Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Generische Annotation von 3D Daten

Marc Stecher

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Thomas Ertl
Betreuer/in:	Dr. Guido Reina, Jun.-Prof. Dr. Michael Krone (Universität Tübingen)
Beginn am:	24. November 2022
Beendet am:	24. Mai 2023

Kurzfassung

Das Erkunden von dreidimensionalen Daten ist ein wichtiger Bestandteil der Visualisierung. Damit hierbei aufkommende Erkenntnisse nicht verloren gehen, muss es eine Möglichkeit geben, diese schnell und einfach aufzuschreiben. Besser ist es noch, wenn diese Anmerkungen direkt in den Daten, welche betrachtet werden, notiert werden können und somit direkt im Zusammenhang der Daten gesehen werden. Darüber hinaus ist es auch wichtig, dass diese so aufgeschriebenen Anmerkungen von anderen eingesehen werden können. Genau solch ein Ansatz wird in dieser Arbeit ausgearbeitet und in dem Prototyping Framework MegaMol implementiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Bei dem Betrachten von 3D Daten können oft neue Hypothesen aufkommen, die am besten direkt aufgeschrieben werden sollten. Da diese 3D Daten oft nicht nur von einer Person betrachtet werden, muss es möglich sein, dass alle hieran beteiligten Personen die Möglichkeit haben die Hypothesen der anderen Betrachter einfach und direkt in den Daten einsehen zu können.

Die Idee hinter dieser Arbeit war, einen Weg zu finden, Erkenntnisse und Anmerkungen die bei dem Betrachten von dreidimensionalen Daten auftreten, direkt innerhalb des hierbei verwendeten Programms festzuhalten, sodass diese zusammen mit den Daten sichtbar sind und keine externen Programme verwendet werden müssen, zwischen denen immer hin- und her-gewechselt werden muss. Damit dies funktionieren kann, muss es ein Objekt geben, dass die Anmerkungen direkt mit dem Raum der Daten verknüpfen kann. Dies sind Annotationen. Diese helfen dadurch, dass zusätzlich zu den Anmerkungen auch verschiedene Daten über den Zustand der aktuellen Simulation abgespeichert werden, um diesen bei Bedarf wieder-herstellen zu können.

Damit der Überblick über alle vorhandenen Annotationen nicht verloren werden kann, muss es auch die Möglichkeit geben, diese in einer Liste oder ähnlichem anzeigen zu lassen. Ebenso muss es möglich sein, dass die Annotationen mit ihren räumlichen Koordinaten innerhalb der Daten verbunden sind, um die Zusammengehörigkeit dieser Annotationen festzustellen.

2 Verwandte Arbeiten

Für Markierungen im 3D Raum gibt es bisher schon in einer Vielzahl an Varianten. Die zwei prominentesten hierbei sind sogenannte Innere und Äußere Markierungen.

Innere Markierungen [CG08][BFH01] zeichnen sich dadurch aus, dass diese direkt auf dem zu markierenden Objekt platziert werden. Somit wird die Markierung direkt in den Zusammenhang mit der Position in den Daten gebracht, ohne dass weitere Funktionen verwendet werden müssen.

Äußere Markierungen [ČB10], versuchen eine leere Fläche zu finden, welche in der Regel außerhalb der angezeigten Daten liegt und platziert die Markierung dort. Damit diese dann immer noch in Verbindung mit dem markierten Punkt gebracht werden kann, wird eine Linie gezeichnet, welche diese miteinander verbindet.

Diese beiden Ansätze haben Ihre Vor und Nachteile. In meiner Arbeit wird hauptsächlich der erste Ansatz verwendet, aber um Überlappungen von Markierungen zu verhindern, werden diese dann verschoben, wodurch die Linien zur Verbindung in Einsatz kommen.

Die meisten dieser Ansätze behandeln jedoch lediglich eine Markierung im Raum, welche einen Titel trägt. Es gibt auch Versionen, die mithilfe von Hyperlinks arbeiten und somit erlauben, dass die Markierungen als Referenz-Punkt gelten, von welchem dann andere Funktionen aus gestartet werden könnten. Ein Beispiel hierfür sind HyperLabels [KIK+21], welche wie ein Ordner agieren. Hierbei sind die Markierungen als Interaktion-Flächen verfügbar, womit unterliegende Strukturen oder weitere Markierungen angezeigt werden können. In diesem Fall handelt es sich bei den verwendeten Datensätzen jedoch um Hierarchische Daten, welche in dieser Arbeit nicht im Mittelpunkt stehen, da es hier mehr um allgemeine 3D Daten geht.

Über das allgemeine Erstellen von Markierungen hinaus, gibt es auch Ansätze die gewisse 3D Datensätze automatisch mit Markierungen versehen können. In [BF14], wird ein Ansatz beschrieben, wie mithilfe von Gruppierungen von Objekten, ein LiDAR Scan mit Markierungen ergänzt werden kann.

Die meisten der genannten Ansätze beschreiben jedoch nur das Platzieren von Markierungen im Raum, welche lediglich einen Titel zugewiesen bekommen und sonst nichts. Die einzige Ausnahme hierbei wären solche, welche die Markierung interaktiv machen und dann mithilfe von weiteren Funktionen, die restlichen Informationen weitergeben.

3 Technische Grundlagen

3.1 MegaMol

MegaMol [GBB+19] ist ein Prototyping Framework für interaktive Visualisierung. Die Entwicklung von MegaMol startete 2006 im Rahmen des Collaborative Research Center 716. Dieses hat sich aber mittlerweile aufgelöst [SFB]. Das Visualisierungsinstitut der Universität Stuttgart (kurz: VISUS) arbeitet weiterhin an MegaMol.

MegaMol wurde zusammen mit Wissenschaftlern aus den Bereichen Biologie, Physik, Materialwissenschaft und Visualisierung entwickelt. Das Ziel hierbei war, die Partikel-basierende Simulationen voranzutreiben um mit immer größeren Datensätzen arbeiten zu können. Mittlerweile hat sich MegaMol weiterentwickelt und wird immer mehr mit neuen Algorithmen und Techniken erweitert, welche Visualisierungen verschiedenster Arten ermöglichen [GBB+19].

Einer der großen Gedanken bei der Entwicklung von MegaMol war es, den Benutzern zu erlauben, alle vorhandenen Ressourcen der Hardware zu verwenden. Somit soll es möglich sein, dass CPU und GPU in ihrem vollständigen Umfang genutzt werden. Damit dies möglich ist, wird ein großer Wert auf die Effizienz und die Benutzbarkeit aller Komponenten gelegt.

Es wurde viel Wert darauf gelegt, dass es Wissenschaftlern schnell möglich ist neue Prototypen an Funktionen austesten. Damit dies geschehen kann, werden alle verwendeten Funktionen und Komponenten als eine Pipeline aneinandergereiht und ergeben dann ein Funktionierendes Projekt. Durch diese hohe Modularität, ist es möglich einige der Funktionen in verschiedenen Bereichen wiederzuverwenden.

3.2 ImGui

[Dear ImGui], im weiteren Verlauf einfach nur ImGui genannt, ist eine Bibliothek für graphische Benutzerinteraktionen in C++. Es kann direkt in die 3D Pipeline der verwendeten Anwendung eingebaut werden, wodurch es jederzeit angezeigt werden kann. Ebenso ist es einfach und besitzt keine zusätzlichen Externen Bedingungen zur Verwendung.

ImGui kann somit verwendet werden um schnell, eine UI zu erstellen, die keine externen Voraussetzungen besitzt. Dies ermöglicht es ohne das Importieren von vielen unnötigen Bibliotheken, das zugrundeliegende Programm um diese UI zu erweitern. Ebenso kann ImGui mithilfe von wenigen Zeilen Code bereits Fenster anzeigen.

ImGui wird innerhalb von MegaMol bereits für verschiedene Anzeigen in anderen Modulen verwendet, weswegen alle Grundlagen hierfür bereits vorhanden waren und es somit möglich war direkt damit zu Arbeiten.

4 Gesamtkonzept

4.1 Annotation

In dieser Arbeit geht es um das Erstellen und verwenden von Annotationen in 3D Daten. Mit Annotationen sind hierbei Anmerkungen gemeint, die ein Benutzer zu interessanten Punkten innerhalb der Daten geben möchte. Damit dies möglich ist, sind in meiner Arbeit Annotationen ein Objekt welches mehrere Komponenten besitzt, um diese Anmerkungen im Raum festzuhalten. Diese Komponenten sind die folgenden:

- **Titel:** Dieser beschreibt den aktuellen Sachverhalt oder Erkenntnis die festgehalten werden soll.
- **Text:** Hier können die genaueren Gedanken, Fragen oder mögliche Erklärungen für diesen Sachverhalt oder Erkenntnis festgehalten werden.
- **Start und End-Zeitpunkt:** Hiermit wird der Zeitraum in dem diese Annotation vorhanden sein soll, festgehalten.
- **Räumlichen Standort:** Hierbei werden die markierten Koordinaten für die Annotation im 3D Raum festgehalten, um jeder Annotation eine räumliche Zuordnung zu geben.
- **Kamera Position:** Die genauen Blickrichtungen und Position der Kamera werden hiermit abspeichert, um diese wieder herstellen zu können, damit die Annotation direkt aus dem gewünschten Blickwinkel wieder gesehen werden kann.

4.2 Design Entscheidungen

Die Annotationen bestehen aus den oben beschriebenen 5 Komponenten, da dies eine Methode war, mit der möglichst wenige Daten die abhängig von dem geladenen Datensatz sind, gespeichert werden müssen. Allgemein sollten Annotationen einen Titel und Anmerkung zusammen mit dem aktuellen Zustand der Daten, speichern. Da MegaMol es erlaubt die Kamera-Positionen und Zeitpunkte der Erstellung der Annotationen zu ermitteln, müssen hierfür lediglich diese Informationen abgespeichert werden, anstatt von einem Schnappschuss der Daten, welcher schwieriger gewesen wäre, als allgemeine Funktion, Variable oder Datei zu erstellen, da dieses Modul es ermöglichen soll, mit jeglicher, von MegaMol unterstützen, Art von 3D Daten arbeiten zu können.

In meinem Programm wurde ImGui zur Darstellung aller von mir erzeugten Fenstern verwendet. Hierfür mussten keine nicht bereits in MegaMol verwendete Bibliotheken importiert werden, da ImGui bereits in anderen Modulen verwendet wird. Ebenso ist es mit ImGui einfach, viele Fenster zu erstellen, ohne dass dies einen zu starken Einfluss auf die Performanz von MegaMol haben sollte.

Es wurde in meinem Modul für alle Darstellungen, die der Benutzer sieht, entweder eine Zeichnung mithilfe OpenGL gemacht oder ein von ImGui erstelltes Fenster, verwendet. Es wurden nur diese zwei Programme benutzt, da dies eine Anpassung durch weitere Entwicklungen einfacher macht.

4.3 High level Überblick

Dieses Modul erlaubt es dem Benutzer, in MegaMol einen beliebigen 3D Datensatz der auch zeitliche Daten in der Form von Animationen beinhalten kann, zu analysieren und annotieren. Hierfür können Annotationen erstellt werden, indem ein Punkt in Raum und Zeit der Daten und Animationen von diesen ausgewählt wird. Diesem Punkt können dann die vorher erwähnten Eigenschaften zugewiesen werden und dann abgespeichert werden. Als zweites gibt es die Möglichkeit, diese erstellten Annotationen im Raum der Daten anzeigen zu lassen. Damit der Überblick über alle Annotationen nicht verloren werden kann, gibt es ein Fenster, welches alle Annotationen unabhängig von ihrer Sichtbarkeit auflistet.

5 Implementierung

Um das manuelle Erstellen von Annotationen zu ermöglichen, wurde MegaMol um das Modul “AnnotationRenderer” erweitert. Dieses Modul implementiert einen eigenen Renderer, welcher in der Pipeline direkt rechts von dem “View Modul” eingefügt werden kann. Dies kann in Abbildung 5.1 am Beispiel des Projektes *TestSpheres* gesehen werden.

Das in Kapitel 4 beschriebene Gesamtkonzept wurde so umgesetzt, dass jede Funktion die hier beschrieben wurde, als einzelne Fenster Implementiert wurde. Somit sind die einzelnen Funktionen voneinander getrennt und der Benutzer kann diese aktivieren und deaktivieren wie es gerade benötigt wird. Das erste Fenster ist für das Erstellen von neuen Annotationen gedacht. Als Zweites gibt es ein Fenster für jede Annotation, die sichtbar sein sollte, welches diese im Raum der Daten anzeigt. Das letzte Fenster beinhaltet eine Liste aller Annotationen, um den Überblick über diese zu behalten. Ebenso ist es möglich, in der Parameterliste von MegaMol, welche in Abbildung 5.2 zu sehen ist, Annotationen in der Form von JSON-Dateien zu importieren und exportieren, sowie verschiedene Einstellungen der anderen Fenster zu steuern.

5.1 Annotation hinzufügen

Dieses ImGui Fenster ist für das Erstellen von neuen Annotationen gedacht. Eine bereits ausgefüllte Version davon ist in Abbildung 5.3 zu sehen. Es besitzt Eingabe-Felder für einen Titel und eine Beschreibung oder Notiz, die mit dieser Annotation festgehalten werden möchte. Danach kommen mehrere Tasten mit welchen die räumlichen und zeitlichen Koordinaten für diese neue Annotation, sowie die Kamera-Position festgelegt werden können. Um die räumlichen Koordinaten zu überprüfen, gibt es ebenfalls eine Taste, die eine Kugel an den zuvor ausgewählten Koordinaten anzeigt. Als Letztes gibt es dann eine Taste der es erlaubt die somit zusammengestellte Annotation abzuspeichern.

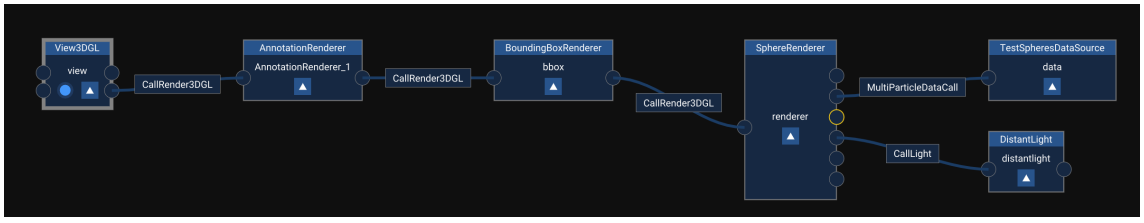


Abbildung 5.1: Rendering Pipeline: Dies ist ein Beispiel wie der AnnotationRenderer in die Pipline eines Projektes eingebaut werden kann. Dieser liegt am besten rechts von dem View Modul.

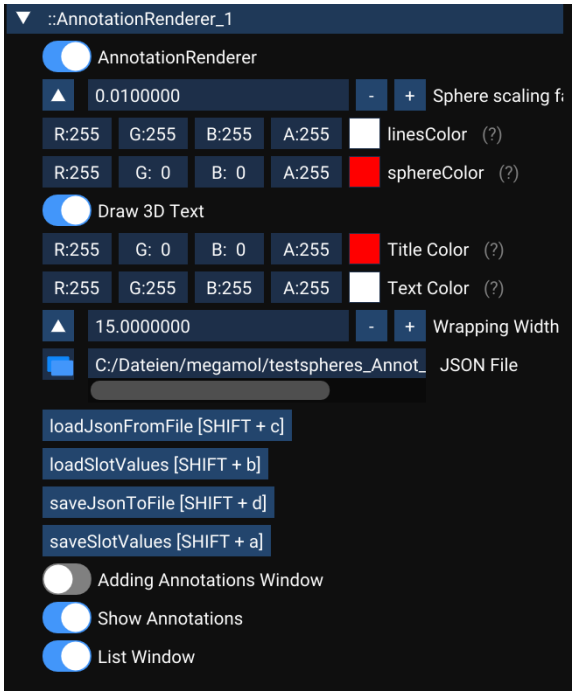


Abbildung 5.2: Das Parameter-Menü des AnnotationRenderers in MegaMol. Hiermit werden die Größen und Farben von Titel, Text, Linien und Kugeln gesteuert. Ebenso wird das Anzeigen der einzelnen Fenster mithilfe der Schalter geregelt. Zusätzlich ist es möglich die Annotationen und Einstellungen dieser Slots, zu Importieren und Exportieren.

Die Auswahl der räumlichen Koordinaten wird mit dem Drücken der entsprechenden Taste gestartet. Hierbei wartet das Programm darauf, dass mit einem Links-Klick eine Position innerhalb der Daten auf dem Bildschirm ausgewählt wird. Anschließend werden die aktuellen Koordinaten der Maus auf dem Bildschirm genommen und der Tiefen-Puffer wird an dieser Koordinate befragt, welche Tiefe er hat. Sobald hier ein Ergebnis zurückkommt, das zeigt, es gibt ein Element an dieser Stelle, dann wird diese Tiefe als neue z Koordinate verwendet. Die Koordinaten der Maus werden nun ebenfalls in Welt-Koordinaten umgerechnet.

Es wird hierbei zuerst der y-Wert geflippt, da ImGui die y-Achse in die gegen-gesetzte Richtung aufspannt, wie es bei OpenGL der Fall ist. Anschließend werden die x und y-Werte in die Form des Einheitswürfels gebracht. Hierbei werden mit den Werten `lhsFBO->getWidth()` und `lhsFBO->getHeight()` die Dimensionen des Framebuffers bestimmt, um die Umrechnung in den Einheitswürfel zu ermöglichen. Anschließend wird die übliche Umrechnung von Bildschirm-Koordinaten in Welt-Koordinaten angewandt. Diese ist das Multiplizieren der inversen Model-View-Projection Matrix mit dem homogenen Vektor $(nx, ny, z, 1.0)$. Zuletzt werden die ersten drei Einträge des resultierenden Vektors durch den vierten Eintrag geteilt.

Für die Auswahl der zeitlichen Koordinaten gibt es zwei Tasten, die jeweils den aktuellen Zeitpunkt der Simulation/Animation abspeichern. Es gibt hier zwei Tasten, da meistens eine Annotation über einen Zeitraum hinweg vorhanden sein sollte und nicht nur zu einem einzelnen Frame.

Die Kamera-Position wird direkt durch das Drücken der entsprechenden Taste abgespeichert.

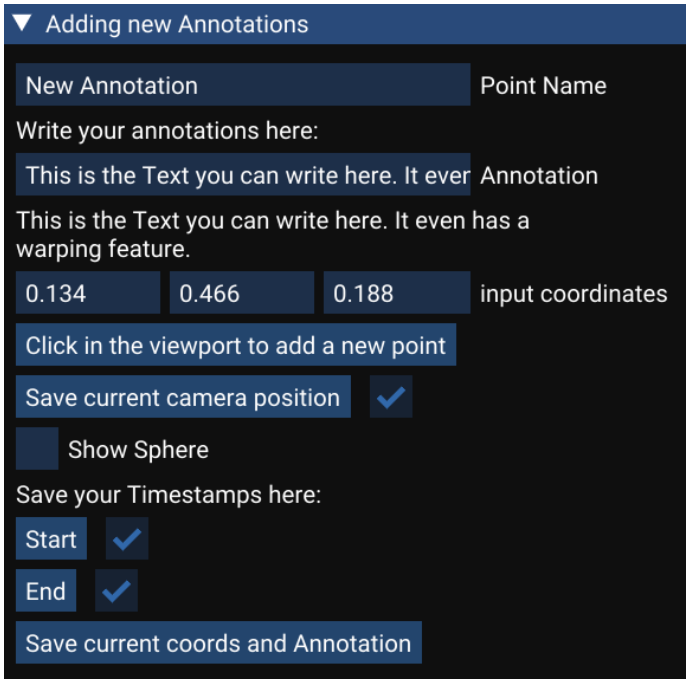


Abbildung 5.3: Fenster zum Erstellen neuer Annotationen: Dieses Fenster erlaubt es neue Annotationen zu erstellen und abzuspeichern. Hierbei ist es möglich den Namen der Annotation, die Notiz in Textfelder einzugeben. Weiter ist es möglich Die aktuelle Kamearposition und Zeitposition zu speichern. Anschließend kann die Annotation gespeichert werden.

Damit der Benutzer direkt sehen kann, welche Knöpfe bereits einen Wert bekommen haben, gibt es hinter jeder Taste eine Box, die immer dann einen Haken bekommt, wenn der entsprechende Taste mindestens einmal gedrückt wurde. Dies hilft es den Überblick zu behalten und es wird verhindert, dass einzelne Parameter nicht angegeben werden. Sobald alle Felder ausgefüllt und alle Daten angegeben wurden, kann der Benutzer diese mithilfe der letzten Taste abspeichern.

5.2 Darstellung von Primitiven

In dieser Arbeit werden hauptsächlich Fenster für die Interaktion mit dem Benutzer und zum Anzeigen der Texte verwendet. Jedoch gibt es auch Informationen, die nicht durch solche Fenster dargestellt werden können. Diese wären die genaue Position der Annotation im 3D Raum oder eine Linie, welche die jeweiligen Fenster der Annotationen mit ihren Koordinaten verbindet um diesen Zusammenhang nicht zu verlieren auch wenn das Fenster verschoben wurde.

Darstellung von Punkten im 3D Raum

Wenn Annotationen Welt-Koordinaten zugewiesen bekommen, dann können diese Koordinaten durch eine Kugel dargestellt werden. Diese Kugel kann mithilfe von zwei Reglern ihre Größe und Farbe verändern.

Das Generieren der Kugeln geschieht hierbei mithilfe des sogenannten *Immediate Mode Renderings* das OpenGL anbietet. Es wird dieser Modus hier verwendet, da er einfacher zu manipulieren war und es somit auch einfacher ist, diesen für jede einzelne Kugel aufzurufen.

Bei dem Zeichnen der Kugeln muss hierfür `glBegin(GL_POINTS)` gefolgt von `glVertex3f(x,y,z)` und vollendet mit `glEnd(GL_POINTS)` werden.

Damit dieser Code nicht nur einen einfachen Punkt, sondern eine Kugel zeichnet, wurden hier die Shader aus dem Modul *megamol101_gl* verwendet. Hierbei handelt es sich um die drei Dateien, die mit *pretty_points* anfangen.

Damit diese gezeichneten Kugeln auch richtig gezeichnet werden, wird der gesamte Code für ihre Generierung noch von einem Tiefen-Test umgeben, sodass diese immer in der richtigen Tiefe angezeigt werden und es keine unerwünschten Überlappungen gibt.

Verbindungslinien

Damit es möglich ist, die Korrelation zwischen Fenstern und den jeweils dazugehörigen Punkten besser nachzuvollziehen, gibt es eine Funktion, die zwischen diesen zwei Objekten eine Linie zeichnet um die Zusammengehörigkeit direkt zu erkennen.

Diese Linien werden somit immer zwischen einem Punkt im 3D Raum und einem Fenster dass sich im Bild-Raum befindet gezeichnet. Damit diese Linien immer an dem jeweiligen Fenster hängen, werden hierbei die Koordinaten des Fensters im Bild-Raum ausgelesen und dann in Welt-Koordinaten umgerechnet. Diese Umrechnung erfolgt wieder mithilfe der Funktion, die bereits in Abschnitt 5.1 erklärt wurde.

Die z-Koordinate wird als Eingabe-Parameter in die Funktion mit eingegeben, da diese nicht abhängig von den (x, y) Koordinaten ist und somit frei gewählt werden kann. In dem Fall des Zeichnens der Linien, wird jedoch ein z-Wert von -1 genommen, da dieser dafür sorgt, dass die Linie immer auf dem Bildschirm endet und es somit den Eindruck erstellt, dass die Linie direkt das Fenster verbindet.

Nachdem die Welt-Koordinaten für das Fenster berechnet wurden, kann mithilfe des *Immediate Mode* in OpenGL eine Linie zwischen den zwei Punkten gezeichnet werden. Mithilfe eines Farb-Feldes in dem Parameter-Menü des Renderers kann ebenfalls eine Farbe für alle Linien ausgewählt werden. Ein Beispiel für eine Linie ist in Abbildung 5.4 zu sehen.

5.3 Annotationen anzeigen im 3D Raum

Alle Annotationen, die im Objekt-Space eine Markierung bekommen haben, werden immer dann angezeigt, wenn die folgenden zwei Bedingungen erfüllt sind. Die Kugel, welche diese Markierung visuell darstellt, wird nicht durch andere Objekte verdeckt. Ebenso muss der aktuelle Zeitpunkt der Animation innerhalb der abgespeicherten Zeitspanne der Annotation liegen.

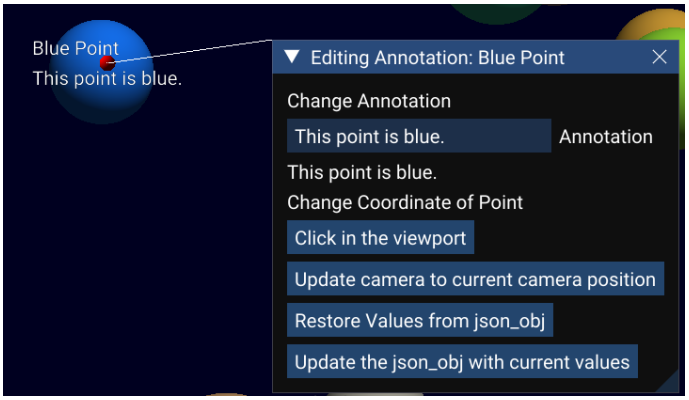


Abbildung 5.4: Dies ist die Verbindungslinie zwischen einem Fenster und seinem entsprechenden Punkt. In diesem Fall ist das Fenster ein Bearbeitung-Fenster.

Sichtbarkeit

Um alle angezeigten Annotationen auf ihre aktuelle Sichtbarkeit zu testen, wird die von OpenGL gegebene Funktion `glBeginQuery` mit dem Parameter `GL_ANY_SAMPLES_PASSED` verwendet. Diese Funktion erlaubt es für jede Kugel die gemalt wird, zu testen, ob mindestens ein Pixel im endgültigen Bild zu sehen sein wird.

Als Vorbereitung für diesen Test, wird ein Struct erstellt, der mehrere Vektoren beinhaltet. Diese Vektoren speichern das folgende ab:

- `query`: Die Anfrage selbst.
- `result`: Das Resultat der Anfrage.
- `resultAv`: Die Aussage, ob das Resultat verfügbar ist.
- `queryStarted`: Einen Boolean der angibt, ob eine Anfrage gestellt wurde.

Die Länge dieser Vektoren entspricht der doppelten Anzahl an aktuell vorhandenen Annotationen. Somit gibt es für jede Annotation zwei Einträge in den Vektoren. Dies wird benötigt, um Einträge für gerade und ungerade Frames unterscheiden zu können, denn `glQuery` gibt immer erst einen Frame später die Antwort auf die gestellte Frage. Diese Vektoren werden immer dann erweitert, wenn neue Annotationen hinzukommen.

Wenn die Funktion, welche das Anzeigen der Beschriftungen steuert, aktiviert wird, wird für jede Annotation die zu dem aktuellen Zeitpunkt in der Animation angezeigt werden soll, die jeweilige Kugel an ihrer entsprechenden Position im 3D Raum mithilfe einer ähnlichen Funktion wie in Abschnitt 5.2 gezeichnet. Der einzige Unterschied hierbei ist, dass

für jede Kugel die gezeichnet wird, zusätzlich ein `glBeginQuery` mit den Parametern `GL_ANY_SAMPLES_PASSED` und dem Eintrag in dem Vektor `query` aufgerufen wird um die jeweilige Anfrage zu starten. Hierbei wird in dem Vektor `query` der index: `2 * i + frameType` verwendet, wobei `frameType` hierbei bestimmt, ob es sich um einen geraden oder ungeraden Frame handelt. Danach wird die Kugel normal gezeichnet und ein `glEndQuery` wird aufgerufen, um die Query zu beenden. Anschließend wird im Vektor `queryStarted` der entsprechende Eintrag auf `true` gesetzt, um zu signalisieren, dass diese Query auch gestartet wurde.

Der zweite Teil der Berechnung findet nun immer um ein Frame versetzt statt, da dadurch mit hoher Wahrscheinlichkeit garantiert werden kann, dass die Ergebnisse der glQuery vorhanden sind. Dieser Teil der Berechnung geht über alle Annotationen und testet für jede getrennt ihren aktuellen Zustand. Hierbei wird zuerst überprüft, ob die Annotation überhaupt sichtbar sein soll. Hierfür werden zuerst die Start- und End-Zeitpunkte der Annotation verglichen, da es die zwei folgenden Fälle gibt. Der normale Fall, in dem gilt: Start $\leq$ Ende. Der zweite Fall ist hierbei interessanter, da hier die Annotation über das Ende der Animation hinausgeht und dann wieder am Anfang dieser weitergeht. In diesem Fall gilt Ende $\leq$ Start. Anschließend wird entsprechend der Verteilung des Start- und End-Zeitpunktes bestimmt, ob der aktuelle Zeitpunkt der Animation sich innerhalb des Bereiches, der durch die zwei Ränder bestimmt wird, liegt. Sollte dies nicht der Fall sein, ist die Annotation nicht sichtbar und wird als solche deklariert. Sollte der aktuelle Zeitpunkt jedoch in diesem Zeitbereich liegen, dann wird der nächste Test durchgeführt, der nun mit den Ergebnissen der vorher generierten glQuery arbeitet. Hier wird zuerst der `FrameType` invertiert, sodass alle weitere Anfragen an die Vektoren immer den vorherigen Frame benutzen. Die erste Anfrage hierbei ist, dass ermittelt wird, ob eine Query überhaupt gestartet wurde. Dies ist notwendig, da ansonsten in den Ersten Frames, nachdem der Vektor erstellt oder verändert wurde, noch keine Anfragen herausgegangen sind und somit auch noch keine Antworten vorhanden sein können, was sonst zu einem Fehler des Programms führen würde. Sollte jetzt eine Query vorhanden sein, wird zuerst abgefragt, ob eine Antwort bereits vorhanden ist. Sollte dies der Fall sein, wird die Antwort der eigentlichen Query ausgelesen. Wenn diese den Wert `GL_TRUE` aufweist, dann ist die Annotation sichtbar und es wird die Variable `show_point` auf `true` gesetzt. Womit dann die Funktion `display_visual_points_windows`, welche die eigentlichen Fenster generiert, diese dann zeichnen kann. Sollte das Ergebnis etwas anderes zurückgeben, dann wird die Variable `show_point` auf `false` gesetzt und es wird kein Fenster gezeichnet.

Sobald bei einer Annotation die Variable `show_point` auf `true` gesetzt ist, wird ein ImGui Fenster über der Stelle der Markierung dargestellt. Dieses Fenster beinhaltet hierbei die Information über den Namen der Annotation und die Notiz, welche hiermit verbunden ist.

Sollten sich mehrere Fenster von Annotationen die eng beieinander liegen überlappen, werden diese mithilfe eines *Force Directed Layout* Ansatzes voneinander weggeschoben, sodass diese Überlappung nicht mehr stattfindet. Damit diese nun verschobenen Fenster immer noch ihren Koordinaten zugeordnet werden können, wird eine Linie zwischen dem Fenster und dem Punkt im Objekt-Space gezeichnet.

Sollte der Benutzer nur die Kugeln der Annotationen, aber nicht den Text dieser sehen, dann gibt es einen Schalter mit dem Namen *Draw 3D Text*, welcher genau dies bewirkt. Somit ist es möglich die Positionen der sichtbaren Annotationen zu sehen, ohne dass Text diese verdeckt.

Verschiebung

Damit die Verschiebung der Fenster im Falle einer Überlappung möglich ist, wird die Größe der Fenster in jedem Frame abgespeichert. Dies ist notwendig, da im ersten Frame in dem die Fenster gezeichnet werden, diese Größe noch nicht festgelegt ist und somit frühestens im zweiten Frame korrekt abgefragt werden kann. Nachdem im vorherigen Teil berechnet wurde, welche Annotationen angezeigt werden sollen, geht diese Funktion nun über alle Annotationen und sucht nur die heraus, welche gezeichnet werden sollen. Hierbei werden für jede dieser Annotationen die ursprünglich

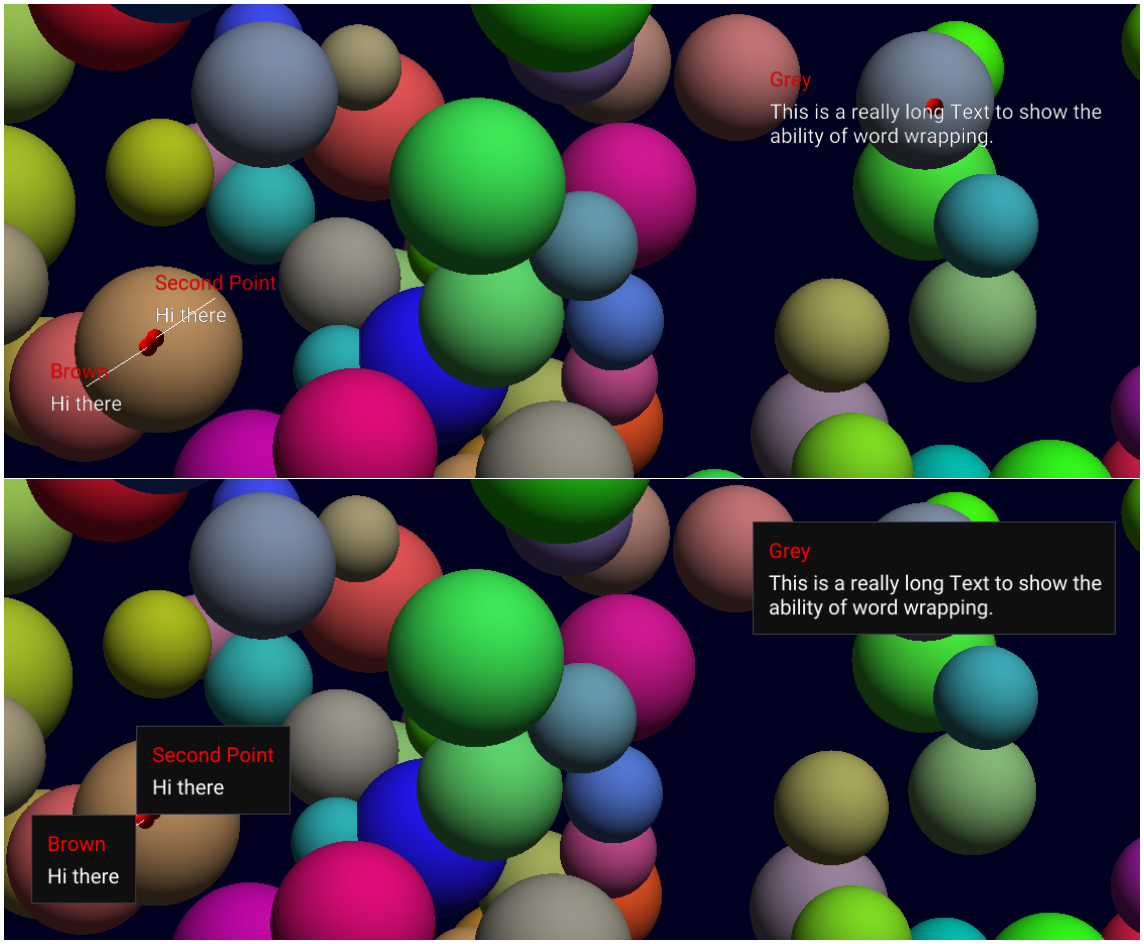


Abbildung 5.5: Anzeigen von Annotationen im Raum. Das erste Bild zeigt die Annotationen mit durchsichtigem Hintergrund, während in dem zweiten der Hintergrund nicht durchsichtig ist. Dies ist für eine bessere Lesbarkeit sinnvoll in dichten Datensätzen.

geplante Position auf dem Bildschirm und zwei weitere Variablen, `Offset` und `totalOffset` für die Schleifen in einem neuen lokalen Vektor gespeichert. Anschließend wird mithilfe diesem, meistens kleineren Vektor, alle Annotationen die gezeichnet werden miteinander verglichen. In diesen Schleifen wird zuerst geprüft, ob sich die zwei Fenster überhaupt überlappen und wenn dies nicht der Fall ist, wird direkt zum nächsten Vergleich übergegangen. Wenn sich die Fenster überlappen, dann wird zuerst der Bereich der Überlappung berechnet. Anschließend wird getestet wie die Fenster zueinander liegen, indem die x und y Koordinaten verglichen werden. Sobald dies feststeht, bekommen beide Fenster einen Offset, der jeweils der Hälfte der Überlappung entspricht. Dies sorgt dafür, dass beide Fenster komplett voneinander weggeschoben werden und danach sich nicht mehr überlagern sollten. Dies wird fortgesetzt, bis jedes Fenster mit jedem anderen Fenster genau einmal verglichen wurde.

Sobald dies vollendet ist, wird der Offset von jedem Fenster auf dessen Bildschirm-Position addiert und ebenfalls in der Variable `totalOffset` abgespeichert. Diese Berechnung wird nun insgesamt 5 mal durchlaufen, damit Fälle, in denen sich Fenster, die nach der Verschiebung wieder überlappen

würden, erneut verschoben werden können um sich nicht zu überlappen. Am Ende des fünften Durchlaufes werden dann alle Fenster mithilfe der Berechneten Offsets gezeichnet. Somit sollte es nun in den meisten Fällen keine Überlappungen mehr geben.

Färbung der Annotationen

Als eine weitere Option für das Anzeigen der Annotationen, gibt es zwei Felder in dem Parameter-Menü des AnnotationRenderes, mit denen die Farbe der Titel und der Notizen von allen angezeigten Annotationen verändert werden kann. Dies ist hilfreich, wenn das aktuelle Projekt und der dazugehörige Datensatz, sehr dunkel ist und somit der schwarze Text schwer zu erkennen ist. Die Option der Farbänderung kann nun in diesen Fällen eine Abhilfe schaffen und besser die Titel und Notizen der Annotationen zum Vorschein bringen.

Ein Beispiel wie dies aussieht, ist in Abbildung 5.5 zu sehen. Hierbei ist auch die Option des An- und Ausschalten des Hintergrunds der Fenster zu sehen. Denn sollten selbst die Farben des Textes nicht helfen, da in dem Datensatz ein zu starker Farbkontrast besteht, dann kann hier der Hintergrund undurchsichtig gemacht werden. Dies kann natürlich Daten verdecken, aber es ist möglich, die Annotationen ohne Probleme zu lesen. Sobald es wieder um die Erkundung der Daten geht, kann dieser Hintergrund wieder durchsichtig gemacht werden, um die Daten wieder besser zu erkennen.

5.4 Liste aller Annotationen

Dieses ImGui Fenster beinhaltet eine Liste aller bisher erstellten Annotationen für das aktuelle Projekt. Ein Beispiel für dieses Fenster ist in Abbildung 5.6 zu sehen. Diese Liste wird in Tabellenform angezeigt. Dies kommt daher, dass es somit möglich ist, mehrere verschiedene Einträge zu erstellen, welche wichtige Informationen für jede Annotation darstellen.

Der erste Eintrag beinhaltet den Name der jeweiligen Annotation.

Der zweite Eintrag ist Farb-kodiert und zeigt die aktuelle Sichtbarkeit der jeweiligen Annotation. Diese Sichtbarkeit ist in die folgenden drei Arten und Farben aufgeteilt.

- **Roter Eintrag:** Diese Annotation ist aufgrund der Zeitlichen Einschränkung nicht sichtbar.
- **Gelber Eintrag:** Diese Annotation ist zum aktuellen Zeitpunkt vorhanden, ist jedoch nicht visuell sichtbar, da sie verdeckt wird.
- **Grüner Eintrag:** Diese Annotation ist zum aktuellen Zeitpunkt mit den aktuellen Kamera-Einstellungen sichtbar.

In dieser Liste wird neben jeder Annotation eine Zeitleiste angezeigt, die darstellt, in welchem zeitlichen Bereich die Annotation zu sehen ist. Diese wird mithilfe der Funktion `ImGui::PlotLines()` gezeichnet. Damit dies variabel funktioniert, wird hierbei zuerst die gesamt-Anzahl der Frames die die Animation besitzt, durch 100 geteilt, welches die Anzahl der Segmente in dieser Zeitleiste ist. Anschließend werden die Start- und End-Zeitpunkte ebenfalls in dieses neue Zahlensystem umgewandelt. Als Nächstes wird überprüft, in welcher Reihenfolge die Start- und End-Zeitpunkte liegen. Dies muss getan werden, da die Animation endlich ist und somit ein Ende hat, nach dem

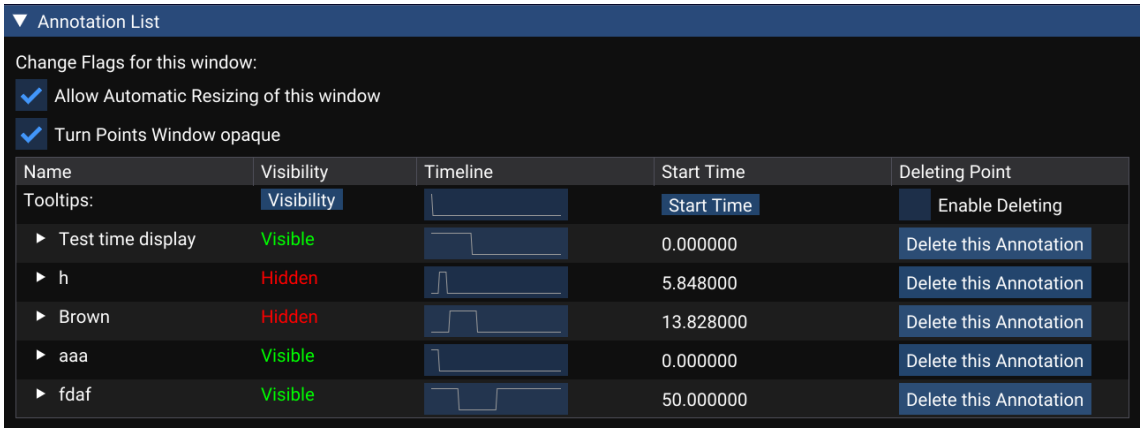


Abbildung 5.6: Liste aller Annotationen. Die erste Spalte beinhaltet den Namen, die zweite die Sichtbarkeit, wobei grün dafür steht, dass der Punkt sichtbar ist, gelb dafür, dass der Punkt von Daten verdeckt wird und rot dafür, dass der Punkt zum aktuellen Zeitpunkt in der Animation nicht vorhanden ist. Ebenso wird eine Zeitleiste dargestellt um zu zeigen wann die Annotation innerhalb der Simulation verfügbar ist. In der letzten Spalte ist es möglich die Annotation zu löschen.

wieder von vorne begonnen wird. Wenn eine Annotation genau über diesen Umbruch verläuft, tritt somit der Fall auf, dass der Wert des Start-Zeitpunktes größer ist als der des End-Zeitpunktes. Nun bekommen alle Zeitpunkte zwischen dem Index 0 und 99 einen Wert zugewiesen, der 1.0 ist, wenn die Annotation sichtbar und 0.0, wenn diese nicht sichtbar ist. Alle Werte hierbei werden in einem Array der Länge 100 gespeichert. Als Letztes wird dieses Array von Werten gezeichnet. Diese Zeitleiste erlaubt es dem Benutzer einen direkten Überblick über die zeitliche Verteilung der Annotationen zu bekommen.

Als letzten Eintrag in jeder Zeile gibt es eine Taste, die es ermöglicht die jeweilige Annotation zu löschen. Damit dies nicht unbeabsichtigt geschehen kann, muss vorher in der ersten Zeile der Liste der Haken bei der Box *Enable Deleting* gesetzt werden. Somit ist es nach Setzen von diesem Haken möglich, schnell mehrere Annotationen ohne großen Aufwand zu löschen. Hierbei muss jedoch beachtet werden, dass diese Annotationen vollständig innerhalb allen Variablen entfernt werden und es hierfür keine Möglichkeit mehr gibt, dies rückgängig zu machen.

Damit ein besserer Überblick über all diese Informationen gehalten werden kann, gibt es in der ersten Zeile der Liste, einen Tooltip für die Spalte der Sichtbarkeit. Ebenso gibt es eine eigene Zeitleiste, welche den Zeitpunkt des aktuell angezeigten Frames darstellt. Somit ist es dem Benutzer möglich einzuschätzen wie weit eine Annotation noch vorhanden sein wird oder wie lange es noch dauert bis diese angezeigt wird.

Jeder Eintrag in der Liste kann aufgeklappt werden, um genauere Informationen über die jeweilige Annotation zu bekommen. Dies ist in Abbildung 5.7 zu erkennen. Hierzu zählen die mit der Annotation verknüpfte Notiz, sowie Tasten die es erlauben, direkt die abgespeicherte Kameraposition wiederherzustellen. Ebenso ist es hier möglich zu dem Anfangszeitpunkt der Annotation zu springen. Mithilfe dieser zwei Funktionen erlaubt es dem Benutzer einfach an die genaue Stelle der Annotation zu springen.

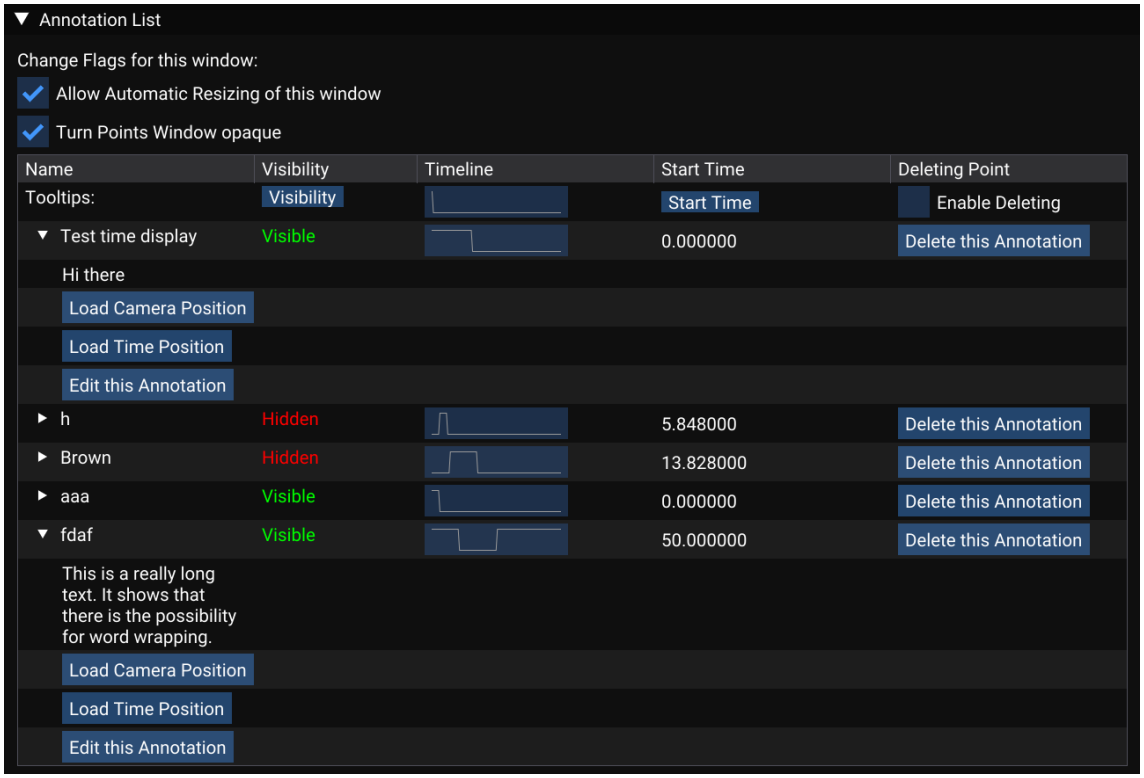


Abbildung 5.7: Hier sind mehrere Annotationen in der Liste aufgeklappt. Direkt unter den Namen sind die Notizen lesbar. Danach kommen zwei Tasten die es erlauben, den ursprünglichen Zustand der Daten innerhalb der Animation wiederherzustellen. Hierfür können die Kamera-Position und der Start-Zeitpunkt geladen werden. Als letztes gibt es noch die Möglichkeit das Bearbeitungsfenster für diese Annotation zu öffnen.

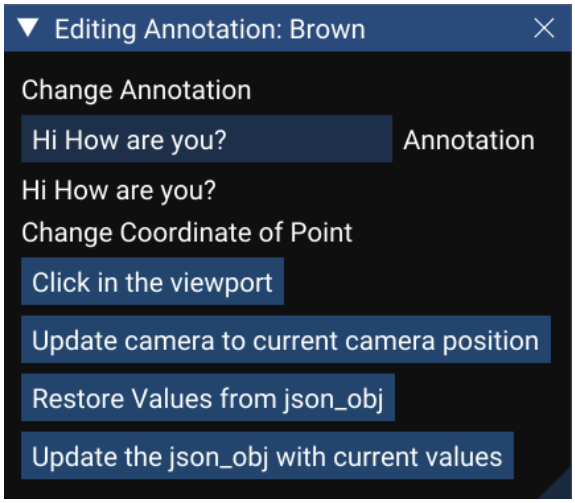


Abbildung 5.8: Bearbeitungsfenster für bestehende Annotationen. Hierbei können der Inhalt der Notiz, die Koordinaten im 3D Raum und die Kamera-Position geändert werden. Ebenso kann die zuletzt gespeicherte Version wiederhergestellt werden.

Aufgrund der Features mit der Farbe und Zeitleiste ist es hier einfach die gewünschten Annotationen zu finden und direkt anzuschauen. Dadurch, dass die Kameraposition und Zeitpunkte innerhalb der Animation direkt mit abgespeichert sind, erspart dies dem Benutzer einiges an Zeit, da dieser lediglich auf eine Taste drücken muss, um direkt zu der gespeicherten Position zurückzukehren, um dann dort weiterarbeiten zu können.

Als letzten Eintrag für jeden Punkt gibt es dann noch eine Taste, die ein neues Fenster öffnet. Diese ist in Abbildung 5.8 zu sehen. Dieses neue Fenster erlaubt es die entsprechende Annotation zu bearbeiten. Die hierbei möglichen Änderungen sind die Kameraposition, die genauen Koordinaten der Kugel im 3D Raum und die Notiz. Ebenso ist es möglich diese Änderungen in der JSON-Variable `json_obj` abzuspeichern. Dies hat keinen Einfluss auf die angezeigten Daten, da diese Variable lediglich im Hintergrund läuft und für das endgültige Speichern in die JSON-Datei zuständig ist. Dadurch dass Änderungen von Annotationen direkt umgesetzt werden, ist es hier nur mithilfe dieser Variable einen älteren Zustand wiederherzustellen. Somit ist es mithilfe der Taste *Restore Values from json_obj* möglich, den Zustand welcher in der Variable `json_obj` abgespeichert ist, zu laden und somit alle Änderungen an dieser Annotation seit dem letzten speichern rückgängig zu machen. Aufgrund dieser Funktionsweise, ist es nicht möglich Änderungen die bereits in `json_obj` abgespeichert wurden, rückgängig zu machen.

Um die Zuordnung des Bearbeitungs-Fensters und seiner entsprechenden Annotation garantiert werden kann, ist der Name der Annotation im Titel des Fensters dargestellt und es wird eine Linie zwischen dem Fenster und den Koordinaten, des mit der Annotation verknüpften Punktes im 3D Raum, gezeichnet.

Damit der Benutzer während dem auf und zu klappen der einzelnen Einträge, nicht immer das Fenster vergrößern oder verkleinern muss, gibt es ganz oben eine Box, in der ein Haken gesetzt werden kann, wenn gewünscht ist, dass sich dieses Fenster automatisch in seiner Größe verändert.

Ebenso gibt es hier noch eine weitere Box, mit der gesteuert wird, ob die Fenster welche in Abschnitt 5.2 angezeigt werden, einen durchsichtigen Hintergrund haben, oder nicht. Dies ist wichtig, da es zu Fällen kommen kann, in denen bereits ein starker Farbkontrast zwischen Daten und Hintergrund besteht. In diesen Fällen, kann es dann sehr wenige bis keine Farbkombinationen geben, mit denen der Text einfach erkennbar ist. Durch das Anzeigen des Hintergrunds in den Fenstern ist es nun möglich, die Titel und Texte der Annotationen einfach zu lesen. Sobald es wieder wichtiger wird, die Daten lesen zu können, kann dieser Hintergrund wieder auf dem selben Weg ausgeschaltet werden, wodurch die Annotationen immer noch sichtbar sind, wenn auch möglicherweise schwerer, jedoch verdecken sie nun die Daten nicht mehr.

5.5 Speichern und Importieren von Daten

Kein Programm dieser Art ist vollständig, ohne es dem Benutzer zu erlauben, die Annotationen in einer Form außerhalb des Programms zu speichern um diese später wiederverwenden zu können. Somit ist dies auch in meinem Programm möglich, in der Form von JSON-Dateien.

Alle Annotationen werden mithilfe von structs intern gespeichert, um die Anzahl an Variablen die global verfügbar sein müssen zu reduzieren. Somit gibt es einen Vektor, der alle diese Strukts von Annotationen speichert. Ebenso gibt es eine zweite Variable, welche alle wichtigen Informationen der Annotationen speichert. Diese ist in der Form von einem JSON-Objekt. Diese JSON Variable

wird immer dann aktualisiert, wenn entweder der Benutzer eine neue Annotation hinzufügt, oder wenn eine bearbeitete Annotation gespeichert werden soll. Denn während dem Bearbeiten von Annotationen werden die Änderungen direkt in den Structs gespeichert da diese für das Anzeigen aller darin gespeicherten Informationen zuständig sind.

Die JSON Variable ist hierbei dafür zuständig das Importieren und Exportieren der Daten zu ermöglichen. Dies geschieht indem der Benutzer auf die passende Taste *saveJsonToFile* drückt. Danach wird der aktuelle Zustand von allen Annotationen in die JSON Variable gespeichert und diese dann in eine JSON-Datei abgespeichert.

Für den Speicher-Pfad der JSON-Datei gibt es hier zwei Möglichkeiten. Als Basis-Pfad wird immer der Pfad zu dem importieren Projekt genommen und dann am Ende des Projektname ein *_annotation.json* angehängt. Sollte der Benutzer einen bestimmten Pfad bereits besitzen, so kann dieser in dem Parameter-Menü von MegaMol direkt eingegeben werden. Sobald dies geschehen ist, wird dieser für alle weiteren Verwendungen des Pfades benutzt.

Genauso können diese abgespeicherten Daten aus der JSON-Datei ausgelesen und wieder Importiert werden. Dies geschieht mithilfe der Taste *loadJsonFromFile*. Hierbei wird dann die JSON-Datei gesucht und wenn diese gefunden wurde, wird sie importiert. Bei diesem Import der Daten, werden alle bisher bestehenden Annotationen nicht beeinflusst. Somit kann es dazu kommen, dass wenn eine Datei zweimal importiert wurde, jeder Punkt doppelt in der Liste aller Annotationen aufzufinden ist. Dies kann dann jedoch mithilfe der in Abschnitt 5.4 beschrieben Lösch-Funktion gereinigt werden.

5.6 Verwendung des Moduls

Hinzufügen des Plugins

Damit das Plugin *annotations* verwendet werden kann, muss dieses zuerst in CMake auf aktiv gesetzt werden und anschließend muss MegaMol neu mithilfe von CMake gebaut werden. Danach kann MegaMol geöffnet werden und das gewünschte Projekt geladen werden. Anschließend muss in dem Konfigurator der AnnotationRenderer in die Pipeline direkt rechts an das View Modul eingebaut werden. Nun ist das Plugin verwendbar.

Laden von Annotationen

Für das Laden von Annotationen gibt es zwei Wege. Der Erste ist, keinen Pfad einzustellen, womit versucht wird, die Annotationen aus dem Standard-Pfad zu Laden. Der Zweite ist, einen Pfad anzugeben, was bei der Angabe keine weiteren direkten Auswirkungen hat, da dies nur ein setzten des Pfades ist. Sollen nun die Annotationen geladen werden, geschieht dies mit der Taste *loadJsonFromFile*. Dabei werden alle Annotationen in der Liste aller Annotationen am Ende dieser hinzugefügt.

Erstellen einer Annotation

In dem Parameter-Menü des AnnotationRenderers befindet sich ein Schalter *Add Annotation*. Dieser muss nun betätigt werden um das Fenster aus Abschnitt 5.1 anzuzeigen. Danach kann in das Erste Textfeld der Name der Annotation eingegeben werden. Dieser ist der Name mit dem die Annotation später in der Liste aller Annotationen angezeigt wird. Danach kann die eigentliche Notiz für die Annotation in das nächste Textfeld eingetragen werden. Danach kann man die Koordinaten die mit dieser Annotation verbunden werden soll, entweder direkt als Koordinaten in das nächste Feld eingeben, oder durch Drücken der Taste *Click in the viewport to add a new point* wird der Prozess der Auswahl gestartet. Anschließend kann mithilfe eines Links-Klicks mit der Maus eine Stelle in den Daten auf dem Bildschirm ausgewählt werden und diese werden dann gespeichert als eine Welt-Koordinate, welche in dem darüber liegendem Feld angezeigt wird. Dieser Prozess kann beliebig oft wiederholt werden, solange die Annotation noch nicht abgespeichert wurde. Danach kann die aktuelle Kamera-Position ermittelt und ebenfalls aufgezeichnet werden. Dies geschieht mit der Taste *Save current camera position*. Um die Koordinaten des vorherigen Schrittes zu überprüfen, kann nun mit der nächsten Taste eine Kugel an den aktuell festgelegten Koordinaten angezeigt werden. Als letzte Angabe, die gespeichert werden kann, kommen der Anfangs und End-Zeitpunkt der Animation des aktuellen Projektes zwischen denen die Annotation in den Daten angezeigt werden soll. Sobald alle Angaben beendet wurden, kann nun auf die letzte Taste gedrückt werden und die Annotation abgespeichert und zu der Liste aller Annotationen hinzugefügt werden.

Liste und Bearbeitung

Die Liste kann mithilfe des korrespondierenden Reglers in dem Parameter-Menü an und ausgeschaltet werden. Innerhalb der Liste kann dann ein Punkt ausgewählt werden und durch Drücken einer Taste die Kameraposition für diese Annotation wiederhergestellt werden. Dasselbe ist möglich für den Startzeitpunkt der Annotation, welcher auch gesetzt werden kann, indem die entsprechende Taste gedrückt wird. Das Fenster für die Bearbeitung der jeweiligen Annotation kann geöffnet werden, indem die letzte Taste die unter der entsprechenden Annotation existiert, gedrückt wird.

In diesem Bearbeitungsfenster können nun die Anmerkung, Kamera- und Punkt-Position im 3D Raum verändert werden. Nicht gespeicherte Änderungen können dann mithilfe der entsprechenden Taste wieder auf den zuletzt gespeicherten Zustand zurückgesetzt werden.

Speichern der Annotationen

Zum Speichern aller aktuellen Annotationen in eine JSON-Datei, kann die im Parameter-Menü liegende Taste *saveJsonToFile* gedrückt werden. Sollte ein Pfad bereits angegeben sein, dann werden die Annotationen in die angegebene Datei geschrieben. Ist jedoch kein Pfad angegeben, dann wird eine Datei erstellt, die unter dem gleichen Pfad wie das geladene Projekt aufzufinden ist.

5.7 Alternative Implementationen

In diesem Abschnitt möchte ich über verschiedene Methoden oder Überlegungen gehen, die ich im Laufe meiner Arbeit versucht hatte, welche aber entweder aus technischen oder optischen Gründen nicht in meinem Endresultat verwendet werden.

Text im 3D Raum

Die Erste Idee für das Anzeigen der Annotationen im Raum war, diese mithilfe von `SDFFont` direkt in die Daten zu malen. Zusätzlich wurde dieser Text auch in dem *Billboard Modus* gemalt, wodurch dieser immer zur Kamera hin-gezeigt hat.

Dieses direkte Zeichnen hatte sich jedoch schnell als problematisch herausgestellt. Hierbei gab es nun entweder die Möglichkeit, dass der Text der angezeigt wird immer im Vordergrund liegt und dabei andere Informationen auf dem Bildschirm verdeckt. Alternative wäre die Option gewesen, dass dieser Text mit einem Tiefen-Test behandelt wird, was wiederum dazu führte, dass der Text in dichten Datensätzen zum Großteil nicht mehr sichtbar war.

Als diese Probleme ersichtlich wurden, kam die Idee auf, auch diesen Teil des Programms mithilfe von ImGui darzustellen und somit kam die aktuelle Version wie sie in Abschnitt 5.2 beschrieben wurde zustande.

Zeitleisten

Die Zeitleisten in ihrer aktuellen Form besteht aus `ImGui::PlotLines()`. Als erste Idee für die Zeitleiste war jedoch der Gedanke aufgekommen, eine ASCII Zeichnung zu erstellen, welche aus 20 Zeichen bestand, wobei ein $ für einen Abschnitt stand, in dem die Annotation vorhanden ist und ein ‾ dafür stand, dass keine Annotationen vorhanden sind. Ein Vergleich dieser Zeitleiste mit der aktuellen Version ist in Abbildung 5.9 zu sehen.

Als Zweites, gabe es die Überlegung eine Variante des `slider` zu verwenden, welche zwei Punkte hat, die miteinander Verbunden sind, womit die Anfangs und Endpunkte, sowie der Zeitraum dazwischen einfach sichtbar war. Hierzu wurden Ideen aus dem GitHub von ImGui verwendet [RangeSlider]. Diese hatten sich jedoch als nicht nützlich herausgestellt, da es an verschiedenen Stellen Probleme gab, die ich nicht lösen konnte. Für eines der Hauptprobleme hat sich später erst herauskristallisiert, warum dieses vermutlich der Fall war.

Daraufhin kam dann die Idee, welche zu der aktuellen Version der Zeitleiste geführt hatte. Bei der Definition der Funktion `ImGui::PlotLines()` steht jedoch dabei, dass empfohlen wird die Funktion `ImPlot::PlotLines()` zu verwenden. Diese erlauben mehr Optionen beim Erstellen der Graphen und auch größere Freiheiten, für das Verwenden von Funktionen zum Zeichnen innerhalb der `PlotLines`.. `ImPlot` hat sich hierbei aber als nicht hilfreich herausgestellt, denn jegliche Form von Graphen die hiermit gezeichnet werden, beinhalten ein `Begin()` und `End()`. Diese führen aber bei meiner Implementation der einzelnen Zeilen in der Liste aller Annotationen zu mehreren Problemen, wodurch diese nicht verwendet werden konnten. Das Hauptproblem hierbei ist, dass wenn immer ein `Begin()` und `End()` benötigt wird um etwas anzuzeigen, werden alle dadurch gemalten Objekte immer ausschließlich in der ersten Zeile angezeigt. Somit sind alle anderen Zeilen frei von Zeitleisten, was

Abbildung 5.9: Vergleich ASCII Zeitline und aktuelle Zeitleiste. In dem linken Bild ist die ASCII Zeitleiste zu sehen, welche nur aus 20 Zeichen bestand. Rechts ist die aktuell verwendete Zeitleiste zu sehen, welche aus 100 Teilen besteht.

den Sinn deren Anzeige zunichtemacht. Der einzige Fall wo diese Zeitleisten doch in den korrekten Zeilen gemalt wurden war, wenn entweder in der Liste die entsprechende oder die darüber liegenden Zeile geöffnet wurde.

Bearbeitung von bereits existierenden Annotationen innerhalb der Liste

Zuerst war die Überlegung, dass Punkte direkt innerhalb der Liste aller Annotationen bearbeitet werden können. Dies wurde später durch die aktuelle Version, in der diese Bearbeitung in einem eigenem Fenster stattfindet ersetzt.

Die Grundidee hinter diesem Ansatz war, dass somit alles was die Punkte angeht direkt in einem einzelnen Fenster war und somit nicht der ganze Bildschirm mit weiteren Fenstern vollgefüllt werden würde. Eines der Probleme, die hierbei aufkamen, war, dass mit diesem Ansatz es sich als schwieriger herausstellte, die Verbindung zwischen der aktuellen zu bearbeiteten Annotation und deren Position im 3D Raum in einem direkten und leicht ersichtlichen Rahmen anzuzeigen. Weitere Probleme kamen daher, dass wenn mehrere Annotationen miteinander verändert werden sollten und diese innerhalb der Liste nicht nahe beieinander lagen, es zu einem Mehraufwand für den Benutzer führte. Dieser musste, nun entweder die Liste sehr groß machen, was einen Großteil des Bildschirmes verdecken würde oder immer in der Liste hoch und runter navigieren um zwischen den relevanten Annotationen zu springen. Diese Probleme wurde durch das Einführen von eigenen Bearbeitungsfenstern für in der Liste ausgewählte Annotationen beseitigt, da es nun möglich war die Korrelation zwischen Annotation und ihren Koordinaten durch eine Linie zu sehen. Ebenso können die Fenster beliebig angeordnet werden, sodass mehrere gleichzeitig sichtbar sind und somit das Navigieren zwischen diesen einfacher ist.

6 Resultate

6.1 Überblick

Mein Programm erlaubt das Erstellen von Anmerkungen, innerhalb von MegaMol, in der Form von Annotationen direkt in dem Raum der dreidimensionalen Daten. Dies verbindet das bereits mögliche Erkunden von Daten innerhalb von MegaMol mit dem Festhalten von Anmerkungen direkt innerhalb der Daten, ohne hierbei externe Programme verwenden zu müssen.

Das Erstellen von neuen Annotationen ist hier sehr einfach möglich. Sie werden, wie in Abschnitt 5.1 beschrieben, durch das Ausfüllen der Textfelder für Namen und Anmerkung zusammen mit dem Festhalten des aktuellen Zustandes des Datensatzes mit wenigen Tastendrücken erstellt. Somit können Anmerkungen und Gedanken direkt aufgeschrieben werden, ohne dass hierbei viel Zeit verloren geht.

Es gibt die Möglichkeit, dass alle Annotationen die innerhalb der Simulation zu dem aktuellen Zeitpunkt vorhanden und nicht verdeckt sind, anzuzeigen. Dies erlaubt das Erkunden aller Annotationen sowohl im Räumlichen als auch im Zeitlichen. Damit es hier zu keinen Überlappungen der angezeigten Fenster kommt, wurde ein einfaches Force-Direkted-Layout implementiert, welches alle überlappenden Fenster von Annotationen so verschiebt, dass diese frei im Raum stehen. Um die Verbindung zwischen den nun verschobenen Fenstern und den jeweiligen Punkten im Raum nicht zu verlieren, wird eine Linie gezeichnet die diese verbinden.

Des Weiteren ist es möglich einen Überblick über alle Annotationen zu bekommen, da diese in einer Liste zusammengefasst werden, in der auch die aktuelle Sichtbarkeit aller Annotationen dargestellt wird. Innerhalb von diesem Fenster ist es möglich den vorher abgespeicherten Zustand der Simulation wiederherzustellen. Ebenso erlaubt es das Löschen und Bearbeiten von Annotationen. Das letztere wird hierbei in einem neuen Fenster erledigt, um die Liste der Annotationen nicht mit zu vielen Informationen zu füllen, wodurch dann der Überblick verloren gehen würde.

Als Letztes besteht die Option, die Annotationen mithilfe einer JSON-Datei abzuspeichern und dann zu einem späteren Zeitpunkt wieder zu importieren. Dies erlaubt es, dass ein Benutzer die Annotationen anfängt und Andere dann an diesen weiterarbeiten, indem neue Annotationen hinzugefügt oder bereits vorhandene geändert werden.

6.2 Beispiele

Eine der wichtigsten Funktionen für die Verwendung dieses Moduls ist, dass der Überblick über alle Annotationen immer gewahrt werden kann. Hierfür zeigt Abbildung 6.1 ein Beispiel, wie die Liste aller Annotationen aussehen kann, wenn in dieser 31 Annotationen abgespeichert sind. Diese wurden von einem Skript erstellt, welches zufällige Titel und Zeiträume für diese Annotationen

Abbildung 6.1: Liste aller Annotationen mit 31 gespeicherten Annotationen.

generiert hat. Wie in diesem Bild gut erkennbar ist, kann mithilfe der Spalte für die Sichtbarkeit, hier *Visibility* genannt, direkt gesehen werden, welche Annotationen aktuell sichtbar und welche durch Objekte verdeckt sind. Ebenso zeigt es den jeweiligen Zeitraum, in dem die Annotation überhaupt sichtbar sein kann in der nächsten Spalte. Somit kann sich der Benutzer leicht einen Überblick über die Zeiträume der Annotationen verschaffen, ohne die Animation der Daten laufen zu lassen.

Bisherige Abbildungen in meiner Arbeit zeigten lediglich die einzelnen Fenster in verschieden Zuständen während dem Gebrauch. Um über das Gesamtprodukt meiner Arbeit einen besseren Überblick zu verleihen, werde ich im Folgenden einen gröberen Überblick geben.

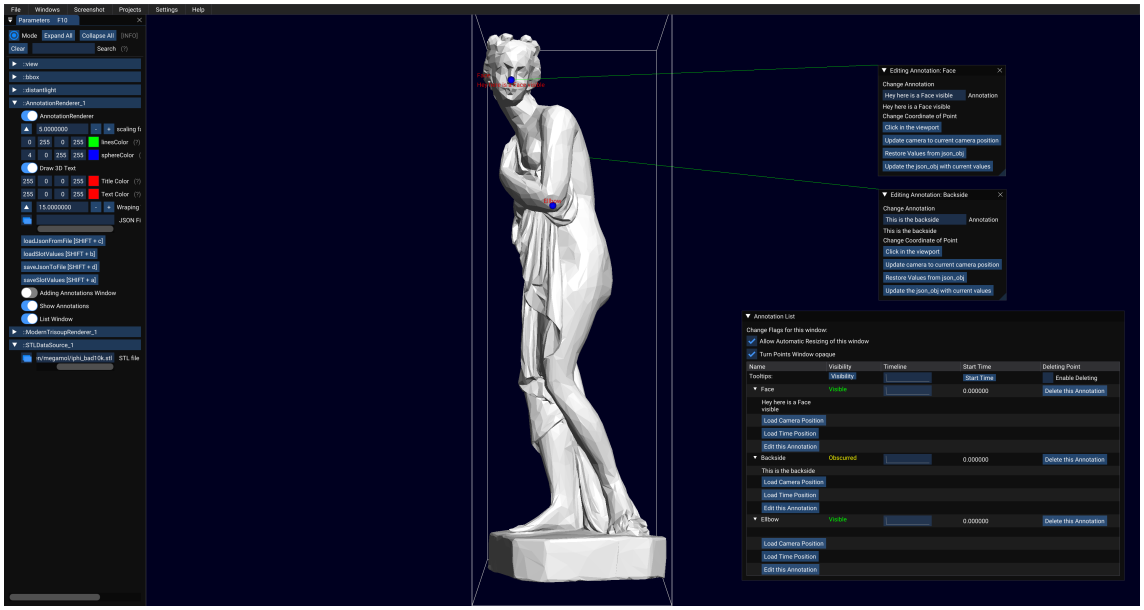
In Abbildung 6.2a ist ein Bildschirmfoto der gesamten MegaMol Anwendung zu sehen. In diesem Bild ist das geladene Projekt ein solches, das mithilfe von STL Dateien eine Oberfläche darstellt. In dem verwendeten Datensatz war dies eine Figur von einem Menschen. Die Abbildung 6.2b zeigt einen Ausschnitt des Bildschirmes, in dem zwei Annotationen zu sehen sind. Hierbei ist erkennbar, dass diese Annotationen aufgrund der unterschiedlichen Schattierung des Gesichtes teilweise schwer zu lesen sind. Dies ist der Fall, obwohl bereits eine Farbe gewählt wurde, welche den Text am deutlichsten sichtbar macht.

Hierbei ist besonders auffällig, warum es sinnvoll ist, dass es die Möglichkeit gibt, die Fenster der Annotationen mit einem Hintergrund anzuzeigen, da ohne diese Option der starke Kontrast zwischen dem dunklen Hintergrund und der hellen Figur, diese selbst mit verschiedenen Farben schwer zu erkennen sind. Durch den nicht durchsichtigen Hintergrund in Abbildung 6.2c, ist es nun möglich diese ganz einfach zu lesen und wenn dies erledigt ist, kann ganz einfach der Hintergrund der Fenster wieder ausgeschaltet werden und die Daten in der Figur wieder betrachtet werden.

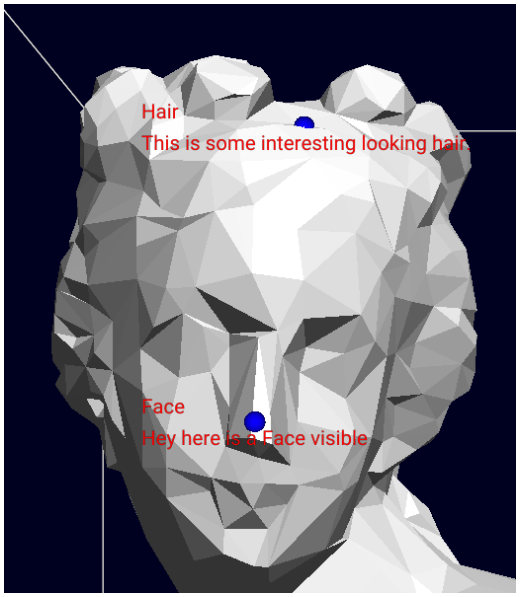
6.3 Limitierungen

Somit gibt es bereits einige Optionen und Funktionen, die verwendet werden können, um Annotationen in einem 3D Datensatz zu Platzieren und Betrachten. Jedoch gibt es auch hier an verschiedenen Stellen Beschränkungen, die meistens aufgrund der Art und Weise wie die jeweiligen Funktionen implementiert wurden, auftreten. Am auffälligsten ist dies bei der Funktion, welche die Annotationen im Fall von Überlappungen verschiebt. Hierbei werden diese immer entlang eines Vektors verschoben, der zwischen den beiden Fenster-Mittelpunkten aufgespannt wurde. Hierbei wird beachtet, dass die Fenster immer voneinander weg geschoben werden, da sonst nicht der gewünschte Effekt erzielt werden würde. Dies funktioniert bei kleinen Mengen an Annotationen die in der gleichen Region des Bildschirmes zu sehen sind recht gut. Sollten jedoch auf einmal sehr viele Annotationen die gleiche Region teilen, dann wird dies eher zu einem Problem. Dieses kommt daher dass nun die Fenster der Annotationen sich gegenseitig wegschieben, aber hierbei kann es nun vorkommen, dass ein Fenster von mehreren Fenstern in gegen gesetzte Richtungen verschoben wird, womit es sich im großen und ganzen sehr wenig bis gar nicht von der Ausgangsposition verschiebt.

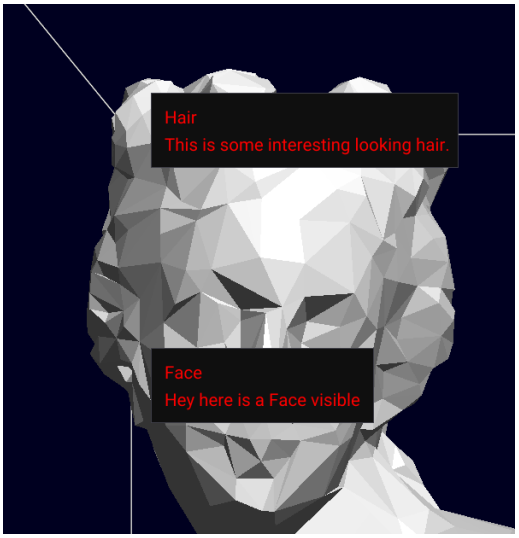
In Abbildung 6.3 ist zu sehen, wie es aussieht, wenn ungefähr 50 Annotationen auf einmal angezeigt werden, die alle unterschiedlich große Anmerkungen und Titel haben. Diese Annotationen wurden mithilfe eines Zufallsgenerators erstellt und im Raum Platziert. In Abbildung 6.4 ist zu sehen, wie diese selben Annotationen aussehen würden, wenn die Verschiebung von Annotationen nicht geschehen würde. Hier ist es sehr schwer auch nur einen Teil der Annotationen zu lesen. Somit



(a) Gesamte Anwendung



(b) Annotationen ohne Hintergrund



(c) Annotationen mit Hintergrund

Abbildung 6.2: Bildschirmfotos von möglichen Zuständen der Annotationen innerhalb von einem STL Datensatz.

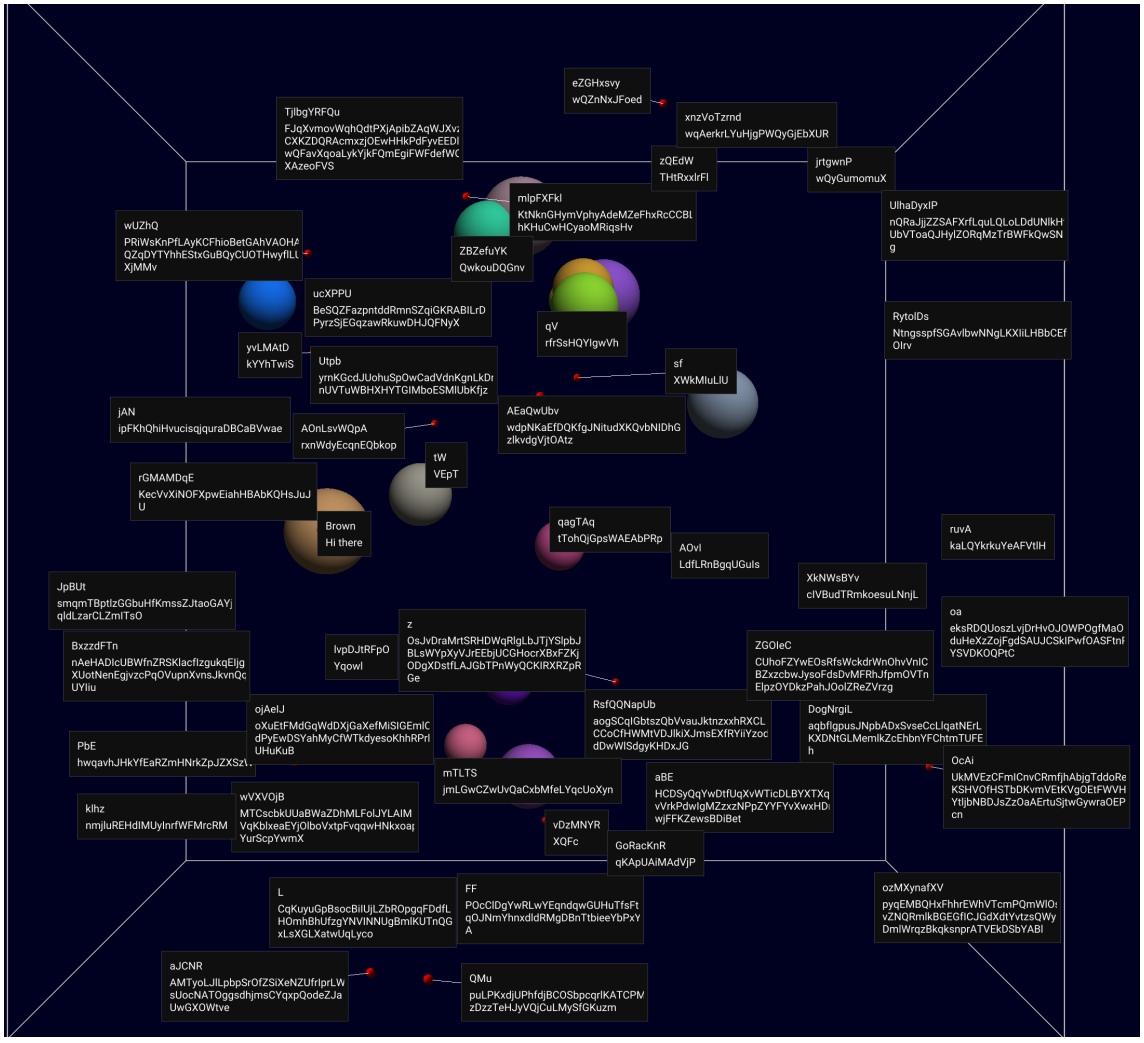


Abbildung 6.3: Hier werden 50 Annotationen angezeigt, welche mithilfe einer Funktion voneinander abgestoßen werden. Dies funktioniert in einem Großteil der Fälle, womit nur wenige Annotationen an gewissen Stellen übereinander liegen.

hat der verwendete Algorithmus zwar gewisse Beschränkungen unter welchen er nicht mehr optimal funktioniert, aber ohne diesen würde das Anzeigen der Annotationen um einiges schlechter aussehen.

Einen weiteren problematischen Effekt, der durch diese Herangehensweise an die Verschiebung entsteht, ist dass es durchaus vorkommen kann, dass bei dem bewegen der Kamera, die Fenster hin- und her-springen können. Somit kann es vorkommen, dass bei vielen Annotationen diese nicht lesbar sind, wenn der Benutzer schnelle Bewegungen innerhalb der Daten ausübt. Sobald aber diese Bewegungen vorbei sind, beruhigen sich die Fenster sofort, da es hier in jedem Frame zu gleichen Ergebnissen der Verschiebung kommt, solange keine Änderungen an der Sichtweise vorgenommen werden. Dies alles ist jedoch erst dann der Fall, wenn viele Annotationen nahe beieinander liegen. Sind alle Annotationen jedoch verteilt über Raum und Zeit der Daten, so ist dieser Fall eher seltener vorhanden.

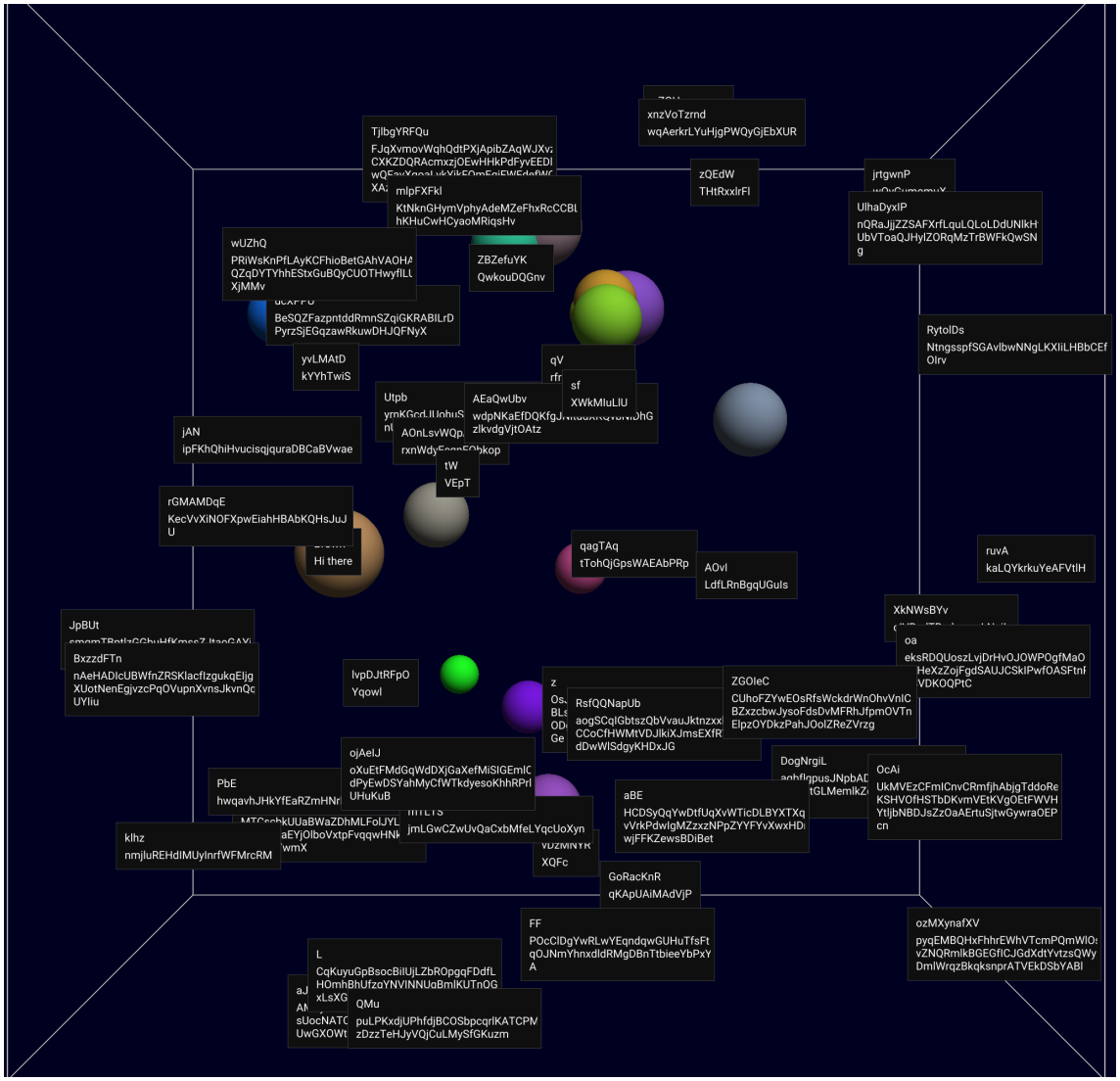
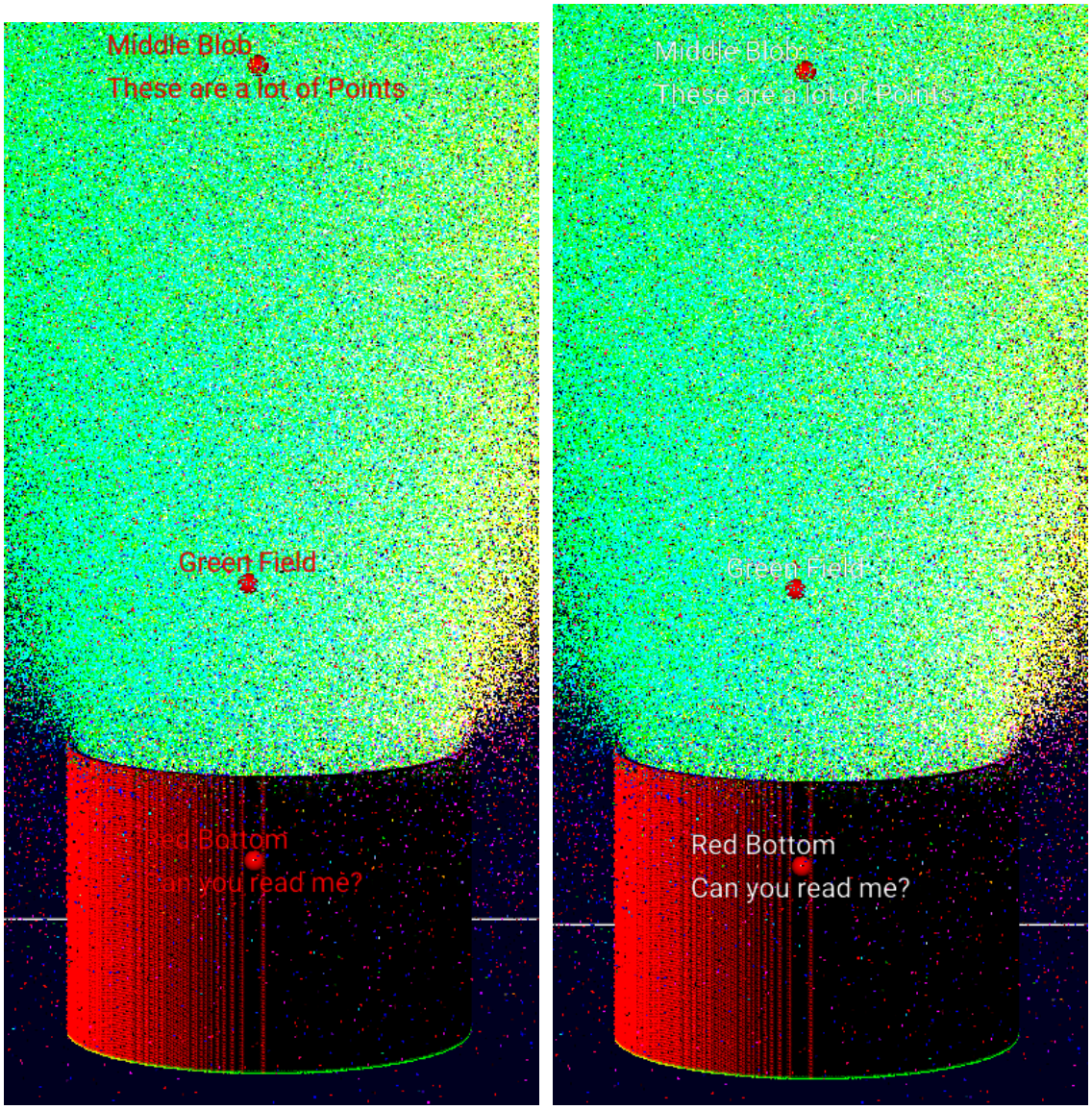


Abbildung 6.4: Hier werden dieselben 50 Annotationen von Abbildung 6.3 angezeigt, hierbei werden diese jedoch direkt über den umgerechneten Bildschirm-Koordinaten der jeweiligen Punkten gezeichnet und nicht verschoben. Somit gibt es eine Vielzahl an Überlappungen was einen Großteil der Annotationen nicht erkennbar macht.

Ein weiteres Problem, welches ebenfalls mit der Darstellung der Annotationen zusammenhängt, ist das Färben der Texte. Hiermit ist gemeint, dass es vorkommen kann, dass ein Datensatz mehrere verschiedene Farben über die gesamte Darstellung verteilt besitzt. Somit wird es schwierig, hier eine optimale Farbe für die Titel und Texte aller Annotationen auf einmal zu finden. Ein Beispiel hierfür ist dargestellt in Abbildung 6.5. Diese zwei Bilder unterscheiden sich lediglich in der Farbe der Texte der angezeigten Annotationen. Abbildung 6.5a hat einen roten Text und Abbildung 6.5b einen weißen Text. Im Abbildung 6.5a sind die meisten Annotationen gut sichtbar, da diese sich auf einem hell-grünen Hintergrund befinden. Jedoch ist die Annotation, welche auf der roten Fläche liegt, eben auf einem roten Hintergrund, weshalb es hier sehr schwierig ist, diese passend zu lesen.



(a) Roter Text **(b)** Weißer Text

Abbildung 6.5: Bilder mit verschiedenen Text-Farben. In (a) sind die zwei oberen Annotationen relativ leicht zu Erkennen, nicht jedoch die untere, welche die gleiche Farbe wie der Hintergrund hat. In (b) ist die untere Annotation nun leichter zu lesen, während die oberen Annotationen beinahe unsichtbar sind.

Im rechten Bild ist dieses Problem genau umgekehrt. Hier sind die Annotationen auf dem hellen Hintergrund sehr schwer zu erkennen, während die Annotation im roten Bereich nun deutlicher hervorsticht.

Als Lösung gibt es in diesen Fällen aktuell lediglich die Option der Nicht-durchsichtigen Fenster für die Annotationen zu verwenden, da hierbei der Text auf dem dunklen Hintergrund der Fenster dann gut zu lesen ist. Dies verdeckt jedoch wiederum die Daten, welche eigentlich betrachtet werden sollen, was auch wieder zu einem Problem wird. Diese Problem ist jedoch nur so lange vorhanden, wie der Benutzer die Fenster mit Hintergrund aktiv hat.

Das Laden der Startzeitpunkte für Annotationen, benutzt einen festen String. Somit wird das Laden nur funktionieren, solange das *View Modul* den Namen *view* besitzt. Sollte das nicht der Fall sein, dann kann der Befehl, welcher aufgerufen wird, nicht die Animation auf den gewünschten Frame setzen. Diese Funktion wird jedoch wieder korrekt arbeiten, sobald das *View Modul* den Namen *view* bekommt. Somit kann dieser Fehler bereits durch eine kleine Änderung an dem Projekt behoben werden.

7 Zusammenfassung

In dieser Arbeit wurde ein Ansatz implementiert, welcher das manuelle Erstellen von Annotationen in MegaMol erlaubt. Dies ermöglicht es, das in MegaMol bereits mögliche Erkunden von Daten, welche mit verschiedensten Funktionen dargestellt werden, so zu erweitern, dass Anmerkungen an diese Daten direkt innerhalb von MegaMol notiert werden können. Somit wird verhindert, dass diese Anmerkungen in getrennten Dateien oder Programmen erstellt werden müssen, was ansonsten den Workflow der Erkundung behindern oder verlangsamen würde. Dadurch dass die Annotationen sowohl als Liste, als auch innerhalb der Daten in der Form von Fenstern angezeigt werden, kann immer der Überblick über alle vorhandenen Annotationen bewahrt werden. Die Abspeicherung der Zeitpunkte der Animation, an welcher die Annotation erstellt wurde, sowie die jeweilige Kameraposition, erlaubt es dass für jede Annotation der Zustand der Daten zum Zeitpunkt der Erstellung direkt wieder hergestellt werden kann. Aufgrund der Funktion die es erlaubt, Annotationen in der Form von JSON-Dateien zu exportieren, gibt es die Möglichkeit, dass mehrere Benutzer an einem Datensatz arbeiten und alle die Annotationen der Anderen betrachten und bearbeiten können.

8 Mögliche Weiterentwicklungen

In diesem Kapitel gebe ich mehrere Ideen für mögliche Änderungen oder zusätzliche Funktionen die verwendet werden können, um das Modul zu verbessern.

Richtungspfeile

Hierbei war die Idee, Pfeile anzuzeigen, die in die Richtung von verdeckten Annotationen deuten. Dies würde es dem Benutzer erlauben, auch ohne Verwendung der Liste, alle zum aktuellen Zeitpunkt der Simulation bestehenden Annotationen zu finden. Somit wäre dies eine weitere Methode einen besseren Überblick über die Annotationen zu gewährleisten.

Eine Methode einer möglichen Implementation hierfür wäre, dass mithilfe der Kamerapositionen eine Richtung für alle aktuell verdeckten Punkte zu bestimmen und dann die Pfeile in Richtung der Kameraposition zeigen zu lassen. Diese Pfeile würden am Bildrand positioniert sein, damit sie nicht die Daten überdecken. Für die Pfeile gäbe es dann auch noch die Option, dass diese entweder für jede einzelne Annotation existieren, oder es existieren nur vier Pfeile die vier Richtungen des Bildschirmes zeigen, also nach oben, unten, rechts und links. Diese würden dann auf alle Annotationen in die jeweilige Richtung hinweisen und nehmen somit weniger Platz auf dem Bildschirm in Anspruch. Ebenso können hierbei dann Zahlen in die Pfeile eingefügt werden, die angeben, wie viele Annotationen in dieser Richtung vorhanden sind.

Die Implementation dieser Funktion kann vermutlich relativ leicht möglich sein, für alle Fälle, in denen die Bounding Box in der Mitte des Bildschirmes ist. Jedoch könnte es schwieriger werden, wenn diese Bounding Box verschoben wird und somit Pfeile in “Drehrichtung” nicht mehr korrekt ist, da nun alle Pfeile in eine Richtung zeigen.

Eine Idee für die Quadranten wäre, den Bildschirm in vier Dreiecke aufzuteilen. Diese würden von den Ecken des Fensters zur Mitte hin aufgespannt werden. Anschließend kann für jeden Punkt getestet werden, in welchem dieser Dreiecke er aktuell liegt und dann zu den jeweiligen Pfeilen hinzugefügt werden.

Automatisches Platzieren

Diese Idee basiert auf der Tatsache, dass es bereits Algorithmen gibt, die für gewisse Datensätze es erlauben, automatisch Markierungen in 3D Datensätzen zu erstellen [BF14]. Meine Implementation erlaubt bisher nur das manuelle Platzieren von Annotationen. Aufgrund der Import-Funktion, können jedoch bereits Annotationen von verschiedenen Benutzern geladen werden. Somit wäre es möglich, dass ein Algorithmus der Annotationen oder Markierungen automatisch erstellt und

platziert, eine Datei erstellt, welche von meinem Modul geladen werden kann. Dies würde dann das Arbeiten mit diesen Annotationen gleichstellen, mit Annotationen die von anderen Benutzern erstellt wurden.

Auswahl von Regionen

Hiermit ist gemeint, dass der Benutzer die Möglichkeit hat, eine Annotation nicht nur wie bisher mit einem Punkt im 3D Raum zu verknüpfen, sondern auch als alternative Option, eine Region auf dem Bildschirm auswählen zu können und diese dann mit einer Annotation zu versehen. Somit können ganze Regionen beschrieben werden und nicht nur einzelne Punkte.

Für eine Implementation dieser Idee wäre es möglich, dass diese Region entweder nur im Screen-Space in der Form von ImGui Fenstern besteht, die jedoch nur sichtbar sind, wenn aus der korrekten Richtung geschaut wird. Alternativ hierzu gäbe es auch die Option dass die Region auf die Oberfläche der Bounding-Box gelegt wird und somit auch sichtbar ist, wenn der Benutzer nicht direkt aus dem gleichen Blickwinkel, wie der mit dem diese Region erstellt wurde, schaut.

Exportieren in einer Markdown Datei

Die Idee hierbei ist, es dem Benutzer zu ermöglichen, eine Datei zu bekommen, die das Betrachten aller Annotationen erlaubt, ohne MegaMol verwenden zu müssen. Hierfür werden dann nur der Titel und die Notiz der jeweiligen Annotationen, zusammen mit einem Bildschirm-Foto, welches die Ansicht des Datensatzes von der Kameraposition aus darstellt, gespeichert. Somit ist es möglich eine statische Version aller Annotationen zu bekommen die ohne MegaMol verwendbar ist.

Verbesserte Algorithmen für die Überlappung von Fenstern

Aktuell wird in dem Fall dass sich Fenster von angezeigten Annotationen überlappen, ein Algorithmus angewandt, der diese Fenster voneinander so wegschiebt, dass sie im Bezug auf die jeweilige Verschiebung gesehen, den kürzesten Weg hierbei zurücklegen. Sollte sich dieser Weg plötzlich in eine andere Ecke verlegen, dann werden die Fenster von einem Frame zum nächsten abrupt in eine ganz andere Richtung verschoben, was den Benutzer möglicherweise irritieren könnte. Ebenso ist der aktuelle Algorithmus nicht wirklich optimiert, außer in der Ansicht, dass jedes Fenster nur einmal pro Durchlauf mit jedem anderen Fenster verglichen wird. Jedoch hat der gesamte Algorithmus immer noch eine kubische Laufzeit, was bei vielen Annotationen zu einer starken Verlangsamung der Simulation führen kann. Hierbei wäre es somit praktisch, einen anderen Algorithmus zu finden der diese Optimierungen in Betracht zieht und implementiert.

Eigene Farbe für jede Annotation

Hierbei ist die Idee, dass jede Annotation die Möglichkeit hat, eine eigen Farbe zu bekommen, womit die Probleme in Abbildung 6.5 verringert werden könnten. Die Schwierigkeit einer Lösung liegt vermutlich darin, dass es so implementiert wird, dass der Benutzer nicht durch das ständige Angeben von Farben in seiner Arbeitsweise eingeschränkt wird. Hierfür wäre die Möglichkeit gegeben, dass die globalen Einstellungen der Farben weiterhin bestehen bleiben, es jedoch die Möglichkeit gibt, mithilfe einer Taste, auf eine andere Einstellung zu springen, welche für jeden Punkt einzeln verändert werden kann.

Diese zweigeteilte Anwendung hätte von Vorteil, dass der Benutzer nur dann die Farben einzeln anpassen muss, wenn dies auch wirklich notwendig ist. Somit könnte einiges an Zeit gespart werden, welche ansonsten Nötig wäre, wenn es nur die Option gäbe, für jede Annotation die Farben einzeln einzurichten.

Die gleichen Probleme können auch bei den Farben für die Kugeln, welche die Koordinaten der Annotationen darstellen, entstehen. Somit wäre es auch bei diesen hilfreich, wenn es die Möglichkeit gäbe, die Farben individuell zu steuern.

Annotationen über Server senden

Dies baut auf der Idee in [EG04] auf, welche davon handelt, Annotationen innerhalb eines Netzwerkes zu speichern, womit jeder auf diese Annotationen zugreifen kann, ohne dass manuell Dateien übergeben werden müssen, wie es aktuell in meiner Version der Fall wäre. Somit würde es auch möglich sein, dass mehrere Personen gleichzeitig an einem Datensatz arbeiten und gemeinsam die Annotationen erstellen und am Ende hat jeder die Annotationen des anderen direkt zur Verfügung. In meiner Implementation wäre es vermutlich möglich, die Annotation-Dateien in einem Netzwerk so abzuspeichern, dass alle darauf zugreifen könnten und dann direkt in diese gemeinsame Datei schreiben und auch lesen können.

Literaturverzeichnis

[BF14] A. Boyko, T. Funkhouser. „Cheaper by the Dozen: Group Annotation of 3D Data“. In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*. UIST '14. Honolulu, Hawaii, USA: Association for Computing Machinery, 2014, S. 33–42. ISBN: 9781450330695. DOI: 10.1145/2642918.2647418. URL: https://doi.org/10.1145/2642918.2647418 (zitiert auf S. 11, 43).

[BFH01] B. Bell, S. Feiner, T. Höllerer. „View Management for Virtual and Augmented Reality“. In: *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*. UIST '01. Orlando, Florida: Association for Computing Machinery, 2001, S. 101–110. ISBN: 158113438X. DOI: 10.1145/502348.502363. URL: https://doi.org/10.1145/502348.502363 (zitiert auf S. 11).

[ČB10] L. Čmolík, J. Bittner. „Layout-aware optimization for interactive labeling of 3D models“. In: *Computers & Graphics* 34.4 (2010). Procedural Methods in Computer Graphics Illustrative Visualization, S. 378–387. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2010.05.002. URL: https://www.sciencedirect.com/science/article/pii/S0097849310000695 (zitiert auf S. 11).

[CG08] G. Cipriano, M. Gleicher. „Text Scaffolds for Effective Surface Labeling“. In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), S. 1675–1682. DOI: 10.1109/TVCG.2008.168 (zitiert auf S. 11).

[Dear ImGui] *Dear ImGui*. URL: https://github.com/ocornut/imgui (zitiert auf S. 13).

[EG04] S. E. Ellis, D. P. Groth. „A Collaborative Annotation System for Data Visualization“. In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI '04. Gallipoli, Italy: Association for Computing Machinery, 2004, S. 411–414. ISBN: 1581138679. DOI: 10.1145/989863.989938. URL: https://doi.org/10.1145/989863.989938 (zitiert auf S. 45).

[GBB+19] P. Gralka, M. Becher, M. Braun, F. Frieß, C. Müller, T. Rau, K. Schatz, C. Schulz, M. Krone, G. Reina, T. Ertl. „MegaMol – a comprehensive prototyping framework for visualizations“. In: *The European Physical Journal Special Topics* 227.14 (März 2019), S. 1817–1829. ISSN: 1951-6401. DOI: 10.1140/epjst/e2019-800167-5 (zitiert auf S. 13).

[KIK+21] D. Kouřil, T. Isenberg, B. Kozlíková, M. Meyer, M. E. Gröller, I. Viola. „HyperLabels: Browsing of Dense and Hierarchical Molecular 3D Models“. In: *IEEE Transactions on Visualization and Computer Graphics* 27.8 (2021), S. 3493–3504. DOI: 10.1109/TVCG.2020.2975583 (zitiert auf S. 11).

[RangeSlider] *ImGui: Range Slider, Issue 76*. URL: https://github.com/ocornut/imgui/issues/76 (zitiert auf S. 30).

[SFB] *Auflösung des SFB 716 und letzte Mitgliederversammlung. 2018.* URL: https://www.sfb716.uni-stuttgart.de/news/SFB-716-ends-on-31th-December-2018/ (zitiert auf S. 13).

Alle URLs wurden zuletzt am 21. 05. 2023 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Druck-Exemplaren überein.

Ort, Datum, Unterschrift