorithmus dient, wurden durch das Argument $flatness$ der Methode $getPolygonization$ der Klasse $NURBSSurface$ erstellt.

Wie in der Abbildung zu sehen, entspricht eine Division mit 10 des *tolerance*-Wertes bei gleichbleibender Größe der Punktwolke in etwa einer Verzehnfachung der Laufzeit des Algorithmus. Im getesteten Wertebereich reicht eine Verdopplung der Größe der Punktwolke aus, um bei gleichbleibendem *tolerance*-Wert die Laufzeit zu verzehnfachen.

Die auf Khachiyans Ellipsoid-Algorithmus basierende Berechnung des Hüllellipsoids ist also für Aufgaben, bei denen eine kurze Berechnungszeit eine hohe Priorität besitzt, für möglichst kleine Punktwolken interessant. Die achseparallelen Hüllquader und orientierten Hüllquader konnten für alle getesteten Größen der Punktwolke im einstelligen Millisekunden-Bereich berechnet werden.

4.3 Auswirkungen eines hohen *tolerance*-Wertes auf die Genauigkeit des Hüllellipsoids



(a) tolerance = 0.01 (b) tolerance = 0.001 (c) tolerance = 0.0001

Abbildung 8: Beispielszene mit Hüllellipsoiden in der Skalenebene des ganzen Baums. Jedes Bild zeigt das Ergebnis einer Division mit 10 des *tolerance*-Wertes zur Berechnung des Ellipsoids im Vergleich zum vorherigen Bild.

Um den *tolerance*-Parameter für die Berechnung des Hüllellipsoids gut wählen zu können, wurden verschiedene Werte ausprobiert und das Ergebnis visuell bewertet. In Abbildung 8 sind die Auswirkungen von drei verschiedenen *tolerance*-Werten auf die Genauigkeit des Hüllellipsoids in der Beispielszene (Abbildung 6 (c)) zu sehen. Bei einem Wert von 0.01 ist deutlich zu sehen, dass ein kleiner Abschnitt des unteres Stammes nicht vom berechneten Hüllellipsoid eingeschlossen wird. Ab einem *tolerance*-Wert von 0.001 ist dieser Fehler behoben und es ist auch bei einem Wert von 0.0001 keine weitere Veränderung zu sehen. Eine Division des *tolerance*-Wertes mit 10 hat eine Verzehnfachung der benötigten Berechnungszeit zur Folge.

5 Zusammenfassung

5.1 Fazit

Im Laufe des Praktikums konnte die Berechnung von Hüllobjekten für die Plattform GroIMP[5] für Pflanzenmodellierung erfolgreich implementiert werden. Die Hüllobjekte wurden in die bestehende Graphstruktur der Szenen mithilfe der Skalierungsobjekte und Skalenebenen eingebettet und stellen die Topologie der sichtbaren Objekte in höheren Skalenebenen dar. Die Nutzung von achsenparallelen und orientierten Hüllquadern sollte durch geringe Berechnungszeiten für viele Anwendungszwecke möglich sein. Hohe Berechnungszeiten für Hüllellipsoide zu Punktwolken mit großer Anzahl an Punkten in der konvexen Hülle könnten allerdings die Menge an Anwendungsmöglichkeiten für diese Form von Hüllkörpern einschränken.

5.2 Ausblick

Alle Arten von implementierten Hüllkörpern bilden eine Basis für mögliche zukünftige Projekte in der GroIMP[5] Plattform. Es können effiziente Lösungen zu Kollisionsabfragen oder Ray-Tracing geschaffen werden. Auch die gröbere Darstellung von Pflanzen oder Teilen von Pflanzen mithilfe der Hüllkörper schaffen eine vereinfachte Version komplexer Szenen und könnte dadurch in Zukunft für Erkenntnisgewinne sorgen.

Literatur

[1] Jung-Woo Chang, Wenping Wang, and Myung-Soo Kim. Efficient Collision Detection Using a Dual Bounding Volume Hierarchy. In *International Conference on Geometric Modeling and Processing*, pages 143–154. Springer, 2008.

[2] Manfred Ernst and Gunther Greiner. Multi Bounding Volume Hierarchies. In *2008 IEEE Symposium on Interactive Ray Tracing*, pages 35–40. IEEE, 2008.

[3] Leonid G Khachiyan and Michael J Todd. On the complexity of approximating the maximal inscribed ellipsoid for a polytope. Technical report, Cornell University Operations Research and Industrial Engineering, 1990.

[4] James T Klosowski, Martin Held, Joseph SB Mitchell, Henry Sowizral, and Karel Zikan. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[5] Ole Kniemeyer, Gerhard Buck-Sorlin, and Winfried Kurth. GroIMP as a Platform for Functional-Structural Modelling of Plants. *Frontis*, pages 43–52, 2007.

[6] Thomas Larsson and Tomas Akenine-Möller. A Dynamic Bounding Volume Hierarchy for Generalized Collision Detection. *Computers & Graphics*, 30(3):450–459, 2006.

[7] Karl Pearson. LIII. On Lines and Planes of Closest Fit to Systems of Points in Space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

[8] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics (TOG)*, 26(1):6–es, 2007.