Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3488

# Interaktive Hierarchievisualisierung auf großen, hochauflösenden Displays

Hansjörg Schmauder

| | |
|---|---|
| **Studiengang:** | Softwaretechnik |
| **Prüfer/in:** | Prof. Dr. Daniel Weiskopf |
| **Betreuer/in:** | Dr. rer. nat. Michael Burch,<br>Dipl.-Inf. Christoph Müller |
| **Beginn am:** | 15. Mai 2013 |
| **Beendet am:** | 14. November 2013 |
| **CR-Nummer:** | C.2.4, E.1, G.1.2, G.2.2, H.5.2, I.3.2, I.3.5, I.3.6 |

## Kurzfassung

Moderne Anzeigegeräte bieten auf großen Flächen hohe Auflösungen, die der von gewöhnlichen Monitoren nahe kommen. Die VISUS-Powerwall ist ein Beispiel hierfür mit einer Anzeigegröße von 5,90 $\times$ 2,24 Metern, sodass auch die kollaborative Verwendung durch mehrere Benutzer gleichzeitig möglich ist. Da die Visualisierung großer Datenmengen aufgrund des beschränkten Platzes eine große Herausforderung in der Informationsvisualisierung darstellt, bieten diese Displays Vorteile für solche Rendering-Vorhaben. Für die Darstellung der hierarchischen Datensätze mit Größenordnungen von hunderttausenden Knoten wurden die fraktalbasierten „verallgemeinerten Pythagoras-Bäume“verwendet. Die Kombination von Tablet-Computern mit einem System zum optischen Tracking ermöglicht neuartige, interaktive Features, die zusammen mit dem großen Display auch bei der Kollaboration von Benutzern eingesetzt werden können.

In dieser Arbeit werden diese Technologien verknüpft und das Potential sowie die Schwierigkeiten des komplexen Systemaufbaus vorgestellt. Die entstandene Software wird außerdem anhand zweier Fallbeispiele studiert und die Ergebnisse präsentiert, wobei Zukunftsempfehlungen für derartige Vorhaben gemacht werden.

## Abstract

State of the art displays provide high resolutions comparable to the resolution known from common monitors, but on larger surfaces thus enabling co-located cooperative work. The powerwall at VISUS is such a display with an output area of 5.90 $\times$ 2.24 meters. As the visualization of large amounts of data is a challenge in information visualization due to limited screen space, these displays can mitigate scalability issues when rendering big data. In this thesis, fractal-based “generalized Pythagoras trees” are applied to hierarchical data sets containing hundreds of thousands of nodes. Also, the use of tablet PCs is combined with an optical tracking system, allowing for novel interactive features. They also allow the users to work collaboratively and share their findings using the large display.

This thesis attempts to simultaneously use the potential of all these modern technologies and also presents the drawbacks and difficulties when developing software for such a complex system. The implemented software is tested with two case studies and their results are presented. The document concludes with an outlook on further work.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis



# Verzeichnis der Listings



# Verzeichnis der Algorithmen

# 1. Einleitung

## 1.1. Motivation

In der Informationsvisualisierung beschäftigt man sich generell mit abstrakten Daten, die häufig diskreter Natur sind. Die verwendeten Visualisierungstechniken benutzen meistens Standard-Displays als Anzeigemedium. Hierunter werden Bildschirme verstanden, wie sie bei gewöhnlichen Desktop-Computern verwendet werden. In der Forschung zur Informationsvisualisierung werden schon Aufbauten mit zwei HD-fähigen Monitoren als „große Displays“ bezeichnet. Geht es jedoch um die Visualisierung großer, abstrakter Datenbestände, stößt man mit einem solchen Display schnell auf das Problem der visuellen Skalierbarkeit, d. h. ab einer bestimmten Datenmenge muss man mittels Aggregation und Overplotting eine geeignete Übersichtsdarstellung über eine möglichst große Datenmenge bekommen. Das Einsehen detaillierterer Informationen wird dann – sofern überhaupt vorgesehen – in der Regel durch Interaktion ermöglicht.

Mit Hilfe von großen, hochauflösenden Displays wie etwa einer Powerwall lässt sich das Problem der visuellen Skalierbarkeit etwas entschärfen, da mehr Platz und eine höhere Auflösung zur Verfügung steht, um die grafischen Repräsentationen der Datenelemente darzustellen. Selbstverständlich kann es sich hierbei aber nicht um eine tatsächliche Lösung handeln, da ein entsprechend noch größerer Datensatz auch diese Größen zu sprengen vermag. Solche Datenmengen existieren zuhauf, und es ist extrapolierend aus den Entwicklungen der Vergangenheit zu erwarten, dass sich die Datensätze bereits in naher Zukunft in einer noch höheren Größenordnung bewegen. Daher müssen auch für ein Powerwall-System geeignete Interaktionen zur Verfügung gestellt werden. Aufgrund der Eigenheiten des Displays tut sich hier ein eigener Strang des Forschungsgebietes der Mensch-Rechner-Interaktion auf, da die interaktive Steuerung eines solchen Systems unter maßgeblich anderen Voraussetzungen stattfindet als bei klassischen Desktop-, Laptop- oder mobilen und Touch-Systemen. Besonders im Bereich der Hierarchievisualisierung, die der Informationsvisualisierung zuzuordnen ist, bietet sich eine interaktive Visualisierung auf großen, hochauflösenden Displays an; hiervon würde zum Beispiel die Visualisierung fraktaler „verallgemeinerter Pythagoras-Bäume“ oder anderer fraktaler Darstellungen profitieren. Jedoch stellt sich die Frage nach geeigneten Interaktionsmedien und damit zusammenhängenden Faktoren, die bislang möglicherweise noch gar keine Rolle gespielt hatten. Sind die Interaktionen z. B. möglicherweise körperlich ermüdend, weil sich Benutzer des Systems viel bewegen müssen? Sind die bereitgestellten Möglichkeiten passend zur Visualisierung gewählt? Ist die Steuerung des Systems intuitiv verständlich oder muss sie zunächst erklärt werden?

Diese Herausforderungen steht gegenüber, dass die Verwendung eines solchen Displays auch entsprechend viele neue Möglichkeiten mit sich bringt, die bislang noch unzureichend erforscht sind, da solche Systeme kostspielig und dementsprechend wenig verbreitet sind. Zu diesen Möglichkeiten zählt beispielsweise das kollaborative Arbeiten, also dass mehrere Benutzer gleichzeitig und am selben Ort mit einem System arbeiten und sich auf ganz natürliche Art und Weise über ihre Vorhaben, Erkenntnisse und Fehlschlüsse austauschen können. Im Rahmen dieser Arbeit soll auch beleuchtet werden, welche Chancen sich für die Zusammenarbeit mehrerer Benutzer an einem solchen System ergeben; hier ist eine große Spanne von Kollaborationsstilen denkbar.

Im Rahmen dieser Diplomarbeit sollen die „generalisierten Pythagoras-Bäume“, eine Hierarchievisualisierung fraktaler Art [BBM+en], auf großen, hochauflösenden Displays darstellbar und interaktiv manipulierbar gemacht werden. Dazu sollen

- zunächst Hierarchiedaten eingelesen werden können,
- Hierarchiedaten als „generalisierter Pythagoras-Baum“ dargestellt werden können,
- hierarchiebezogene interaktive Features implementiert werden,
- weitere Powerwall-spezifische Interaktionstechniken eingebaut werden, die beispielsweise ein optisches Tracking-System nutzen,
- eine Fallstudie mit großen Hierarchien wie etwa der NCBI-Taxonomie durchgeführt werden,
- verteiltes Rendering auf geteilten Displays (mit Datenreplikation, Synchronisation der Anzeigeparameter) implementiert werden.

## 1.2. Gliederung

Die Diplomarbeit gliedert sich in folgende Abschnitte:

**Kapitel 2 – Verwandte Arbeiten** beschreibt bestehende Ansätze, wie sie in der Informationsvisualisierung angetroffen werden und beleuchtet die Problemstellung im Rahmen anderer Arbeiten aus diesem Gebiet.

**Kapitel 3 – Interaktion** gibt einen Überblick über Möglichkeiten zur Interaktion an großen Displays, auf denen Visualisierungen dargestellt werden. Hier werden insbesondere auch die in dieser Arbeit implementierten Interaktionen vorgestellt.

**Kapitel 4 – Implementierung** beleuchtet die technischen Aspekte, die bei der Implementierung der Visualisierung mit ihren Interaktionsmöglichkeiten in einem verteilten System zu beachten sind. Darin finden sich eine Vorstellung der Struktur der Software sowie Details zur Implementierung der Softwarekomponenten. Außerdem werden dort die mathematischen Hintergründe für die vielen Berechnungen der Anwendungen dargelegt.

**Kapitel 5 – Alternatives System** behandelt ein System, das implementiert wurde, um unter technologisch abweichenden Voraussetzungen die gewünschten Funktionalitäten bereitzustellen. Da die Software nicht die erhoffte Performance bieten konnte, wurde sie nicht fertigentwickelt. Dieses Kapitel diskutiert die Gründe, die hierfür verantwortlich sind, und vergleicht wichtige Größen des Systems mit dem tatsächlich implementierten, das in Kapitel 4 vorgestellt wurde.

**Kapitel 6 – Fallstudien** untersucht die implementierten Visualisierungstechniken anhand zweier Datensätze, die als „große Hierarchien" betrachtet werden können. Verwendet werden eine Taxonomie über Spezies und Arten der Biologie mit über 300.000 Knoten und ein hierarchisches Dateisystem-Abbild mit über 450.000 Knoten.

**Kapitel 7 – Zusammenfassung und Ausblick** fasst die Ergebnisse der Arbeit zusammen. Im Anschluss daran werden Konsequenzen daraus gezogen und mögliche Anknüpfungspunkte vorgestellt.

## 1.3. Abkürzungen

| Abkürzung | Erläuterung |
|---|---|
| CPU | Central Processing Unit (Hauptprozessor) |
| D3D | Direct3D |
| DOF | Degree of Freedom (Freiheitsgrad) |
| DPI | Dots per Inch (Bildpunkte pro Zoll) |
| eMMC | Embedded Multimedia Card |
| FoV | Field of View (Öffnungswinkel der Sichtpyramide) |
| FPS | Frames pro Sekunde |
| GDI | Graphics Device Interface |
| GUI | Graphical User Interface (Benutzungsschnittstelle) |
| HD | High Definition |
| IPv4 | Internet Protocol Version 4 |
| MSDN | Microsoft Developer Network |
| NCBI | National Center for Biotechnology Information |
| SSD | Solid State Drive |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| VAST | Visual Analytics Science and Technology |
| VISUS | Visualisierungsinstitut der Universität Stuttgart |
| VLSI | Very Large Scale Integration |
| VRPN | Virtual Reality Peripheral Network |
| WXGA | Wide XGA = Wide Extended Graphics Array |

## 1.4. Konventionen

Autoren einer Publikation werden in Kapitälchen geschrieben, z. B. SHNEIDERMAN.

Vektoren werden mit fettgedruckten Kleinbuchstaben bezeichnet, z. B. $\mathbf{t}$.

Werden einzelne Komponenten eines Vektors benutzt, so ist dieser Vektor durch den Index des Symbols erkenntlich. $x_\mathbf{t}$ bezeichnet beispielsweise die X-Komponente eines Vektors $\mathbf{t}$.

Quaternionen werden in der Regel mit $q$ bezeichnet und sind Elemente aus $\mathbb{H}$.

Matrizen werden mit fettgedruckten Großbuchstaben bezeichnet, z. B. $\mathbf{W}$. Da es sich hierbei in dieser Arbeit ausschließlich um $4 \times 4$-Transformationsmatrizen handelt, werden einige Kurzschreibweisen verwendet, die wie folgt zu verstehen sind:

- $\mathbf{T}(x_\mathbf{t}, y_\mathbf{t}, z_\mathbf{t}) = \begin{pmatrix} 1 & 0 & 0 & x_\mathbf{t} \\ 0 & 1 & 0 & y_\mathbf{t} \\ 0 & 0 & 1 & z_\mathbf{t} \\ 0 & 0 & 0 & 1 \end{pmatrix}$ bezeichnet eine Translation um $\mathbf{t} = \begin{pmatrix} x_\mathbf{t} \\ y_\mathbf{t} \\ z_\mathbf{t} \end{pmatrix}$.
- Da die Translationen in der Regel zweidimensional sind, wird auch $\mathbf{T}(x_\mathbf{t}, y_\mathbf{t}) = \mathbf{T}(x_\mathbf{t}, y_\mathbf{t}, 0)$ verwendet.
- $\mathbf{S}(x_\mathbf{s}, y_\mathbf{s}, z_\mathbf{s}) = \begin{pmatrix} x_\mathbf{s} & 0 & 0 & 0 \\ 0 & y_\mathbf{s} & 0 & 0 \\ 0 & 0 & z_\mathbf{s} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ bezeichnet eine Skalierung mit $\mathbf{s} = \begin{pmatrix} x_\mathbf{s} \\ y_\mathbf{s} \\ z_\mathbf{s} \end{pmatrix}$.
- In diesem Dokument sind insbesondere solche Skalierungen von Interesse, die gleichermaßen in X- wie in Y-Richtung strecken bzw. stauchen und keine Skalierung auf der Z-Achse vornehmen. Daher wird kurz auch $\mathbf{S}(s) = \mathbf{S}(s, s, 1)$ geschrieben.
- $\mathbf{R}(\varphi) = \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 & 0 \\ \sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ bezeichnet eine Rotation um die Z-Achse um den Winkel $\varphi$. Rotationen um andere Achsen werden in dieser Arbeit nicht verwendet.
- $\mathbf{L}(\mathbf{p}, \mathbf{t}, \mathbf{u})$ stellt eine Kameramatrix dar, die die Kamera an die Stelle $\mathbf{p}$ (*position*) transliert, wobei von dort aus auf den durch $\mathbf{t}$ (*target*) dargestellten Punkt geschaut wird und $\mathbf{u}$

(*up*) die Richtung nach oben angibt[1]. Hierfür werden zunächst die X-, Y- und Z-Achsen transformiert und normiert:

$$\begin{aligned} \mathbf{z} &= \frac{\mathbf{t}-\mathbf{p}}{\|\mathbf{t}-\mathbf{p}\|} \\ \mathbf{x} &= \frac{\mathbf{u}\times\mathbf{z}}{\|\mathbf{u}\times\mathbf{z}\|} \\ \mathbf{y} &= \mathbf{z}\times\mathbf{x} \end{aligned}$$

Damit ist $\mathbf{L}(\mathbf{p}, \mathbf{t}, \mathbf{u}) = \begin{pmatrix} x_\mathbf{x} & x_\mathbf{y} & x_\mathbf{z} & 0 \\ y_\mathbf{x} & y_\mathbf{y} & y_\mathbf{z} & 0 \\ z_\mathbf{x} & z_\mathbf{y} & z_\mathbf{z} & 0 \\ -\langle \mathbf{x}, \mathbf{p}\rangle & -\langle \mathbf{y}, \mathbf{p}\rangle & -\langle \mathbf{z}, \mathbf{p}\rangle & 1 \end{pmatrix}$.

- $\mathbf{P}(\varphi_{\text{FoV}}, r, z_\text{n}, z_\text{f}) = \begin{pmatrix} r\cdot\cot\frac{\varphi_{\text{FoV}}}{2} & 0 & 0 & 0 \\ 0 & \cot\frac{\varphi_{\text{FoV}}}{2} & 0 & 0 \\ 0 & 0 & \frac{z_\text{f}}{z_\text{f}-z_\text{n}} & 1 \\ 0 & 0 & -\frac{z_\text{n}\cdot z_\text{f}}{z_\text{f}-z_\text{n}} & 0 \end{pmatrix}$ stellt eine perspektivische Projektion dar[2]. Dabei ist $\varphi_{\text{FoV}}$ der vertikale Öffnungswinkel der Sichtpyramide und $r = \frac{w}{h}$ das Seitenverhältnis des Displays mit Breite $w$ und Höhe $h$. $z_\text{n}$ bzw. $z_\text{f}$ bezeichnen die nahe (*near*) bzw. ferne (*far*) Clipping-Ebene des Frustums.
- $\mathbf{E}_n$ bezeichnet die $n \times n$-Einheitsmatrix $\mathbf{E}_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$. Wenn $\mathbf{E}$ verwendet wird, ist hiermit $\mathbf{E}_4$ gemeint.

[1] Siehe MSDN: http://msdn.microsoft.com/en-us/library/windows/desktop/bb281710%28v=vs.85%29.aspx
[2] Siehe MSDN: http://msdn.microsoft.com/en-us/library/windows/desktop/bb281727%28v=vs.85%29.aspx

# 2. Verwandte Arbeiten

Gemäß ihrem Titel versucht diese Arbeit, einige Parameter gleichzeitig abzudecken: *Große, hierarchische Daten* (2.2) sollen *interaktiv* (2.3) und *kollaborativ* (2.4) mit *fraktalbasierten Visualisierungen* (2.1) auf *großen Displays* (2.5) dargestellt werden. Dementsprechend stellt dieses Kapitel Arbeiten vor, in denen diese Gesichtspunkte eine Hauptrolle spielten bzw. im Rahmen eines anderen Anliegens berücksichtigt wurden.

## 2.1. Visualisierungen

Es existiert eine Vielzahl unterschiedlicher Visualisierungstechniken, von denen wiederum ein beträchtlicher Teil zur Darstellung hierarchischer Daten ausgelegt ist. Obwohl viele dieser Hierarchievisualisierungen im engeren Sinne als fraktalbasierte Visualisierungen verstanden werden können, kommen sie ohne eine spezielle Erläuterung ihrer fraktalen Natur aus[1]. Diese werden im folgenden Unterabschnitt 2.1.1 separat beschrieben, Unterabschnitt 2.1.2 stellt dann solche Visualisierungen vor, die sich explizit Fraktalen bedienen, um Hierarchien darzustellen.

### 2.1.1. Hierarchievisualisierungen

Die Darstellung hierarchisch organisierter Daten stellt ein klassisches Aufgabengebiet der Informationsvisualisierung dar. Sie sind ein in der Natur häufig beobachtbarer Sonderfall von Graphen im Allgemeinen. Auch sind sie tief verankert im menschlichen Denken; so wird praktisch jeder in der Lage sein, einen Mahagoni- oder Ebenholz-Baum als einen Baum zu identifizieren, auch wenn der Betrachter noch nie zuvor diesen speziellen Baum gesehen hat. Auch in einer darunterliegenden Ebene fällt es Menschen leicht, Laub- von Nadelbäumen zu unterscheiden. Die semantischen Taxonomien, die sich im Denken über Wahrgenommenes bilden, stellen eine sehr natürliche Hierarchie dar. Entsprechend zugänglich sind hierarchische Strukturen für Personen, die mit diesem Denken vertraut sind.

Eine der bekanntesten Visualisierungsmethoden sind *Knoten-Kanten-Diagramme*, wie sie auch für die Visualisierung allgemeiner Graphen verwendet werden. Aus ihnen wird schnell die Zusammenhangsstruktur des Graphen klar, da die zusammenhängenden Knoten durch Kanten verbunden sind. Solche Diagramme sind dafür bekannt, bei kleinen Graphen gut

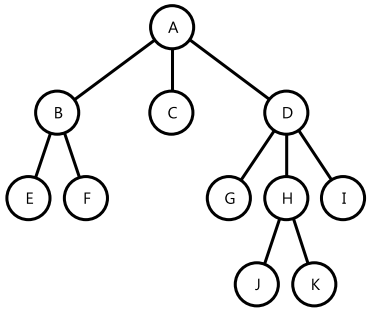
[1] Dieser Umstand kommt daher, dass Fraktale aufgrund ihres selbstähnlichen Aufbaus eine gewisse natürliche Gleichartigkeit mit hierarchischen Strukturen aufweisen.

lesbar zu sein, aber schlecht für große Graphen zu skalieren, da jeder Knoten eine grafische Repräsentation besitzt und immer mehr Kanten gerendert werden müssen, sodass der genaue Kantenverlauf durch viele Kreuzungen und kurze Winkel zwischen gekreuzten Kanten schwierig zu verfolgen wird; man spricht hier auch von „visual clutter“ [RLN07]. GHONIEM et al. berichten davon, dass für gewöhnliche (d. h. nicht-hierarchische Graphen) bereits ab einer Größe von 20 Knoten matrixbasierte Visualisierungen in sechs von sieben Aufgaben besser abschneiden als das entsprechende Knoten-Kanten-Diagramm [GFC04]. Auch wenn das Problem des „visual clutter“ durch ein geeignetes Graphenlayout zur Widerspiegelung der hierarchischen Struktur entschärft werden kann, ist der Platz auf einem großen Display immer noch zu stark begrenzt, als dass große Hierarchien, beispielsweise die NCBI-Taxonomie mit einer Größe von 324.276 Knoten, gut als Knoten-Kanten-Diagramm dargestellt werden könnten. Andere Techniken versuchen, verschiedene Visualisierungen zu kombinieren, wodurch ebenfalls eine Entwirrung erreicht werden kann. Ein Beispiel hierfür ist das von HENRY et al. vorgestellte System *NodeTrix*, in dem Knoten-Kanten- und Matrix-Darstellungen vereint werden [HFM07].

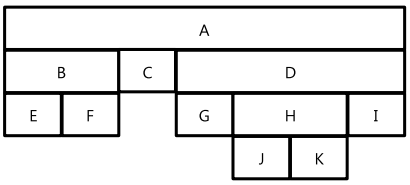
Neben dieser grundlegenden Technik beschäftigen sich viele Visualisierungen aber auch mit Hierarchien im Speziellen. Zunächst gibt es Methoden, die auf Knoten-Kanten-Diagrammen aufsetzen, um die dabei anzutreffenden Schwierigkeiten in den Griff zu bekommen (siehe beispielsweise Abbildung 2.1a). Hier sind z. B. *Edge-Bundling*-Methoden zu nennen, die die hierarchische Struktur beachten, um Kanten sinnvoll und ästhetisch ansprechend zusammenzubündeln [Hol06]. Dabei wird eine Kante zwischen zwei Knoten im Allgemeinen nicht durch eine gerade Linie zwischen den Knoten dargestellt, sondern durch eine B-Spline-Kurve, deren Kontrollpolygon sich aus der logischen Verbindung der Knoten über den kleinsten gemeinsamen Vorfahren in der Hierarchie ergibt. Solche Methoden adressieren aber eben lediglich die Symptome der Darstellungsart und können ohne grundlegende Änderungen an der Visualisierung keine Verbesserung der Ursache bewirken.

*Tree-Maps*, wie vorgestellt von JOHNSON und SHNEIDERMAN, verfolgen einen platzfüllenden Ansatz, bei dem die gesamte Breite und Höhe der Visualisierungsfläche bzw. des Bildschirms verwendet wird [JS91]. Die Grundidee besteht darin, dass der Wurzelknoten die gesamte Fläche ausfüllt. Diese Fläche wird in kleinere Rechtecke aufgeteilt, von denen jedes einen Kindknoten der Wurzel darstellt. Diese kleineren Flächen werden wiederum in ihre Kindknoten unterteilt und so weiter. Abbildung 2.1c illustriert diese Visualisierungsmethode. Die Kategorie platzfüllender Ansätze ist attraktiv, da sie, wenn die Bildschirmgröße nicht mehr ausreicht, um einen zu großen Datensatz darzustellen, auf einem größeren Display wieder präzise Resultate erzielen kann. Die Idee der *Tree-Maps* wurde in einigen Arbeiten erweitert, so z. B. von VAN WIJK und VAN DE WETERING, die mit *Cushion-Tree-Maps* einen Schattierungseffekt einführten, der die hierarchische Struktur stärker hervorhob [WW99]. BALZER et al. entfernten sich mit den *Voronoi-Tree-Maps* von der orthogonalen Anordnung der Knoten in der Visualisierung. Stattdessen wurden Voronoi-Diagramme als Grundlage für das Layout der Darstellung verwendet [BDL05].

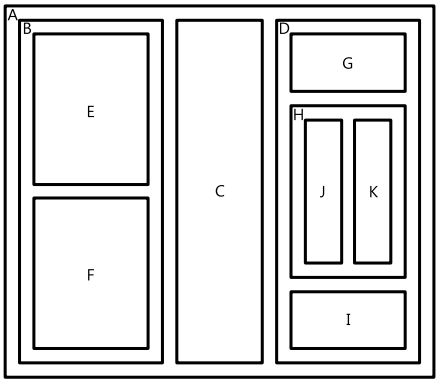
Ein weiterer, platzfüllender Ansatz (jedoch nur in einer Dimension) sind die von BURCH et al. vorgestellten *Indented Pixel Tree Plots* [BRW10]. Dabei wird eine Hierarchie mit Hilfe der z. B.



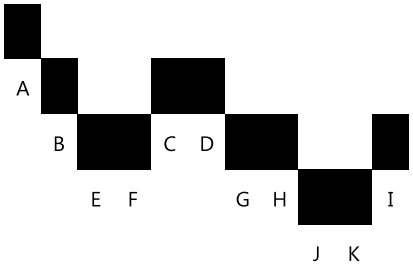

**(a)** Knoten-Kanten-Diagramm




**(b)** Icicle Plot




**(c)** Tree-Map




**(d)** Indented Pixel Tree Plot

**Abbildung 2.1.:** Verschiedene Hierarchievisualisierungen, die alle die gleiche Hierarchie zeigen [BSW11].

aus Dateisystem-Darstellungen bekannten Einrückungsmetapher visualisiert (siehe Abbildung 2.1d). Die Knoten der Hierarchie werden auf der gesamten Breite verteilt und durch gleichgroße Rechtecke dargestellt. Die Tiefe eines Knotens in der Hierarchie bestimmt, auf welche Höhe das Rechteck gezeichnet wird. Wenn die Breite des Bildschirms nicht ausreicht und mehrere Knoten in einen Pixel der Breite fallen, wird statt der Rechtecke an der horizontalen Position eine Linie gezeichnet, die das Minimum und Maximum der Tiefen der in den Pixel fallenden Knoten umfasst. Zu dieser Visualisierungsmethode existiert auch ein Werkzeug, das die interaktive Arbeit mit Indented Pixel Tree Plots erlaubt [BSW11]. Zu den verfügbaren Interaktionsmethoden zählen vor allem hierarchisch schachtelbare Detailansichten, die unter der Übersichtsdarstellung kleinere Teile daraus mit einem höheren Detailgrad visualisieren können. Wird dies weiter fortgesetzt, kann hiermit auch eine Auflösung erreicht werden, in der textuelle Informationen zu den Hierarchieknoten dargestellt werden können.

Ein ähnlicher Ansatz, der sich ebenfalls die Metapher des Stapelns zu Nutzen macht, wurde von KRUSKAL und LANDWEHR vorgestellt [KL83]. *Icicle Plots* stellen eine Hierarchie als untereinander geschichtete Blöcke dar, wie sie in Abbildung 2.1b gezeigt werden. Die Kindknoten werden auf die gesamte Breite des Elternknotens verteilt, was entweder äquidistant oder anhand einer bestimmten Metrik wie beispielsweise der Unterbaumgröße des Knotens geschehen kann. Mit jeder Hierarchieebene wächst die Darstellung um eine Schicht, was der Entstehung von Eiszapfen ähnelt. Im Gegensatz zu *Indented Pixel Tree Plots* wird nicht jedem Hierarchieknoten ein exklusiver Repräsentant in der Horizontalen spendiert, weshalb insgesamt eine schlankere Darstellung ermöglicht wird, jedoch benötigen Knoten mit großen Unterbäumen viel Breite.

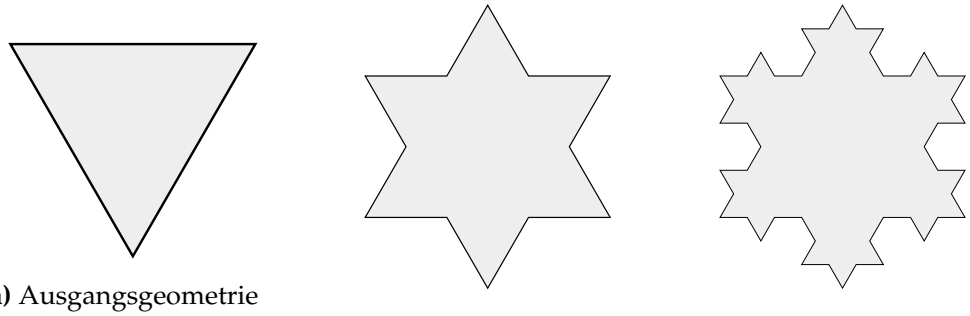
Mit *InterRing* wurde dieser Ansatz von YANG et al. in eine radiale Darstellung überführt [YWR02]. Anstatt die Knoten auf der Horizontalen zu verteilen, wächst die Visualisierung von innen nach außen, wobei ein innerer Ring die Wurzel markiert, um den herum die Kinder verteilt werden. Während bei *Icicle Plots* die Knoten auf der Breite des Elternknotens zum Liegen kommen, decken sie bei *InterRing* den entsprechenden Winkelanteil ab.

### 2.1.2. Fraktale Visualisierungen

MANDELBROT erklärt *Fraktale* – ohne eine mathematisch präzise Definition – als selbstähnliche, häufig auch in der Natur anzutreffende geometrische Figuren und Muster [Man91]. Klassische Beispiele für Fraktale sind Bäume, Farne und andere Gewächse, aber auch die Gestalt und das Problem der Länge von Küstenlinien wird mit dem Begriff des Fraktals erläutert. Eng daran gekoppelt ist die „Dimension“ und im Zusammenhang mit Fraktalen insbesondere die *Hausdorff-Dimension* für metrische Räume. Bildlich kann man sich hierunter die Dimension des Raumes vorstellen, der durch die Figur gefüllt wird. Übliche, geometrische Figuren besitzen eine meist ganzzahlige Hausdorff-Dimension; eine Linie spannt beispielsweise einen eindimensionalen Raum auf und besitzt die Hausdorff-Dimension $D = 1$. Die KOCH-Schneeflocke (siehe Abbildung 2.2), bei der pro Iteration jede Seite eines Dreiecks in vier neue Strecken mit jeweils einem Drittel der ursprünglichen Seitenlänge geteilt werden, besitzt hingegen eine Hausdorff-Dimension von $D = \frac{\log 4}{\log 3} \approx 1{,}262$.

Visualisierungen, die sich explizit eines Fraktals bedienen, sind im Vergleich zu den gängigen Hierarchievisualisierungen eher unüblich. Wie im vorherigen Abschnitt bereits beschrieben wurde, existiert das Konzept der *Tree-Maps* in mannigfaltigen Ausformungen. Von besonderem Interesse ist dabei die Art und Weise, wie die Form und Anordnung der Knotenrepräsentanten gefunden wird. Neben den *Voronoi-Tree-Maps* [BDL05] werden auch andere Layout-Algorithmen genutzt, um eine möglichst „perfekte“ Anordnung für die Visualisierung zu finden. Hierbei kommen insbesondere auch fraktale Layouts in Betracht.

WATTENBERG löst das Layout-Problem zunächst im Eindimensionalen, indem er die Hierarchieknoten anhand einer Metrik linear auf einer Linie verteilt. Diese optimale Lösung wird dann mithilfe der Hilbert-Kurven in eine zweidimensionale Darstellung überführt (siehe Abbildung 2.3) [Wat05]. Die Hilbert-Kurve ist ein Fraktal mit der Hausdorff-Dimension $D = 2$,



**(a)** Ausgangsgeometrie bzw. nullte Iteration **(b)** Erste Iteration **(c)** Zweite Iteration

**Abbildung 2.2.:** Die Entstehung der Koch-Schneeflocke in verschiedenen Iterationen. Die Abbildungen sind `http://commons.wikimedia.org/wiki/File:Koch_Snowflake_7th_iteration.svg` entnommen.

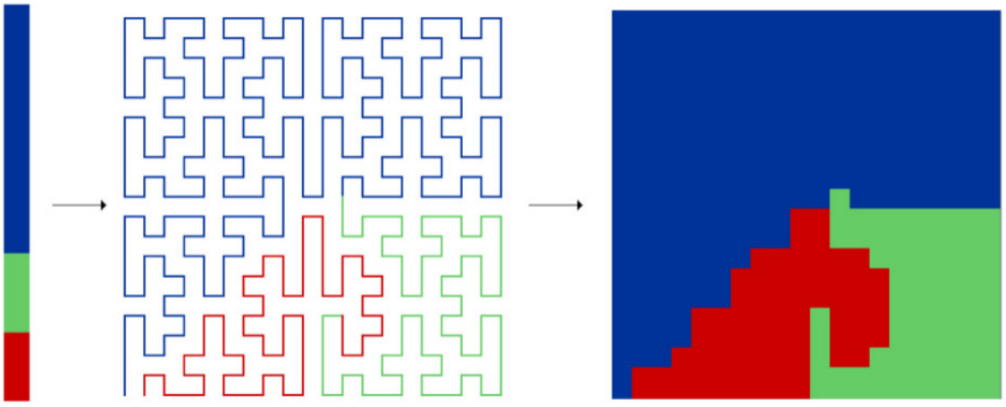


**Abbildung 2.3.:** Das Layout-Problem wird eindimensional gelöst und mithilfe einer Hilbert-Kurve platzfüllend angeordnet. [Wat05]

sie ist also im Grenzübergang lückenlos platzfüllend. Sie erfüllt außerdem die in [Wat05] geforderten Bedingungen für eine „optimale“ Layout-Funktion.

Ein anderes Fraktal, die so genannten *H-Bäume* (siehe Abbildung 2.4), finden Verwendung in Werkzeugen zum Entwurf von Systemen mit hohem Integrationsgrad, insbesondere im Hinblick darauf, wie Signale in VLSI-Schaltkreisen effizient an Chips weitergereicht werden können.

KOIKE und YOSHIHARA nutzen ein kegelartiges Fraktal, um große Hierarchien zu visualisieren [KY93]. In ihrer Arbeit wird ein 3D-Visualisierungswerkzeug vorgestellt, mithilfe dessen eine Dateihierarchie dargestellt wird. Der Elternknoten der aktuell gezeigten Hierarchie bildet die Spitze des Kegels, die Kindknoten werden äquidistant auf dem Rand der Grundfläche verteilt. Dabei gestaltet sich insbesondere die Interaktion bei jeder Tiefe gleich wie in den darüber- und den darunterliegenden Ebenen. Diese interaktiven Möglichkeiten des Systems

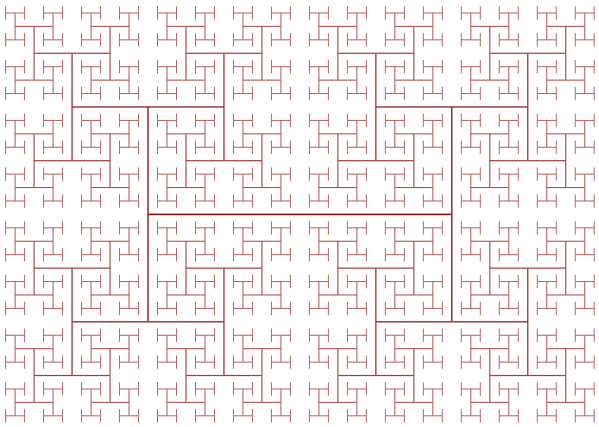


**Abbildung 2.4.:** Ein H-Baum in der zehnten Iteration. Bildherkunft: http://commons.wikimedia.org/wiki/File:H_tree.svg.

umfassen neben der Suche bestimmter Knoten im Datensatz und der Transformation der Ansicht auch das Öffnen von Dateien, also datensatzspezifische Features.

BECK et al. nutzen Pythagoras-Bäume und verallgemeinern sie von binären Fraktaldarstellungen zu $n$-ären Bäumen, die zur Darstellung von Visualisierungen geeignet sind [BBM+en]. Diese Technik ist für diese Arbeit grundlegend und wird daher im Implementierungskapitel unter Unterabschnitt 4.5.1 näher beschrieben.

## 2.2. Große Datensätze

Nicht erst seitdem der Begriff der „Informationsexplosion“[2] zum geflügelten Wort in der Informationstechnologie geworden ist, sind große Datensätze in den Visualisierungsdisziplinen von Interesse. GANTZ und REINSEL schreiben 2012, dass zwischen 2005 und 2020 ein Anstieg des umlaufenden Datenvolumens um einen Faktor 300 stattfinden wird, die Daten also von anfänglich 130 Exabyte ($= 130 \cdot 10^{15}$ Byte) auf 40 Zettabyte ($= 40 \cdot 10^{18}$ Byte) wachsen werden (siehe Abbildung 2.5) [GR12]. Ein Problem hierbei ist die Schere, die zwischen dem Datenvolumen und dem mit der verfügbaren Hardware analysierbaren Datenvolumen klafft; letzteres wächst zwar auch und ebenfalls exponentiell, allerdings nicht so schnell wie die erzeugten Daten. Um diese Lücke zu überwinden, bedarf es angemessener Methoden aus dem Visualisierungs- und Visual-Analytics-Umfeld.

Seit 2006 wird jährlich die so genannte „VAST-Challenge“ veranstaltet, in der Forscher aus dem Visual-Analytics-Bereich zu einem gegebenen (fiktiven) Szenario anhand eines zur Verfügung gestellten Datensatzes eine Lösung für die Bewältigung der Datenmengen einreichen

[2] Unter der „Informationsexplosion“ versteht man das überproportionale Wachstum der erzeugten und verfügbaren Daten, dem oft exponentieller Charakter attestiert wird. Siehe auch http://www.zakon.org/robert/internet/timeline/.

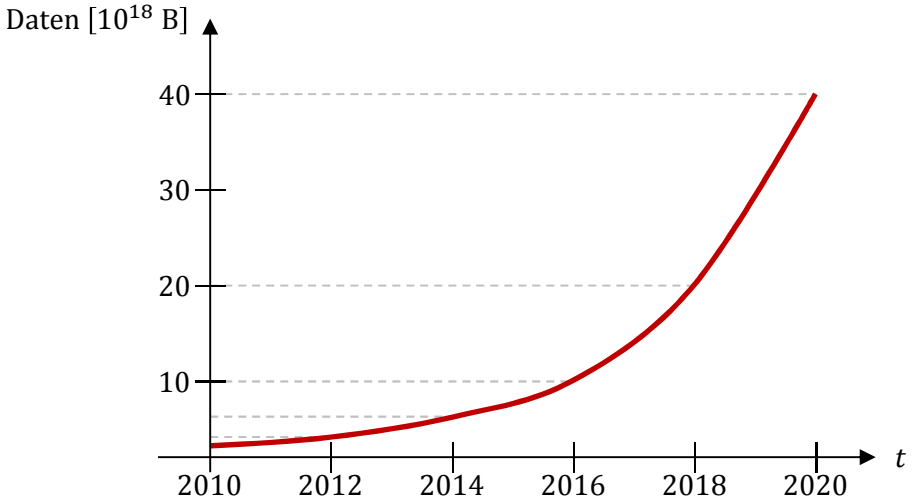



**Abbildung 2.5.:** Der Exponentieller Anstieg des Datenvolumens, der als „Informationsexplosion“ bezeichnet wird. Prognostiziert ist eine Verdoppelung der Datenmengen alle zwei Jahre [GR12].

können, die den Betrachtern Einsichten in die vielseitigen Daten gibt. So stellt beispielsweise KEIM bereits 2002 eine Kategorisierung von Visualisierungs- und Data-Mining-Techniken anhand dreier Dimensionen vor, nämlich des dargestellten Datentyps, der Visualisierungstechnik und der Interaktionstechnik [Kei02]. Auch die bei der VAST-Challenge verwendeten Datensätze wurden von Jahr zu Jahr umfangreicher [CGW13]. Um die mannigfaltigen Facetten zu adressieren, die sich bei der Arbeit mit großen Datensätzen ergeben, gibt es seit 2008 auch „Mini-Challenges“, in denen das Augenmerk auf präziser umrissenen Teilen liegt. Das zeigt auch, dass universelle Lösungen für große, komplexe Daten äußerst schwierig und oft unrealistisch sind.

## 2.3. Interaktion

Zwar wird in der Informationsvisualisierung häufig versucht, dem Betrachter Muster, Ausreißer etc. im Datensatz durch statische Darstellungen näherzubringen, jedoch spielt seit jeher auch die Interaktion eine große Rolle. Sie wird häufig dann eingesetzt, wenn statische Darstellungen nicht in ausreichendem Umfang in der Lage sind, den Informationsgehalt adäquat wiederzugeben. Das ist z. B. der Fall, wenn

- Datensätze so groß sind, dass sie nicht befriedigend oder nur unvollständig auf einem Display dargestellt werden können,
- Daten eine inhärente Komplexität besitzen, die durch die Visualisierung nicht gut auf visuelle Parameter abgebildet werden kann, oder

- Daten hochdimensional sind und relevante Informationen keinen gut erkennbaren Eingang in die Darstellung finden.

SHNEIDERMAN koppelt in [Shn96] Interaktion an sieben Typen von Aufgaben, die der Benutzer damit zu bewältigen versucht. Diese werden im Folgenden kurz und abstrakt beschrieben; je nach gewählter Visualisierung gestalten sich diese Aufgaben natürlich möglicherweise grundlegend unterschiedlich.

1. **Übersicht** (*overview*): Der Benutzer möchte einen Überblick über den Gesamtdatensatz erlangen.
2. **Zoom**: Vergrößerung von interessanten Elementen.
3. **Filter**: Reduktion der Darstellung auf die relevanten Elemente.
4. **Details auf Anfrage** (*details-on-demand*): Auswahl relevanter Elemente, um darüber detailliertere Informationen zu erhalten.
5. **Beziehungen** (*relate*): Der Benutzer möchte Beziehungen und Zusammenhänge unter den Elementen sichtbar machen.
6. **Versionen** (*history*): Aktionen können rückgängig gemacht und wiederhergestellt werden (*undo* und *redo*).
7. **Extraktion** (*extract*): Teildatensätze können anhand bestimmter Anfrageparameter extrahiert werden.

Diese Aufgaben folgen dem von SHNEIDERMAN in derselben Arbeit vorgelegten *Information Seeking Mantra*, das für viele Arbeiten der grundlegende Gedanke hinter der Implementierung interaktiver Features geworden ist:

> Overview first, zoom and filter, then details-on-demand.

YI et al. schlagen eine andere Klassifikation vor und geben Beispiele für Arbeiten, in denen repräsentative Interaktionstechniken umgesetzt werden [YKSJ07]. Die Kategorien ergeben sich aus den Intentionen der Benutzer beim Interagieren; mit ihrer Hilfe wird gemäß den Empfehlungen von THOMAS und COOK [TC05] versucht, eine Interaktionstaxonomie auf einer höheren Abstraktionsebene darzustellen. Diese wird im Folgenden wiedergegeben und ihre Relevanz für diese Arbeit erörtert:

1. **Auswählen** (*select*): Markieren von interessanten Elementen – In dieser Arbeit ist hier besonders das Selektionsfeature mithilfe des Tablets interessant (siehe Unterabschnitt 3.2.1).
2. **Erkunden** (*explore*): Einblicke in andere Teile des Datensatzes erlangen – Hierbei handelt es sich um das bisherige Hauptaugenmerk dieser Arbeit, die versucht, die interaktive Navigation durch die Darstellung eines Datensatzes zu ermöglichen.

3. **Umordnen** (*reconfigure*): Neue Anordnung der sichtbaren Informationen – Mit Ausnahme der Umordnung der Benutzeransichten werden derartige Features bislang noch nicht verwendet und die gegenwärtig angedachte Arbeitsweise mit dem System verlangt auch nicht danach. Allerdings könnte es in folgenden Arbeiten eine größere Rolle spielen, wenn der kollaborativen Arbeit mehr Aufmerksamkeit beigemessen wird.
4. **Ansicht wechseln** (*encode*): Anzeige der Daten mit einer anderen Darstellung – Konzeptionell ist eine solche Interaktion vorgesehen, jedoch wird am Ende dieser Arbeit auch diskutiert, warum diese Interaktionskategorie sich im Rahmen dieser Arbeit schwierig gestaltet; siehe hierfür Kapitel 7.
5. **Detailgrad regulieren** (*abstract/elaborate*): Mehr oder weniger detailliertere Informationen anzeigen – Da bislang lediglich die verallgemeinerten Pythagoras-Bäume zur Visualisierung verwendet werden, kann ein Teil dieser Kategorie durch die geometrischen Transformationen erreicht werden. Die Sinnhaftigkeit von Auf- und Zuklapp-Interaktionen von Teilbäumen wird in dieser Arbeit unter dem Gesichtspunkt der Stabilität der Visualisierung erörtert; siehe hierzu Abschnitt 4.5.1 und Abschnitt 5.2.1. Weitere, semantische Features, die beispielsweise die Domäne des Datensatzes beachten, können vor allem auf dem Tablet angewandt werden. Siehe hierzu Abschnitt 3.2.
6. **Filter**: Nur Elemente anzeigen, die bestimmten Bedingungen genügen – Ein solches Feature ist in dieser Arbeit nicht vorgesehen, hätte aber konzeptionell an zwei Stellen Platz: Erstens in der Remote-Anwendung (siehe Unterabschnitt 4.8.1), von der aus die Daten an die Rendering-Clients verteilt werden; hier könnte eine Vorfilterung vorgenommen werden. Zweitens könnte auch jeder Benutzer die Möglichkeit haben, auf dem in seinem Arbeitsbereich sichtbaren Datensatz Filteroperationen anzuwenden.
7. **Verbindungen** (*connect*): Anzeigen von Beziehungen zwischen den Elementen – Die hierarchische Beziehung zwischen den Knoten innerhalb des Datensatzes wird durch die Visualisierung wiedergegeben. Weitere Verbindungen waren in den in dieser Arbeit verwendeten Fallbeispielen (siehe Kapitel 6) nicht relevant.

Weitere Interaktionstechniken sind üblich und weit verbreitet. Sie gehen nicht speziell aus den Informationsvisualisierungs-Anwendungen hervor, beispielsweise das Rückgängigmachen oder Wiederherstellen von Aktionen.

Die technologischen Fortschritte auf dem Gebiet Multitouch-fähiger Anzeigegeräte lassen gestenbasierte Interaktionen zu einem wichtigen Forschungsschwerpunkt in der Mensch-Rechner-Interaktion werden. Durch die immense Verbreitung von Smartphones, bei denen Benutzerinteraktion fast ausschließlich über den Touch-Bildschirm geschieht, haben sich inzwischen feste Gesten für bestimmte Aufgaben etabliert. Hierunter zählen insbesondere auch Gesten zur Transformation von Ansichten, wie sie in dieser Arbeit verwendet werden [HSH+04]:

- Wisch- und Panning-Gesten zum Scrollen und Verschieben
- Zieh-Gesten zum Zoomen
- Gegenläufige Gesten zur Rotation

NORMAN beschreibt generelle Richtlinien, die beim Entwurf von Schnittstellen zu beachten sind [Nor02]. Dabei handelt es sich zwar nicht um eine Arbeit aus der Mensch-Rechner-Interaktion, aber die Erkenntnisse daraus lassen sich häufig analog darauf übertragen.

## 2.4. Kollaboration

Zusammenarbeit im Zusammenhang mit Informationsvisualisierung ist vornehmlich ein Anliegen jüngerer Forschung und wird eher im Umfeld von Visual Analytics versucht. STUSAK identifiziert in [Stu09] zwei wichtige Dimensionen, die bei der Kollaboration beachtet werden müssen:

**Zeit** Arbeiten die verschiedenen Nutzer gleichzeitig (*synchronous*) oder zeitversetzt (*asynchronous*) mit der Visualisierung?

**Ort** Befinden sich die Nutzer am selben Ort (*co-located*) oder sind sie über verschiedene Standorte verteilt (*distributed*)?

Jede Kombination hieraus birgt eigene Herausforderungen, die beim Entwurf und der Implementierung eines Informationsvisualisierungs-Systems beachtet werden müssen. So nennen ISENBERG und CARPENDALE Richtlinien für die Hardware sowie für die Gestaltung der Visualisierung und der kollaborativen Umgebung [IC07]. So wird beispielsweise bei der Hardware verlangt, dass ein ausreichend großes Display verwendet wird, um die häufig großen Datenmengen adäquat darstellen zu können. Außerdem bringen verschiedene Arten von Displayaufbauten verschiedene Tendenzen bezüglich der Art der Kollaboration mit sich. So können Powerwalls von größeren Gruppen verwendet werden, während Tabletops durch ihre Größe und die Positionierung des Displays die Anzahl der kollaborierenden Benutzer natürlich begrenzen, da bei vielen Nutzern das Display unerreichbar bzw. durch die anderen verdeckt wird. Die große Vielfalt der Interaktionsmöglichkeiten mit einer Powerwall bringt allerdings auch die daraus folgenden Schwächen mit sich. So kann sich eine Gestensteuerung mit einer Wand als körperlich zu aufwändig erweisen, als dass sie als Hauptinteraktion genutzt werden kann.

Kollaboratives Arbeiten hat natürlich auch Konsequenzen für die Interaktion. Auch wenn sich die Zusammenarbeit in dieser Arbeit in einem überschaubaren Rahmen bewegt, gibt es Einsichten aus Veröffentlichungen über öffentliche Displays, die hier ebenfalls relevant sind. PELTONEN et al. beschreiben beispielsweise anhand eines Aufbaus in Helsinki, wie Benutzer eines großen, 2,5 m breiten, Multitouch-fähigen Displays von sich aus beginnen, zu kollaborieren, sich gegenseitig mit ihrem erlangten Wissen über das System Hilfestellung geben, es aber auch zu Konfliktsituationen über den „Besitz“ dargestellter Elemente kommt [PKS+08]. Eine grundlegende Folgerung aus ihrer Arbeit ist aber, dass sich derartige Displays eignen, mehreren Benutzern die Zusammenarbeit zu ermöglichen. Ebenso spielt dort der Aspekt des *social learning*s, d. h. dass sich die Nutzer gegenseitig unterrichten und helfen bezüglich der Benutzung des Systems, eine große Rolle.

Eben dieser Ansatz ist auch im Rahmen der kollaborativen Arbeit in dieser Arbeit angedacht: Die Benutzer des Systems arbeiten logisch getrennt voneinander. Im Gegensatz zu [PKS+08],

wo keine explizite Trennung zwischen den Nutzern vorgesehen ist, passiert diese hier sehr streng, sodass jeder seinen eigenen Arbeitsbereich besitzt, in dem er unbeeinflusst von seinen Kollegen Interaktionen durchführen kann. Nutzer können sich ihre eigenen Erkenntnisse dann aber durch spezielle Interaktionen gegenseitig zeigen und so ein *social learning* stattfinden lassen, wenn auch nicht in Bezug auf das System und seine Benutzung, sondern auf die sichtbaren Informationen und deren Bedeutungsgehalt. Siehe hierzu auch die in dieser Arbeit angedachten Features für einzelne Benutzer (Unterabschnitt 4.7.1) und für die Kollaboration mehrerer Nutzer (Unterabschnitt 4.7.2).

## 2.5. Große Displays

Für viele Visualisierungstechniken ist der zur Verfügung stehende Bildschirmplatz das ausschlaggebende Maß für die Menge an Information, die sie darzustellen vermag. Diese Aussage trifft insbesondere auf Visualisierungen zu, die den ihnen zur Verfügung stehenden Platz komplett belegen. Beispiele hierfür sind *Tree-Maps* und diverse ihrer vielen Ableger (Cushion-, Voronoi-, Jigsaw-Tree-Maps etc.). Es gibt aber auch Knoten-Kanten-Darstellungen, deren Layout-Algorithmen den Graphen in einer bestimmten Geometrie bzw. – als Spezialfall hiervon – dem Bildschirm anordnen. Kräftebasierte Layouts sind ein Beispiel für diese Kategorie des Graphenlayouts.

In einer von BALL und NORTH im Jahre 2005 vorgelegten Arbeit wurde untersucht, wie sich die Verwendung eines großen Displays – dort ein Aufbau von $3 \times 3$ Monitoren – auf die Navigation in hochauflösenden Visualisierungen, das Suchen einer bestimmten Stelle oder das Finden verwandter visueller Elemente darin auswirkt. Ihre Evaluation zeigt, dass die Nutzer auf das System gut ansprechen und die geforderten Aufgaben schneller als mit herkömmlichen Displays bearbeitet werden konnten [BN05].

Als Vorzüge großer Displays führen CZERWINSKI et al. an, dass sie zunächst zu einer generellen Verbesserung der Produktivität von Nutzern gegenüber herkömmlichen Displays führen und darüber hinaus eine besondere Befriedigung mit sich bringen. Das liegt nicht zuletzt daran, dass große Displays in der Lage sind, dem Betrachter in einem größeren, seiner Wahrnehmung angepassteren Blickwinkel Daten zu präsentieren. Dieser kognitive Vorteil macht sich insbesondere bei Frauen bemerkbar, die bei Versuchen an großen Displays ähnliche Resultate erzielten wie Männer, die gegenüber Frauen bei kleineren Bildschirmen sonst tendenziell besser abgeschnitten haben [CRM+06]. Ebenso wie [BN05] stellen CZERWINSKI et al. fest, dass Nutzer die Arbeit mit größeren Displays gegenüber üblichen Anzeigegeräten als befriedigender wahrnehmen und dabei häufig eine höhere Produktivität zeigen [CSR+03].

LEHMANN et al. stellen ein System vor, das einen optischen Tracker von NaturalPoint und dessen Daten für die Interaktion mit der Visualisierung nutzt. Hierbei wurden detaillierte Informationen zur Verfügung gestellt, wenn sich der Benutzer dem Display näherte, Daten hingegen aggregiert, wenn er sich davon entfernte. Hierzu wurde der Platz vor dem Display in drei Abstandsbereiche eingeteilt. Im Gegensatz zur vorliegenden Arbeit stand dort ein Displaysystem mit 24 Monitoren, die zu einer großen Fläche angeordnet waren, zur Verfügung; entsprechend befinden sich zwischen je zwei Monitoren noch die Ränder des Bildschirms. Es

ist also kein übergangsfreies Display und es wurden in der Arbeit von LEHMANN et al. auch Anstrengungen unternommen, beispielsweise die Beschriftungen so anzuordnen, dass diese nicht bildschirmbereichübergreifend gezeichnet werden [LSST11].

Die für diese Arbeit zur Verfügung stehende Powerwall wurde zunächst unter dem Gesichtspunkt genutzt, dass dort ein bedeutend größerer Datensatz dargestellt werden kann als auf niedriger auflösenden Displays. Bei fraktalbasierten Visualisierungen, die nicht platzfüllend sind, ist dies allerdings gegenwärtig nicht der Hauptvorteil. Er könnte aber an Bedeutung gewinnen: Da das System im Sinne der Erweiterbarkeit darauf ausgelegt ist, künftig auch noch andere Visualisierungen als die verallgemeinerten Pythagoras-Bäume darstellen zu können, könnte sich die Darstellung mit insgesamt $10.800 \cdot 4.096$ Pixel $\approx$ 44,24 Megapixel als sehr vorteilhaft bei pixelbasierten oder platzfüllenden Visualisierungen erweisen. Die Größe der zur Verfügung stehenden Fläche ermöglicht es hier vor allem, dass mehrere Benutzer gleichzeitig und kollaborativ (siehe Abschnitt 2.4) mit dem System arbeiten können.

# 3. Interaktion

Dieses Kapitel behandelt die in Betracht gezogenen Vorgehen, wie der Betrachter mit der Visualisierung interagieren könnte. Hierbei werden die beteiligten Geräte bzw. Methoden vorgestellt und diskutiert, welche Möglichkeiten sie speziell für die Anwendung an einer Powerwall bieten. Da es sich bei dem Tablet um das endgültig verwendete Gerät handelt, wird es in Abschnitt 3.2 gesondert beschrieben.

In diesem Kapitel wird nicht das optische Tracking-System beschrieben, da es nicht im engeren Sinne ein vom Benutzer bedienbares, interaktives Gerät ist. Da der Tracker trotzdem relevant für die Interaktion im Gesamtsystem ist, sei an dieser Stelle auf Unterabschnitt 4.1.4 verwiesen, wo der Aufbau des Kamerasystems und das dafür benötigte Zubehör beschrieben ist.

Details zur Implementierung der hier beschriebenen, interaktiven Features, soweit sie in der Software umgesetzt wurden, finden sich in Abschnitt 4.7.

## 3.1. Geräte

Dieser Abschnitt stellt die verschiedenen Geräte vor, die zur Interaktion mit der Visualisierungssoftware zur Verfügung standen oder hierfür verwendet werden könnten. Ausgenommen hiervon ist das Tablet (siehe Abschnitt 3.2). Es wurde als Hauptinteraktionsmethode verwendet und wird daher gesondert und in größerem Umfang beschrieben.

### 3.1.1. Flight-Stick

Beim Flight-Stick handelt es sich um einen Stab, der kabellos in der Hand gehalten wird und durch das optische Tracking-System im Raum verfolgt wird. Er bietet intuitiv die dadurch zur Verfügung stehenden sechs Freiheitsgrade, jeweils drei für die Position im Raum (X-, Y- und Z-Koordinaten) und weitere drei für die Orientierung des Stabs (jeweils die Rotation um die X-, Y- und Z-Achse). Außerdem besitzt er vorne zwei gegenüberliegende Tasten, die der Benutzer wie die einer gewöhnlichen Computermaus bedienen kann. Diese können softwareseitig einfach z. B. durch VRPN in Ereignisse umgesetzt werden.

Der Flight-Stick besticht durch seine Einfachheit und sein geringes Gewicht. Einem Benutzer ist sofort klar, dass er mit den zwei Knöpfen Interaktionsmöglichkeiten hat, die – sobald der Stab einmal richtig orientiert ist – gut voneinander abgrenzbar sind. Der Stick ist handlich und besitzt eine klare Grundausrichtung in der Tiefenachse, die durch den Tracking-Starrkörper an der Vorderseite gegeben ist.

Durch diesen einfachen Aufbau sind natürlich die Möglichkeiten einer differenzierten Interaktion eingeschränkt. Eine Folge hiervon ist, dass bei der Verwendung des Geräts mittelbare Interaktionsebenen wie Menüs, Bewegungsgesten o. Ä. eingerichtet werden müssten, um breiter gefächert Aktionen anzubieten.

### 3.1.2. 3D-Maus

Die 3D-Maus ähnelt optisch einer Art verkürztem Joystick. Der Benutzer hält einen Knauf in der Hand, der auf einem Boden dreh- und neigbar angebracht ist. Auch dieses Gerät bietet 6 DOF sowie zwei Tasten an der linken und rechten Seite des Bodens. Im Gegensatz zur klassischen Maus wird die 3D-Maus nicht insgesamt bewegt, sondern lediglich der Knauf gedreht oder in die gewünschte Richtung geneigt.

Im Positiven ist hervorzuheben, dass es sich bei der 3D-Maus für computererfahrene Benutzer um eine recht intuitive Bedienmöglichkeit handelt, da die Verwendung der Tasten offensichtlich ist und auch die Neigung des Knaufs das von klassischen Mäusen gewohnte Verhalten widerspiegelt. Auch die räumliche Bewegung bildet sie gut ab, sodass auch für die Drehungen klar ist, welche Handbewegungen welche Freiheitsgrad-Semantik besitzen. Um allerdings die große Powerwall nutzen zu können, sollte die Maus nicht durch ein Kabel an den Remote-Rechner gebunden sein, sondern über Funk die Interaktionsdaten verfügbar machen, und selbst dann wären beide Hände zur Bedienung nötig.

### 3.1.3. Sprachsteuerung

Zur Steuerung der Visualisierungssoftware durch Sprache wird hardwareseitig lediglich ein Mikrofon benötigt. Da auf Grund von fehlendem Weltwissen des Rechners, Performancegründen etc. in der Regel keine semantische Analyse eines natürlichsprachlichen Satzes in Echtzeit durchgeführt werden kann, ist ein solches System darauf angewiesen, einen geringen Befehlssatz bereitzustellen, der von der Anwendung erkannt wird.

Ein positiver Aspekt dieser Interaktionsmethode ist die physische Freiheit des Betrachters, abgesehen beispielsweise von einem Headset-Mikrofon, das er am Kopf trägt. Es fallen keine weiteren Geräte an, die mit den Händen bedient oder anderweitig getragen werden müssten.

Eine Schwierigkeit bei diesem Konzept stellt jedoch die Identifikation eines Datenelements dar. Beispielsweise verwenden Pythagoras-Bäume aus semiotischer Sicht Quadrate bzw. Rechtecke als Zeichen für ein Element des Datensatzes, das dessen semantischen Inhalt darstellt. Am Beispiel der phylogenetischen Taxonomie (siehe Fallstudie 6.1) stellt sich die Frage, wie dieses geometrische Objekt in ein Sprachkommando einfließen soll. Eine Möglichkeit wäre ein Befehl im Stile von „Zoome auf *Knoten*“, wobei der *Knoten* durch den Namen des Organismus identifiziert würde. Dann müsste dem Betrachter aus der Visualisierung ersichtlich sein, welchen Namen das betroffene Element trägt. Wenn dieser einer Fremdsprache (in diesem Beispiel: dem Lateinischen) entlehnt ist, stellt die Aussprache und Erkennung potentiell eine

weitere Herausforderung dar. Eine Identifikation des Knotens durch eine Nummer oder durch visuelle Elemente wie dessen Größe oder Farbe sprengen die Grenzen der Benutzbarkeit und des Leistungsvermögens einer Sprachsteuerung.

Am Beispiel der in dieser Arbeit verwendeten Daten und Visualisierungstechniken wäre eine Steuerung durch Sprache also – wenn überhaupt – nur in Kombination mit anderen Interaktionsmedien brauchbar, wodurch der Vorteil der Unabhängigkeit von weiteren Geräten wieder eingeschränkt wird oder ganz wegfällt.

Ein weiterer Ansatz für Sprachsteuerung sind so genannte „Mausraster“, d. h. der Bildschirm (hier eventuell auch nur der sichtbare Arbeitsbereich) wird beispielsweise in neun Felder unterteilt, die über Sprache identifiziert werden. Diese Unterteilung wird dann im gewünschten Feld weiter fortgesetzt, bis der Benutzer genau die Stelle getroffen hat, die ihn interessiert bzw. die er manipulieren möchte. Dieser Ansatz lässt sich bei überschaubaren Displays gut umsetzen, skaliert jedoch nicht gut mit der Auflösung. So wären viele Verfeinerungsschritte notwendig, wenn der Benutzer sich für ein klein dargestelltes Element interessiert.

### 3.1.4. Gestensteuerung

Es gibt verschiedene Systeme zur Gestenerkennung, die sich maßgeblich in ihrem jeweiligen Bedienkomfort und ihrer Präzision unterscheiden. Eine Kategorie von Lösungen bestimmt die Gesten mithilfe von Kameras, die den Benutzer aufnehmen und beispielsweise durch Methoden der Bildverarbeitung oder durch komplexere Kamerasysteme, die eine Tiefeninterpretation der Aufnahmen erlauben, die Körperhaltung und somit die Gestik des Benutzers herauszufinden versuchen. Eine erschwingliche Lösung wurde beispielsweise von Microsoft mit dem Kinect™-Projekt auf den Markt gebracht. Leider ist dieses System aufgrund der Interferenz im Infrarot-Gitter nicht in der Lage, den Raum vor der Powerwall geeignet aufzunehmen.

Würde die Gestenerkennung allerdings funktionieren, wäre eine große Vielfalt interaktiver Features denkbar. Jedoch muss dann darauf geachtet werden, dass die verwendeten Gesten nicht zu ausfallend sind bzw. nicht zu häufig durchgeführt werden müssen, da sie sonst eine schnelle körperliche Ermüdung hervorrufen können oder sich mehrere Benutzer gegenseitig beim Arbeiten mit der Software behindern. Dies gilt insbesondere für Gesten zur Translation, Rotation oder Skalierung der Ansicht.

## 3.2. Tablet

Ein optisch getracktes Tablets impliziert eine hybride Interaktionsform:

- Es handelt sich um einen *vollwertigen Computer*, der aufwändige Berechnungen durchführen kann und anspruchsvolle Aufgaben zu lösen vermag. Das versetzt das Tablet in die Lage, über das Netzwerk mit den anderen Rechnern zu kommunizieren und eine Darstellung des Datensatzes sowie Detailinformationen dazu zu bieten.

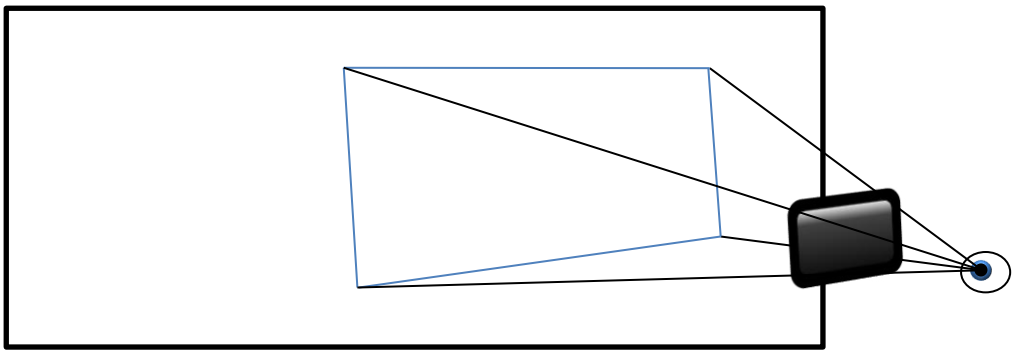


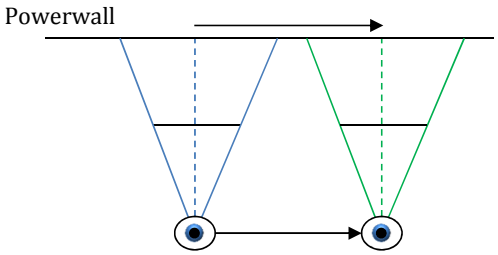
**Abbildung 3.1.:** Die virtuelle „Sichtpyramide“ eines Benutzers, der durch sein Tablet auf die Darstellung auf der Powerwall sieht.

- Gleichermaßen erfüllt es nicht nur den Charakter eines gewöhnlichen Rechners, sondern hat auch Eigenschaften eines *tangible UI*s. Zwar handelt es sich um einen festen Körper, der nicht verformbar ist und dementsprechend keine Elemente der grafischen Repräsentation der Darstellung auf der Powerwall nachahmt, jedoch ermöglicht die Kombination mit dem optischen Tracking-System eine Interaktionsform, die im Kontext digitaler Darstellungen für die Benutzer ungewohnt sein dürfte, da sich physische Eigenschaften wie die Lage und Orientierung, also die Art und Weise, wie das Tablet gehalten wird, auf die Visualisierung auswirken.

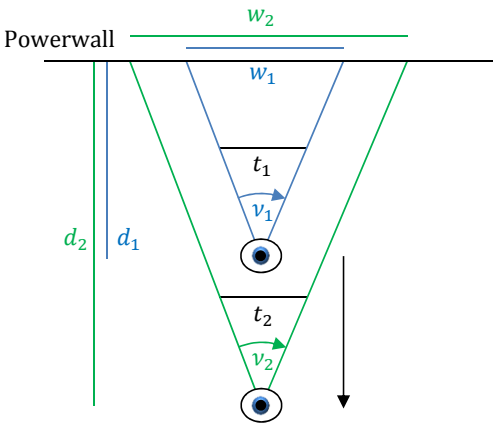
Das Tablet schafft also eine Art „semidigitales Interaktionsmedium“, wodurch Interaktion, die mit materiellen Gegenständen stattfindet, wieder stärker in den Mittelpunkt der Aufmerksamkeit rücken muss. Ein solches Umfeld lässt die ansonsten häufig gewählten Metaphern beim Erledigen von Aufgaben noch realer erscheinen. Entsprechend kommt klassischen Begriffen aus dem Umfeld des Designs von Benutzungsschnittstellen wie z. B. der von NORMAN geprägten *perceived affordance*, dem „wahrgenommenen Angebotscharakter“, eine große Bedeutung zu [Nor02].

Eine weitreichende Einsicht über die Metaphern-basierte Interaktion findet sich in BEDERSONs und HOLLANs Arbeit über *Pad++* [BH94], die auch im Rahmen dieser Diplomarbeit beachtet wurde: Anstatt also die üblichen Interaktionsmethoden auf dem Tablet nachzuimplementieren, sollen dem Benutzer seinen Möglichkeiten entsprechende, spezifische Interaktionen angeboten werden. Dementsprechend ist es hierbei sehr wichtig, die Brauchbarkeit und Akzeptanz durch die Nutzer dieser neuartigen Interaktionstechniken in einer Studie zu bestimmen und im Idealfall zu belegen.

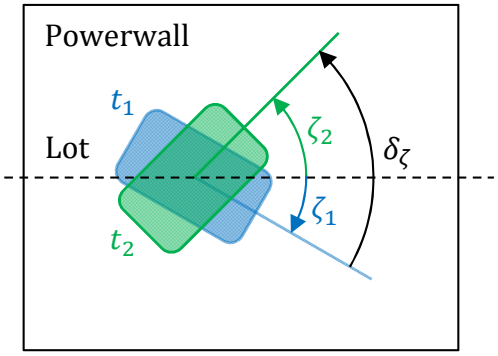
Die grundlegende Metapher im Umgang mit dem Tablet soll in dieser Arbeit ein Sichtfenster sein, durch das der Benutzer hindurchsehen kann (siehe Abbildung 3.1), jedoch mit zusätzlichen, transformativen oder semantischen Features, wie sie auch in *Augmented-Reality*-Anwendungen zu finden sein könnten. Auf dem Tablet soll er gegenüber der großen Darstellung auf der Powerwall detaillierte Informationen sehen, während die Powerwall – dank ihrer hohen Auflösung – eine bessere Übersicht über den Datensatz bieten kann. Die



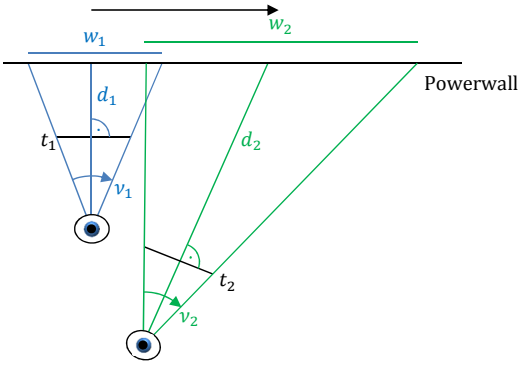

**(a)** Translation durch Bewegung des Tablets in der Horizontalen und Vertikalen.




**(b)** Skalierung durch Vor- und Rückwärtsbewegung mit dem Tablet.




**(c)** Z-Achsen-Rotation durch Drehung des Tablets.




**(d)** Translation und Skalierung durch Nutzung verschiedener Freiheitsgrade.

**Abbildung 3.2.:** Möglichkeiten zur Transformation der Ansicht durch Interaktion, die mithilfe des optischen Tracking-Systems umgesetzt wird.

Visualisierungen sollen sich so verhalten, als würde der Betrachter durch das Tablet auf die Wand sehen und könnte hierdurch zusätzliche Informationen à la *details-on-demand* sehen. Eine ähnliche Arbeit Metapher wurde von EISSELE et al. zur Visualisierung von Luftströmungen in physischen Objekten verwendet [EKE08].

Zusätzlich zur Bereicherung der Visualisierung soll das Tablet dem Benutzer erlauben, gängige Transformationen auf die Darstellung anzuwenden. Hierunter fallen vor allem die Translation der Ansicht in X- und Y-Richtung, deren Skalierung sowie ihre Rotation um die Z-Achse.

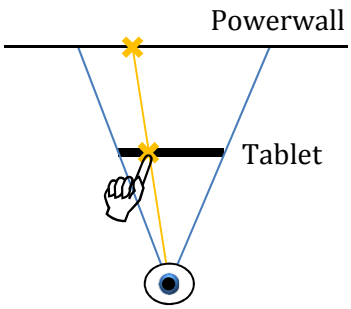



**Abbildung 3.3.:** Mittelbare Selektion eines Knotens auf der Powerwall durch den Touch-Bildschirm des Tablets.

Durch die Verbreitung von interaktiven Geräten mit Multitouch-fähigen Displays haben sich bereits bestimmte Gesten für diese Transformationen durchgesetzt. Benutzer erwarten daher, dass ihnen diese Funktionalität mithilfe der entsprechenden Gesten zur Verfügung gestellt wird. Daher bietet auch diese Software Wisch-, Zoom- und Drehungsgesten, die diese Transformationen durchführen. Da diese von der Windows-API vorgegeben, berechnet und an die Anwendung weitergereicht werden, handelt es sich dabei um technische Details, die in Abschnitt 4.7.1 näher beschrieben werden.

Allerdings wird insbesondere auch das optische Tracking-System für transformative Interaktion genutzt. Abbildung 3.2 skizziert, wie die Tracker-Interaktionen durchgeführt werden.

Die grundlegende Idee hierbei ist, dass der auf dem Tablet sichtbare Ausschnitt „festgehalten“ wird und der Benutzer sich daraufhin bewegt, um den Bereich auf eine andere Stelle der Powerwall zu transformieren. Hierzu kann er alle 6 DOFs nutzen, die ihm das Tracking-System bietet: Die Veränderung der Position in X-, Y- oder Z-Richtung und die Orientierung des Tablets, d. h. die Rotation um die drei Achsen. Dadurch kann er schnell entlegene Stellen der Visualisierung an einen für ihn sichtbaren Ort bringen, indem er auf eine interessante Stelle zeigt und sie „auf die Powerwall zieht“.

### 3.2.1. Selektion

Mithilfe des Tablets ist es möglich, auf der Powerwall dargestellte Knoten zu selektieren (siehe Abbildung 3.3). Hierzu zeigt der Benutzer mit dem Tablet in die Nähe des auszuwählenden Knotens; sobald dieser auf dem Bildschirm sichtbar ist, kann er mit einem Finger berührt werden und wird dadurch markiert. Die Selektion ist Ausgangspunkt für Details-on-Demand, die zu markierten Knoten eingeblendet werden können.

### 3.2.2. Details-on-Demand

Während auf der Powerwall die Hierarchie lediglich durch Farben dargestellt wird, könnte auf dem Tablet eine texturierte Variante der Visualisierung gezeigt werden. Die Bilder könnten beispielsweise durch eine Web-Suche nach dem einem Datenknoten angehängten Namen gefunden, heruntergeladen und für die Darstellung verwendet werden. Dieser Name könnte gleichzeitig, womöglich mit zusätzlichen Informationen, als Beschriftung bei dem Knoten zu lesen sein. Auch hier ist eine Internetsuche eine denkbare Quelle für diese Angaben.

# 4. Implementierung

In diesem Kapitel wird die Software beschrieben, die im Rahmen dieser Diplomarbeit zur Visualisierung der Daten auf der verfügbaren Powerwall geschrieben wurde. Kapitel 5 behandelt einen alternativen Ansatz dazu, der zuerst verfolgt, dann aber aufgrund von Performance-Problemen eingestellt wurde. Danach wurde ein Wechsel zu performanteren Technologien und Softwarestrukturen vollzogen, die in dem System resultierten, das in diesem Kapitel beschrieben wird.

## 4.1. Technische Grundlagen

Dieser Abschnitt stellt die verwendeten Technologien vor. Dabei werden auch die zur Verfügung stehende Hardware und der physische Aufbau rund um die Powerwall beschrieben.

### 4.1.1. Technologien

Für die Entwicklung der Software wurde das .NET-Framework 4.0 und die Programmiersprache C# verwendet. Für das Rendering wird DirectX verwendet, daher enthält der Quellcode auch Effect-Dateien, die die Vertex- und Pixel-Shader enthalten. Als GUI-Bibliothek zur Erstellung von grafischen Benutzungsschnittstellen wurde Windows Forms verwendet. Da diese Technologie von Haus aus keine Touch-Gesten unterstützt, wurde diese Funktionalität mithilfe der Windows-API nachgebildet. Die hierzu verwendeten Aufrufe sind jedoch erst ab Windows 7 verfügbar, weshalb dies die Mindestversion für den interaktiven Teil der Anwendung ist.

Der Einsatz von DirectX ermöglicht Hardwareunterstützung und flüssige Animationen, z. B. bei Interaktionen. Als Wrapper-Bibliothek für das nicht verwaltete DirectX kommt SlimDX[1] zum Einsatz. Darin enthalten sind

- Direct2D zum Zeichnen zweidimensionaler Geometrie,
- Direct3D 9–11 für das Rendering dreidimensionaler Szenen,
- DirectInput zur Verarbeitung von Eingaben durch Maus, Tastatur, Joysticks etc.,

[1] SlimDX ist ein Open-Source-Framework, das die Möglichkeiten von DirectX in einer .NET-Umgebung zur Verfügung stellt. Siehe auch http://slimdx.org/.

- DirectSound zur Wiedergabe und Anwendung von Effekten auf Musik und Aufnahmen sowie
- DirectWrite für die Darstellung von Text.

### 4.1.2. Interaktions-Hardware

Zur Interaktion mit der Visualisierung verwendet jeder Benutzer ein eigenes Tablet. Zur Verfügung standen hier zwei Geräte:

Erstens ein Microsoft Surface Pro mit einer SSD-Festplattenkapazität von 128 GB. Dieses Gerät bietet auf einem 10,6-Zoll-Bildschirm eine Full-HD-Auflösung von 1920 × 1080 Pixeln und Multitouch-Funktionalität. Als Prozessor ist ein Intel® Core™ i5 der 3. Generation verbaut, der mit zwei Kernen eine Taktfrequenz von jeweils 1.7 GHz erzielt. Zur Verfügung stehen 4 GiB Arbeitsspeicher; als Grafikkarte ist der Intel®-HD-Graphics-Chip 4000 verbaut. Als Betriebssystem wird Windows 8 Pro in der 64-Bit-Version ausgeführt. Das Gerät an sich wiegt 916 g, mit einer eventuell angeschlossenen Tastatur mit einem Eigengewicht von 218 g erhöht sich das Gewicht auf ca. 1,13 kg.

Zweitens ein Lenovo ThinkPad Tablet 2 mit einem 64-GB-eMMC-Speicher und einem Bildschirm mit einer Diagonale von 10,1 Zoll. Dieser bietet eine HD-WXGA-Auflösung von 1366 × 768 Pixeln und 5-Punkt-Multitouch-Funktionalität. Der Intel®-Atom™-Prozessor Z2760 taktet auf 1,8 GHz und besitzt zwei Kerne. Als Grafikchip ist ein Intel® HD GMA verbaut, durch den höchstens Direct3D 9c unterstützt wird. Es stehen 2 GiB Arbeitsspeicher zur Verfügung und als Betriebssystem wird Windows 8 in der 32-Bit-Version verwendet. Gegenüber dem Surface-Pro-Tablet zeichnet es sich jedoch durch ein deutlich geringeres Gewicht von lediglich 600 Gramm bzw. 580 Gramm ohne Eingabestift aus.

Die Geräte wurden mit einem Starrkörper versehen, um deren Lage und Orientierung im Raum durch ein spezielles Kamerasystem berechnen zu können; siehe hierzu Abschnitt 4.7.1.

### 4.1.3. Powerwall

Als Anzeigegerät wird eine Powerwall verwendet, wie sie in Abbildung 4.1 dargestellt ist. Dabei handelt es sich um eine von der Rückseite mit Beamern beleuchtete Wand, die auf 5,90 m Breite und 2,24 m Höhe eine Nettoauflösung von 10.800 × 4.096 Pixeln erzielt. Als Projektoren werden hier fünf DLA-SH4K „4K“ von JVC verwendet, die äußerst präzise justiert werden müssen, um über ein Spiegelsystem das Bild korrekt auf der Wand darzustellen. Um Verzerrungen entgegenzuwirken, sind ihre Linsen mit einem Gel gefüllt. Jeder dieser Beamer erzeugt ein Bild mit einer Auflösung von 2.400 × 4.096 Pixeln und die Bildbereiche von je zwei benachbarten Projektoren überlappen sich um 300 Pixel in der Horizontalen, in dem das Bild des einen Beamers ausgeblendet und das des anderen Beamers eingeblendet wird. Jeder Projektor besitzt vier Eingänge, die von den dahinterliegenden Rechnern bzw. deren Grafikkarten gespeist werden. Insgesamt sind noch fünf weitere dieser Projektoren und

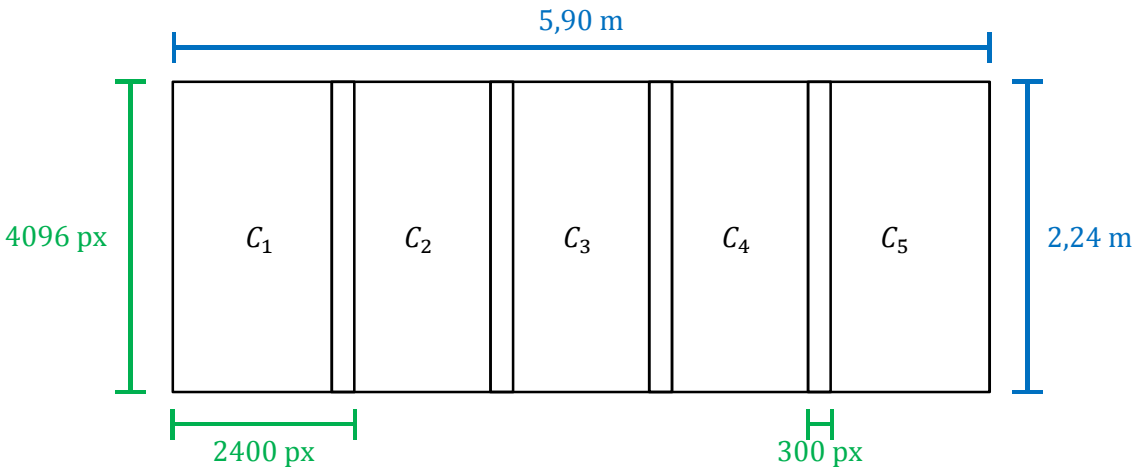



**Abbildung 4.1.:** Darstellung der VISUS-Powerwall mit den relevanten Größen.

Rechner verfügbar, wodurch stereoptische 3D-Visualisierungen ermöglicht werden, diese werden jedoch in dieser Arbeit nicht verwendet. Die netzwerkbedingte Trennung in Remote- und Client-Rechner und dieser spezielle Display-Aufbau muss natürlich im Entwurf der Software berücksichtigt werden – hierauf geht Abschnitt 4.2 ein.

Für die Interaktion ist relevant, dass sich vor der Wand eine freie Fläche befindet, sodass sich der Betrachter vor der Wand bewegen kann.

### 4.1.4. Optischer Tracker

Als optischer Tracker stand das OptiTrack-Infrarot-Kamerasystem von NaturalPoint zur Verfügung. Der Aufbau sollte geeignet sein, bestimmt ausgezeichnete Objekte im Raum vor der Powerwall nachverfolgen zu können. Daher ist das Umfeld der Powerwall mit 16 Kameras ausgestattet, von denen 14 Kameras, jeweils zwei Stück gruppiert, direkt über der Wand angebracht sind und von oben auf einen grob 4 m tiefen Bereich sehen. Jeweils eine der verbliebenen zwei Kameras befindet sich links und rechts neben der Powerwall und blickt von der Seite in den Raum. Abbildung 4.2 illustriert den Aufbau.

Das System ist imstande, mehrere Objekte gleichzeitig zu verfolgen. Hierfür wird spezielles Zubehör eingesetzt, das sich mit dem zu trackenden Objekt in der Szene befindet, diesem also in der Regel anhaftet. Es handelt sich dabei um unveränderliche Gebilde, die mit mehreren, kleinen Kugeln ausgestattet sind. Die Kugeln sind mit einem besonderen Lack überzogen, um derartige Reflektionseigenschaften aufzuweisen, dass sie von den Infrarot-Kameras optimal gesehen werden können. Abbildung 4.3 zeigt einen solchen Starrkörper.

Diese Starrkörper bzw. Rigid Bodys müssen bei unterschiedlichen Geräten auch von unterschiedlicher Gestalt sein, um gesondert erkannt und behandelt werden zu können. Jedem Gerät, das verwendet werden soll, wird ein solcher Starrkörper angeheftet.

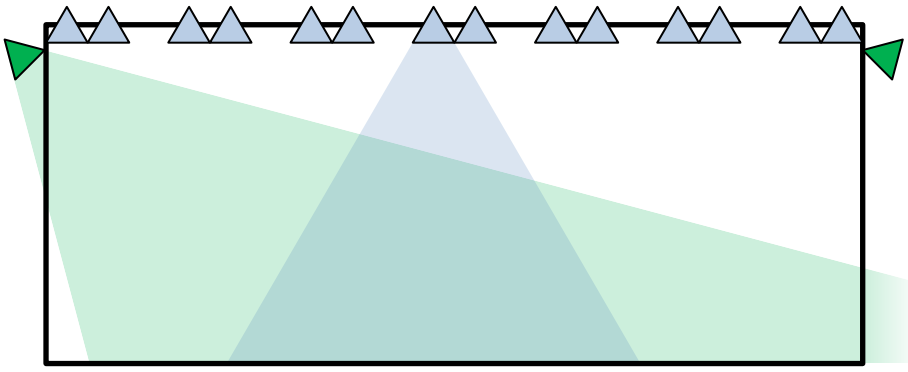


**Abbildung 4.2.:** Der Aufbau der Kameras im optischen Tracking-Systems bei der Powerwall und Andeutungen der entsprechenden Sichtkegel. Die grauen Kameras befinden sich frontal über der Projektionsfläche, die schwarzen sehen seitlich auf die Tracking-Szene.

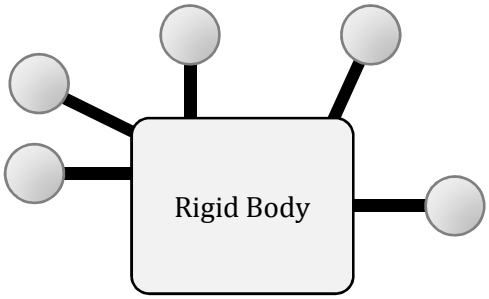



**Abbildung 4.3.:** Schematische Darstellung eines vom Kamerasystem anhand der Lage der speziell lackierten Kugeln wahrgenommenen Starrkörpers.

Auf einem eigenen Rechner im Netzwerk findet die Verarbeitung der Kamera-Rohdaten durch das OptiTrack-Motive-Softwaresystem von NaturalPoint statt. Hierfür müssen die Kameras bzw. die Software kalibriert werden, sodass später korrekte und in ihrer Größenordnung sinnvolle Werte geliefert werden können. Damit die Daten für projektive Transformationen genutzt werden können, muss die Achsenorientierung die in der Software verwendeten Koordinatensysteme geeignet darstellen. Das Koordinatensystem im kalibrierten Tracking-System wird in Abbildung 4.4 gezeigt. Dieses stimmt hinsichtlich der Achsenorientierung mit dem von Direct3D verwendeten, linkshändigen Koordinatensystem überein.

Die Ergebnisse der Bilddaten-Auswertung durch die Kamerasoftware werden über das VRPN-Protokoll im Netzwerk verteilt. Die gesendeten Daten und ihre Interpretation im entwickelten Visualisierungssystem werden in Abschnitt 4.7.1 beschrieben.

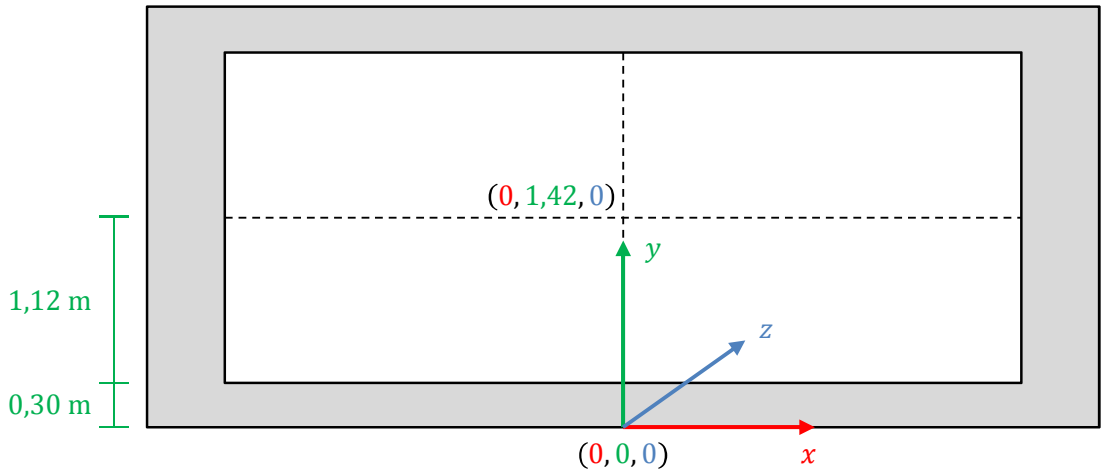



**Abbildung 4.4.:** Das Koordinatensystem, auf das das Kamerasystem kalibriert wurde. Der Ursprung liegt genau in der Mitte der Powerwall im Boden, welcher sich etwa 30 cm unter dem unteren Rand der Darstellungsfläche liegt.

### 4.1.5. Netzwerkaufbau

Die Organisation der Rechner sieht eine zentralisierte Verwaltung vor: Gesteuert werden die verbundenen Geräte und Rechner durch einen zentralen Computer, auf dem ein der Verwaltungsteil der Software läuft, der nicht zur Visualisierung von Daten, sondern zur Organisation der Netzwerkkommunikation, zur Synchronisation der Daten etc. dient. Dieser Rechner wird im Folgenden als Remote bezeichnet. Das Rendering erfolgt hingegen auf den fünf in Unterabschnitt 4.1.3 beschriebenen Computern, deren Ausgabe durch die Projektoren auf der Powerwall sichtbar wird. Diese Rechner, auf denen das Powerwall-Rendering stattfindet, werden im Kommenden auch als Clients bezeichnet. Ein weiterer Teil des Netzwerksystems sind die Tablets, die von den Benutzern zur Interaktion mit der Visualisierung genutzt werden. Abbildung 4.5 zeigt das System und die möglichen Kommunikationsrichtungen.

Die gesamte Kommunikation zwischen Clients, Remote und Tablets findet über die Netzwerkschnittstellen statt, die in Abschnitt 4.4 genauer beschrieben werden.

## 4.2. Software-Architektur

Ein beträchtlicher Aufwand floss in die Überlegung einer geeigneten Architektur für die Software, da mit dem Programmiervorhaben unterschiedliche Konzepte in einer Anwendung vereint werden sollten. Der Entwurf des Systems muss es ermöglichen, einen großen Datensatz auf den verteilten Rechnern der Powerwall für mehrere Benutzer gleichzeitig mit möglicherweise unterschiedlichen Visualisierungsmethoden darzustellen. Diese Darstellungen sollen für die Betrachter Ausgangspunkt von Interaktionsmöglichkeiten sein, die ebenfalls über das Netzwerk synchronisiert werden müssen. Als Interaktionsmedium werden

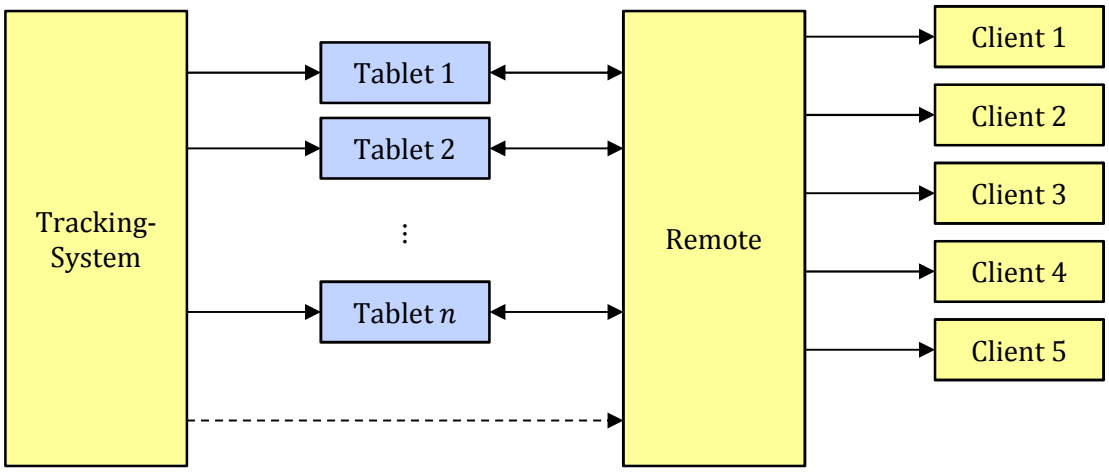



**Abbildung 4.5.:** Die Kommunikationswege im Netzwerk des gesamten Softwaresystems. Die farbliche Hinterlegung der Rechner zeigt die Aufteilung in die zwei unterschiedlichen Subnetze.

optisch getrackte Tablets verwendet, die wiederum in der Lage sind, die Visualisierungen zu rendern.

Diesen Anforderungen entsprechend weist das entstandene System eine natürliche Komplexität auf. Diese entsteht beispielsweise dadurch, dass die Visualisierungen stark an die möglichen Interaktionstypen gekoppelt sind, wobei sie völlig unterschiedliche Geometrien aufweisen können. Für Rückschlüsse der Interaktion auf die Bedeutung in einer Visualisierung muss also deren Schnittstelle geeignet entworfen werden. Andere Teile sind weniger komplex, sondern lediglich unabwendbare Aufwände, die z. B. daher rühren, dass Nachrichten vom Remote gesendet werden und clientseitig interpretiert und in entsprechende Events etc. umgesetzt werden müssen. Auch die Verwendung von Direct3D bringt einen gewissen Codeumfang für die Initialisierung und das Rendering mit sich.

Abbildung 4.6 zeigt eine Übersicht über die Komponenten des Softwaresystems sowie wichtige Klassen daraus und Verbindungen zwischen ihnen. Abschnitt A.2 gibt eine Einführung in die Verwendung der Software.

### 4.2.1. Aufgabengebiete

In diesem Abschnitt wird eine kurze Einführung in die Verteilung der Aufgaben im Hinblick auf die Softwarestruktur gegeben.

Das Rendering findet auf den Client- und Tablet-Rechnern statt. Die Rendering-Clients, die das Bild auf der Powerwall produzieren, verwenden die `ClientForm`-Klasse, die alle hierfür nötigen Direct3D-Aufrufe tätigt. Im `ClientForm` werden auch die entsprechenden Matrizen

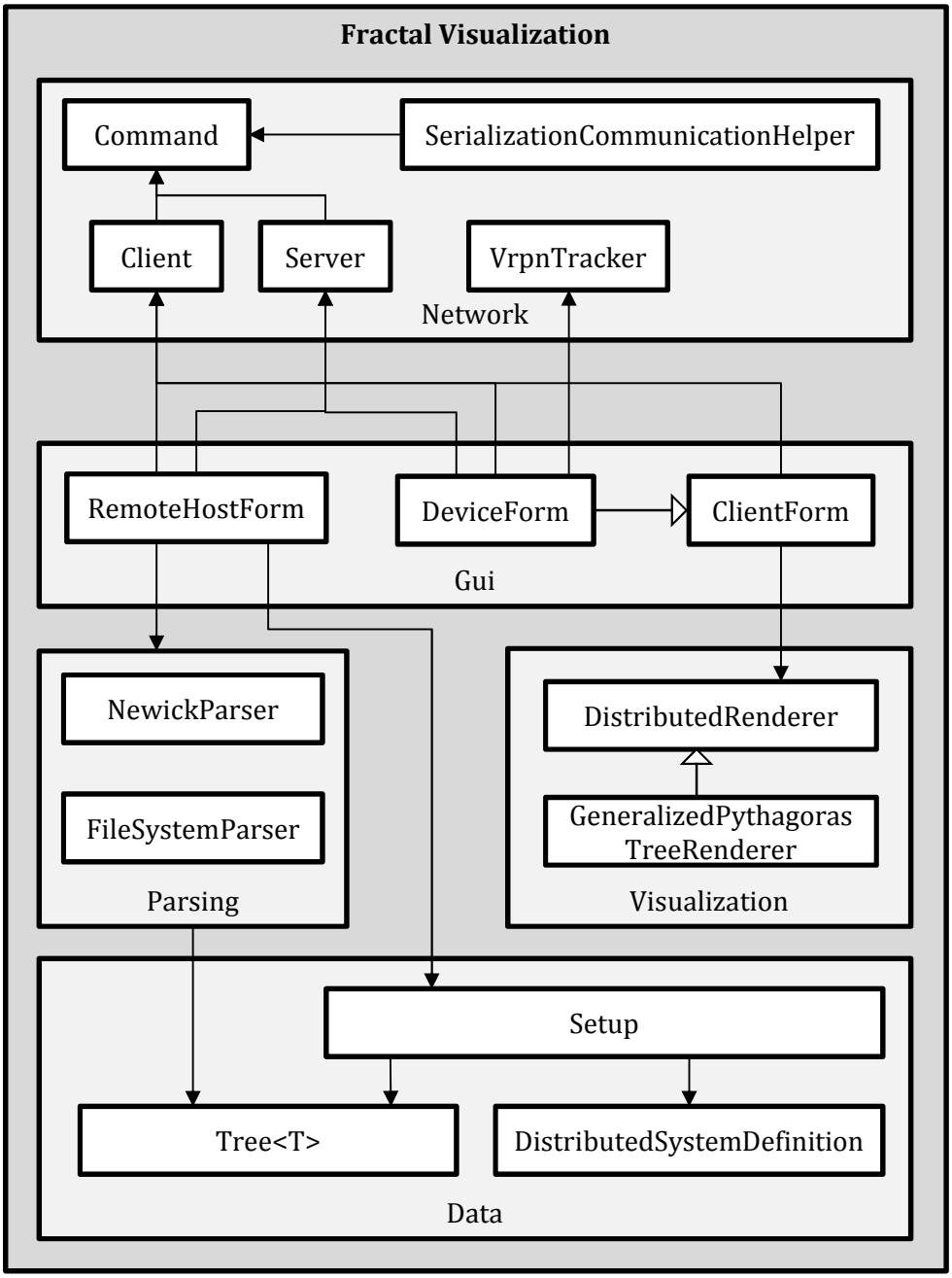



**Abbildung 4.6.:** Eine grobe Architekturübersicht über das Softwaresystem.

berechnet, die für die Darstellungen relevant sind und die `UserViews` verwaltet, durch die ein Benutzer des Systems und seine Ansicht dargestellt wird.

Das Tablet verwendet hingegen die `DeviceForm`-Klasse für das Rendering und die Interaktionen. Das `DeviceForm` ist abgeleitet vom `GestureForm`; dadurch ist es in der Lage, Gesten-Ereignisse zu empfangen und zu verarbeiten. Das `GestureForm` ist wiederum vom `ClientForm` abgeleitet, sodass die ähnlichen, aber doch nicht ganz gleichen Vorgänge beim Device-Rendering nicht ganz neu erstellt werden mussten, sondern sich teilweise von den Mechanismen des Client-Renderings ableiten. Im `DeviceForm` sind die Funktionen, die von der Basisimplementierung abweichen, entsprechend überschrieben, sodass andere Matrizen verwendet werden oder das Rendering der Benutzeransichten anders gestaltet werden kann. Auch das Empfangen und Verarbeiten von VRPN-Ereignissen, die vom optischen Tracking-System kommen, geschieht hier.

Das `RemoteHostForm` läuft auf dem Remote-Rechner und bietet keine Rendering-Funktionalität, sondern dient lediglich der Initialisierung und Synchronisation von Rendering-Clients und Tablets.

## 4.3. Daten und Parser

Das Ziel beim Lesen der Daten ist die speicherinterne Repräsentation in einer Baumstruktur. Hierfür wurde die Klasse `Tree<TData>` implementiert, die einen Knoten in einer Hierarchie darstellt. Dabei kann es sich sowohl um den Wurzel-, einen inneren oder einen Blattknoten handeln. Die hierarchische Struktur wird in dessen `Parent`- und `Children`-Eigenschaft abgebildet. Der Erweiterbarkeit wegen wurde die Klasse generisch programmiert, in den Fällen der vorliegenden Arbeit beschränkt sich die Belegung des generischen Typparameters, der die Zuordnung zusätzlicher Daten zu einem Knoten erlaubt, auf Gleitkommazahlen. Wird also im Folgenden von einem `Tree` gesprochen, ist immer die Rede von einer `Tree<double>`-Instanz.

Wie aus den beiden Fallbeispielen (siehe Kapitel 6) ersichtlich ist, können die Datenquellen je nach Datensatz variieren – im einen Fall handelt es sich um eine bestimmt formatierte Datei, im anderen Fall um eine Dateisystem-Struktur. Daher war eine praktikable Vorgehensweise, für die verschiedenen Eingabeformate jeweils eigene Parser zu erstellen, die im Falle des Dateisystems einen Ordner und seine untergeordneten Verzeichnisse und Dateien in einen `Tree` überführen (`FileSystemParser`), im Falle der phylogenetischen Taxonomie hingegen eine Datei im Newick-Format (`NewickParser` – siehe Abschnitt A.1) einlesen. Wichtig ist die konsistente Speicherhaltung, die im Endeffekt durch die `Tree`-Klasse gegeben ist. Diese ist zudem in der Lage, den durch sie dargestellten Baum wiederum in das Newick-Format zu überführen.

Da der zunächst implementierte `FileSystemParser` eine zu lange Laufzeit für den brauchbaren Einsatz hatte, wurde er in ein externes Projekt ausgelagert. Dann wurde er einmal ausgeführt und die dabei entstandene Hierarchie in eine Newick-Datei überführt.

## 4.4. Kommunikation

Vom gesamten System werden mit Ausnahme des optischen Trackers ausschließlich TCP/IP-Verbindungen benutzt. Beim Start der Anwendung auf einem Client wird in den Kommandozeilen-Parametern die IP-Adresse und der Port des Remote-Rechners angegeben, sodass der Client dorthin eine TCP/IP-Verbindung herstellen kann. Alle weiteren Daten wie z. B. Transformationen, Anweisungen nach Interaktionen etc. erhält er dann vom Remote. In Unterabschnitt 4.4.1 wird beschrieben, wie die Netzwerkkommunikation technisch realisiert wurde.

### 4.4.1. TCP/IP

Der `ClientManager` wird beim Remote mit einem bestimmten Port initialisiert und startet einen asynchronen Vorgang, der via `TcpListener.BeginAcceptTcpClient` auf diesem Port auf eingehende TCP/IP-Verbindungen horcht. Wenn ein Gerät mithilfe eines `Client`-Objekts eine Verbindung auf diesen Port herstellt, erstellt der `ClientManager` einen `Server` und löst das `ConnectionCreated`-Ereignis aus. Um leicht zu unterscheiden, ob sich ein Rendering-Knoten oder ein Tablet zum Remote verbinden möchte, werden zwei `ClientManager` verwendet, die auf unterschiedlichen Ports auf Verbindungen lauschen.

Der Remote hält sich zwei Listen von `Server`-Objekten – eine für die Clients, die andere für die Tablets – und nutzt sie, um Informationen an das Zielgerät zu schicken. Die Remote-Client-Kommunikation unterscheidet sich von der Remote-Tablet-Kommunikation dadurch, dass in letzterem Fall beide Kommunikationsrichtungen zugelassen sind. Während ein Client also vom Remote lediglich Instruktionen empfangen kann, kann ein Tablet auch dem Remote Informationen – beispielsweise zur Interaktion – zukommen lassen.

### 4.4.2. UDP

Zunächst war angedacht, frequent auftretende Daten via UDP zu broadcasten. Vor allem für die Interaktionen hätte dies Vorteile erbringen können, da so nicht vielfach pro Sekunde Pakete auf allen Rendering-Clients synchronisiert werden müssten. Jedoch birgt die Verwendung des Protokolls auch Nachteile.

Bei einer Kommunikation per UDP besteht die Möglichkeit, dass gebroadcastete Pakete verloren gehen. Aufgrund der Kommunikationsdichte bei Transformationen hätte das am wahrscheinlichsten zur Folge gehabt, dass eine Verschiebung, Skalierung oder Rotation nicht durchgehend von allen Clients verarbeitet wird und sich inkonsistente Transformationen auf den verschiedenen Display-Teilen ergeben. Als Lösung hierfür käme eine Synchronisierung mit dem Remote-Rechner in Frage, die von Zeit zu Zeit ausgeführt würde.

Darüber hinaus bringt es der komplexe Netzwerkaufbau (siehe Abbildung 4.5) mit sich, dass die Rendering-Rechner in mehreren Netzwerken gleichzeitig vertreten sind. Laut einigen Tests, die institutsintern durchgeführt wurden, bringt die Verwendung von UDP-Broadcastings

aufgrund des Multihomings keinen Performance-Vorteil gegenüber synchronisierten TCP/IP-Paketen. Außerdem befinden sich die Tablets in einem anderen Subnetz, weshalb mit IPv4 Broadcasts an die Rendering-Clients gar nicht ohne Weiteres möglich gewesen wären.

Aus diesen Gründen wurde ganz auf Kommunikation via UDP verzichtet und ausschließlich der Datenaustausch via TCP/IP umgesetzt.

### 4.4.3. VRPN

Von der Software des optischen Trackers wird VRPN, ein spezielles Virtual-Reality-Protokoll, verwendet. Nähere Informationen hierzu finden sich im Implementierungsabschnitt über das Tracking (siehe 4.7.1).

### 4.4.4. Datenaustausch

Entsprechend obiger Erläuterung gibt es grundlegend zwei Arten, wie im System kommuniziert werden. Da es sich beim Austausch via VRPN um Aufrufe an eine bestehende Bibliothek handelt, wird in diesem Abschnitt ausschließlich die Kommunikation über TCP/IP vorgestellt. Eine Übersicht über die Kommunikationsrichtungen findet sich in Abbildung 4.5.

Da es sich bei den Nachrichten um Instruktionen für den empfangenden Rechner handelt, wurde die Klasse `Command` implementiert, die auf einer Abbildung von Schlüsseln auf Objekte basiert. Um klarzumachen, wie ein empfangener `Command` zu interpretieren ist, gibt es unterschiedliche `CommandTypes`. Tabelle 4.1 zeigt, welche Befehle mit welchen Daten verschickt werden.

Einen Sonderfall in der Liste nimmt der `ExecutionRequest`-Befehl ein. Dieser wird vom Tablet an den Remote gesendet und enthält eine Instruktion, die es auf den Clients auszuführen gilt. Dieser Umweg über den Remote als Moderator ist notwendig, da die Tablets keine eigenen Verbindungen zu den Clients halten. So kann der Moderator die Steuerung übernehmen und den beinhalteten Befehl an alle Clients korrekt weiterleiten. Die `ExecutionRequest`-Befehle sind besonders performancekritisch, da in einer Schleife der beinhaltete Befehl über alle Rendering-Client-Verbindungen übertragen wird.

Im üblichen Programmablauf wird jedem Rendering-Client initial der Gesamtaufbau der Powerwall und die Lage des von ihm zu rendernden Bildausschnittes bekannt gemacht mit dem `InitializeClient`-Befehl. Dabei wird auch der vom Remote-Rechner geladene Datensatz an den neu verbundenen Client übermittelt sowie die zu verwendende Visualisierung und deren Parameter. Auch die Tablets werden entsprechend (`InitializeClient` und `InitializeDevice`) initialisiert. Im explorativen Betrieb, in dem die Benutzer durch das Tablet mit der Visualisierung interagieren, werden hauptsächlich `Transformation-Commands` versendet. Beim Beenden der Remote-Anwendung wird ein `Shutdown`-Befehl an alle Clients verschickt.

| Typ | Beschreibung | Eigenschaften |
|---|---|---|
| `InitializeClient` | Initialisiert einen Client mit dem gesamten Setup des Systems und den Visualisierungsinformationen. Auch die Tablets werden hiermit initialisiert, wobei aber die Systemdefinition und die Lage des Fensters ignoriert werden. | `Setup`, `RendererType` |
| `InitializeDevice` | Initialisiert ein Interaktions-Tablet und weist ihm die Ansicht zu. Hierbei werden zusätzliche Informationen zum Tablet gesendet. | `ViewIdentifier`, `TrackerConnection`, `RigidBodyOffset` |
| `Shutdown` | Beendet die Anwendung auf dem Zielgerät. | (keine) |
| `DataUpdated` | Zeigt an, dass ein neuer Datensatz geladen werden soll. | `Data` |
| `SystemDefinition` | Weist auf die Änderung der Definition des verteilten Systems hin. | `SystemDefinition`, `DisplayPosition` |
| `Snapshot` | Erstellt einen Wiederherstellungspunkt für eine Ansicht. | `ViewIdentifier` |
| `Transformation` | Zeigt den Clients die Veränderung der angewandten Transformation für eine Ansicht an. | `Translation`, `Scaling`, `Rotation`, `ScalingOperation`, `ViewIdentifier` |
| `ExecutionRequest` | Wird zur Weiterleitung von `Commands` von einem Tablet an den Remote gesendet, der den enthaltenen Befehl an alle Clients weiterreichen soll. | `RequestedCommand` mit dessen untergeordneten Eigenschaften |
| `ReorderViews` | Veranlasst die Umsortierung der Benutzeransichten. | `ViewOrder` |

**Tabelle 4.1.:** Die verschiedenen Arten von Befehlen, die über das Netzwerk verschickt werden können.

## 4.5. Visualisierungen

Die implementierte Software beschränkt sich in ihren Visualisierungskomponenten auf Hierarchievisualisierung, d. h. dass der zugrundeliegende Datentyp immer ein `Tree` ist. Entsprechend verlangt die Klasse `DistributedRenderer`, die die Basisklasse aller Visualisierungen ist, ein `Setup` mit einem hierarchischen Datensatz. Für zukünftige Arbeiten könnte eine Verallgemeinerung auf jede Art von Daten leicht dadurch erreicht werden, dass diese Beschränkung aufgehoben und stattdessen `object` als Typ verwendet wird. Um eine Visualisierung darzustellen, wird dem Renderer das `Setup` übergeben, welches insbesondere auch den zu verwendenden Datensatz enthält. Darin ist alles enthalten, was von der Visualisierung benö-

tigt wird, um die benötigte Geometrie zu erzeugen. Mithilfe der `GetVertices`-Methode des `DistributedRenderers` werden alle Vertizes der Visualisierungsgeometrie berechnet und als Array zurückgegeben.

### 4.5.1. Verallgemeinerte Pythagoras-Bäume

Im Software-System ist zwar die Nutzung unterschiedlicher Visualisierungsmethoden für jeden Benutzer konzeptionell vorgesehen, jedoch wurde nur eine Visualisierungstechnik umgesetzt: Die verallgemeinerten Pythagoras-Bäume.

Verallgemeinerte bzw. „generalisierte“ Pythagoras-Bäume stellen eine Verallgemeinerung des Konzepts gewöhnlicher Pythagoras-Bäume dar. Während letztere sich in jeder Stufe in zwei Unterstrukturen aufgliedern und so höchstens zur Darstellung binärer Hierarchien zu verwenden sind, erlaubt die Generalisierung auf jeder Stufe eine $n$-äre Aufteilung. Hierbei gibt es verschiedene visuelle Parameter, die variiert werden können, beispielsweise:

**Welchen Anteil erhält eines der $n$ Rechtecke an den $180°$ des Halbkreises?** Hier wird eine einfache Metrik verwendet: Die Größe eines Unterbaums, verglichen mit den Größen seiner Geschwisterknoten, ist ausschlaggebend für den Winkelanteil, der einem Rechteck zuteilwird. Im kommenden Abschnitt zur Geometrie wird dieser Aspekt genauer beleuchtet.

**Welche Höhe erhält ein Rechteck?** Die Höhe kodiert in dieser Implementierung keine zusätzliche Information, da sie mit der Breite des Rechtecks, also der anteiligen Bogensehnenlänge, gleichgesetzt wird. So entstehen immer Quadrate und die der Berechnung der Breite zugrundeliegende Metrik ist in zwei visuellen Attributen der Visualisierung vertreten.

**Mit welcher Farbe wird ein Rechteck dargestellt?** Von dieser Software wird eine einfache Metrik zur Einfärbung der Quadrate verwendet, die die hierarchische Tiefe des Knotens repräsentiert. Der Abschnitt zu Farbskalen erklärt die Metrik und die dahinter stehende Implementierung.

**Wie werden die Teilbäume sortiert?** Die Frage nach einer geeigneten Sortierung der Teilbäume ist gleichzeitig die algorithmische Problemstellung zur Reduktion von Überlappungen im Baum und wird unten (siehe Abschnitt „Überlappungen“) genauer beschrieben.

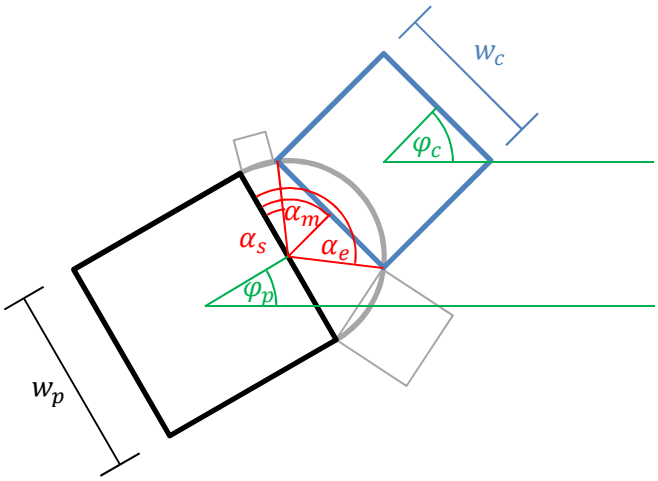



**Abbildung 4.7.:** Die relevanten Winkelgrößen (rot) bei der Berechnung der Geometrie eines Kindknotens $c$ (blau) relativ zu dessen Elternknoten $p$ (schwarz).

### Geometrie

Die Geometrie der dargestellten Baumknoten wird ohne rechenintensive Matrixmultiplikationen berechnet. Als Ausgangspunkt für die Rekursion durch die Baumstruktur wird die Wurzel der Hierarchie konstant positioniert. Ihr Mittelpunkt ist stets $\mathbf{m}_{\text{root}} = (0,\, 0)^T$ und ihre Seitenlängen betragen $h_{\text{root}} = w_{\text{root}} = 1$, sodass sich ein Quadrat ergibt. Für jeden Knoten wird eine Orientierung angegeben, die relativ zur Welt-Abszissenachse im mathematisch positiven Sinn nach oben zeigt. Im Falle der Wurzel, aus der die Knoten „nach oben wachsen“, liegt dieser Wert entsprechend bei $-\frac{\pi}{2}$ rad. Die rekursive `PrepareChildren`-Methode wird zunächst für die Wurzel aufgerufen. Sie positioniert alle Kinder relativ zum bereits positionierten, übergebenen Knoten und ruft sich dann für alle Kindknoten wieder auf. Ihre Hauptaufgabe besteht also darin, die Kinder auf dem Halbkreis über dem ausgerichteten Elternknoten zu verteilen. Abbildung 4.7 zeigt die hierfür relevanten Größen und ihre geometrische Bedeutung. Im Folgenden werden Winkel, die relativ zur Abszissenachse zu verstehen sind, stets mit $\varphi$ bezeichnet, Winkel relativ zur Orientierung eines Knotens hingegen mit $\alpha$.

Nicht jedes Kind $c$ (*child*) bekommt einen gleich großen Anteil am 180°-Winkelbereich des Halbkreises. Der tatsächliche Anteil errechnet sich aus der Größe des Kindes $S_c$ relativ zur gesamten Unterbaumgröße $S_p$ des Elternknotens $p$ (*parent*). Die Größe eines Knotens zählt auch den Knoten selbst mit, weshalb immer $S_p \geq 1$ bzw. $S_c \geq 1$ gilt. Dieser Winkelanteil am Halbkreis errechnet sich zu $\frac{S_c}{S_p - 1} \cdot 180°$, was bedeutet, dass ein Kind mit einem großen Unterbaum mehr Platz eingeräumt bekommt als Geschwister mit kleineren Unterbäumen. Dasselbe gilt für die Höhe des Rechtecks, mit dem der Kindknoten dargestellt wird – in diesem Fall wird vereinfachend die Breite des Rechtecks auch für dessen Höhe verwendet; es entstehen also ausschließlich Quadrate. Für jedes Kind $c$ werden der Anfangs- und Endwinkel $\alpha_s,\, \alpha_e \in [0, \pi]$ sowie der Mittelwinkel $\alpha_m = \frac{\alpha_s + \alpha_e}{2}$ berechnet. Liegt der Wert eines Winkels bei 0, so handelt es sich um den Anfang des Halbkreises, bei $\pi$ um dessen Ende. Diese Winkel

sind also relativ zur Objekt- bzw. Halbkreisgeometrie; dadurch wird verhindert, dass sich Floating-Point-Rundungsfehler, die bei der Geometrieberechnung gemacht werden, durch die Hierarchie propagieren. Stattdessen stellt die Vorgehensweise sicher, dass der erste und letzte Kindknoten exakt am Anfang bzw. Ende des Halbkreises positioniert werden. Aus den Anfangs- und Endwinkeln $\alpha_s$ und $\alpha_e$ lässt sich die Breite des Kindknotens ausrechnen, die der Länge der Halbkreissehne zwischen den Winkeln entspricht. Diese berechnet sich als $w_c = w_p \cdot \sin\left(\frac{\alpha_e - \alpha_s}{2}\right)$.

Für die Orientierung werden wieder Winkel relativ zur Abszisse benötigt. Aus der Ausrichtung des Elternknotens $\varphi_p$ und dem Mittelwinkel lässt sich die Orientierung des Kindknotens berechnen als $\varphi_c = \varphi_p + \alpha_m - \frac{\pi}{2}$. Aufwändiger gestaltet sich die Berechnung der Mittelpunkte und der Eckpunkte der Rechtecke.

**Mittelpunkte** Für einen Kindknoten wird der Mittelpunkt $\mathbf{m}_c$ ausgehend von $\mathbf{m}_p$ als Summe dreier Verbindungsvektoren berechnet (siehe Abbildung 4.8), nämlich jeweils zwischen

- $\mathbf{m}_p$ und dem Halbkreismittelpunkt ($\mathbf{d}_1$),
- dem Halbkreismittelpunkt und dem Schnitt in Mittelwinkelrichtung mit der Sehne, also der unteren Kante von $c$ ($\mathbf{d}_2$) und
- der Sehnenmitte und $\mathbf{m}_c$ ($\mathbf{d}_3$).

Aus der Geometrie heraus ergeben sich für die Verbindungen folgende Vektoren:

$$\begin{aligned}
\mathbf{d}_1 &= \begin{pmatrix} \frac{h_p}{2} \cdot \cos(\varphi_p) \\ \frac{h_p}{2} \cdot \sin(\varphi_p) \end{pmatrix} \\
\mathbf{d}_2 &= \begin{pmatrix} h_i \cdot \cos(\varphi_m) \\ h_i \cdot \sin(\varphi_m) \end{pmatrix} \qquad \text{mit } h_i = \frac{w_c}{2 \cdot \tan(\alpha_e - \alpha_m)} \text{ und } \varphi_m = \varphi_c \\
\mathbf{d}_3 &= \begin{pmatrix} \frac{h_c}{2} \cdot \cos(\varphi_c) \\ \frac{h_c}{2} \cdot \sin(\varphi_c) \end{pmatrix}
\end{aligned}$$

Entsprechend obiger Erläuterung erhält man $\mathbf{m}_c = \mathbf{m}_p + \mathbf{d}_1 + \mathbf{d}_2 + \mathbf{d}_3$.

**Eckpunkte** Die Eckpunkte werden berechnet, wenn die `GetVertices`-Methode aufgerufen wird. Dabei wird eine zusammenhängende Geometrie erzeugt, die in einen Direct3D-Vertex-Buffer geschrieben und auf die Grafikkarte geladen wird, wo sie dann für das Rendering verwendet wird.

Ähnlich der Berechnung der Mittelpunkte wird auch hier eine Summe von Verbindungsvektoren gebildet, die aus der Mitte $\mathbf{m}$ eines Knotens über die Ordinaten- und Abszissenachse des Objekts zu einem der Eckpunkte führen. Die Berechnung der horizontalen Koordinaten entspricht dem gängigen Verständnis, bei der vertikalen Koordinate muss jedoch beachtet werden, dass die Ordinatenachse am Bildschirm in der Regel nach unten wächst. Dies führt

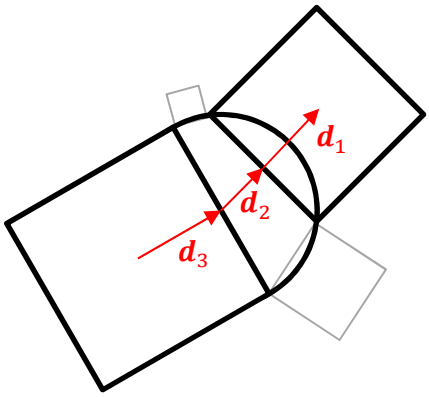



**Abbildung 4.8.:** Verbindungsvektoren $d_1$, $d_2$ und $d_3$ zur Berechnung des Mittelpunkts des Kindknotens relativ zu den Größen des Elternknotens.

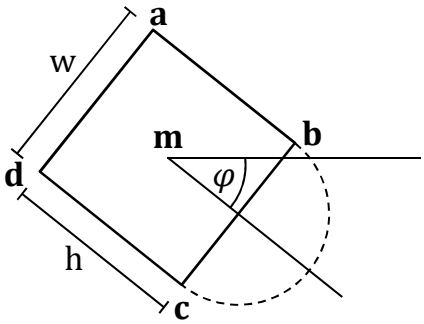



**Abbildung 4.9.:** Darstellung eines Knotens in einem verallgemeinerten Pythagoras-Baum mit den Parametern, aus denen die Eckpunkte errechnet werden.

in diesem Fall zu einer Vorzeichenvertauschung des Kosinus-Terms. Ausgehend von der Eckpunktbenennung **a**, **b**, **c** und **d** gemäß Abbildung 4.9 berechnen sich ihre Koordinaten folgendermaßen:

$$
\begin{aligned}
\mathbf{a} &= \begin{pmatrix} x_\mathbf{m} - \frac{h}{2} \cdot \cos\varphi + \frac{w}{2} \cdot \sin\varphi \\ y_\mathbf{m} - \frac{h}{2} \cdot \sin\varphi - \frac{w}{2} \cdot \cos\varphi \end{pmatrix} \\
\mathbf{b} &= \begin{pmatrix} x_\mathbf{m} + \frac{h}{2} \cdot \cos\varphi + \frac{w}{2} \cdot \sin\varphi \\ y_\mathbf{m} + \frac{h}{2} \cdot \sin\varphi - \frac{w}{2} \cdot \cos\varphi \end{pmatrix} \\
\mathbf{c} &= \begin{pmatrix} x_\mathbf{m} + \frac{h}{2} \cdot \cos\varphi - \frac{w}{2} \cdot \sin\varphi \\ y_\mathbf{m} + \frac{h}{2} \cdot \sin\varphi + \frac{w}{2} \cdot \cos\varphi \end{pmatrix} \\
\mathbf{d} &= \begin{pmatrix} x_\mathbf{m} - \frac{h}{2} \cdot \cos\varphi - \frac{w}{2} \cdot \sin\varphi \\ y_\mathbf{m} - \frac{h}{2} \cdot \sin\varphi + \frac{w}{2} \cdot \cos\varphi \end{pmatrix}
\end{aligned}
$$

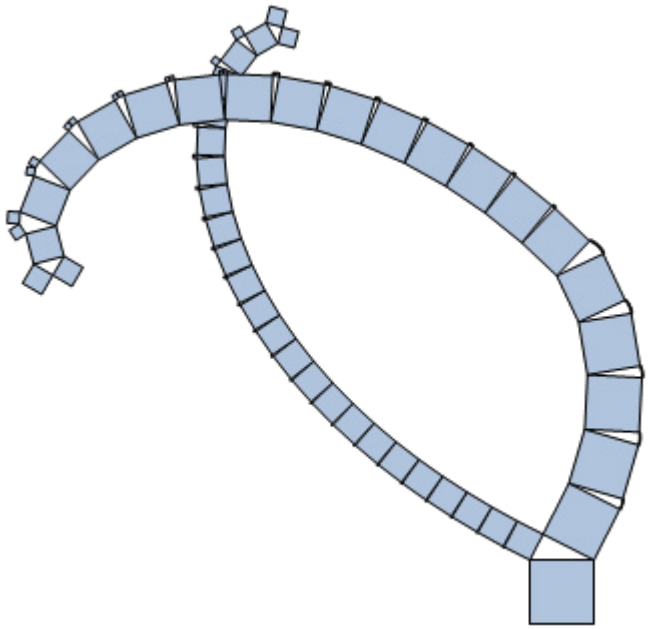


**Abbildung 4.10.:** Überlappung in einem generalisierten Pythagoras-Baum durch einseitenlastige Hierarchien.

**Parallelität** Die Mittel- und Eckpunkte der Vertizes können problemlos parallel [Tou10] berechnet werden, da im `GeneralizedPythagorasTreeRenderer` die Quadrate bereits in einer speziellen Datenstruktur als Array vorliegen. Jedoch muss hierbei darauf geachtet werden, dass die `GeneralizedPythagorasTreeNodes` ordnungserhaltend in den Vertex-Buffer übertragen werden. Dies ist relevant, da verallgemeinerte Pythagoras-Bäume im Allgemeinen nicht überlappungsfrei sind und die Ordnung der Vertizes im Vertex-Buffer die Zeichenreihenfolge bestimmt. Wenn die parallele Vorgehensweise auf verschiedenen Rechnern zu verschiedenen Ergebnisordnungen geführt hat, können Knoten sichtbar bzw. unsichtbar werden, wenn die Darstellung Powerwall-Client-berichsübergreifend transliert wird.

Die Implementierung der Parallelität brachte tatsächlich die erhoffte Beschleunigung. Hierbei wurde die `System.Threading.Tasks.Parallel.For`-Methode benutzt, die unterschiedliche Iterationen auf verschiedenen Prozessorkernen ausführt. Im Gegensatz zur seriellen Methode, bei der die Erzeugung der Vertizes nacheinander abgearbeitet wurde und dabei Zeiten im Bereich von 1–2 Minuten erzielte, benötigt das parallele Verarbeiten lediglich zirka 10–15 Sekunden. Dies entspricht der erwarteten Dauer, da die Powerwall-Rechner über zwei Intel®-Xeon®-X5650-Prozessoren mit jeweils sechs Kernen verfügen und sich die Zeit entsprechend auf gut ein Zwölftel reduzieren sollte.

#### Überlappungen

Die Implementierung des `GeneralizedPythagorasRenderers` nimmt zurzeit keine eigenständigen Umsortierungen in der Datenstruktur vor, die ihm übergeben wird. Die gezeigte Ordnung ist also „willkürlich“ in dem Sinne, dass die gegebene Ordnung beibehalten wird. Diese wird im Softwaresystem vom jeweiligen Parser angelegt und spiegelt beispielsweise die Sortierung der Teilbäume wider, wie sie in der geladenen Datei angetroffen wurde.

Aufgrund der im Gegensatz zu klassischen Pythagoras-Bäumen nicht mehr binären Unterteilung, den unterschiedlichen Größe der Knoten und den damit verbundenen Winkeln kann es zu Überlappungen in der Darstellung kommen, die aus naheliegenden Gründen zu vermeiden sind. Das passiert beispielsweise dann, wenn die Hierarchie in einem Unterbaum sehr einseitig geformt ist (siehe z. B. Abbildung 4.10). Es gibt gegenwärtig keine veröffentlichten Arbeiten, die dieses Problem adressieren und zu lösen versuchen, aber möglicherweise handelt es sich um ein NP-schweres Problem und es existiert keine einfache Lösung. Entsprechend würde ein solcher Versuch den Umfang dieser Arbeit übersteigen.

#### Stabilität

Bei verallgemeinerten Pythagoras-Bäumen handelt es sich im Sinne von BÖHRINGER und PAULISCH um eine *dynamisch instabile* Layoutmethode, da sich kleine bzw. lokal begrenzte Änderungen im Datensatz global auf die Darstellung auswirken [BP90]. Das liegt daran, dass sich die Geometrie sowohl aus den Elternknoten als auch aus den Geschwisterknoten ergibt. Zudem ist ein Knoten in der hier verwendeten Metrik in seiner Größe abhängig von der Anzahl der Knoten in seinen Unterbäumen. Die Geometrie eines einzelnen Knotens der Hierarchie hängt also von allen anderen Knoten der Hierarchie ab, sowohl nach oben und unten (Eltern- und Kindknoten) als auch nach links und rechts (Geschwister). In Abbildung 5.5 wird dieses Problem im Zusammenhang mit dem interaktiven Auf- und Zuklappen von Teilhierarchien anhand von Vergleichsbildern verdeutlicht.

#### Farbskalen

Wie eingangs bereits erwähnt lassen sich für die Einfärbung der Rechtecke verschiedene Farbskalen finden, die je nach Anwendungsfall unterschiedlich gut geeignet sein oder variiert werden können. Abbildung 4.11 zeigt Beispiele für Skalen, deren Eignung abhängig vom Szenario beurteilt werden muss. In dieser Arbeit wird eine einfache Farbskala, nämlich die in Abbildung 4.11 in der Mitte gezeigte, verwendet, durch die die Tiefe eines Knotens kodiert wird. Dabei werden die Knoten, die der Wurzel nahe sind, rot bzw. rötlich eingefärbt; je tiefer ein Knoten sich in der Hierarchie befindet, desto weiter geht seine Farbe ins Gelbliche. Der in einer Hierarchie am tiefsten gelegene Knoten ist komplett gelb. Die kräftigen Farben heben sich gut vom schwarzen Hintergrund der Darstellung auf der Powerwall und den Tablets ab. Besonders die sehr tief gelegenen, gelb eingefärbten Knoten haben dazu einen starken Kontrast und stechen dem Betrachter ins Auge.

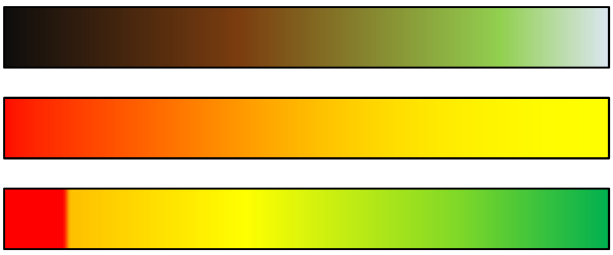


**Abbildung 4.11.:** Drei Beispiele für Farbskalen, oben die „linear optimale“ wie beispielsweise verwendet in [DBB10], in der Mitte eine einfache Farbskala, wie sie in dieser Arbeit verwendet wurde, und unten eine Skala zur Hervorhebung von Schwellwerten, die bei der 10-Perzentile einen Farbsprung macht.

Die Bestimmung der Farbe geschieht bei der Erzeugung der Vertizes, also in der `GetVertices`-Methode. Anhand der zuvor bestimmten Maximaltiefe des geladenen Datensatzes wird eine Farbtabelle, dargestellt durch die `ArrayColorTable`-Klasse, mit der Beschreibung des gewünschten Farbverlaufs erstellt. Die Farbtabelle stellt dann für den diskretisierten Zugriff via Index, der der Tiefe des einzufärbenden Knotens entspricht, die jeweilige Farbe aus dem Farbverlauf bereit.

## 4.6. Rendering

Dieser Abschnitt beschreibt die Rendering-Techniken und das Prinzip von Ansichten, wie sie in dieser Software verwendet werden.

### 4.6.1. Ansichten

Jeder Benutzer, für das System identifiziert durch die Bezeichnung des von ihm verwendeten Geräts mit optischem Tracker, besitzt eine eigene Ansicht auf den Datensatz auf der Powerwall. Mit dieser kann er durch sein Tablet interagieren. Ein solcher `UserView` enthält einige Daten, die für das Direct3D-Rendering relevant sind. Der Client iteriert beim Rendern über alle ihm bekannten Ansichten und zeichnet den Arbeitsbereich jedes Benutzers. Dementsprechend muss sich ein `UserView` auf jeden Fall seine spezielle Geometrie zur Verfügung stellen. Im Kontext von Direct3D bedeutet das, dass in jeder Ansicht das `VertexBufferBinding` zur Geometrie der Ansicht gespeichert ist. Außerdem sind hier die Transformationen definiert, die – resultierend in der View-Transformationsmatrix $\mathbf{V}$ – auf die Ansicht angewendet werden sollen.

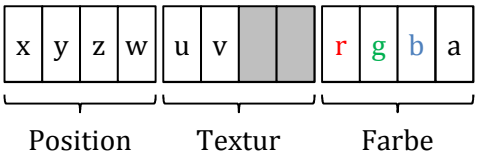



**Abbildung 4.12.:** Der Aufbau eines Vertex, wie er im Speicher abgelegt wird. Die grauen Felder werden zurzeit nicht verwendet.

### 4.6.2. Client-Rendering

Dieser Abschnitt beschreibt das Rendering auf den Powerwall-Rechnern. Die Art und Weise, wie auf den Interaktions-Tablets die Visualisierungen dargestellt werden, ist hiermit verwandt, aber insbesondere im Hinblick auf die Transformationen nicht identisch. Die Eigenheiten des Device-Renderings werden in Unterabschnitt 4.6.3 beschrieben.

Für das Rendering wird SlimDX, eine .NET-Wrapper-Bibliothek für DirectX, verwendet. Von jeder Visualisierung kann mithilfe der `GetVertices`-Methode des `DistributedRenderers` die gesamte Geometrie erfragt werden, die als Liste von Dreiecken ausgegeben wird. Ein `Vertex` (siehe Abbildung 4.12) besitzt dabei drei Felder mit den dazugehörigen Eingabesemantiken für das `InputLayout`:

1. `POSITION`: Ein Vektor $\mathbf{p} = (x, y, z, w)^T$, der die Lage des Vertex in homogenen Koordinaten beinhaltet.
2. `TEXCOORD`: Ein Quadrupel $\mathbf{t} = (u, v, \cdot, \cdot)$, dessen $u$- und $v$-Komponenten die Texturkoordinaten in horizontaler bzw. vertikaler Richtung enthalten. Die anderen beiden Felder sind reserviert, werden zurzeit allerdings nicht verwendet. Hier könnte in zukünftigen Versionen beispielsweise der Selektionszustand gespeichert werden.
3. `COLOR`: Ein Quadrupel $\mathbf{c} = (r, g, b, \alpha)$, der die Farbkanäle Rot, Grün und Blau sowie einen Faktor $\alpha$ für die Deckkraft der Farbe beinhaltet.

Die genaue Anordnung dieser Felder ist sehr wichtig, da vom Direct3D-`InputAssembler`, der die Vertizes später verarbeitet, eine bestimmte (jedoch vom Entwickler festzulegende) Reihenfolge erwartet wird. Jede einzelne dieser Komponenten wird in einfacher Gleitkommagenauigkeit mit 32 Bit Speicherbedarf abgelegt. Somit ergibt sich pro Vertex ein Speicherbedarf von $m_v = 3 \cdot 4 \cdot 4\ \text{B} = 48\ \text{B}$. Da nicht a priori klar ist, ob eine Geometrieoptimierung, beispielsweise durch aneinandergereihte Dreiecke, vorgenommen werden kann, wird immer von der Topologie einer Dreiecks-Liste ausgegangen. Zukünftige Erweiterungen könnten vorsehen, dass vom `DistributedRenderer` angegeben wird, welche Geometrietopologie geliefert wird (siehe Unterabschnitt 4.9.1).

Da für jeden Benutzer eine eigene Darstellung gerendert wird, mit der er interaktiv arbeiten kann, ergibt sich für jeden Benutzer eine eigene View-Transformation $\mathbf{U}$. Abgesehen von Kollaborations-Szenarien sollten sich die Ansichten der verschiedenen Benutzer im Regelfall

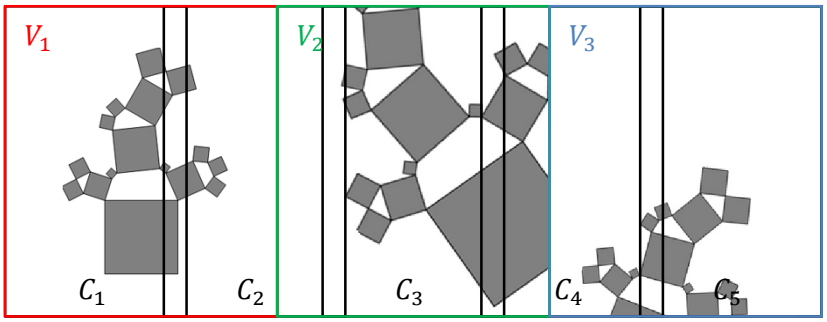



**Abbildung 4.13.:** Drei Benutzeransichten und die Verteilung der zugehörigen Arbeitsbereiche auf der Powerwall. An den Bereichsgrenzen werden die Darstellungen abgeschnitten.

nicht überlappen, weshalb sie durch Arbeitsbereiche wie in Abbildung 4.13 auf verschiedene Orte der Powerwall verteilt sind. Dieser Arbeitsbereich wird pro `UserView` durch die `WorkspaceBounds`-Eigenschaft definiert. Dabei handelt es sich um ein Rechteck, das in Pixelkoordinaten relativ zur oberen linken Ecke der Powerwall angibt, wo der Bereich eines Benutzers beginnt und welche Größe er hat. Beim Erstellen der Ansichten wird der verfügbare Platz im verteilten System mithilfe der `ResizeViews`-Methode im `ClientForm` zu gleichen Anteilen auf alle Benutzer verteilt; dabei werden ausgeblendete Ansichten ignoriert. Wenn $B = (x_B, y_B, w_B, h_B)$ die Bounding-Box des Zielsystems in Pixelkoordinaten darstellt und $n_V$ die Anzahl der sichtbaren Ansichten ist, ergibt sich der Arbeitsbereich $W_i$ des $i$-ten Benutzers zu

$$W_i = \left( i \cdot \frac{w_B}{n_V}, 0, \frac{w_B}{n_V}, h_B \right).$$

Dieser wird beim Rendern um die negativen linken und oberen Koordinaten des Rendering-Client-Bereichs $(-x, -y)$ verschoben, wodurch die Transformationen des Arbeitsbereich-Rechtecks in die Client-Koordinaten transliert werden.

Eine erste Translation verschiebt die Mitte der Geometrie in den Ursprung, sodass der Visualisierungsmittelpunkt und nicht der (möglicherweise willkürlich gewählte) Ursprung der Geometrie das Zentrum für benutzerinduzierte Transformationen ist. Wenn $G = (x_G, y_G, w_G, h_G)$ die Bounding-Box der Geometrie ist, leistet diese Verschiebung also

$$\mathbf{T}_G = \mathbf{T}\left( -\left( x_G + \frac{w_G}{2} \right), -\left( y_G + \frac{h_G}{2} \right) \right).$$

Danach können alle Benutzer-Transformationen angewandt werden, bevor die Vertizes durch die Anwendung der Tile-Transformation $\mathbf{M}_i$ in das Viewport-Koordinatensystem überführt werden. Diese benutzerdefinierten Teiltransformationen einer Ansicht sind eine Skalierung um die Faktoren $x_\mathbf{s}^\mathbf{U}$ und $y_\mathbf{s}^\mathbf{U}$ in horizontaler bzw. vertikaler Richtung, eine Rotation um die Z-Achse um den Winkel $\varphi^\mathbf{U}$ sowie eine Verschiebung der Ansicht in Pixelkoordinaten um $x_\mathbf{t}^\mathbf{U}$ und $y_\mathbf{t}^\mathbf{U}$ in X- bzw. Y-Achsen-Richtung:

$$\begin{aligned}\mathbf{U} &= \mathbf{R}\cdot\mathbf{S}\cdot\mathbf{T}\\ &= \mathbf{S}\left(x_{\mathbf{s}}^{\mathbf{U}}, y_{\mathbf{s}}^{\mathbf{U}}\right)\cdot\mathbf{R}\left(\varphi^{\mathbf{U}}\right)\cdot\mathbf{T}\left(x_{\mathbf{t}}^{\mathbf{U}}, y_{\mathbf{t}}^{\mathbf{U}}\right)\end{aligned}$$

Jeder `UserView` beinhaltet zusätzlich einen Satz von Transformationen, die in der Regel nicht durch den Benutzer verändert werden können. Dabei handelt es sich um die initiale Einpassung **I** der Visualisierung in den Arbeitsbereich der Ansicht, die durch die `Initial*`-Eigenschaften dargestellt werden:

$$\begin{aligned}x_{\mathbf{s}}^{\mathbf{I}} = y_{\mathbf{s}}^{\mathbf{I}} &= \min\left\{\frac{w_B}{w_G}, \frac{h_B}{h_G}\right\}\\ x_{\mathbf{t}}^{\mathbf{I}} &= x_W - \frac{w_B - w_W}{2}\,; \quad y_{\mathbf{t}}^{\mathbf{I}} = 0\\ \varphi^{\mathbf{I}} &= 0\ \text{rad}\end{aligned}$$

$$\begin{aligned}\mathbf{I} &= \mathbf{R}_{\mathbf{I}}\cdot\mathbf{S}_{\mathbf{I}}\cdot\mathbf{T}_{\mathbf{I}}\\ &= \mathbf{R}(0)\cdot\mathbf{S}\left(x_{\mathbf{s}}^{\mathbf{I}}, y_{\mathbf{s}}^{\mathbf{I}}\right)\cdot\mathbf{T}\left(x_{\mathbf{t}}^{\mathbf{I}}, 0\right)\end{aligned}$$

Für die Gesamttransformation **V** werden zunächst die Mittelpunkttranslation angewandt, danach die initiale und die Benutzer-Transformation, jedoch nicht durch Hintereinanderausführung, sondern durch Kombination:

$$\begin{aligned}\mathbf{V} &= \mathbf{T}_G\cdot\mathbf{R}_{\mathbf{V}}\cdot\mathbf{S}_{\mathbf{V}}\cdot\mathbf{T}_{\mathbf{V}}\\ &= \mathbf{T}_G\cdot\mathbf{R}\left(\varphi^{\mathbf{I}}+\varphi^{\mathbf{U}}\right)\cdot\mathbf{S}\left(x_{\mathbf{s}}^{\mathbf{I}}\cdot x_{\mathbf{s}}^{\mathbf{U}}, y_{\mathbf{s}}^{\mathbf{I}}\cdot y_{\mathbf{s}}^{\mathbf{U}}\right)\cdot\mathbf{T}\left(x_{\mathbf{t}}^{\mathbf{I}}+x_{\mathbf{t}}^{\mathbf{U}}, y_{\mathbf{t}}^{\mathbf{I}}+y_{\mathbf{t}}^{\mathbf{U}}\right)\end{aligned} \tag{4.1}$$

Prinzipiell ist es nicht verboten, einen anderen Ursprung zu wählen, z. B. durch die Anwendung einer Translation vor der Ansichts-Transformation, allerdings wird diese Ursprungsverschiebung durch die Skalierungs- und Rotationsmatrizen in der View-Transformation mitgestreckt, -gestaucht und -gedreht. Die Ansicht erscheint demnach also voraussichtlich nicht an der vom Benutzer erwarteten Stelle. Dasselbe gilt, wenn statt der erwähnten Kombination der initialen und View-Transformation deren Produkt $\mathbf{I}\cdot\mathbf{U}$ angewendet wird.

Nach den initialen Transformationen ist die geometrische Bounding-Box in das Arbeitsflächenrechteck eingepasst. Was noch fehlt, ist die Tile-Transformation $\mathbf{M}_i$, wobei $i$ den Teil des verteilten Systems bezeichnet, für den die Matrix berechnet werden soll. Prinzipiell ist sie auf jedem Anzeigeknoten sehr ähnlich. Die Aufgabe von $\mathbf{M}_i$ ist der Koordinatensystemwechsel von einer Geometrie, die in Pixelkoordinaten gegeben ist, in die Direct3D-Viewport-Koordinaten. Sie muss also eine Verschiebung enthalten, die die Visualisierungen an die Stelle schiebt, an der sie im Gesamtsystem erscheinen sollen, deshalb wird zunächst eine Translation des Ursprungs an den linken oberen Rand des verteilten Systems durchgeführt, von dem dann der linke und obere Rand des Ausschnitts $i$ abgezogen wird. Anschließend leistet eine Skalierung die Stauchung der Vertizes in das Viewport-Koordinatensystem mit

$x, y \in [-1, 1]$. Da gemäß der ersten Translation die Ecke der Geometrie noch am Ursprung ausgerichtet ist, muss im Viewport noch eine Verschiebung auf dessen Ecke geschehen. Seien $D = (x_D, y_D, w_D, h_D)$ und $P_i = (x_i, y_i, w_i, h_i)$ die Bounding-Boxen des verteilten Systems ($D$) und des Systemausschnitts ($P_i$), für den die Tile-Transformation $\mathbf{M}_i$ berechnet werden soll. Dann gilt:

$$\begin{aligned} \mathbf{M}_i &= \mathbf{T}_i \cdot \mathbf{S}_i \cdot \mathbf{T}_{\text{Viewport}} \\ &= \mathbf{T}\left(\frac{w_D}{2} - x_i, \frac{h_D}{2} - y_i\right) \cdot \mathbf{S}\left(\frac{2}{w_i}, \frac{2}{h_i}\right) \cdot \mathbf{T}(-1, -1) \end{aligned}$$

Die beim Rendern auf dem $i$-ten Rendering-Client verwendeten Transformationsmatrizen sind dann die Produkte

$$\mathbf{V}_k \cdot \mathbf{M}_i \; \forall k.$$

### 4.6.3. Device-Rendering

Im Gegensatz zu den Powerwall-Clients, auf denen im Prinzip eine zweidimensionale Darstellung das Ziel des Renderings ist, werden auf den Tablets andere Transformationen verwendet, um das in Abschnitt 3.2 beschriebene Verhalten zu erzeugen. Gegenüber dem Powerwall-Rendering sind hier zwei zentrale Unterschiede zu beachten:

- Es wird in physischen Koordinaten gerechnet.
- Die zweidimensionale Visualisierung wird dreidimensional als auf einer Fläche gelegen verstanden.
- Im Endeffekt wird eine perspektivische Projektion durchgeführt, deren Kamera sich aus den Daten des optischen Tracking-Systems ergibt.

Die Powerwall stellt hierbei die Ebene mit $z = 0$ dar. Entsprechend werden auch die Vertizes der Visualisierungsgeometrie an $z = 0$ statt an $z = \frac{1}{2}$ erstellt, wie das bei den Clients der Fall ist, die mit einer orthografischen Kamera rendern.

Da auf dem Tablet geometrisch genau derselbe Visualisierungszustand dargestellt werden soll wie auf der Powerwall, werden die benutzerdefinierten Transformationen der View-Matrix $\mathbf{V}$ beibehalten, außer dass die View-Transformation keine Umrechnung in logische, also Pixel-Koordinaten, sondern in das physische Koordinatensystem vornimmt. Das Koordinatensystem des optischen Trackers arbeitet in physischen Koordinaten und gibt für die Position Werte in Metern mit negativen Z-Anteilen aus. Die Ruhehaltung, also das aufrechte Halten des Tablets auf Augenhöhe, sodass der Durchschau-Vektor orthogonal auf der Powerwall steht, ist so kalibriert, dass sie in die positive Z-Richtung zeigt. Damit entspricht das Tracker-Koordinatensystem genau dem linkshändigen Koordinatensystem in Direct3D.

Neu ist beim Device-Rendering also die Projektionstransformation $\mathbf{D}$, die sich aus physischen Größen ergibt. Sie beinhaltet (in dieser Reihenfolge)

1. die Korrektur des Ursprungs des verteilten Systems, das nicht notwendigerweise auf dem Boden beginnen muss, also eine Translation $\mathbf{T_D}$,
2. $\mathbf{L_D}$, die Platzierung der Kamera in der Szene mit dem Sichtvektor und der Orientierung des Tablets, sowie
3. eine perspektivische Projektion $\mathbf{P_D}$ der Szene zum Auge mit dem Sichtfeld des Betrachters und dem Seitenverhältnis des Tablets.

Wie auch bei den Powerwall-Clients wird zunächst beim $k$-ten Benutzer die Ansichtstransformation $\mathbf{V}_k$ angewandt, danach die projektive Transformation $\mathbf{D}$.

Die Ursprungskorrektur ergibt sich aus der `DistributedSystemDefinition`, die dem Device vom Remote bekanntgemacht wird. Sie enthält den Vektor $\mathbf{t}_0$, dessen Komponenten für die Translation verwendet werden; es ist also $\mathbf{T_D} = \mathbf{T}(x_{\mathbf{t}_0}, y_{\mathbf{t}_0})$.

Um den Sichtfenster-Effekt beim Benutzer zu erzeugen, darf für die Kameraposition nicht einfach die vom Tracking-System gelieferte Position verwendet werden. Die Augen des Betrachters, die die eigentliche Kameraposition ausmachen, liegen hinter dem Tablet, daher muss die Kameraposition $\mathbf{e}$ noch weiter hinten entlang des Blickvektors angesetzt werden. Jedoch muss hierfür noch eine weitere Transformation vorgenommen werden, da der vom Kamerasystem wahrgenommene Starrkörper nicht notwendigerweise in der Mitte des getrackten Tablets platziert ist. Grundsätzlich ist der Augvektor also die Summe $\mathbf{e} = \mathbf{p} + \mathbf{t}_d + \mathbf{t}_e$, wobei $\mathbf{t}_d$ die Verschiebung zum Mittelpunkt des Tablets leistet und $\mathbf{t}_e$ den Vektor von diesem Mittelpunkt zum Auge hinter dem Tablet beschreibt.

Das Tracking-System liefert eine Position $\mathbf{p}$ und ein Orientierungsquaternion $q$. Aus $q$ lassen sich der Blickvektor $\mathbf{z}_d$ und der tabletplanar orientierte Rechts- bzw. Hoch-Vektor $\mathbf{x}_d$ bzw. $\mathbf{y}_d$ berechnen:

$$\mathbf{x}_d = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \cdot q, \qquad \mathbf{y}_d = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \cdot q, \qquad \mathbf{z}_d = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \cdot q$$

$\mathbf{x}_d$ und $\mathbf{y}_d$ werden für die Verschiebung des Rigid Bodys in die Mitte des Tablets benötigt, da diese auf der vom Tablet dargestellten Ebene geschehen muss. Wenn $\mathbf{r}_0 = (x_{\mathbf{r}_0}, y_{\mathbf{r}_0})^T$ das Offset des Starrkörpers auf der Tablet-Ebene ist, lässt sich dieser Blickmittelpunkt berechnen als

$$\mathbf{t}_d = x_{\mathbf{r}_0} \cdot \frac{\mathbf{x}_d}{\|\mathbf{x}_d\|} + y_{\mathbf{r}_0} \cdot \frac{\mathbf{y}_d}{\|\mathbf{y}_d\|}.$$

Von diesem Punkt ausgehend befindet sich das Auge in $-\mathbf{z}_d$-Richtung. Wie weit das Auge vom Tablet entfernt ist, ist benutzerabhängig und wird daher benutzerbasiert in den Anwendungseinstellungen gespeichert. Gebe $d_e$ den Abstand zwischen Auge und Tablet in physischen Größen an. Dann ist

$$\mathbf{t}_e = -d_e \cdot \frac{\mathbf{z}_d}{\|\mathbf{z}_d\|}.$$

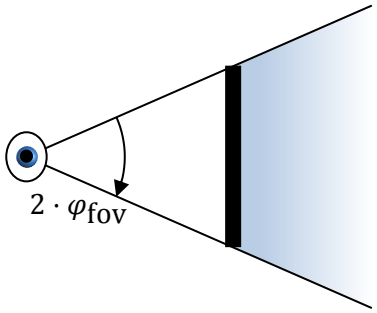



**Abbildung 4.14.:** Abhängigkeit und Bedeutung des Öffnungswinkels im Hinblick auf die Position des Betrachterauges hinter dem Tablet.

Die Kamera- bzw. Augenposition lässt sich insgesamt also berechnen als

$$\mathbf{e} = \mathbf{p} + \mathbf{t}_d + \mathbf{t}_e = x_\mathbf{r} \cdot \frac{\mathbf{x}_d}{\|\mathbf{x}_d\|} + y_\mathbf{r} \cdot \frac{\mathbf{y}_d}{\|\mathbf{y}_d\|} - d_e \cdot \frac{\mathbf{z}_d}{\|\mathbf{z}_d\|}.$$

Um die Kamera weiter zu beschreiben, muss der zurzeit betrachtete Punkt $\mathbf{l}$ gefunden werden. Im gegebenen Koordinatensystem ist dies der Schnittpunkt des Blickvektors mit der Powerwall. Dieser wird wiederum angesetzt in der Mitte des Tablets $\mathbf{p} + \mathbf{t}_d$. Gesucht ist die Stelle, an der die Gerade $g: \ \mathbf{x} = \mathbf{p} + \mathbf{t}_d + t \cdot \mathbf{z}_d$ die Powerwall schneidet, also $z_\mathbf{x} \stackrel{!}{=} 0$ gilt.

$$\begin{aligned} & z_\mathbf{p} + z_{\mathbf{t}_d} + t \cdot z_{\mathbf{z}_d} \stackrel{!}{=} 0 \\ \Leftrightarrow \quad & t = -\tfrac{z_\mathbf{p} + z_{\mathbf{t}_d}}{z_{\mathbf{z}_d}} \\ \Rightarrow \quad & \mathbf{l} = \mathbf{p} + \mathbf{t}_d - \tfrac{z_\mathbf{p} + z_{\mathbf{t}_d}}{z_{\mathbf{z}_d}} \cdot \mathbf{z}_d. \end{aligned}$$

Als letztes fehlt noch der im aktuellen Sichtsystem nach oben gerichtete Vektor $\mathbf{u}$. Dieser berechnet sich mit einem Zwischenschritt aus der Orientierung und der Y-Achse. Zunächst wird der nach rechts gerichtete Vektor

$$\mathbf{r} = \frac{\mathbf{z}_d}{\|\mathbf{z}_d\|} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

berechnet, mithilfe dessen sich $\mathbf{u}$ bestimmen lässt zu

$$\mathbf{u} = \mathbf{r} \times \frac{\mathbf{z}_d}{\|\mathbf{z}_d\|}.$$

Insgesamt lässt sich jetzt die Kameramatrix berechnen als $\mathbf{L_D} = \mathbf{L}(\mathbf{e}, \mathbf{l}, \mathbf{u})$.

Der finale Teil der Gesamttransformation am Tablet, die Projektionsmatrix, bedarf weniger aufwändigen Berechnungen. Abbildung 4.14 illustriert die Bedeutung des FoV-Winkels. Er

lässt sich aus der Höhe des Tablets $h_d$ und dem Abstand zwischen Auge und Tablet $d_e$ berechnen als

$$\varphi_{\text{FoV}} = \arctan\left(\frac{h_d}{2d_e}\right).$$

Das Seitenverhältnis $r$ ist definiert als der Quotient der (bekannten) Maße des Tablets $r = \frac{w_d}{h_d}$. Die Tiefenwerte der nahen und fernen Clipping-Ebene $z_\text{n}$ und $z_\text{f}$ sind hier einfach gewählt als $z_\text{n} = 0$ und $z_\text{f} = \infty$. Letzterer dieser Werte wurde programmatisch als `float.MaxValue` umgesetzt.

Die Projektionsmatrix steht somit auch fest:

$$\mathbf{P_D} = \mathbf{P}\left(\varphi_{\text{FoV}},\ \frac{w_d}{h_d},\ 0,\ \infty\right)$$

Nach obiger Einführung lässt sich jetzt die gesamte, projektive Transformation auf den Tablets bestimmen:

$$\begin{aligned}\mathbf{D} &= \mathbf{T_D} \cdot \mathbf{L_D} \cdot \mathbf{P_D} \\ &= \mathbf{T}(x_{\mathbf{t}_0},\ y_{\mathbf{t}_0}) \cdot \mathbf{L}(\mathbf{e},\ \mathbf{l},\ \mathbf{u}) \cdot \mathbf{P}\left(\varphi_{\text{FoV}},\ \frac{w_d}{h_d},\ 0,\ \infty\right)\end{aligned}$$

Wie eingangs erwähnt wird hiervor die View-Transformation $\mathbf{V}_k$ angewandt, sodass sich beim Rendering insgesamt die Transformation $\mathbf{V}_k \cdot \mathbf{D}$ ergibt.

### 4.6.4. Shader

Von der Software werden Vertex- und Pixel-Shader verwendet, die in den Dateien *shaders11.fx* bzw. *shaders9.fx* für D3D 11 bzw. D3D 9 zu finden sind. Diese Effekt-Dateien enthalten außerdem den `SamplerState`, der die Funktionsweise des Textur-Samplings beschreibt, und den `RasterizerState`, in dem festgelegt wird, dass Primitive beidseitig gezeichnet und Scissor-Tests beim Rendern durchgeführt werden. Letzteres wird benutzt, um die Ansichten visuell auf den zugeordneten Arbeitsbereich zu beschränken.

Der Vertex-Shader `VShader` transformiert die Vertizes der Geometrie anhand der `transformationMatrix` aus dem Matrizen-`ConstantBuffer`.

Gegenwärtig sind zwei Pixel-Shader implementiert:

- `PColorShader` färbt die Geometrie anhand der Farbinformationen der Vertizes ein.
- `PTextureShader` führt ein Sampling auf der `shaderTexture` anhand der Texturkoordinaten der Vertizes durch, um die Farbe eines Pixels zu bestimmen.

Dementsprechend gibt es auch zwei Effekt-Pässe, die zum Einfärben bzw. Texturieren der Geometrie verwendet werden.

## 4.7. Interaktion

Die Benutzer interagieren mit der Visualisierung mithilfe von Tablets, die sie bei sich tragen. Bei aller Interaktion muss sichergestellt werden, dass alle Clients mit gleichen Informationen arbeiten. Unterschiede würden hier zu Inkonsistenzen in der Darstellung oder im Verhalten der Anwendung auf dem betroffenen Powerwall-Knoten führen. Daher kommunizieren die Tablets nicht direkt mit den Clients, sondern senden ihre Interaktionsinformationen zunächst an den Remote-Rechner, der sich um die Verarbeitung der Interaktion und deren Synchronisierung mit den rendernden Rechnern kümmert.

### 4.7.1. Einzelnutzer-Features

#### Gesten

Da die Tablets Multitouch-fähige Bildschirme bieten, können gängige Interaktionsgesten verwendet werden. Weil es sich bei der Software um eine Windows-Forms-Anwendung handelt und Touch- bzw. Gesteninteraktion in dieser Technologie noch nicht vorgesehen waren, müssen die von Windows gelieferten Window-Messages durch die entsprechenden Aufrufe an die Windows-API manuell übersetzt werden. Hierzu wurde die `GestureForm`-Klasse entwickelt, das die `WM_GESTURE`-Nachrichten verarbeitet und in Ereignisse übersetzt, die in einer abgeleiteten Klasse – in diesem Fall dem `DeviceForm` – interpretiert werden können.

Von der Verwendung von Bewegungsträgheit wurde abgesehen, da nicht erwartet wurde, dass die intertiale Weiterbewegung der Ansicht ein von Nutzern erwünschtes Feature darstellt. Sollte sich in einer Studie das Gegenteil erweisen, wäre das Feature aus technischer Sicht leicht nachzurüsten, da Windows die entsprechenden Nachrichten in der Gesture-API schon bereitstellt und senden kann, wenn das Fenster entsprechend konfiguriert wird. Bestimmte Gesten bzw. deren Attribute explizit aktiviert werden, damit sie vom Formular empfangen werden können; hiervon ist insbesondere die Rotationsgeste betroffen. Hierzu dient die Übergabe eines Arrays von Gestenkonfigurationen im Konstruktor der `GestureForm`-Klasse, wodurch genau angegeben kann, welche Gestenfunktionalitäten zur Verfügung stehen und welche nicht an das Fenster gesendet werden sollen.

Die Interpretation der Gesten gliedert sich in der Regel in drei Stufen, die durch die `GestureFlags` abgebildet werden:

1. Eine neue Geste wird begonnen.
2. Der Benutzer führt die Geste aus, wodurch sich ihre gestenspezifischen Parameter ändern.
3. Die Geste wird abgeschlossen.

Die nachfolgenden Abschnitte beschreiben die implementierten Gesten und deren Interpretation. Generell wurde das Konzept verfolgt, die Interpretation der Geste auf den Tablets sofort zu zeigen, während die Transformationen auf der Powerwall erst beim Abschluss einer Geste angewendet werden. Dies hat den Grund, dass sich bei der Echtzeit-Synchronisierung besonders bei langen Gesten eine Verzögerung einstellte, durch die die einzelnen Teile der Powerwall unterschiedliche Transformationen zeigte, das Bild also nicht zusammenpasste (siehe Unterabschnitt 4.9.4).

**Pan** Mit der Pan-Geste kann der Benutzer seine Ansicht verschieben. Hierzu berührt er den Bildschirm mit (mindestens) einem Finger und bewegt ihn daraufhin in eine beliebige Richtung. Dabei wird die in der Ansicht gespeicherte Translation entsprechend verändert.

Die Geste kann mit einem Finger aufgrund der Verwendung der Pan-Konfigurationsattribute `PanWithSingleFingerHorizontally` und `PanWithSingleFingerVertically` durchgeführt werden; ansonsten müsste der Benutzer zwei Finger aufsetzen und diese in dieselbe Richtung bewegen. Die beiden Attribute `PanWithGutter` und `PanWithInertia` wurden nicht verwendet:

- `PanWithInertia` führt dazu, dass das verschobene Bild als träge angenommen wird und sich daher noch in die Panning-Richtung weiterbewegt, wenn die Geste schon abgeschlossen ist. Da dies nicht das intendierte Verhalten war, wurde von der Verwendung dieses Attributs abgesehen.
- Die `PanWithGutter`-Option hat zur Folge, dass eine begonnene Verschiebung wie auf einer „Schiene“ geführt wird, die je nach erster Bewegung in die horizontale oder vertikale Richtung verläuft. Möchte der Benutzer die Visualisierung gleichermaßen in beide Richtungen schieben können, muss er mit diesem Attribut sehr weit von dieser Schiene abweichen; daher wurde auch diese Option deaktiviert.

**Zoom** Mithilfe der Zoom-Geste wird eine Reskalierung der Ansicht vorgenommen. Auch hierzu muss der Benutzer den Bildschirm mit zwei Fingern berühren, diese dann aber in entgegengesetzte Richtungen bewegen. Entfernt er die Finger voneinander, werden der horizontale und vertikale Skalierungsfaktor gleichermaßen erhöht und damit die Ansicht vergrößert. Führt der Benutzer die Finger zusammen, werden die Faktoren entsprechend verringert.

**Rotation** Beim Ausführen der Rotationsgeste kann der Benutzer die Ansicht drehen. Dazu setzt der Benutzer einen Finger auf und dann einen weiteren, den er um den zuerst aufgesetzten Finger herumbewegt. Der erste Finger wird als Rotationszentrum interpretiert, der zweite bestimmt den Winkel, um den rotiert wird. Mithilfe dessen kann er den Rotationswinkel seiner Ansicht anpassen.

Bei dieser Geste muss beachtet werden, dass erst ein Makro[2] zur Berechnung des Rotationswinkels $r$ aus der `Arguments`-Eigenschaft $a$ verwendet werden muss. Außerdem hat die Berechnung des Winkels unterschiedliche Semantiken, je nachdem, ob die Geste gerade begonnen wird oder sich in der Ausführungsphase befindet. Beim Beginn der Geste wird im `Argument` der Winkel gegenüber der Horizontalachse übergeben, wohingegen während der Ausführung die Winkeldifferenz zum Beginn der Geste berechnet wird.

#### Optischer Tracker

Die Lokalisierung der Tablets geschieht über einen optischen Tracker, der einen am jeweiligen Tablet befestigten Starrkörper via Infrarot-Kamerasystem wahrnimmt. Dieser Starrkörper besteht aus fünf speziell lackierten und in einer bestimmten Weise angebrachten Kugeln, die vom Kamerasystem durch Bildverarbeitung zuverlässig erkannt werden können. Aus den gewonnenen Daten lässt sich im Idealfall bis auf ein Rauschen von etwa $\pm 2$ mm der Ort des Trackers bestimmen sowie dessen Orientierung berechnen. Diese Daten liegen dann als Vektor bzw. Quaternion vor und werden vom Tracking-System über VRPN, ein spezielles Protokoll, gebroadcastet.

Die Remote-Anwendung verwendet die Bibliothek *VrpnNet*, die eine VRPN-Anbindung für .NET-Anwendungen zur Verfügung stellt. Aus den Broadcasts des Tracking-Systems werden nur diejenigen Nachrichten verarbeitet, die mit einem bestimmten Tracker assoziiert sind. Dazu muss in einem bestimmten Tool zunächst der Tracker angelegt und konfiguriert werden, kann dann gespeichert und immer wiederverwendet werden. Insbesondere sind den Trackern eindeutige Namen zugewiesen, über die identifiziert werden kann, welcher Benutzer sich gerade mit seinem Tablet bewegt. Mithilfe von *VrpnNet* werden auf dem Remote-Rechner vom `VrpnHostControl` Ereignisse ausgelöst. Diese umfassen als Ereignisparameter folgende Informationen:

- `Position`: Ein dreidimensionaler Vektor $\mathbf{p} = \left(x_\mathbf{p}, y_\mathbf{p}, z_\mathbf{p}\right)^T$, der den Ort des Starrkörpers relativ zum Ursprung des Raumkoordinatensystems angibt.
- `Orientation`: Ein Quaternion $q = \left(x_q, y_q, z_q, w_q\right)$, das die Ausrichtung des Starrkörpers beschreibt.
- `Sensor`: Dieses Feld wird von der Anwendung nicht verwendet.
- `Time`: Der Zeitstempel des Tracker-Ereignisses.

Daraus kann berechnet werden, auf welche Stelle $\mathbf{s}$ der Powerwall der Benutzer mit seinem Tablet zeigt: Von der Position $\mathbf{p}$ des Tablets wird entlang seines Orientierungsvektors $q$ eine Gerade $\mathbf{p} + r \cdot q$ gedacht und diese mit der Powerwall-Ebene geschnitten, die durch konstantes $z = 0$ charakterisiert ist. Die genaue Geometrie und die damit verbundenen Berechnungen werden in Unterabschnitt 4.6.3 ausgeführt. Die Sonderfälle, in denen der Orientierungsvektor keine Tiefe, also $\mathbf{z}_d = 0$ besitzt, werden von der Software entsprechend

[2] Die im MSDN vorgeschlagene Berechnung lautet $r = \frac{a \cdot 4\pi}{2^{16}-1} - 2\pi$.

abgefangen. Aus dem Schnittpunkt **s** mit der Powerwall in physischen Koordinaten lassen sich die getroffenen Bildschirmkoordinaten und daraus über die Transformationsinvertierung die Visualisierungskoordinaten berechnen.

Mithilfe dieser Berechnungen können die in Abschnitt 3.2 beschriebenen Interaktionen umgesetzt werden.

**Ansichtstransformation** Aufgrund der Unzulänglichkeit der Daten des optischen Trackers (siehe hierzu Unterabschnitt 4.9.5) konnte die Transformation der Benutzeransicht durch physische Bewegung des Tablets nicht getestet werden; daher wird hier nur die grundlegende Idee vorgestellt, um die in Kapitel 3 beschrieben und in Abbildung 3.2 skizzierten Interaktionen umzusetzen. Gesucht ist eine Abbildung

$$\begin{aligned} \delta_{\mathbf{M}} : \mathbb{R}^3 \times \mathbb{H} \times \mathbb{R}^3 \times \mathbb{H} &\to \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R} \\ (\mathbf{p}_1, q_1, \mathbf{p}_2, q_2) &\mapsto (\mathbf{t}, \mathbf{s}, \varphi)\,, \end{aligned}$$

die zu der Differenz zwischen zwei Tracking-Zuständen die auszuführende Transformationsdifferenz berechnet.

Der Translationsanteil berechnet sich aus dem Abstand zwischen den Schnittpunkten (für deren Berechnung und die anderer Größen, siehe Unterabschnitt 4.6.3) mit der Powerwall zwischen den Zuständen $(\mathbf{p}_1, q_1)$ und $(\mathbf{p}_2, q_2)$. Die Umrechnung in Pixelkoordinaten geschieht durch einfache Multiplikation mit einem linearen Faktor $\lambda^{\text{px}}_{\text{phys}}$, für den vereinfachend angenommen wird, dass die Pixel der Powerwall physisch quadratisch sind. Sei $w_{\text{phys}}$ die physische Breite des Gesamtdisplays und $w_{\text{px}}$ die Breite der logischen Bounding-Box des Systems in Pixeln. Dann ist

$$\lambda^{\text{px}}_{\text{phys}} = \frac{w_{\text{px}}}{w_{\text{phys}}} \tag{4.2}$$

und die Translationsdifferenz beträgt

$$\mathbf{t} = \lambda^{\text{px}}_{\text{phys}} \cdot (\mathbf{s}_2 - \mathbf{s}_1)\,.$$

Für den Skalierungsanteil wurde das Verhältnis der Längen der Abstandsvektoren $\mathbf{d}_1$ und $\mathbf{d}_2$ zwischen dem Mittelpunkt des Tablets und den Schnittpunkten mit der Powerwall $\mathbf{s}_1$ und $\mathbf{s}_2$ herangezogen:

$$x_{\mathbf{s}} = y_{\mathbf{s}} = \frac{|\mathbf{d}_2 - \mathbf{s}_2|}{|\mathbf{d}_1 - \mathbf{s}_1|} = \frac{\sqrt{(x_{\mathbf{d}_2} - x_{\mathbf{s}_2})^2 + (y_{\mathbf{d}_2} - y_{\mathbf{s}_2})^2 + z^2_{\mathbf{d}_2}}}{\sqrt{(x_{\mathbf{d}_1} - x_{\mathbf{s}_1})^2 + (y_{\mathbf{d}_1} - y_{\mathbf{s}_1})^2 + z^2_{\mathbf{d}_1}}}$$

Damit zoomt der Benutzer, indem er sich von der Powerwall entfernt, also den Abstand zum Schnittpunkt vergrößert.

Eine günstige Rechenvorschrift für die Anwendung der Rotation der Visualisierung um die Z-Achse wurde noch nicht implementiert, muss sich aber konzeptionell aus $q_1$ und $q_2$ ergeben. Es fehlt also noch der Rotationsanteil $(q_1, q_2) \overset{?}{\mapsto} \varphi$.

**Selektion** Da die Selektion eines bestimmten Knotens zur Schnittstelle der Renderer gehören, muss die mit dem Tablet getroffene Stelle in Geometrie-Koordinaten zurückgerechnet werden. Dies wird mithilfe der Inversion der aktuellen Transformation, die dem Tablet bekannt ist, erreicht.

Als Eingabe dient der berechnete, physische Schnittpunkt mit der Powerwall $\mathbf{s}_{\text{phys}}$. Durch die Transformation von $\mathbf{s}_{\text{phys}}$ mit $\mathbf{V}^{-1}$ kann der Schnittpunkt auf Geometriekoordinaten abgebildet werden. Dazu werden die gemachten Transformationen in umgekehrter Reihenfolge und mit invertierten Parametern angewandt. Diese „Parameter-Invertierung“ hat je nach Transformation unterschiedliche Bedeutungen:

$$\begin{aligned}
\mathbf{T} \cdot \mathbf{T}^{-1} = \mathbf{E} \quad &\Leftrightarrow \quad \mathbf{T}(x,\, y)^{-1} = \mathbf{T}(-x,\, -y) \\
\mathbf{S} \cdot \mathbf{S}^{-1} = \mathbf{E} \quad &\Leftrightarrow \quad \mathbf{S}(x,\, y)^{-1} = \mathbf{S}\left(\frac{1}{x},\, \frac{1}{y}\right) \\
\mathbf{R} \cdot \mathbf{R}^{-1} = \mathbf{E} \quad &\Leftrightarrow \quad \mathbf{R}(\varphi)^{-1} = \mathbf{R}(-\varphi)
\end{aligned}$$

Die in Gleichung 4.1 beschriebene Transformation

$$\mathbf{V} \;=\; \mathbf{T}_G \cdot \mathbf{R}\left(\varphi^{\mathbf{I}} + \varphi^{\mathbf{U}}\right) \cdot \mathbf{S}\left(x_{\mathbf{s}}^{\mathbf{I}} \cdot x_{\mathbf{s}}^{\mathbf{U}},\, y_{\mathbf{s}}^{\mathbf{I}} \cdot y_{\mathbf{s}}^{\mathbf{U}}\right) \cdot \mathbf{T}\left(x_{\mathbf{t}}^{\mathbf{I}} + x_{\mathbf{t}}^{\mathbf{U}},\, y_{\mathbf{t}}^{\mathbf{I}} + y_{\mathbf{t}}^{\mathbf{U}}\right)$$

wird also invertiert zu

$$\mathbf{V}^{-1} \;=\; \mathbf{T}_G^{-1} \cdot \mathbf{R}\left(-\varphi^{\mathbf{I}} - \varphi^{\mathbf{U}}\right) \cdot \mathbf{S}\left(\frac{1}{x_{\mathbf{s}}^{\mathbf{I}} \cdot x_{\mathbf{s}}^{\mathbf{U}}},\, \frac{1}{y_{\mathbf{s}}^{\mathbf{I}} \cdot y_{\mathbf{s}}^{\mathbf{U}}}\right) \cdot \mathbf{T}\left(-x_{\mathbf{t}}^{\mathbf{I}} - x_{\mathbf{t}}^{\mathbf{U}},\, -y_{\mathbf{t}}^{\mathbf{I}} - y_{\mathbf{t}}^{\mathbf{U}}\right).$$

Der so transformierte Punkt in Geometriekoordinaten $\mathbf{s}_{\text{geom}} = \mathbf{s}_{\text{px}} \cdot \mathbf{V}^{-1}$ wird als Eingabe für die `Pick`-Methode des verwendeten `DistributedRenderers` verwendet; diese liefert das Datenelement, also ein `Tree`-Objekt zurück, welches der `Select`-Methode übergeben wird. Dadurch wird der Vertex-Buffer aktualisiert mit dem neu selektierten Knoten, der daraufhin mit der Farbe der Ansicht hervorgehoben wird.

### 4.7.2. Kollaborative Features

#### Umordnen von Ansichten

Als kollaboratives Feature wurde mithilfe des optischen Trackers umgesetzt, dass sich die Benutzer des Systems frei bewegen und ihre eigenen Ansichten zu anderen Benutzern „mitnehmen“ können. Ausschlaggebend hierfür sind die X-Koordinaten der Positionsvektoren $\mathbf{p}_i$ aller Benutzer. Der Remote merkt sich für jeden Benutzer den letzten bekannten Standort, den er über die VRPN-Broadcasts des optischen Trackers erfahren hat. Bei jeder Positionsänderung eines Benutzers wird dann geprüft, ob sich die Ordnung der Ansichten, gegeben durch die Sortierung der Komponenten $x_{\mathbf{p}_i}\, \forall i$, geändert hat. Falls dem so ist, wird die neue Ordnung der Ansichten mit den Render-Clients synchronisiert.

In der Regel bedeutet das, dass zwei Benutzer die Plätze getauscht haben bzw. dass sich ein Benutzer zu einem anderen Benutzer bewegt. Durch die Umordnung der Ansichten

nehmen die Benutzer ihre Ansichten mit, wenn sie sich vor der Powerwall bewegen, wodurch ermöglicht wird, dass entfernte Benutzer, die sich beispielsweise über ihre Erkenntnisse austauschen möchten, in physische Nähe rücken, um besser kommunizieren zu können.

### 4.7.3. Generelle Interaktionen

#### Übersichts-Ansichten

Zwischen zwei benachbarten Ansichten wird immer die komplette Hierarchie dargestellt. Diese sind dort fest platziert, können also nicht verschoben oder durch Interaktion transformiert werden. Sie zeigt neben der Visualisierung auch jeweils einfarbige, viereckige Überlagerungen an, die genau den Bereich abdecken, der vom entsprechenden Benutzer in seiner Ansicht gerade sichtbar ist. So dient die Darstellung den Betrachtern als Orientierungsmöglichkeit in den Hierarchien.

Technisch können die Übersichts-Ansichten einmal beim Laden des Datensatzes mithilfe eines Off-Screen-Render-Targets in eine Textur gezeichnet und dann zwischen den Ansichten in einem einfachen Quadrat dargestellt werden. Der Aufruf der `CreateOverview`-Methode geschieht daher beim Empfang der `InitializeClient`-Nachricht. Hierbei wird eine neue Textur und darauf ein `RenderTargetView` erstellt. Der Einfärbungs-Pixel-Shader wird verwendet, um die geladene Geometrie darzustellen, die über eine Transformation $\mathbf{O}$ in die erstellte Textur eingepasst wird. Diese ähnelt konzeptionell der initialen Ansichtstransformation und besteht aus zwei Teilen, wobei $G = (x_G, y_G, w_G, h_G)$ die Bounding-Box der Geometrie darstellt:

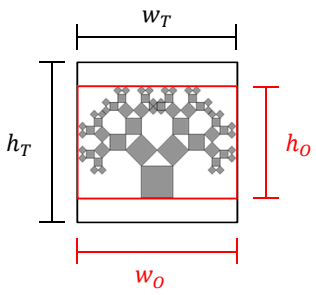
$$
\begin{aligned}
s_{\mathbf{O}} &= \min\left\{\frac{1}{w_G}, \frac{1}{h_G}\right\} \\
\mathbf{O} &= \mathbf{T}_{\mathbf{O}} \cdot \mathbf{S}_{\mathbf{O}} \\
&= \mathbf{T}\left(-\left(x_G + \frac{w_G}{2}\right), -\left(y_G + \frac{h_G}{2}\right)\right) \cdot \mathbf{S}(2 \cdot s_{\mathbf{O}}, 2 \cdot s_{\mathbf{O}})
\end{aligned}
$$

Der Faktor 2 bei der Skalierung um $s_{\mathbf{O}}$ rührt daher, dass der Viewport-Bereich keine Einheitslänge, sondern in X- und Y-Richtung jeweils die Länge 2 (da $x, y \in [-1, +1]$) besitzt.

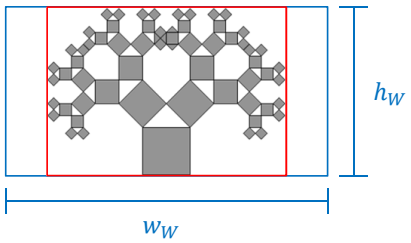
Da die Übersichten immer zwischen zwei benachbarten Ansichten dargestellt werden sollen, wird beim zweiten Benutzer begonnen und auf die linke Seite die Übersicht platziert. Hierzu wird ein Quadrat mit der Seitenlänge 2 verwendet, das durch die Matrix $\mathbf{O}'$ an die richtige Stelle transformiert wird:

$$
\begin{aligned}
\mathbf{O}' &= \mathbf{S}_{\mathbf{O}'} \cdot \mathbf{T}_{\mathbf{O}'} \cdot \mathbf{W} \\
&= \mathbf{S}\left(\frac{w_T}{2}, \frac{h_T}{2}\right) \cdot \mathbf{T}\left(x_W - \frac{w_B}{2}, \frac{h_O - h_B}{2}\right) \cdot \mathbf{W}
\end{aligned}
\tag{4.3}
$$

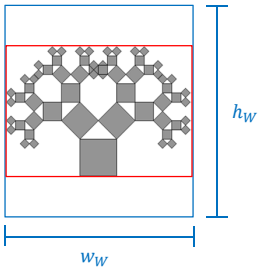
Dabei sind $w_T$ bzw. $h_T$ die Breite bzw. Höhe der Übersichtsdarstellung in Pixeln, $B$ stellt die Bounding-Box des verteilten Systems in Pixeln dar und $w_B$ bzw. $h_B$ davon die Breite bzw. Höhe. $W$ ist der Arbeitsbereich des Benutzers, an dessen linken Rand die Textur gezeichnet werden



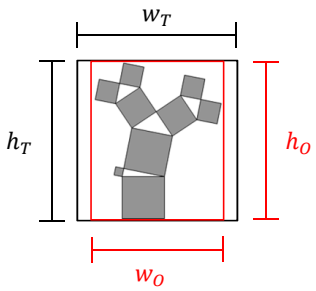

**(a)** Fall 1: $w_O \geq h_O$.



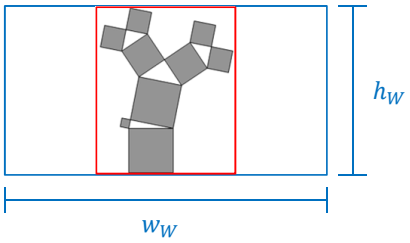

**(b)** Fall 1 a: $w_W \geq h_W$.



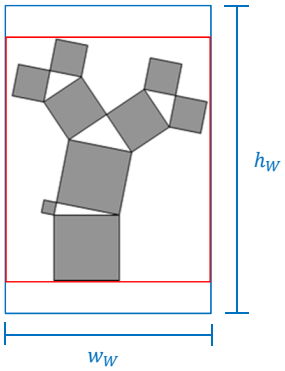

**(c)** Fall 1 b: $w_W < h_W$.




**(d)** Fall 2: $w_O < h_O$.




**(e)** Fall 2 a: $w_W \geq h_W$.




**(f)** Fall 2 b: $w_W < h_W$.

soll. Die Transformation $\mathbf{O}'$ leistet die Verschiebung des Ursprungs und eine Skalierung, sodass das oben erwähnte Quadrat genau die Ausmaße $(w_T, h_T)$ annimmt und auf dem Display ganz unten und am linken Ende des Arbeitsbereiches eines Benutzers dargestellt wird. Der Texturierungs-Effekt wird eingesetzt, um das Bild dort zu rendern.

**Überlagerungen** Nachdem die Übersichtstexturen gerendert wurden, müssen noch die halbdurchsichtigen Rechtecke gezeichnet werden, die die Übersichtsdarstellung so überlagern, dass derselbe Abschnitt abgedeckt wird, wie gerade im Arbeitsbereich des jeweiligen Benutzers sichtbar ist. Hierzu werden die inversen Teiltransformationen der Ansicht verwendet; die Grundidee hierbei ist, dass das Einheitsquadrat um die Faktoren $x_\mathbf{s}$ und $y_\mathbf{s}$ gestreckt bzw. gestaucht wird, um die Größe des Arbeitsbereichs wiederzugeben. Aufgrund der Einpassung der gesamten Visualisierung in die in der Regel quadratische Übersichtstextur und der initialen Einpassung der Visualisierung in den Arbeitsbereich jedes Benutzers müssen hier vier Fälle unterschieden werden: Die oben bereits erwähnten Quadrate mit Seitenlänge 2 sollen durch eine Skalierung so in ihrer Größe angepasst werden, dass sie beide Einpassungen widerspiegeln. Die Skalierungsfaktoren sind $x_\mathbf{s}$ und $y_\mathbf{s}$ und müssen folgendem Verhältnis genügen:

$$\frac{w_V}{h_V} = \frac{x_\mathbf{s}}{y_\mathbf{s}} \quad \Leftrightarrow \quad x_\mathbf{s} = y_\mathbf{s} \cdot \frac{w_V}{h_V} \quad \Leftrightarrow \quad y_\mathbf{s} = x_\mathbf{s} \cdot \frac{h_V}{w_V}$$

Einer der Faktoren kann je nach angetroffenem Fall der Verhältnis berechnet bzw. gewählt werden:

1. Im ersten Fall ist die Geometrie breiter als hoch oder gleich breit wie hoch (siehe Abbildung 4.15a), es gilt also $w_O \geq h_O$.
   a) Wenn $w_W \geq h_W$ ist und damit das Seitenverhältnis des Arbeitsbereichs zugunsten der Breite fällt (siehe Abbildung 4.15b), kann $y_\mathbf{s} = h_O$ gewählt werden und es ergibt sich $x_\mathbf{s} = h_0 \cdot \frac{w_V}{h_V}$.
   b) Ansonsten ist der Arbeitsbereich höher als breit (siehe Abbildung 4.15c) und es wird $x_\mathbf{s} = 1$ gesetzt. Dann ergibt sich $y_\mathbf{s} = \frac{h_V}{w_V}$.
2. Der zweite Fall beschreibt das Gegenteil, also dass die Geometrie schmal und somit höher als breit ist (siehe Abbildung 4.15d).
   a) Ist zusätzlich der Arbeitsbereich eher breit als hoch (siehe Abbildung 4.15e, kann $y_\mathbf{s} = 1$ gewählt werden. $x_\mathbf{s}$ berechnet sich dann zu $x_\mathbf{s} = \frac{w_V}{h_V}$.
   b) Ist $w_W < h_W$ (siehe Abbildung 4.15f), gilt $x_\mathbf{s} = w_O$ und $y_\mathbf{s} = w_O \cdot \frac{h_V}{w_V}$.

Die insgesamt resultierende Transformation $\mathbf{O}_V$ enthält dann diese Skalierung, die inversen Transformationen des Benutzers $\mathbf{U}_V$ sowie die aus Gleichung 4.3 bekannte Matrix $\mathbf{O}$ zur Translation und Skalierung zur Übersichtsdarstellung:

$$\begin{aligned}\mathbf{O}_V &= \mathbf{T}_{\mathbf{U}_V}^{-1} \cdot \mathbf{S}_{\mathbf{U}_V}^{-1} \cdot \mathbf{S}_{\mathbf{O}_V} \cdot \mathbf{R}_{\mathbf{U}_V}^{-1} \cdot \mathbf{O} \\ &= \mathbf{T}\left(-\frac{2 \cdot x_\mathbf{t}^{\mathbf{U}_V}}{w_W}, -\frac{2 \cdot y_\mathbf{t}^{\mathbf{U}_V}}{h_W}\right) \cdot \mathbf{S}\left(\frac{1}{x_\mathbf{s}^{\mathbf{U}_V}}, \frac{1}{y_\mathbf{s}^{\mathbf{U}_V}}\right) \cdot \mathbf{S}(x_\mathbf{s}, y_\mathbf{s}) \cdot \mathbf{R}\left(-\varphi^{\mathbf{U}_V}\right) \cdot \mathbf{O}\end{aligned}$$

## 4.8. Grafische Benutzungsschnittstelle

In diesem Abschnitt wird die Benutzungsschnittstelle (GUI) für die Remote-Anwendung vorgestellt. Der Teil der Software, der auf der Powerwall läuft, besitzt keine hervorzuhebende GUI und wird daher nicht beschrieben. Es handelt sich dabei lediglich um ein SlimDX-`RenderForm`, das keine weiteren Steuerelemente besitzt und komplett von der Remote-Anwendung ferngesteuert wird. Die Benutzungsschnittstelle der Tablet-Anwendung wird in Abschnitt 4.7 beschrieben.

Die Remote-Anwendung läuft auf einem gewöhnlichen Desktop-Rechner und muss die Möglichkeit bieten, die beteiligten Anwendungen zu initialisieren und im weitesten Sinne zu überwachen. Sie bietet einen dreigeteilten Aufbau, der in den folgenden Abschnitten erläutert wird. Wo es möglich war, wird von der `Reflection`-API Gebrauch gemacht, um keine unnötigen Anpassungsaufwände in der GUI zu verursachen, wenn z. B. neue `DistributedRenderer`-Implementierungen hinzugefügt werden.

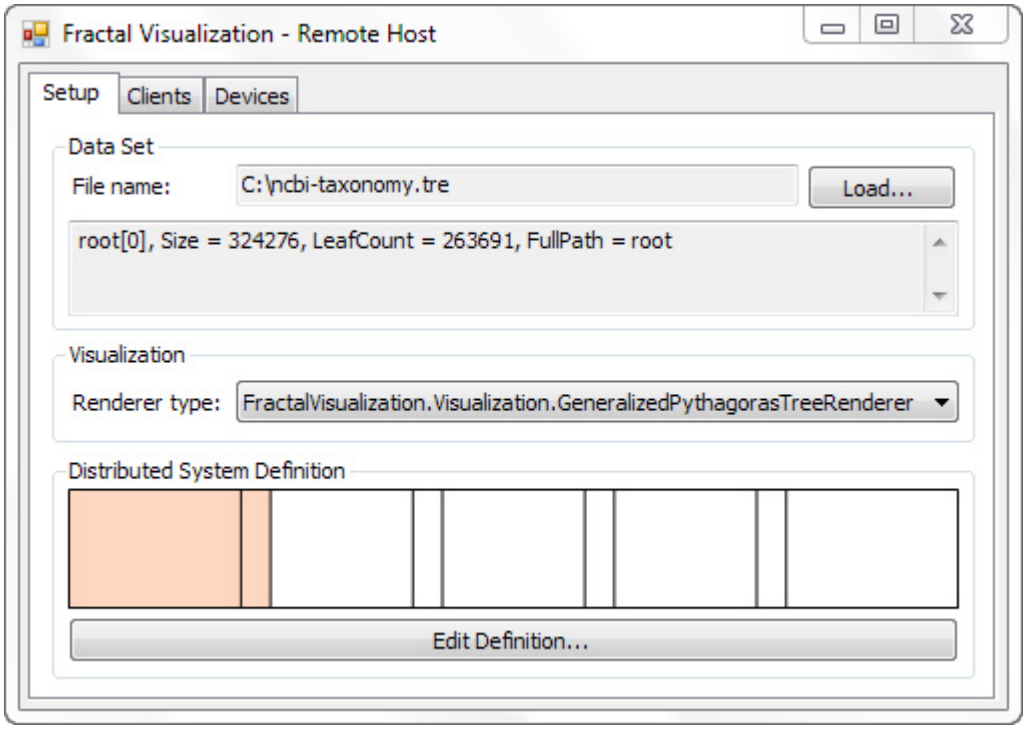



**Abbildung 4.15.:** Der „Setup“-Reiter in der GUI der Remote-Anwendung.

### 4.8.1. Setup

Unter dem *Setup*-Reiter werden die zur Initialisierung der Clients benötigten Daten verwaltet.

Wie in Abbildung 4.15 zu sehen ist, kann hier ein Datensatz geladen werden; in diesem Fall wurde die NCBI-Taxonomie aus Fallbeispiel 6.1 geladen. Da es sich hierbei um den Referenzdatensatz für diese Arbeit handelt, wird er – sofern vorhanden – automatisch beim Programmstart geladen.

Als nächstes kann der zu verwendende Renderer ausgewählt werden. Die Einträge der Auflistung werden beim Start der Anwendung mithilfe der Reflektionsmethoden von .NET automatisch generiert. Die Wahl des Renderers bestimmt die Art und Weise, wie der Datensatz auf den verschiedenen Geräten und Rendering-Clients dargestellt wird. Da für diese Arbeit ausschließlich die verallgemeinerten Pythagoras-Bäume umgesetzt wurden, wird dieser Renderer standardmäßig verwendet.

Außerdem kann die Definition des verteilten Rendering-Systems gesehen und bearbeitet werden. Als Standardwert wird hier die VISUS-Powerwall mit fünf Projektorenspalten in einer Zeile mit einer jeweiligen Auflösung von $2400 \times 4096$ Pixeln und einer horizontalen Überlappung von 300 Pixeln verwendet. Das System kann aber auch mit dem in Abbildung 4.16 gezeigten Dialog bearbeitet werden. Dieser ermöglicht das Bearbeiten des geladenen Systems, indem für jeden einzelnen Projektor die Rechtecke verändert werden. Gleichermaßen kann

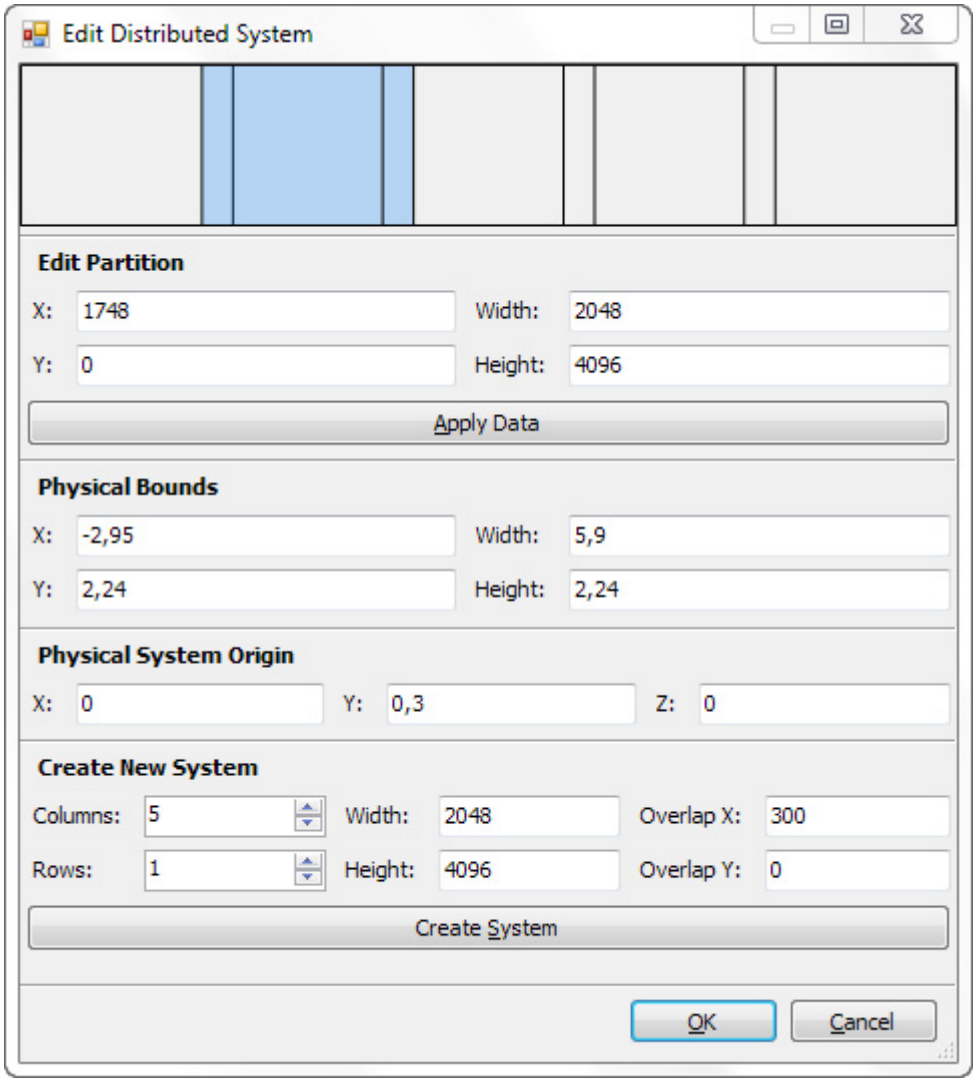



**Abbildung 4.16.:** Ein Dialog zum Erstellen und Bearbeiten von Definitionen für das verteilte Rendering-System.

aber auch ein neues System mithilfe der Anzahl der Spalten und Zeilen sowie der Größe und Überlappung der einzelnen Rechtecke berechnet werden.

### 4.8.2. Rendering-Clients

Der zweite Reiter, zu sehen in Abbildung 4.17, dient zur Verwaltung der Verbindungen mit den Rendering-Clients der Powerwall. Hier kann der `ClientManager` für die Rendering-Clients konfiguriert, d. h. der Port angegeben werden, auf den sich die Clients verbinden müssen. Neben Funktionalitäten zum lokalen Testen der Anwendung wird hier vor allem für jeden verbundenen Client ein `ServerControl` zur Verfügung gestellt, über den die Clients initialisiert

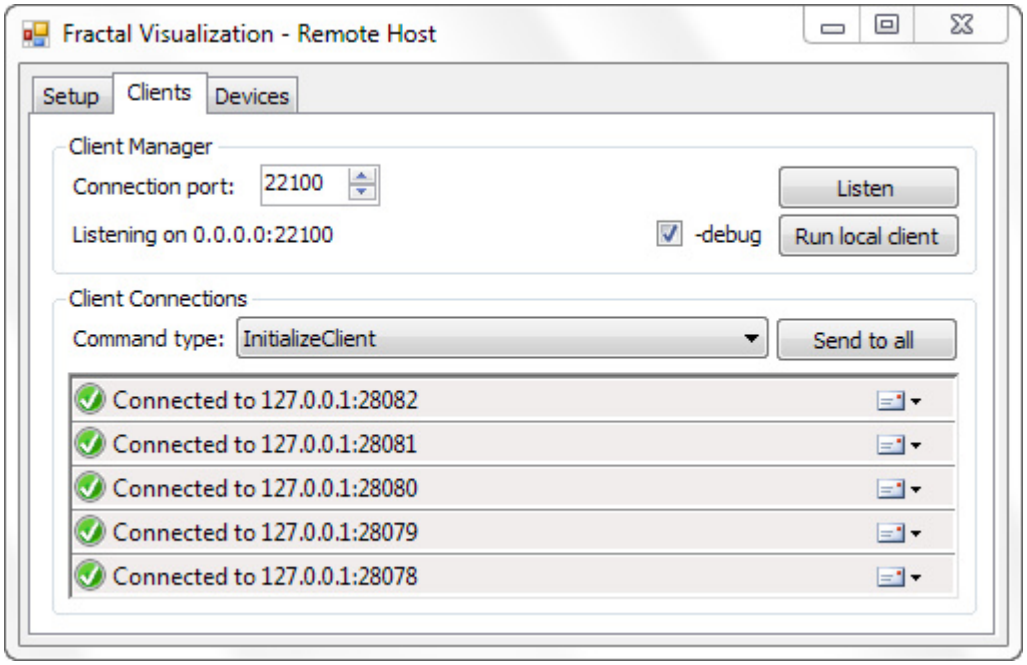



**Abbildung 4.17.:** Der „Clients“-Reiter in der GUI der Remote-Anwendung.

oder mit neuen Werten aktualisiert werden können. Darüber findet sich ein Auswahlfeld, über das Befehle an alle Clients gleichzeitig versendet werden können.

### 4.8.3. Tablets

Der dritte und letzte Reiter der Remote-Anwendung bietet eine GUI für die Kommunikation mit den Tablets (siehe Abbildung 4.18). Für jeden verbundenen Device-Client wird ein `TrackableDeviceControl` angelegt, welches ein `ServerControl` für den Verbindungsstatus und das manuelle Versenden von `Commands` und ein `VrpnHostControl` für die Verarbeitung von Tracking-Daten beinhaltet. Letzteres findet sich dort aufgrund der zuerst angedachten Synchronisierung von VRPN-Ereignissen über den Remote (siehe Diskussion bei 4.9.5), wird aber im endgültigen System nicht verwendet.

## 4.9. Diskussion

Dieser Abschnitt diskutiert die im Rahmen der Implementierung getroffenen Entscheidungen zum Entwurf des Systems, die Probleme, die bei der Implementierung auftraten und die Konsequenzen für das System.

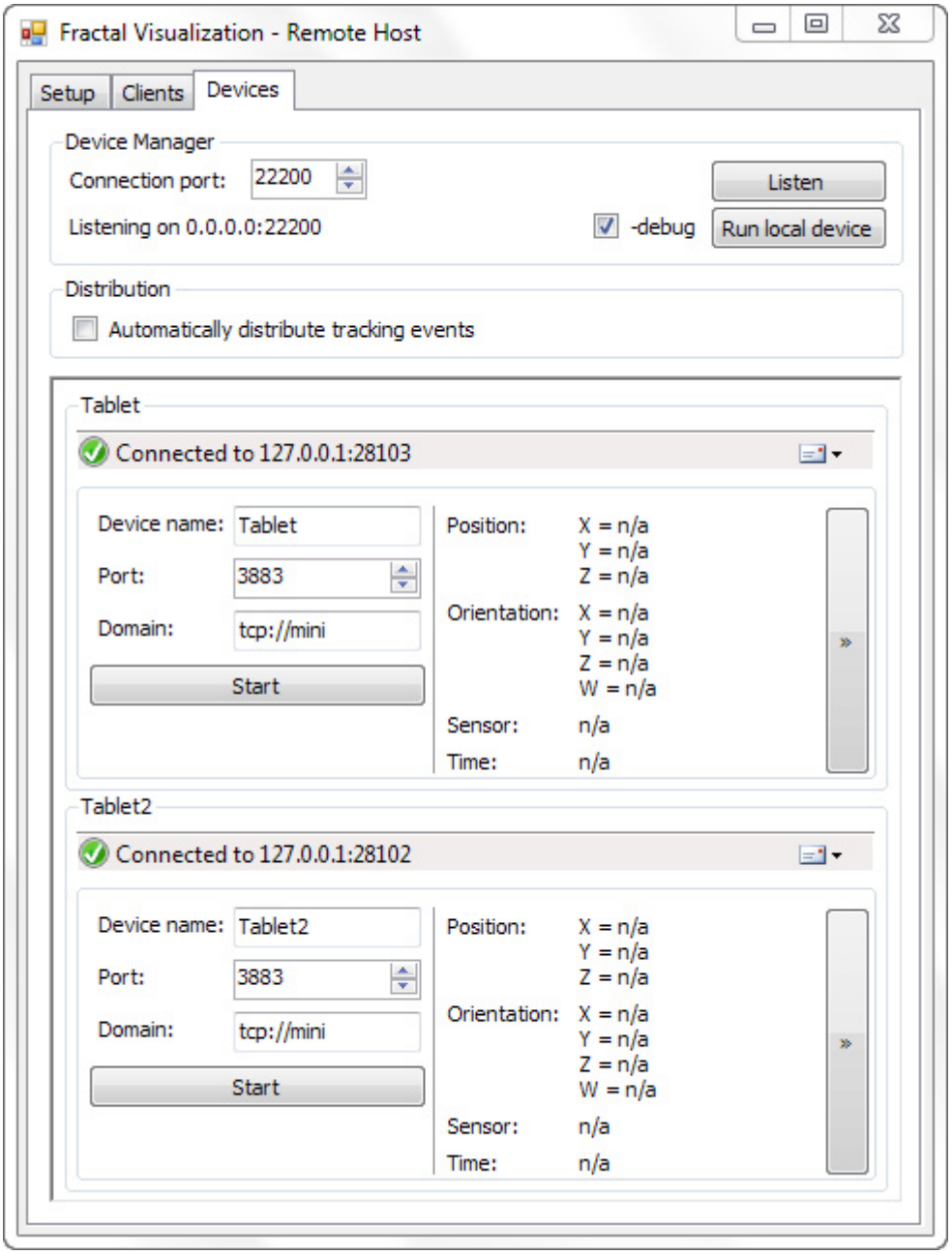



**Abbildung 4.18.:** Der „Devices“-Reiter in der GUI der Remote-Anwendung.

### 4.9.1. Rendering

Wie bereits in Unterabschnitt 4.6.2 angedeutet wurde, besteht teilweise eine ungünstige Kopplung zwischen Direct3D-Rendering und den Visualisierungsklassen des Systems. Diese rührt hauptsächlich daher, dass bei der Implementierung zunächst eine ganz andersartige Rendering-Logik im Mittelpunkt stand, die den Entwurf des Systems maßgeblich beeinflusste (siehe hierzu Kapitel 5 – „Alternatives System“). Entsprechend viele Ansatzpunkte für Verbesserungen gibt es beim Rendering:

Die Renderlogik, die bislang im `ClientForm` bzw. den davon abgeleiteten Klassen liegt, könnte direkt an die `DistributedRenderer` ausgelagert werden. Da jeder Benutzer prinzipiell in der Lage sein soll, mit seiner präferierten Visualisierung den Datensatz zu untersuchen, würde es so genügen, pro `UserView` einen `DistributedRenderer` zu speichern, der in der Lage ist, auf einen gegebenen Direct3D-Kontext zu rendern. Dann könnte jeder Renderer optimiert auf seine Geometrie auch ein eigenes Eingabelayout und dementsprechend auch eigene Vertex-Strukturen verwenden. Ganz vermeiden lässt sich ein gewisses Mindestmaß an Kopplung nicht, da der Renderer im Falle der Verwendung von Texturen auch das `Device` kennen müsste, für das die Ressourcen erstellt werden sollen. Auch die Shader könnten – je nach Bedarf – beim Renderer definiert sein. Fraglich ist, wie hier mit dem Vertex-Shader verfahren werden soll:

- Zum einen kann der Vertex-Shader im Renderer definiert werden. Dann müsste ihm allerdings auch die View-Transformation bekanntgemacht werden, damit die Vertizes im Shader entsprechend transformiert werden können.
- Zum anderen könnte der Vertex-Shader auch zentral im `ClientForm` gesetzt werden. Das würde dem `DistributedRenderer` allerdings einen Teil der Flexibilität und Optimierung nehmen, die er aus einem eigenen Vertex-Shader möglicherweise gewonnen hätte.

Bei einer Definition der Direct3D-Shader bei der jeweiligen Visualisierung könnte als weiterer Vorteil die Tesselations-Phase mit dem Hüllen-Shader zur Erzeugung der Geometrie genutzt werden. Der Hüllen-Shader könnte wie in Abschnitt 4.5.1 beschrieben aus dem Mittelpunkt und der Orientierung eines Quadrates für den verallgemeinerten Pythagoras-Baum die Vertizes des Vierecks berechnen.

Gegenwärtig ist ausschließlich das Rendering mit Direct3D 11 implementiert, das auf dem Surface-Pro-Tablet funktioniert. Das andere Tablet unterstützt lediglich Direct3D 9c. Zwischen D3D 11 und D3D 9 klafft eine derartige Lücke, dass eine Umstellung des Renderings nicht einfach zu erreichen ist, sondern große Teile der Direct3D-Initialisierung und des Renderings neu implementiert werden müssten. Dies liegt insbesondere an den Effekten, wie sie für Direct3D 11 in der Datei *shaders11.fx* definiert wurden. Mit *Effects 11*[3] existiert beispielsweise ein eigenes Projekt, das die Migration von D3D-9-Effekten zu D3D 11 adressiert.

[3]Siehe `https://fx11.codeplex.com/`.

### 4.9.2. Mehrbenutzerfähigkeit

Ein System, das mehreren Benutzern gleichzeitig die kollaborative Verwendung ermöglicht, stellt sich damit vor Situationen, die in einem Einzelnutzersystem keine Rolle spielen und daher von besonderer Bedeutung für die Benutzbarkeit sind.

### 4.9.3. Visualisierungen

Die Allgemeinheit, mit der die Software im Hinblick auf die Vielfalt möglicher Visualisierungsmethoden entworfen wurde, bringt für die Anwendung einige Tücken mit sich. Eine große Herausforderung stellt das Wechseln zwischen zwei Visualisierungen dar (siehe Kapitel 7), da hier unklar ist, wie der Erhalt der *mental map* des Betrachters angemessen unterstützt werden kann, da die Visualisierungen konzeptionell so weit voneinander abweichen können, dass sich das Mapping der sichtbaren Ausschnitte schwierig oder mehrdeutig gestalten kann. Fraglich ist auch die softwarearchitektonische Kopplung, die sich daraus ergibt, dass interaktive Features – wo auch immer diese definiert sein mögen – sich auf die visuellen Darstellungen auswirken. Die Struktur einer Software, die all dies zu kombinieren versucht, ist also inhärent recht komplex. Speziell bei der hier implementierten Software wird die Architektur den Aufgaben, die in ihr abgebildet sein sollten, nicht gerecht, da viele Umbauten und Paradigmenwechsel während der Arbeit daran stattfanden.

### 4.9.4. Netzwerk

Durch die Verwendung von `Commands` ist durch die Objekterzeugung für jede Nachricht ein relativ großer Aufwand gegeben, sodass die Größe der über das Netzwerk verschickten Pakete die benötigte Mindestgröße um ein Vielfaches übersteigt. Die Auswirkungen hiervon machen sich insbesondere bei der Interaktion auf dem Tablet bemerkbar: Dort wird in Echtzeit ein `Transformation-Command` erstellt und innerhalb eines `ExecutionRequest-Commands` an den Remote versendet, der die innere Nachricht an alle verbundenen Clients verteilt, also im Fall der VISUS-Powerwall verfünffacht. So wird vielfach pro Sekunde eine mehrere Kilobyte große Nachricht über ein Drahtlosnetzwerk verschickt, in ein anderes Subnetz geroutet, und von dort aus in fünffacher Ausführung weiterverteilt. Diese Pakete werden alle asynchron versendet, d. h. im Endeffekt arbeiten alle verbundenen Geräte mit derselben Transformation, aber innerhalb einer Interaktion ist keine Garantie für den gleichzeitigen Erhalt der Nachricht auf den Rendering-Clients gegeben. Besonders stark ist der Effekt bei einer sehr langen Geste, beispielsweise beim Zoomen, zu sehen: Am Anfang der Geste und innerhalb einiger Sekunden wird die Erwartungshaltung gut erfüllt, da die zu sehenden Transformationen auf der Powerwall zeitlich gut mit der ausgeführten Geste korrelieren. Verlängert man die Geste, so stellt sich jedoch eine Latenz zwischen der Ausführung und der Darstellung auf der Powerwall ein. Zusätzlich dazu ergibt sich durch die Asynchronität des Vorganges eine Zeitverschiebung zwischen den einzelnen Clients, sodass die verschiedenen Teile des verteilten Systems stark abweichende Transformationen anwenden und zeigen.

Außerdem war anfangs angedacht, die VRPN-Tracking-Daten auf dem Remote-Rechner zu empfangen und diese über die bestehende TCP/IP-Verbindung zu den Clients zu synchronisieren. Hieraus ergibt sich eine große Netzwerkauslastung, da pro Sekunde eine große Menge an Informationen versendet wird. Diese Auslastung hat Auswirkungen, wenn neben der Tracking-Synchronisation auch noch andere Nachrichten über denselben Socket versendet werden sollen. Es gibt mehrere Lösungsansätze für diese Problematik.

**Puffern** Anstatt bei jedem Gesten-Ereignis eine Aktualisierung des Transformationsunterschiedes über das Netzwerk zu veröffentlichen, könnten die Gesten-Daten in einem Puffer gesammelt und in größeren Zeitabständen akkumuliert verteilt werden. Hierbei müsste vor dem tatsächlichen Versenden der Gesamtunterschied, der sich durch die angefallenen Interaktionen ergeben hat, berechnet werden, was aber einen unbedeutenden Rechenzeit-Anteil im Vergleich zur sofortigen Verteilung der Nachrichten über die .NET-Socket-API ausmachen würde.

Bei dieser Variante handelt es sich um einen Trade-Off: Es wird verhindert, dass kontinuierliche Synchronisation zu inkonsistenten Latenzen in der Anzeige führen; hierdurch geschieht allerdings auch keine wirkliche Echtzeit-Interaktion mehr. Jedoch werden die Interaktionen auch nicht erst dann synchronisiert, wenn die Transformation abgeschlossen ist.

**Prioritäten-Warteschlange** Der `Server` könnte sich, anstatt jede Nachricht sofort zum asynchronen Versenden an den Socket zu geben, eine Warteschlange mit den noch zu sendenden Objekten halten, wobei jede Nachricht gekennzeichnet ist, ob sie mit hoher Priorität, also synchronisationskritisch, oder mit niedriger Priorität behandelt werden soll, also durchaus zu einem späteren Zeitpunkt nachgeholt oder sogar verworfen werden kann. Dieses Vorgehen macht insbesondere dann Sinn, wenn keine Unterschiede zwischen aufeinanderfolgenden Zuständen, sondern die absoluten Zustände versendet werden. Dies ist beispielsweise bei den Tracking-Daten der Fall: Dort werden immer die absoluten Daten über den wahrgenommenen Starrkörper gebroadcastet. Um die Netzwerklast zu reduzieren und dabei immer noch eine annähernd in Echtzeit stattfindende Synchronisation der Zustände zu erreichen, könnten sämtliche Befehle mit niedriger Priorität gekennzeichnet werden, die abgearbeitet werden, wenn der Socket sonst nicht beschäftigt wäre. In bestimmten Abständen sollte dann eine Nachricht mit hoher Priorität gekennzeichnet werden, sodass alle noch ausstehenden Aktualisierungen verworfen werden und der aktuellste Stand synchronisiert wird.

### 4.9.5. Optischer Tracker

Bei der Implementierung der Interaktion, die mit dem optischen Tracking-System ausgeführt wird, traten einige Probleme auf, deren Ursprünge teilweise nicht ganz geklärt werden konnten.

Zunächst zeigte das Kamerasystem das eigentümliche Verhalten, nach einer gewissen Zeit, in der hochfrequent Tracking-Daten gebroadcastet wurden, nur noch sehr spärlich Updates zu

senden. Dabei wurden für eine große Zeitspanne (Größenordnung 0,5–1,5 Minuten) überhaupt keine Ereignisse ausgelöst, dann kamen wieder Pakete mit einer hohen Framerate für etwa eine Sekunde und das Warten begann von Neuem. Die Ursache hierfür ist unbekannt, kann aber seitens der Tracking-Software vermutet werden. Für einen kleinen Teil des Problems konnte Abhilfe geschafft werden, indem die Tracking-Update-Ereignisse im `DeviceForm` nicht mehr synchron, sondern asynchron weiterverarbeitet wird. Hierdurch wurde das beschriebene Verhalten gelindert, existiert allerdings prinzipiell immer noch.

Ein weiteres Problem war, dass die Tracking-Software die getrackten Objekte teilweise sehr schlecht erkannte oder sie mit anderen Trackables verwechselte. Vor allem Letzteres legte die Vermutung nahe, dass die Starrkörper, an denen die lackierten Kugeln befestigt sind, einander zu ähnlich waren, da sie insbesondere lediglich zweidimensional waren, d. h. alle Kugeln in einer Ebene positioniert waren. Daher wurden zunächst zwei jeweils dreidimensionale Starrkörper gebaut, um diese Verwechslungsgefahr zu verringern, was jedoch keinen Erfolg brachte – die Tablets wurden nach wie vor häufig als andere Geräte interpretiert. Das Problem konnte erst dadurch beseitigt werden, dass die anderen Starrkörper, die dem Kamerasystem bekannt waren, aus dessen Erkennungsliste gelöscht wurden, was natürlich keine Lösung darstellt. Die genaue Ursache dieses Verhaltens wurde bislang noch nicht gefunden.

Das gravierendste, weiterhin bestehende Problem, für das kein Workaround gefunden werden konnte, ist die das gewaltige Rauschen, das bei bestimmten Positionen und Orientierungen im Raum im Orientierungsquaternion auftritt. Dies führte leider dazu, dass sich die mit dem Tracker angedachte Interaktion in den meisten Situationen als gänzlich unbrauchbar entpuppte. Hielt man beispielsweise das Tablet gerade auf die Wand und zielte damit etwa auf den Ursprung in der Mitte der Projektionsfläche, so erhielt man derartige Abweichungen im Orientierungsquaternion, dass die berechneten Schnittpunkte mit der Wand zwischen zwei Kameraframes über 30 m auseinanderlagen.

Als erste Idee wurde ein Plausibilitätstest implementiert, der Kamera-Updates nur dann zuließ, wenn sich zwischen dem neuen und dem letzten bekannten Zustand eine Veränderung des Look-Targets von höchstens 1 m ergeben hatte. Daraus resultierte eine Startschwierigkeit des interaktiven Systems, sodass der Benutzer zum Beginnen seiner Interaktionen zunächst in die Nähe des initial anvisierten Punktes – dem Koordinatenursprung – kommen musste. Schwerer wog jedoch die Anzahl der Frames, die aufgrund des Tests verworfen wurde: Je nach Lage und Orientierung des Tablets waren etwa 3 %, an anderen Stellen weit über 99 % hiervon betroffen. Diese Messung wurde wie folgt vorgenommen: Der Benutzer bewegte sich einige Zeit (etwa 1–2 Minuten) frei vor der Powerwall. Es wurden bei jedem VRPN-Ereignis parallel zwei Zählvariablen verwendet, von denen die eine die längste Serie valider Daten, die andere die längste Serie von Daten, die durch den Plausibilitätstest verworfen wurden, akkumulierte. Die Messungen ergaben, dass an brauchbaren Stellen, also beim besten Ergebnis, unter 34 Frames ein ungültiger Frame war, an den schlechten Stellen hingegen nur ein brauchbarer unter 541 unplausiblen Frames war. Bei so vielen beschädigten Frames, dass das Rendering mehrere Sekunden lang keine Veränderung zeigte, war sowohl mit als auch ohne Plausibilitätstest keine brauchbare Interaktion möglich.

# 5. Alternatives System

Im Rahmen dieser Diplomarbeit wurde zunächst ein Hybrid-Rendermechanismus aus GDI+ und Direct3D zu implementieren versucht. Die Tests, bei denen dieses System auf der Powerwall ausgeführt wurde, offenbarten jedoch, dass die Performance des Renderings nicht ausreichte, um für einen Betrachter eine angenehm flüssige Interaktion zu erreichen. Aufgrund der Vermutung, dass unter diesen Bedingungen kollaboratives Arbeiten zwischen mehreren Benutzern gar nicht erst zustande kommen würde, wurde die Entwicklung eingestellt und auf die ausschließliche Verwendung von Direct3D ausgewichen; siehe hierzu Kapitel 4.

Der Grund für die ursprüngliche Entscheidung, GDI+ und Direct3D zu kombinieren, rührte von der Anforderung her, Details-on-Demand, darunter insbesondere Text, auf der Powerwall darzustellen. Die hierfür benötigten Funktionalitäten bietet zwar DirectWrite, jedoch besteht eine nichttrivial zu behebende Inkompatibilität zwischen Direct3D-11-`Devices` und DirectWrite unter Windows 7. Unter Windows 8 ist das Problem behoben, sodass Textrendering auf den Tablets zur Verfügung steht, nicht aber auf den Powerwall-Rechnern, auf denen Windows 7 installiert ist. Außerdem hatte die Idee, Texturen statt ausschließlich eingefärbter Geometrie zu verwenden, durchaus ihren Reiz, wenn sich zweidimensionale Visualisierungen schlecht durch eine Primitivengeometrie darstellen lassen. Dementsprechend ist von den Änderungen besonders die Visualisierungskomponente des Systems betroffen.

## 5.1. Visualisierungsansatz

Wie im implementierten System wird die `DistributedRenderer`-Klasse verwendet, die jedoch eine `Render`-Methode enthält, die nach der Initialisierung des Renderers mit einem `Setup` verwendet werden kann. Diese erhält als Parameter ein GDI+-`Graphics`-Objekt, das für eine zuvor im Arbeitsspeicher erstellte `Bitmap` erzeugt wurde.

Für die Darstellung der Visualisierungen auf den verschiedenen Geräten bzw. Bildschirmen wurde Direct3D 11 verwendet. Im Gegensatz zu GDI+ sollten dadurch schnelle Transformationen für flüssige Effekte, insbesondere auf der Powerwall, ermöglicht werden. Auch im alternativen System waren Benutzeransichten, wie sie in Unterabschnitt 4.6.1 beschrieben werden, analog konzipiert. Zusätzlich zum Vertex-Buffer mit der Geometrie der Ansicht, die sich aus ihren `ViewTiles` (siehe Unterabschnitt 5.1.2) ergab, wurden darin auch die mit GDI+ erzeugten Texturen gespeichert. Zunächst wurde eine einzige Textur, die die gesamte Visualisierung zeigte, auf einem einzelnen Quadrat gezeigt. Da es sich bei der Textur um eine

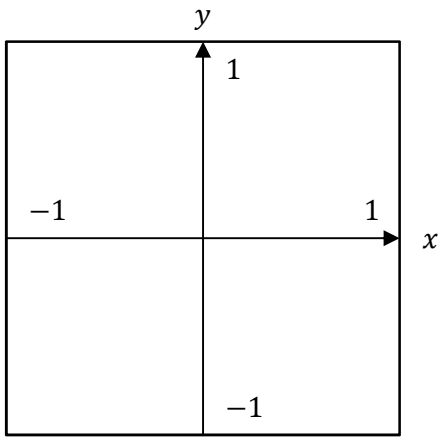



**Abbildung 5.1.:** Der sichtbare Bereich und das Koordinatensystem im Direct3D-Viewport.

einmal vorgerenderte Rastergrafik handelte, konnte durch Zoomen die Grenze überschritten werden, ab der die Texel beim Rasterisieren mehr als einen Pixel beanspruchten. Aus diesem Grund wurde beim Rendern auf ein iteratives Verfahren zurückgegriffen, das die beliebig hochauflösende Darstellung einer Ansicht ermöglicht. Diese Vorgehensweise wird in Unterabschnitt 5.1.2 näher erläutert.

### 5.1.1. Transformationen

Jedem Client war bekannt, wie die Powerwall aufgebaut ist und welcher Teil daraus in seinen Renderbereich fällt. Die Vertizes der Geometrie enthielten Pixelkoordinaten, die, um Berechnungen zu begünstigen, relativ zum Ursprung des Gesamtsystems angegeben waren; dieser lag in der oberen, linken Ecke der Powerwall. Der im verwendeten Direct3D-`Viewport` sichtbare Koordinatenbereich lag in horizontaler und vertikaler Ausdehnung im Intervall $[-1, 1]$ (siehe Abbildung 5.1). Relevant für die Transformation der Koordinaten war das Rechteck $r_i = (x_i, y_i, w_i, h_i)$, das die Lage des Displays in Powerwall-Pixeldimensionen beschreibt. Hierbei war $(x_i, y_i)$ der Ursprung des lokalen Pixelkoordinatensystems eines Clients relativ zur oberen, linken Ecke der Powerwall. Insgesamt resultierte eine Tile-Transformationsmatrix $\mathbf{M}_i$, die eine affine Abbildung darstellte, die die Pixelgeometrie des $i$-ten Clients in Direct3D-Viewport-Koordinaten transformierte. Diese ergibt sich aus der Translation der Geometrie in den Clientbereich, der Normierung der Vertex-Koordinaten und Umkehrung der Achsen auf die Spezifikation des Direct3D-Koordinatensystems und der Verschiebung dieser normierten Koordinaten in das richtige Achsenintervall:

$$\begin{aligned} \mathbf{M}_i &= \mathbf{T}_{\text{Client}_i} \cdot \mathbf{S}_{\text{Viewport}_i} \cdot \mathbf{T}_{\text{Viewport}} \\ &= \mathbf{T}(x_i, y_i) \cdot \mathbf{S}\left(\frac{2}{w_i}, -\frac{2}{h_i}\right) \cdot \mathbf{T}(1, 1) \end{aligned}$$

Durch die Tile-Transformation geschieht der Wechsel aus dem pixelorientierten Geometriekoordinatensystem in das im Direct3D-Viewport sichtbare Koordinatensystem.

### 5.1.2. Auflösungsrekursion

Bei der Initialisierung einer Ansicht wurde die gesamte Visualisierung in eine Textur mit einer vorher bestimmten, von der Hardware des ausführenden Rechners abhängigen Größe gerendert. Hierdurch sollte der Benutzer in die Lage versetzt werden, eine Übersicht über den gesamten Datensatz zu erlangen. Wollte er jedoch Detailelemente wie beispielsweise sehr klein dargestellte Knoten betrachten, musste er hierzu in die Darstellung hineinzoomen. Technisch bedeutete das, dass die Transformation der Ansicht eine größere Skalierung beschrieb. Sei $(w_q, h_q)$ die Größe des Quadrats, auf dem die Rastertextur der Größe $(w_t, h_t)$ gerendert werden soll. Dabei wird die gesamte Geometrie durch die Tile-Transformation durch einen Skalierungsfaktor $s_W$ gestreckt. Wenn durch die Benutzerinteraktion die Reskalierungsbedingung

$$\frac{s_W \cdot w_q}{w_t} > 1 \quad \vee \quad \frac{s_W \cdot h_q}{h_t} > 1 \tag{5.1}$$

erfüllt wird, wird die Textur nicht mehr pixelgenau dargestellt. Durch Zoomen ergibt sich also kein Detailzugewinn mehr. Dann musste der auf dem Bildschirm sichtbare Teil der Visualisierung mit einer größeren Auflösung neu gezeichnet werden. Hierfür wurde die Klasse `ViewTile` implementiert.

`ViewTiles` stellten auf hierarchische Weise die Geometrie einer Ansicht dar, die von Direct3D effizient zum Zeichnen benutzt werden konnte. Die Kernidee der Auflösungsrekursion bestand darin, die Ansichtsgeometrie beim Zoomen immer weiter zu verfeinern, sodass die Reskalierungsbedingung aus Gleichung 5.1 aufgehoben wird. Hierzu wurden die zum Zeitpunkt des Eintretens der Bedingung auf dem Bildschirm sichtbaren `ViewTiles` durch vier wiederum quadratische, kleinere Tiles ersetzt, die jeweils ebenfalls eine Textur mit einer Auflösung von $(w_t, h_t)$ beinhalteten. Mit dieser Methode wurden demnach immer quaternäre Bäume erzeugt; diese Datenstruktur ist als *quadtree* bekannt [Sam84]. Abbildung 5.2 illustriert das Vorgehen.

Ein `ViewTile` konnte demnach zwei „Zustände“ einnehmen, je nachdem, wo es in der Hierarchie stand:

- Ein *innerer Knoten* enthält vier untergeordnete `ViewTiles`, die seine Fläche einnehmen.
- Ein *Blattknoten* stellt ein Quadrat in der Geometrie dar, dessen Größe sich aus seinem Elternknoten ergibt. Darauf wird eine Textur der Größe $(w_t, h_t)$ gezeichnet.

Eine Sonderstellung nahm die Wurzel der `ViewTile`-Hierarchie ein. Sie war der einzige Knoten, der explizit durch einen Konstruktoraufruf erzeugt werden konnte. Dabei wurden der verwendete Renderer sowie die zu verwendende Texturauflösung übergeben. Die Wurzel enthielt im Gegensatz zu allen anderen Knoten festgelegte Werte für die Lage des Tiles, während

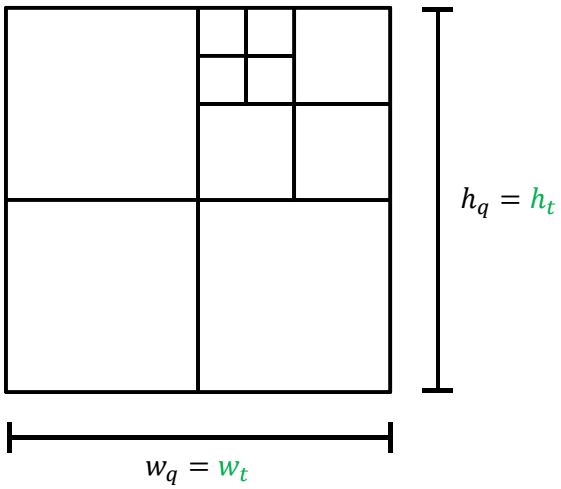



**Abbildung 5.2.:** Rekursive Verfeinerung der Auflösung durch zusätzliche Geometrie mithilfe von ViewTiles.

alle ihr untergeordneten Knoten diese Werte relativ zu ihrem jeweiligen Elternknoten berechneten. Bei der Instanziierung der Wurzel fand auch die Berechnung des Skalierungsfaktors $s_t$ statt, der vor dem Rendern der Texturen mit GDI+ angewendet wurde, sodass die Visualisierung in ihrer längsten Ausdehnung genau die Grenzen der Textur ausfüllt. So wurde das Seitenverhältnis der Visualisierung nicht zerstört. Hierzu wurde die vom jeweiligen Renderer berechnete Bounding-Box $G = (x_G, y_G, w_G, h_G)$ der gesamten Visualisierung benötigt:

$$s_t = \min\left\{\frac{w_t}{w_G}, \frac{h_t}{h_G}\right\}$$

Außerdem musste der Koordinatenursprung vor dem Rendern der Textur um $(x_T, y_T)$ so verschoben werden, dass der Mittelpunkt der Bounding-Box genau in der Mitte der Textur zum Liegen kam. Hierzu wurde zunächst der Bounding-Box-Mittelpunkt zum Texturursprung und danach in die Texturmitte verschoben:

$$\begin{aligned} x_T &= -\left(x_G + \frac{w_G}{2}\right) \cdot s_t + \frac{w_t}{2} \\ y_T &= -\left(y_G + \frac{h_G}{2}\right) \cdot s_t + \frac{h_t}{2} \end{aligned}$$

Abbildung 5.3 zeigt das Resultat dieser Transformationen. Dabei bedeutet $s_t \cdot B_V$ die um den Skalierungsfaktor $s_t$ gestreckte Bounding-Box der Visualisierung.

`ViewTiles`, die der Wurzel oder anderen, inneren Knoten untergeordnet sind konnten nicht via Konstruktor erzeugt werden, sondern entstanden durch Aufruf der `Split`-Methode im Elternknoten. Dabei wurde die `Children`-Eigenschaft des Elternknotens auf ein Array mit vier untergeordneten Tiles gesetzt, die durch Aufruf des privaten Konstruktors instanziiert und initialisiert wurden. Aus dem als Parameter übergebenen übergeordneten Knoten ließen sich die meisten Eigenschaften des Kindknotens berechnen oder direkt übernehmen. So renderte

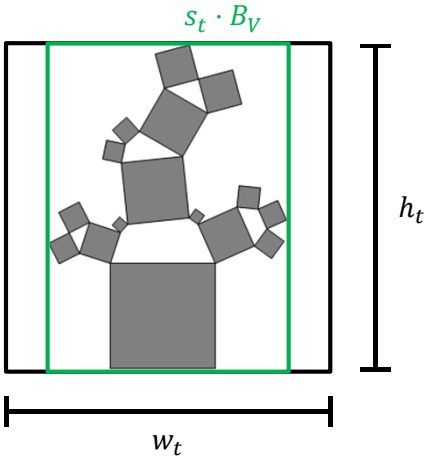



**Abbildung 5.3.:** Einpassung der Visualisierung in die Geometrie der Wurzel der ViewTile-Hierarchie.

der untergeordnete Knoten mit demselben `Renderer`, übernahm also auch die Bounding-Box der Visualisierung vom Elternknoten. Um eine Verfeinerung der Darstellung zu erreichen, bot der Kindknoten die gleiche Texturauflösung wie der Elternknoten, besaß aber einen doppelt so großen Skalierungsfaktor. Die geometrischen Eigenschaften leiten sich von den entsprechenden Werten des übergeordneten Knotens ab. Sei $B_p = \left(x_{B_p}, y_{B_p}, w_{B_p}, h_{B_p}\right)$ das Rechteck, das die Vertizes des Elternknotens definiert, und $C_p = \left(x_{C_p}, y_{C_p}, w_{C_p}, h_{C_p}\right)$ mit $x_{C_p}, y_{C_p}, w_{C_p}, h_{C_p} \in [0, 1]$ das Rechteck mit den Texturkoordinaten an den Rändern des Tiles. Gebe außerdem $a_c \in [0, 3]$ die Anordnung des Kindknotens relativ zum übergeordneten Knoten an, wobei die Werte für oben links (0), oben rechts (1), unten links (2) und unten rechts (3) stehen. Dann sind

$$B_c = \begin{cases} \left(x_{B_p}, y_{B_p}, \frac{w_{B_p}}{2}, \frac{h_{B_p}}{2}\right), & \text{falls } a_c = 0 \\ \left(x_{B_p} + \frac{w_{B_p}}{2}, y_{B_p}, \frac{w_{B_p}}{2}, \frac{h_{B_p}}{2}\right), & \text{falls } a_c = 1 \\ \left(x_{B_p}, y_{B_p} + \frac{h_{B_p}}{2}, \frac{w_{B_p}}{2}, \frac{h_{B_p}}{2}\right), & \text{falls } a_c = 2 \\ \left(x_{B_p} + \frac{w_{B_p}}{2}, y_{B_p} + \frac{h_{B_p}}{2}, \frac{w_{B_p}}{2}, \frac{h_{B_p}}{2}\right), & \text{falls } a_c = 3 \end{cases}$$

$$C_c = \begin{cases} \left(x_{C_p}, y_{C_p}, \frac{w_{C_p}}{2}, \frac{h_{C_p}}{2}\right), & \text{falls } a_c = 0 \\ \left(x_{C_p} + \frac{w_{C_p}}{2}, y_{C_p}, \frac{w_{C_p}}{2}, \frac{h_{C_p}}{2}\right), & \text{falls } a_c = 1 \\ \left(x_{C_p}, y_{C_p} + \frac{h_{C_p}}{2}, \frac{w_{C_p}}{2}, \frac{h_{C_p}}{2}\right), & \text{falls } a_c = 2 \\ \left(x_{C_p} + \frac{w_{C_p}}{2}, y_{C_p} + \frac{h_{C_p}}{2}, \frac{w_{C_p}}{2}, \frac{h_{C_p}}{2}\right), & \text{falls } a_c = 3 \end{cases}$$

die entsprechenden Geometriewerte beim Kindknoten.

Jeder Knoten besitzt eine bestimmte Tiefe $d$ in der `ViewTile`-Hierarchie. Diese berechnet sich als

$$d = \begin{cases} 0, & \text{falls Wurzelknoten} \\ d_p + 1, & \text{sonst.} \end{cases}$$

Mithilfe der Tiefe und den bereits genannten Größen ließ sich eine allgemeine Berechnungsvorschrift für die Translation der Textur beim Rendern formulieren, die sowohl auf innere Knoten, insbesondere auch die Wurzel, als auch auf Blattknoten zutrifft:

$$\begin{aligned} x_T &= -\left(x_{B_V} + \frac{w_{B_V}}{2}\right) \cdot s_t + \left(\frac{1}{2} - x_C\right) \cdot w_t \cdot 2^d \\ y_T &= -\left(y_{B_V} + \frac{h_{B_V}}{2}\right) \cdot s_t + \left(\frac{1}{2} - y_C\right) \cdot h_t \cdot 2^d \end{aligned}$$

#### Parallelität

Für eine bessere Performance beim Rendern war eine zweistufige Parallelisierung angedacht, indem das GDI+-Rendering, welches auf der CPU stattfindet, auf mehrere Threads verteilt wird. Die Idee und die Gründe, warum diese Methode unpraktikabel war, wird im Folgenden erläutert.

In der ersten Stufe wurden vier Threads angestoßen, um bei einer Unterteilung eines `ViewTiles` die vier neuen Texturen parallel zu rendern. Während das Rendering der feiner aufgelösten Texturen noch nicht abgeschlossen war, wurden in den Kindknoten stattdessen noch die alten Texturen mit gestreckten Pixeln angezeigt. Da nicht für jeden gerenderten Frame in Direct3D pro `ViewTile` der Vertex-Buffer oder eine entsprechende Transformation angewendet werden sollte, enthielt der `UserView` die gesamte Geometrie seiner Tiles. Dieser Vertex-Buffer wurde neu geschrieben, wenn ein Tile unterteilt wurde. Dabei wurden statt der Texturkoordinaten für die feinere Textur zunächst (angegeben für die diagonal gegenüberliegenden Vertizes jeweils links oben und rechts unten) die Koordinaten $(0, 0)$ bis $\left(\frac{1}{2}, \frac{1}{2}\right)$, $\left(\frac{1}{2}, 0\right)$ bis $\left(1, \frac{1}{2}\right)$, $\left(0, \frac{1}{2}\right)$ bis $\left(\frac{1}{2}, 1\right)$ und $\left(\frac{1}{2}, \frac{1}{2}\right)$ bis $(1, 1)$ eingetragen und die grobe Textur in der Texturliste an den entsprechenden Indizes erneut eingefügt. Wenn der Thread eines neuen Tiles das Rendering der Textur abgeschlossen hatte, wurden im Vertex-Buffer an der Stelle, wo die Vertizes des Tiles standen, Vertizes mit denselben Positionen, aber Texturkoordinaten zwischen $(0, 0)$ und $(1, 1)$ geschrieben und in der Texturliste die alte durch die neue Textur ersetzt. Auf diese Weise kann die Geometrie- und Texturverfeinerung stattfinden, ohne das Direct3D-Rendering zu blockieren, wodurch ein flüssiges Interaktionsverhalten sehr gestört würde. Außerdem lässt sich so theoretisch die Dauer, bis alle Kindtexturen gerendert sind, um drei Viertel verkürzen. Aufgrund der Threadaffinität der GDI+-API und der erzeugten GDI+-Objekte[1] müsste hierzu allerdings pro Tile in einem eigenen GUI-Thread ein Renderer-Objekt mit eigenen `GraphicsPaths` gestellt werden, was eine größere Speicherbelastung und

[1] Siehe http://social.msdn.microsoft.com/Forums/windows/en-US/25cceaa4-f058-482a-8381-03698f73651a/advanced-gdi-rendering-in-multiple-threads.

hohe Aufwände für das Vorbereiten der Geometrie zur Folge hätte. Außerdem können sehr wohl bei einem Reskalierungsvorgang mehr als vier neue Texturen gerendert werden müssen, wenn mehr als ein `ViewTile` zu teilen ist. Das parallele Rendering der Texturen stellte also eine große Herausforderung an die Ressourcenverwaltung des Systems dar, da bei $n$ neuen Knoten auch $n$ Renderer bereits vorbereitet zur Verfügung stehen müssten. Die Geometrie on-the-fly erneut vorzubereiten, war keine Option, da dies von allen Prozessen, die am Rendering beteiligt waren, mit Abstand am zeitaufwändigsten war und deshalb auch einmalig initial stattfand.

Die zweite Stufe der Parallelisierung lief auf dasselbe Problem der Threadaffinität hinaus: Da der `GeneralizedPythagorasTreeRenderer` seine vorbereiteten `GraphicsPaths` in einem Array speicherte, hätte beim Rendering selbst eine Parallelisierung vorgenommen werden können, indem mehrere Threads parallel über unterschiedliche Bereiche des Pfade-Arrays iterierten. Hier hätte noch großer Aufwand hätte geleistet werden müssen, da wieder die Pfad-Daten hätten kopiert werden müssen. Außerdem wurde das `Graphics`-Objekt, das zum Zeichnen verwendet werden sollte, in einem anderen Thread erstellt als die Pfade, weshalb Zwischenergebnisse im finalen Bild übereinandergeblendet hätten werden müssten. Insgesamt hätte sich bei der Parallelisierung grob eine Beschleunigung des Renderings um den Faktor der Anzahl der Prozessorkerne ergeben können.

### 5.1.3. Snapshots

Um sich besondere Stellen zu merken oder um in der Aktionshistorie rückwärts zu navigieren, konnte der Benutzer für seine Ansicht Snapshots der aktuellen Darstellung anlegen. Mithilfe eines Snapshots war es möglich, den früheren Zustand der Ansicht wiederherzustellen. Bei der Erstellung des Wiederherstellungspunktes wurden daher folgende Daten erhoben:

- `RendererType`: Der Typ des Renderers, der zur Visualisierung verwendet wurde.
- `RendererState`: Ein Objekt, mit dem sich der Renderer in den gleichen Zustand versetzen kann wie bei der Erstellung des Snapshots.
- `Owner`: Der Name der Ansicht, für den der Snapshot erstellt wurde.
- `CreationTime`: Der Zeitpunkt der Erstellung.
- `Translation`, `Rotation` und `Scaling`: Werte, die die Berechnung der gesamten Transformation erlauben, die auf die Ansicht angewendet waren.

Beim `RendererState` handelt es sich um ein Objekt beliebigen Typs, das über die Schnittstelle der `DistributedRenderer`-Klasse erhalten werden konnte. Zum Laden und Speichern von `RendererStates` wurden dort die Methoden `GetCurrentState` und `ApplyState` angeboten. Das Statusobjekt musste die gesamte Konfiguration des Renderers enthalten, daher mussten die Methoden von jeder abgeleiteten Visualisierungsklasse unterschiedlich implementiert werden und waren in der Basisklasse als abstrakt gekennzeichnet.

Der Code zur Erstellung des Snapshots fand sich im `ClientForm`, da für dessen Darstellung auf der Powerwall bzw. auf dem Tablet eine Textur gerendert wurde, die den aktuellen Zustand der Ansicht zeigt. Dazu wurde die Methode `CreateSnapshot(UserView)` aufgerufen, die eine Direct3D-`Texture2D` instanziierte und darauf ein Off-Screen-Render-Target erstellte. Dieses wurde im `OutputMerger` als Render-Target eingestellt, bevor die `RenderView`-Methode aufgerufen wurde, ganz ähnlich der Erstellung von Übersichtstexturen im tatsächlichen System (siehe Abschnitt 4.7.3). Damit die gesamte Ansicht $V$ auf dem Snapshot sichtbar war, wurde die Bounding-Box der Ansicht $B_V$ berechnet, wobei alle Transformationen Beachtung fanden. Aus der Größe der Bounding-Box ergeben sich die Skalierungsfaktoren, die in den Transformations-`MatrixBuffer` geschrieben werden müssen, bevor die Ansicht in die Textur gerendert wird. Da zum Rendern dasselbe `Device` verwendet wurde, auf das die gesamte Ausgabe gerendert wurde, musste vor dem Rendering für das Render-Target ein neuer `Viewport` erstellt werden.

Soll ein früherer Zustand wiederhergestellt werden, konnte aus der `UserView`-Klasse die Methode `LoadSnapshot` verwendet werden. Diese bot zwei Überladungen:

- `LoadSnapshot(int)` lud den Wiederherstellungspunkt aus der Historie der `UserView`-Instanz, für die er erstellt wurde, der sich an der angegebenen Position befindet.
- `LoadSnapshot(Snapshot)` lud einen beliebigen Wiederherstellungspunkt. Diese Überladung diente der Anforderung, dass ein Benutzer auch die Snapshots beliebiger anderer Nutzer laden können sollte. Hierbei musste darauf geachtet werden, wie mit der Snapshot-Historie der Ansicht weitergearbeitet wurde.

## 5.2. Interaktion

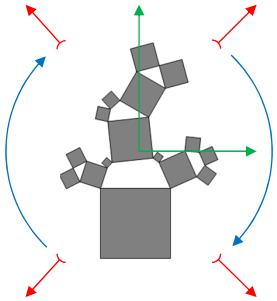
Für das alternative System waren zunächst Interaktionen angedacht, die von den für das endgültige System implementierten Interaktionen abweichen. Diese werden im Folgenden vorgestellt.

### 5.2.1. Einzelnutzer-Features

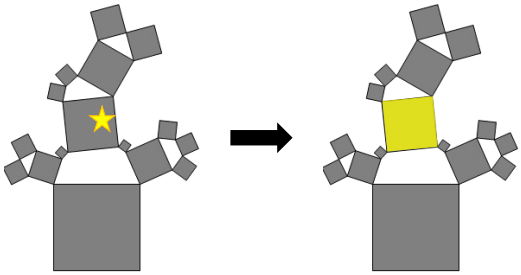
Einige der interaktiven Funktionen sollten die Arbeit eines einzelnen Benutzers unterstützen. Eine visuelle Übersicht über die Features wird in Abbildung 5.4 gegeben.

#### Transformationen

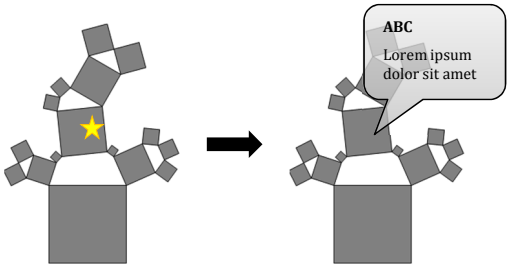
Durch Gesten war es möglich, die Ansicht auf der Powerwall zu transformieren. Abbildung 5.4a zeigt die möglichen Transformationen, zu denen Translation, Skalierung und Rotation zählten. Die Implementierung ist dieselbe, wie sie im neuen System zur Verwendung von Gesten entwickelt wurde; siehe hierzu Abschnitt 4.7.1.



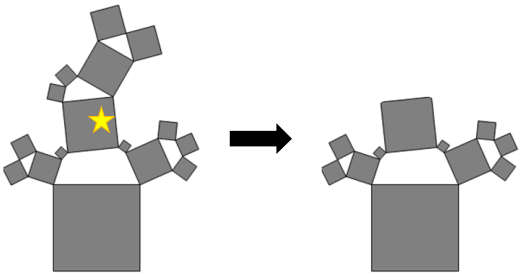
**(a)** Translation, Skalierung und Rotation der Ansicht



**(b)** Selektion bzw. Markierung von Hierarchieelementen




**(c)** Details-on-Demand zu Hierarchieelementen



**(d)** Auf- und Zuklappen von Hierarchieelementen

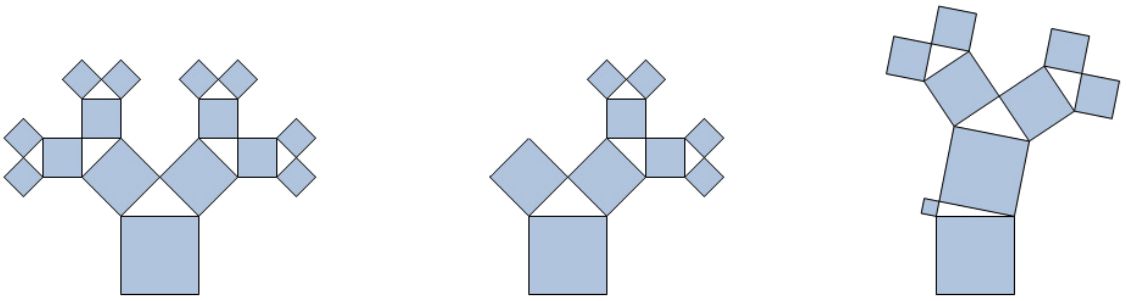
**Abbildung 5.4.:** Eine Übersicht über die interaktiven Features des alternativen Systems.

#### Selektion

Mithilfe eines Klicks, d. h. einer Tap-Geste auf dem Tablet, konnten Knoten in der Hierarchie ausgewählt werden, die dann durch eine Bildüberlagerung auf der Powerwall sichtbar gemacht wurden (siehe Abbildung 5.4b). Zu jeder Benutzeransicht gehörte dabei eine eigene Farbe, in der die Hervorhebungen halbtransparent gezeichnet wurden. Details zur Implementierung finden sich in Abschnitt 4.7.1.

#### Details-on-Demand

Zu einzelnen Knoten der Hierarchie sollten genauere Informationen angezeigt werden können. Dazu sollte ein zweidimensionales Overlay gerendert werden, das im Datensatz gespeicherte oder aus dem Datensatz generierte bzw. akquirierte Informationen beinhaltete (siehe Abbildung 5.4c). Dieses konnte dann beispielsweise zum Fallbeispiel der NCBI-Taxonomie Daten über die jeweilige Art, entsprechende Bilder oder Hyperlinks umfassen.



**(a)** Die vollständige („aufgeklappte“) Hierarchie

**(b)** Zuklappen durch Ausblenden des Unterbaums

**(c)** Zuklappen durch Entfernen des Unterbaums

**Abbildung 5.5.:** Auf- und Zuklappen von Hierarchieknoten mit unterschiedlicher Kollabierungs-Semantik.

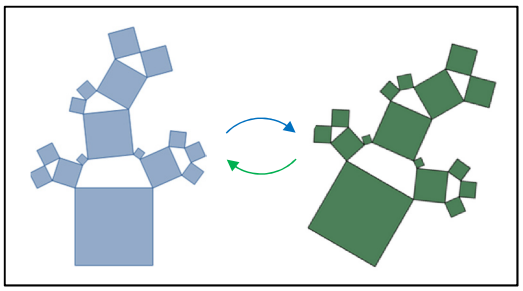
#### Auf- und Zuklappen

Als hierarchiebezogenes Feature war das Expandieren und Kollabieren von inneren Knoten angedacht, wodurch der gesamte Unterbaum ausgeblendet (wie illustriert in Abbildung 5.4d) bzw. wieder angezeigt wurde. Diese Funktionalität brachte einige Schwierigkeiten technischer Natur mit sich, die in Performance-Problemen resultierten.

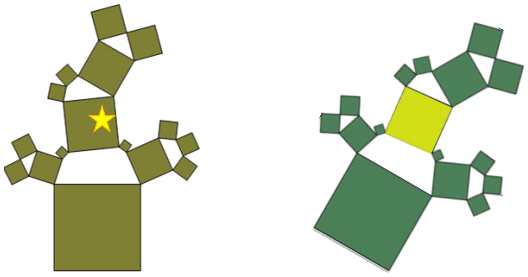
Im Falle des verallgemeinerten Pythagoras-Baums als Visualisierung kann das Expandieren und Kollabieren von Knoten auf unterschiedliche Weisen interpretiert werden, die in Abbildung 5.5 gezeigt sind:

**Kollabieren durch Ausblenden** Die zum Unterbaum eines eingeklappten Knotens gehörenden Elemente werden ausgeblendet (siehe Abbildung 5.5b). Infolgedessen müssten die GDI+-Texturen – zunächst zumindest diejenigen, die geometrisch einen Anteil an dem Unterbaum hatten – neu gerendert werden, um den versteckten Teil nicht mehr auf der Powerwall anzuzeigen. Je nach Anwendungsszenario kann sich dies positiv oder negativ auf das Verständnis des Benutzers mit seiner *mental map* auswirken [HMM00]. Nutzt er die Interaktion beispielsweise, um den Teilbaum lediglich als uninteressant zu markieren und möchte ihn daher nicht mehr sehen, spiegelt diese Variante des Ausblendens vermutlich am ehesten die Intentionen des Nutzers wider. Das restliche Layout der Visualisierung wird dabei beibehalten, die Interaktion hat folglich geringe Auswirkungen auf sein Verständnismodell der Darstellung. Fraglich ist, ob Nutzer nicht eher eine Platzersparnis durch das Ausblenden eines Knotens intendieren; hierfür wäre diese Art des Zuklappens nicht geeignet.

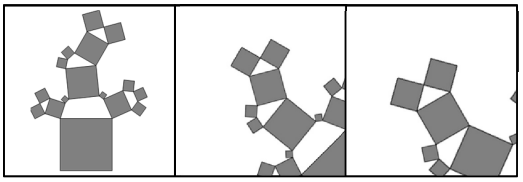
**Kollabieren durch Entfernen** Die zweite Variante bringt noch größere Performance-Einbußen mit sich: In diesem Fall wird der kollabierte Unterbaum nicht einfach ausgeblendet,



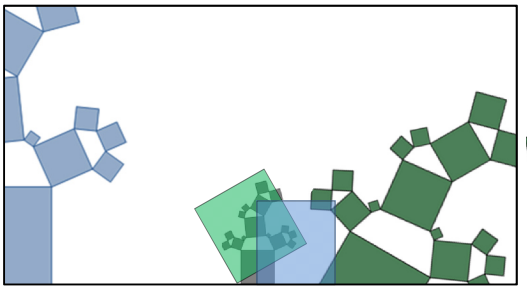
**(a)** Umordnen von Benutzeransichten



**(b)** Einblenden von Markierungen anderer Benutzer in der eigenen Ansicht



**(c)** Snapshots zum Rückgängigmachen von Interaktionen



**(d)** Übersichtsansichten mit Bildausschnitten zwischen benachbarten Ansichten

**Abbildung 5.6.:** Eine Übersicht über die kollaborativen bzw. benutzerübergreifenden, interaktiven Features des alternativen Systems.

sondern vom Verständnis her aus dem Datensatz „gelöscht“. Im Gegensatz zur ersten Methode bleibt mit dieser Variante die Visualisierungsmetapher der verallgemeinerten Pythagoras-Bäume erhalten. Wenn wie in dieser Arbeit die Größe des Unterbaums als Metrik verwendet wird, die die Größe eines Knotens in der Visualisierung bestimmt, kann der kollabierte Knoten „zusammenschrumpfen“, da er keine sichtbaren Kindknoten mehr besitzt. Dies wiederum wirkt sich negativ auf das Verständnis des Datensatzes aus, da sich hierdurch die Darstellung – je nach Größe des eingeklappten Teilbaums – stark ändern kann (siehe Abschnitt 4.5.1 zur Stabilität).

### 5.2.2. Kollaborative Features

Es waren auch interaktive Features angedacht, die der Kollaboration mehrerer Benutzer dienlich sein sollten. Abbildung 5.6 zeigt diese Funktionalitäten.

#### Umordnen von Ansichten

Das Umordnen der Ansichten (siehe Abbildung 5.6a) war im alternativen System gleich angedacht, wie es jetzt im neuen System umgesetzt wurde; die Beschreibung findet sich in Abschnitt 4.7.2.

#### Daten anderer Nutzer

Es sollte einem Benutzer möglich sein, aufbauend auf dem Arbeitsfortschritt anderer Benutzer Selektionen anderer Nutzer zu sehen (siehe Abbildung 5.6b) und deren Snapshots zu benutzen, um auf denselben Stand zu wechseln. Dieses Feature war noch nicht implementiert, als die Arbeit am alternativen System eingestellt wurde. Das Konzept beim Teilen der Selektionen war, dass beispielsweise eine grafische Überlagerung in der Farbe des jeweiligen Benutzers, der den Knoten markiert hatte, angezeigt wurde. So konnten mehrere Benutzer einander wichtige oder interessante Teile des Datensatzes visuell annotieren.

### 5.2.3. Generelle Interaktionen

Einige der Interaktionen lassen sich nicht genau diesen Kategorien zuordnen, insbesondere, wenn sie sowohl die Arbeit einzelner Benutzer als auch das kollaborative Arbeiten unterstützen.

#### Snapshots

Snapshots, also Bildschirmfotos des Arbeitsbereiches eines Benutzers, waren ein grundlegendes Navigationskonzept im alternativen System. Durch sie konnte der Benutzer seinen Fortschritt beim Arbeiten mit der Visualisierung verfolgen (siehe Abbildung 5.6c) und einen früheren Stand wiederherstellen oder wieder zu einem fortgeschritteneren Stand wechseln. Die gesamte Historie wurde auf der Powerwall in kleinen Bildern beim Arbeitsbereich des Benutzers dargestellt. Es gab mehrere Ideen zur Umsetzung dieser Interaktion, beispielsweise durch eine Darstellung auf dem Tablet und einfache Klicks auf das jeweilige Bild oder durch andere Metaphern wie Filmstreifen, durch deren Einzelbilder mit einer Wisch-Geste geblättert werden konnte.

Für die Implementierung des Snapshot-Renderings und softwarearchitektonische Anmerkungen, siehe Unterabschnitt 5.1.3.

#### Übersichts-Ansichten

Das Konzept der Übersichts-Ansichten im alternativen System stimmt mit dem im endgültigen System überein und ist daher in Abschnitt 4.7.3 beschrieben.

## 5.3. Diskussion

In diesem Abschnitt werden die Stärken und Schwächen der alternativen Software diskutiert. Insbesondere werden hier detailliert die Gründe betrachtet, die dazu geführt haben, dass die Arbeiten an dem System eingestellt wurden. Stattdessen wurden einige technologische Änderungen vorgenommen, sodass daraus die in Kapitel 4 vorgestellte Software entstand.

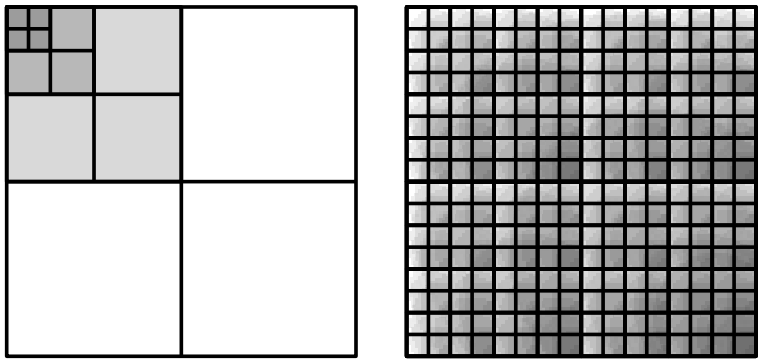
### 5.3.1. Performance

Das bei weitem gravierendste Problem des alternativen Systems war die Renderdauer der `ViewTiles`, wenn diese aufgeteilt wurden. Die in Abschnitt 5.1.2 angesprochene Beschleunigung des Renderings einer einzelnen Textur durch paralleles Iterieren über das Array der Pfade war aus den dort genannten Gründen der benötigten Datenvervielfältigung nicht praktikabel. Die Rendergeschwindigkeit einer einzelnen Textur reichte nicht aus, um flüssiges Interaktionsgefühl bei Ansichtstransformationen entstehen zu lassen. Die Dauer beim neuen Rendern der feiner aufgelösten Texturen hängt maßgeblich von der verfügbaren Prozessorleistung des Systems ab und belief sich auf den Powerwall-Rechnern mit dem Referenzdatensatz, der NCBI-Taxonomie, auf etwa 1–2 Sekunden. Auf dem Entwicklungssystem, einem Lenovo Thinkpad R500 mit einem Intel®-Core™-2-Duo-Prozessor mit einer Taktfrequenz von 2 GHz, benötigte das Rendering pro Textur sogar 3–5 Sekunden.

Solche Verzögerungen sind im Kontext von Anwendungen, die interaktiv zu bedienen sind, inakzeptabel. Die Vermutung war, dass kollaboratives Arbeiten gar nicht erst zustande kommen würde, wenn die Benutzer ständig damit beschäftigt sind, in ihrer Wahrnehmung der Daten und bei der Arbeit mit dem System die Interaktionseinbußen zu kompensieren. Da Interaktion und Kollaboration ein Fokus dieser Arbeit sein sollten, war dies der Hauptgrund, aus dem für die weitere Entwicklung performantere Technologien gesucht wurden.

### 5.3.2. Präzision

Ein Vorteil dieser Technik gegenüber dem tatsächlich implementierten System besteht in der Präzision, mit der die Geometrie berechnet werden kann. Während in Direct3D die Komponenten der Vertizes, die die Visualisierungsgeometrie definieren, üblicherweise in 32-Bit-Gleitkommazahlen angegeben werden, wurden im alternativen System Zahlen mit doppelter Genauigkeit verwendet, die durch die Nutzung der Eltern-Kind-Beziehung relativ zum Elternknoten zu genauen Ergebnissen führen. Zwar stößt auch diese Präzision bei einem ausreichend großen Datensatz an ihre Grenzen, doch sie kann durch die Wahl eines geeigneten Datentyps (wie z. B. `decimal`) noch weiter verbessert werden. Die Verschlechterung der Performance durch die Verwendung von Datentypen, für deren Rechenoperationen keine performanten Maschinenbefehle zur Verfügung stehen, könnte jedoch hingenommen werden, da es sich bei der Vorbereitung der Daten um eine einmalige und vor der eigentlichen Benutzung des Systems durchführbare Operation handelt.



**(a)** Der beste Fall **(b)** Der schlechteste Fall

**Abbildung 5.7.:** Veranschaulichungen für die Speicherbedarf-Abschätzungen im besten und schlechtesten Fall der Hierarchieaufteilung.

### 5.3.3. Skalierbarkeit

Die für das Alternativsystem relevante Größe der Renderzeit wurde in Unterabschnitt 5.3.1 diskutiert; eine weitere relevante Dimension ist die Speicherauslastung der Grafikkarte. Diese entsteht hier maßgeblich durch die Texturen, die auf den `ViewTile`s zu sehen sind. Im Anfangsszenario, bei dem es noch keine echte Hierarchie, sondern ein einzelnes `ViewTile` gibt, wird eine Textur gerendert. Diese hat eine in der Anwendungskonfiguration festgelegte Größe $(w_t, h_t)$, wobei beispielsweise auf der Powerwall eine Größe von $w_t = h_t = 4096$ Pixeln verwendet wurde. Die Texturen wurden im RGBA-Farbformat zu jeweils 8 Bit bzw. 1 Byte pro Kanal gespeichert. Pro Textur ergibt sich demnach ein Speicherbedarf von $m_t = w_t \cdot h_t \cdot 4\,\frac{\mathrm{B}}{\mathrm{Pixel}} = 4096 \cdot 4096\,\mathrm{Pixel} \cdot 4\,\frac{\mathrm{B}}{\mathrm{Pixel}} = 64\,\mathrm{MiB}$. Wird dieses `ViewTile` unterteilt, entstehen vier neue Texturen derselben Auflösung. Vereinfacht gesagt kann die alte kann dann entsorgt werden, im System bleibt sie jedoch noch einige Zeit bestehen, bis die letzte Kindtextur gerendert ist. Einzig in dem Augenblick, in dem die letzte Kindtextur fertiggerendert ist und auf die Grafikkarte geladen wurde, besteht eine Überverwendung von $m_t$, bis die alte Textur aus dem Speicher entfernt wurde (was natürlich dann aber auch sofort geschehen kann). Bis auf diesen Sekundenbruchteil beträgt der neue Speicherbedarf also $4 \cdot m_t = 256\,\mathrm{MiB}$. Auf den Powerwall-Rechnern, auf denen $2 \cdot 6$ GiB Grafikspeicher und 24 GiB Arbeitsspeicher zur Verfügung stehen, sind das überschaubare Größenordnungen, doch speziell auf den Tablets, die mit deutlich geringerer Grafikleistung und kleinerem Grafikspeicher auskommen müssen, benötigen die Texturen schnell mehr Speicher, als zur Verfügung steht; dann müsste eine aufwändige Ressourcenverwaltung für die Bilder implementiert werden, die wiederum CPU-Zeit beansprucht und eine bessere Leistung erzielt als übliche Paging-Mechanismen.

Generell lässt sich der von den Texturen verwendete Speicher auf der Grafikkarte im schlechtesten Fall einer balancierten und im besten Fall einer einseitig expandierten `ViewTile`-Hierarchie abschätzen. Sei $d$ die nullbasierte Tiefe dieser Hierarchie. Im schlechtesten Fall (Abbildung 5.7b) beläuft sich der Speicherbedarf für einen komplett aufgeteilten Baum auf $m_{\max} = m_t \cdot 4^d$. Aufgrund der möglicherweise exponentiell abnehmenden Größe der Qua-

drate in den tieferen Ebenen der verallgemeinerten Pythagoras-Baum-Visualisierung ist es nicht unüblich, dass ein Benutzer auch einen entsprechend großen Zoom-Faktor verwendet, woraus tatsächlich eine schlechte Skalierbarkeit bezüglich der Speicherauslastung resultiert. Im besten Fall (Abbildung 5.7a) beträgt für $d \geq 1$ die Speicherauslastung $m_{\min} = (3d + 4) \cdot m_t$. Für diese Auslastung könnten aber Speicherverwaltungs-Methoden implementiert werden, die dieses Problem entschärfen.

### 5.3.4. Auflösungsrekursion

Die Erzeugung der View-Geometrie in quaternären Bäumen (siehe Unterabschnitt 5.1.2) hätte – statt auf der CPU ausgeführt zu werden – in den Tessellierungs-Shader ausgelagert und auf der Grafikkarte berechnet werden können. Diese minimale Verbesserung hätte sich allerdings nicht spürbar auf die Arbeit mit dem System ausgewirkt, da das Rendern der Texturen die bei Weitem größte Rechenzeit beanspruchte. Diese Funktionalität wurde allerdings im Rahmen der Einstellung der Arbeit am alternativen System diskutiert und wurde daher nicht implementiert.

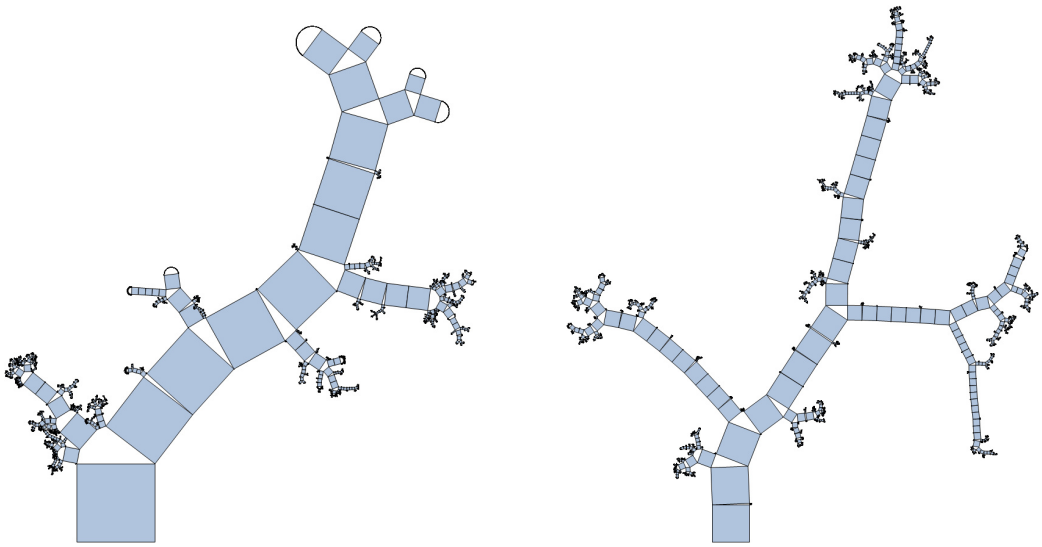
# 6. Fallstudien

Das System wurde im Laufe der Entwicklung mit verschiedenen Datensätzen getestet, um einen Überblick dafür zu bekommen, welche Größenordnungen von Daten von der Visualisierung und der Interaktion gut verarbeitet werden können und wo die Grenzen der Anwendung liegen. Die beiden Datensätze werden im Folgenden vorgestellt. Am Beispiel der phylogenetischen Taxonomie wird die Arbeit mit der Software exemplarisch vorgestellt, beim anderen Fall werden grob die gewonnenen Einsichten präsentiert.

@articlesamet1984quadtree, title=The quadtree and related hierarchical data structures, author=Samet, Hanan, journal=ACM Computing Surveys (CSUR), volume=16, number=2, pages=187–260, year=1984, publisher=ACM

## 6.1. Phylogenetische Taxonomie

Für die erste Fallstudie wurde eine phylogenetische Taxonomie, also eine hierarchische Ordnung von Spezies, herangezogen. Der Datensatz ist eine Taxonomie des NCBI (siehe



**(a)** Der Dateisystem-Datensatz **(b)** Die NCBI-Taxonomie

**Abbildung 6.1.:** Die Darstellung der verwendeten Datensätze als verallgemeinerte Pythagoras-Bäume.

Abbildung 6.1b), die 324.276 Elemente umfasst. Die Hierarchie hat eine maximale Tiefe von 42 Ebenen und verfügt über 263.691 Blattknoten, die jeweils eine bestimmte Spezies bzw. Lebensform darstellen. Gespeichert ist die Hierarchie in einer knapp 7 MiB großen Datei im Newick-Format. Im Folgenden wird exemplarisch das Vorgehen beschrieben, wie mit der Software vom Starten bis zum Beenden Daten visualisiert und interaktiv untersucht werden können.

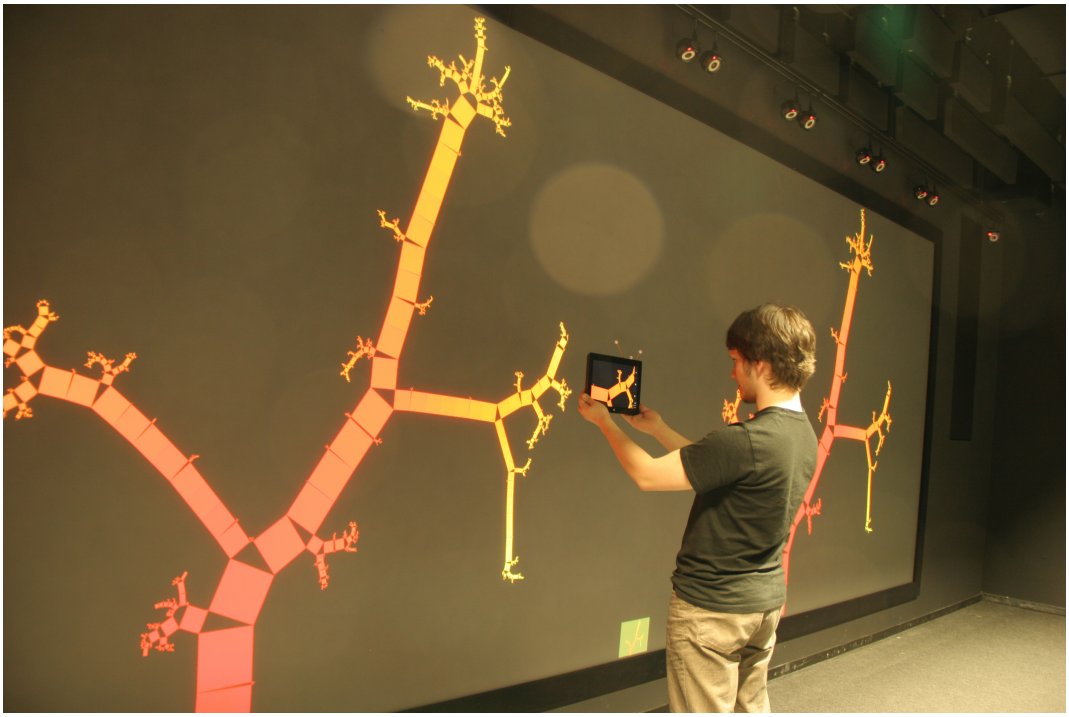
Zunächst wird auf dem Remote-Rechner die Remote-Anwendung gestartet, woraufhin als erstes der *Setup*-Reiter zu sehen ist. Falls nicht schon beim Programmstart geschehen, kann dort der gewünschte Datensatz geladen werden. Auch die Definition des verteilten Rendering-Systems kann bei Bedarf hier angepasst und der initial zu verwendende `DistributedRenderer` ausgewählt werden.

Sobald die Projektoren und Rendering-Rechner der Powerwall verfügbar sind, können darauf die Client-Anwendungsteile gestartet werden. Dabei muss in den Kommandozeilen-Parametern übergeben werden, wohin sich der Client verbinden soll, also die IP-Adresse und der Port des Remote-Rechners. Der Port kann bei Bedarf im *Clients*-Reiter der Remote-Anwendung geändert werden. Wenn die Verbindung zum Remote-Rechner hergestellt ist, werden die Clients von dort aus initialisiert. Die Informationen, die über das Netzwerk kommuniziert werden, enthalten auch den kompletten Datensatz; die Client-Rechner müssen also keinen Zugriff auf die entsprechenden Dateien haben. Wenn die Initialisierung abgeschlossen ist, wird auf der Powerwall der Datensatz so groß wie möglich dargestellt.

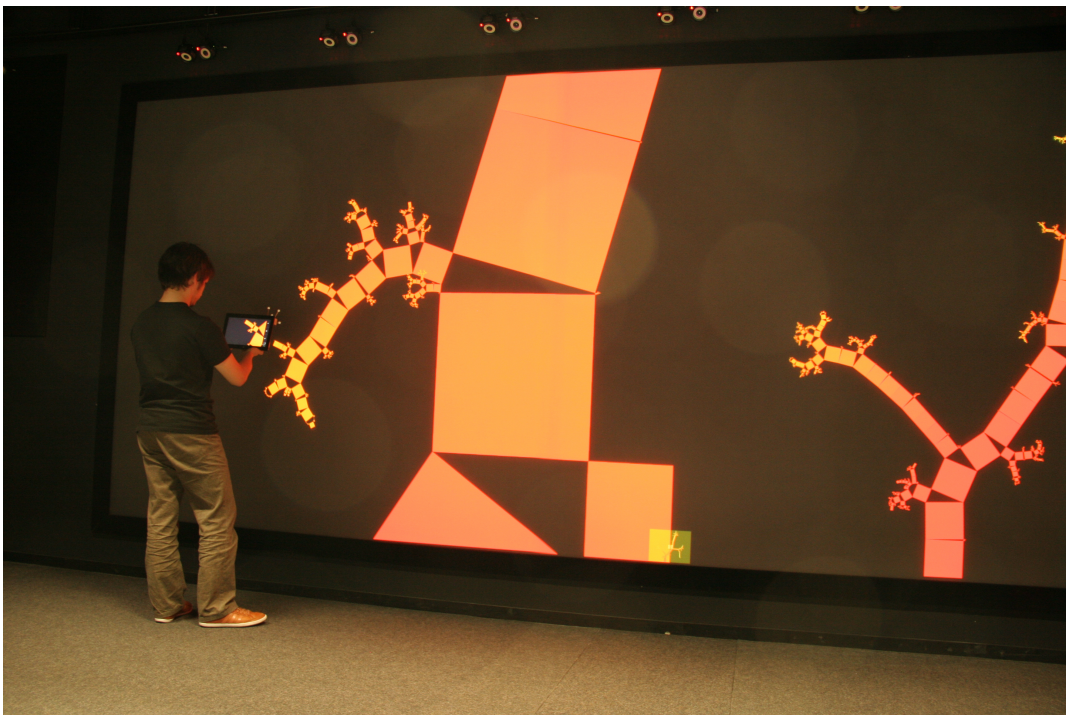
Parallel dazu können auch die Device-Anwendungen auf den Tablets gestartet werden. Auch hier muss die Konfiguration der Verbindung zum Remote-Rechner über die Kommandozeile geschehen. Für jedes verbundene Gerät erscheint ein Bedienfeld im *Devices*-Reiter, in dem der Name des Tablets gesetzt werden muss und die VRPN-Verbindung für die Abnahme der Daten des optischen Trackers festgelegt wird. Dabei muss der festgelegte Name des Tablets mit dem Name des getrackten Starrkörpers übereinstimmen. Mithilfe dieser Steuerelemente muss nach der Konfiguration analog zu den Rendering-Clients auch das Tablet initialisiert werden. Nachdem der Datensatz übertragen und vorbereitet wurde und die Verbindung zum optischen Tracker hergestellt wurde, zeigt das Tablet als initiale Transformation das Bild so an, als würde der Betrachter durch sein Tablet genau auf die Mitte der Powerwall blicken.

Nachdem jetzt auf allen beteiligten Systemen die Software ausgeführt wird, kann der Benutzer mit der Untersuchung beginnen und die interaktiven Features nutzen, um Einblicke in den Datensatz zu gewinnen. Ein Beispiel hierfür zeigt Abbildung 6.2a: Der Benutzer besieht sich hier einen Ast des verallgemeinerten Pythagoras-Baumes, der eine weitreichende Abspaltung einer Art in der Spezies-Taxonomie markiert. Auf dem Tablet kann er zusätzliche Informationen zu den Knoten sehen, die er gerade anvisiert, beispielsweise den Namen der Spezies oder ein hierzu passendes Bild. Wenn er möchte, kann er außerdem den Knoten für eine spätere Untersuchung oder zur einfacheren Beschreibung für andere Benutzer markieren. Diese Markierung können die anderen Benutzer bei sich einblenden, sodass sie sehen, welchen Knoten er meinte, und können sich wiederum detaillierte Informationen hierzu einblenden lassen.

Als nächstes zeigt er auf einen anderen Unterbaum, den er genauer untersuchen möchte. Durch Aktivierung der Haltefunktion kann er die Darstellung jetzt um diesen Punkt herum



**(a)** Initial wird auf der Powerwall für zwei Benutzer der gesamte Datensatz arbeitsbereichfüllend angezeigt.



**(b)** Der linke Benutzer hat auf eine interessante Stelle des Datensatzes gezoomt, damit er sie genauer untersuchen kann.

**Abbildung 6.2.:** Bilder von der Software im laufenden Betrieb mit dem NCBI-Datensatz.

transformieren. Da er ihn vergrößert und in der Mitte seines Arbeitsbereiches sehen möchte, zeigt er mit aktivierter Haltefunktion jetzt auf die Mitte seiner Displayhälfte und bewegt sich einige Schritte von der Wand weg. Dadurch findet eine Verschiebung des zuerst fixierten Punktes zum neuen Zielpunkt sowie eine Skalierung der gesamten Darstellung statt. Auf dem Tablet ändert sich die Darstellung hierbei bis auf kleine Ungenauigkeiten nicht. Der Grundgedanke hinter dieser Interaktion ist, dass der gegenwärtig auf dem Tablet sichtbare Bereich die Bewegungen des Benutzers mitvollzieht – bewegt er sich also nach hinten, soll sich die Darstellung gleichermaßen „auf ihn zu“ bewegen, was der Transformation einer Skalierung gleichkommt. Den jetzt vergrößerten Visualisierungsausschnitt kann er wiederum mit seinem Tablet untersuchen (siehe Abbildung 6.2b).

## 6.2. Dateisystem

Der Dateisystem-Datensatz (siehe Abbildung 6.1a) wurde auf dem Entwickler-Rechner erzeugt und spiegelt die hierarchische Ordner- und Dateistruktur der 250-GB-Hauptfestplatte unter der Laufwerksbezeichnung „C:\“ wider. Die Hierarchie hat eine Gesamtgröße von 459.329 Knoten, von denen 388.424 Blätter sind. Ein Blatt bedeutet hier, dass es sich bei dem Knoten entweder um eine Datei oder um einen leeren Ordner handelt. Die geladene Hierarchie wurde für die schnellere Wiederherstellung in einer Newick-Datei (siehe Abschnitt A.1) mit einer Größe von zirka 15,5 MiB gespeichert.

Auch mit dem Dateisystem-Datensatz wurde eine kurze Fallstudie durchgeführt. Mithilfe der transformativen, interaktiven Features konnte hier festgestellt werden, dass es einige Ordner gibt, die extrem viele Dateien enthalten. Es gibt auch einige Ordner, die aufgrund der Farbkodierung als hierarchisch sehr tief gelegen hervorstechen, aber so gut wie überhaupt keine Dateien enthalten.

Ein Gebiet, das genauer untersucht wurde, da es sofort ins Auge sticht, sind die in Abbildung 6.1a ganz oben sichtbaren Ordner, die kaum Unterteilungen in weitere Ordner enthalten, aber enorm viele Blattknoten – in diesem Fall Dateien: Die Ordner enthielten Graphdaten aus einem Softwarevisualisierungs-Projekt, aus dem über viele Revisionen hinweg dynamische Aufruf- und Vererbungsgraphen extrahiert wurden. Diese Ordner enthielten zirka 35.500 bzw. 16.000 Dateien. Ein ähnlicher Anblick findet sich etwa in der Mitte der Visualisierung: Hierbei handelt es sich um den Ordner im Benutzerverzeichnis, in dem temporäre Dateien aus der Browserverwendung zwischengespeichert werden. Auch dieser Ordner enthielt über 11.000 Dateien.

Außer den Einsichten über den Datensatz machte diese Fallstudie allerdings auch klar, dass die Überlappung, die bei verallgemeinerten Pythagoras-Bäumen zwischen zwei Ästen herrschen kann, ein großes Problem für die Praktikabilität der Visualisierung für Analysen darstellt. Hier waren insbesondere oft feine Strukturen, die der Benutzer untersuchen wollte, von in der Nähe befindlichen, größeren Knoten überdeckt, sodass eine Analyse dieser Verästelungen nicht möglich war.

# 7. Zusammenfassung und Ausblick

In dieser Diplomarbeit wurde eine Software entwickelt, die große Datensätze auf einem großen Display rendert. Das Display besteht aus fünf Projektoren, die von fünf verschiedenen, über ein Netzwerk verbundenen Rechnern angesteuert werden. Das dabei entstehende Bild wird auf einer Powerwall mit den Maßen 5,90 $\times$ 2,24 Meter mit einer Auflösung von 10.800 $\times$ 4.096 Pixeln dargestellt. Als Visualisierungsmethode wurden verallgemeinerte Pythagoras-Bäume, eine fraktalbasierte Visualisierung, implementiert. Um mit der Größe der Datensätze zurechtzukommen, wurden interaktive Features implementiert, die sich den speziellen Aufbau des Systems zunutze machen und daher erheblich von den von Interaktionstechniken, die von gewöhnlichen Desktop-Rechnern bekannt sind, abweichen. Die Software ist mehrbenutzerfähig und bietet dem Anwender die Möglichkeit, über eine speziell für Tablets entwickelte Anwendung mit den Visualisierungen zu interagieren. Dabei gehört zu jedem Betrachter eine eigene Ansicht auf der Powerwall, die räumlich und logisch von den Arbeitsbereichen der anderen Nutzer getrennt ist, sodass diese sich nicht gegenseitig bei ihrer Arbeit stören.

Die interaktiven Features sehen vor, dass für benutzerdefinierte Transformationen der Ansicht ein optisches Tracking-System eingesetzt wird, das dem getrackten Tablet über VRPN seine Position und seine Orientierung bekannt macht. Der Benutzer interagiert also nicht hauptsächlich über ein metaphern- oder fensterbasiertes GUI, sondern durch seine physischen Bewegungen, die er mit dem Tablet ausführt. Die Arbeit mit dem Tablet soll sich so anfühlen, als sähe man als Benutzer durch den Bildschirm wie durch ein Fenster auf die Powerwall, wobei man auf dem Tablet neben den Interaktionsmöglichkeiten zusätzlich eine Art „semantische Lupe“ hat, durch die zusätzliche Information sichtbar wird.

Während der Entwicklung wurden einige technische Unwägbarkeiten angetroffen, die die Weiterentwicklung des Systems verzögerten oder gar zurückwarfen. Hierbei gab es zwei besonders gravierende Ereignisse:

**GDI+-Rendering** Aufgrund der Performanceunzulänglichkeit des zunächst mit Direct3D und GDI+ implementierten Systems wurde eine Umstellung vorgenommen, nach der ausschließlich Direct3D für das Rendering verwendet wurde. Hierdurch konnte die Performance erheblich verbessert werden, sodass flüssige Transformations-Interaktionen bei allen verwendeten Datensätzen möglich waren.

**Optischer Tracker** Beim Testen der Interaktion mit dem installierten Tracking-System stellte sich heraus, dass die von den Kameras gelieferten Daten teilweise sehr unzuverlässig und damit für die Interaktion nicht zu gebrauchen waren. Die für die Orientierung gelieferten Werte führten zu Schnittpunkten des Sichtvektors mit der Powerwall, die

eine (bei mehr als 80 FPS absolut unrealistische) Verschiebung von über 20 m zwischen zwei Frames beschrieben.

Andererseits konnte die Arbeit auch zeigen, dass die verfügbare Grafik-Hardware auf den Powerwall-Clients problemlos geeignet ist, Informationsvisualisierung mit relativ großen Datensätzen an der Powerwall zu betreiben.

## Ausblick

**Software-Architektur** Die in der Diskussion der Implementierung in Abschnitt 4.9 beschriebenen Probleme sollten adressiert werden und die Struktur der Software im Rahmen eines umfangreichen Refactorings auf einen Stand gebracht werden, der die Erkenntnisse aus der Implementierung des alternativen und des tatsächlichen Systems beinhaltet. Die implementierte Software enthält beispielsweise einige Funktionalitäten (wie z. B. das in Abbildung 4.18 sichtbare Kontrollkästchen zur automatischen Synchronisierung von VRPN-Ereignissen), die im gegenwärtigen Stand nicht genutzt werden oder obsolet sind.

**Optischer Tracker** Für die Zukunft müssen dringend die Probleme mit dem Tracking-System in den Griff bekommen werden. Leider ist die Ursache für die fragwürdigen Daten noch ungeklärt und es kann sich dabei sowohl um ein Software- als auch um ein Hardwareproblem, möglicherweise sogar um ein externes Problem handeln. Bislang wird vermutet, dass die Anordnung der Infrarot-Kameras nicht geeignet ist, die Bewegungen des getrackten Starrkörpers ausreichend abzubilden bzw. zu berechnen. Falls dies die Ursache ist, könnte man die Schwierigkeiten womöglich durch weitere Kameras, vor allem an der Seite und idealerweise auch hinten, überwinden. Sollte es sich allerdings um ein Softwareproblem handeln, muss geprüft werden, ob ein Softwareupdate die Probleme behebt oder der Hersteller des Kamerasystems eine andere Lösung kennt.

**Verallgemeinerte Pythagoras-Bäume** Wie bereits in Abschnitt 6.2 angedeutet wurde, stellt das Überlappungsproblem der verallgemeinerten Pythagoras-Bäume je nach Fallbeispiel ein solches Hindernis dar, dass es Analysen mithilfe der Software praktisch unmöglich machen kann. Daher sollte auch dieses Verhalten der Visualisierung in zukünftigen Arbeiten Beachtung finden. Sollte das Problem im Allgemeinen nicht zu lösen sein oder eine Lösung nicht in akzeptabler Zeit berechnet werden können, sollte überlegt werden, ob stattdessen eine andere Visualisierungstechnik verwendet werden soll.

**Interaktive Features** Die bislang noch an diesen Schwierigkeiten gescheiterte Interaktion könnte dann auch noch erweitert werden. Auch die Einführung weiterer, interaktiver Features ist eine durchaus denkbare Erweiterung; hierfür wäre jedoch interessant, ob die bisher geplante Interaktion bei Benutzern Anklang findet.

Speziell für die verallgemeinerten Pythagoras-Bäume, aber auch für andere Visualisierungen, die dies ihrer Geometrie entsprechend zulassen, könnten zusätzliche Informationen in Texturen kodiert werden. Bislang wird ausschließlich ein Effekt-Pass verwendet, der die Primitive einfärbt. Je nach Datensatz kann es aber möglich sein, dass die Texturierung der grafischen Elemente einen Zugewinn bedeutet; so könnten beispielsweise bei der NCBI-Taxonomie Bilder der Spezies gerendert werden, die durch den jeweiligen Knoten dargestellt werden. Dieser Vorgang könnte automatisiert werden, indem z. B. die *Google Search API*[1] verwendet wird. Aber auch für andersartige, beispielsweise multivariate Daten können mithilfe von Texturen weitere visuelle Merkmale (z. B. in Form von Glyphen) angebracht werden.

Eine wichtige, aber schwierige Erweiterung ist die Möglichkeit, dass Benutzer zu einer anderen Visualisierungsmetapher bzw.-methode wechseln können. Jedoch muss dann auf irgendeine Weise, die idealerweise zur Schnittstelle der Visualisierungsimplementierungen gehört, eine Abbildung des sichtbaren Bereichs auf den entsprechenden Bereich in der neuen Visualisierung geschehen. Dass dies im Allgemeinen nicht eindeutig möglich ist, zeigt ein Beispiel, in dem vom sichtbaren Bereich eines verallgemeinerten Pythagoras-Baums, in dem zwei Äste, nicht aber deren gemeinsame Wurzel zu sehen ist, zu einer Tree-Map gewechselt werden soll. Wenn beim Wechsel keine Knoten verloren gehen sollen, zeigt die Tree-Map immer eine Übermenge der im verallgemeinerten Pythagoras-Baum sichtbaren Knotenmenge.

**Kollaborative Features** Umfangreiche Erweiterungen des Systems sind auch in Richtung der kollaborativen Interaktionsfeatures möglich. So könnte es Nutzern beispielsweise möglich sein, sich auf den Stand eines Kollegen zu bringen, indem dessen Ansichtstransformation und möglicherweise auch dessen Aktionshistorie übernommen wird. Auch wäre es denkbar, dass ein Benutzer während der Arbeit seine Erkenntnisse vortragen möchte, und es deshalb hilfreich wäre, wenn seine Ansicht für diese Zeit über die ganze Powerwall dargestellt wird. Diese und weitere Ideen sollten allerdings bei einer Studie mit einem realistischen Anwendungsszenario mit plausiblen Aufgabe- und Fragestellungen evaluiert werden.

**Präsentationen** Nicht zuletzt wegen der Ausstattung des Powerwall-Raumes, der bestuhlt ist und ein Rednerpult hat, kann die Software prinzipiell auch für Demonstrationen von Datenvisualisierung und Einsichten vor einem Publikum genutzt werden. Eine interessante Möglichkeit, die sich hier auftut, ist die Einbeziehung des Publikums in die Demonstration. So könnten auch Zuhörer Tablets besitzen, das Geschehen mitverfolgen sowie eigene Interaktionen durchführen, beispielsweise eigene Transformationen, wenn sie einen anderen als den gezeigten Bereich sehen möchten, oder Anmerkungen zur Demonstration festhalten.

**Hardware-Upgrades** Großes Potential besitzen auch Hardware-Upgrades an der Powerwall, die bislang für den Nutzer noch wie ein großer Bildschirm ist; mithilfe neuer Technologie könnte diese Wand auch eine Multitouch-Unterstützung erhalten, was eine riesige Bandbreite

[1] Siehe `http://code.google.com/p/google-api-for-dotnet/`.

an Interaktionsmöglichkeiten eröffnen würde. Auch die Verwendung anderer Geräte als einem Tablet ist denkbar, wenn sich z. B. in einer Studie herausstellen sollte, dass diese Interaktionsmethode für die Aufgaben nicht gut geeignet ist.

**Nutzerstudie** Sobald die Benutzbarkeit des Systems einen für den Praxiseinsatz hinreichenden Stand erreicht hat, ist eine Nutzerstudie zur Feststellung der Brauchbarkeit und Bewertung der interaktiven Features unerlässlich. Da im Rahmen dieses Systems einige neuartige Interaktionsmethoden realisiert wurden, bestehen hierzu noch keine Studien oder etablierte Bewertungsgrundlagen. Beispielsweise könnte untersucht werden, inwiefern die Arbeit mit einem Tablet, das ein gewisses Eigengewicht besitzt, zu einer schnelleren körperlichen Ermüdung führt oder ob die physischen Transformationsmethoden intuitiv und für die Exploration des Datensatzes angemessen sind.

# A. Anhang

## A.1. Newick-Dateiformat

Im Rahmen dieser Diplomarbeit wurde ein Parser für das Newick-Dateiformat erstellt, um den phylogenetischen Baum einzulesen. Dabei wurde die online verfügbare Beschreibung der Sprache von FELSENSTEIN zu Grunde gelegt[1].

Das Newick-Format ist geeignet, Hierarchien in Textdateien kompakt abzubilden. Diese sind so aufgebaut, dass am Ende der Datei die Wurzel der Hierarchie steht und durch eine Klammerungssyntax im Teil davor die Hierarchie definiert wird. Am Ende der Datei steht ein abschließendes Semikolon, am Anfang eine Zeile mit dem Inhalt „`newick;`“. Das Beispiel in Listing A.1 stellt eine einfache Hierarchie dar, wie sie in Abbildung 2.1 auf Seite 19 gezeigt wird. Tabelle A.1 gibt eine Übersicht über die syntaktischen Möglichkeiten des Formats.

Kern des `NewickParsers` ist die `CreateNode`-Methode, die aus dem Inhalt der Datei und dem gegebenen Lese-Index den Knoten, der an diesem Index endet, erzeugt sowie rekursiv dessen Kinder erstellt. Aufgrund des referenziell mitgegebenen Index besitzt Algorithmus A.1 eine Laufzeitkomplexität von $\mathcal{O}(n)$, wobei $n$ die Länge der zu lesenden Datei ist. Für den *content*-Parameter wird dabei die „untere Zeile“, also die reinen Hierarchieinformationen, übergeben. Der initiale, nullbasierte Laufindex beginnt bei $i = \text{len}(content) - 1$. Von hinten nach vorne wird zunächst der Name, dann sein eventuell vorhandener Parameter gelesen. Wenn eine öffnende Klammer (von hinten: „)“) gelesen wird, wird die Methode auf der ersten Stelle des darin enthaltenen, inneren Knotens wieder aufgerufen.

| **Syntax** | **Beschreibung** |
|---|---|
| `;` | Ein einzelner, unbenannter Wurzelknoten. |
| `root;` | Ein einzelner Knoten auf der Wurzelebene. |
| `[2.5]root;` | Ein einzelner Knoten mit dem Parameter 2,5 auf der Wurzelebene. |
| `()root;` | Wurzelknoten mit unbenanntem Unterknoten. |
| `(,,)root;` | Wurzelknoten mit drei unbenannten Unterknoten. |
| `(B,([24.25]D,C)[30]A)root;` | Hierarchie mit einigen Parametern. |

**Tabelle A.1.:** Die syntaktischen Möglichkeiten des Newick-Dateiformats.

[1] Siehe http://evolution.genetics.washington.edu/phylip/newicktree.html.

**Listing A.1** Ein Beispiel für eine Newick-formatierte Datei. Ihr Inhalt beschreibt die Hierarchie aus den Abbildungen in 2.1.

```
newick;
((I,(K,J)H,G)D,C,(F,E)B)A;
```

**Algorithmus A.1** Rekursive Methode zum Parsen eines Newick-Dateiinhalts in Linearzeit.

```
function CreateNode(content, i) : Tree
    result ← new Tree
    name ← ""
    done ← false

    while i ≥ 0 do
        c ← content[i]

        if c = ')' then
            while content[i] ≠ '(' do
                i ← i − 1
                child ← CreateNode(content, i)
                child.Parent ← result
            end while
        else if c = ',' ∨ c = '(' then
            done ← true
        else
            name ← c + name
        end if
        i ← i − 1

        if done then
            break
        end if
    end while

    result.Name ← name
    return result
end function
```

## A.2. Kommandozeilen-Parameter

Für Anwender der Software besteht eine logische Dreiteilung, je nachdem, welchen Einsatzzweck die Anwendungsinstanz erfüllen soll. Die verschiedenen „Modi“ der Software werden über die Kommandozeilen-Parameter gesteuert.

- Als erster Parameter wird eine Option übergeben, die den Anwendungsmodus festlegt.
- Die `-debug`- bzw. `-d`-Option setzt die `Debug`-Einstellung für das gesamte Programm.

**Remote** Die Remote-Anwendung, die als Mittler zwischen den Rendering-Clients und den Interaktions-Tablets steht, wird standardmäßig ausgeführt, wenn keine Parameter übergeben werden, sie kann aber auch explizit über die `-remote`- bzw. `-r`-Option angefordert werden.

Das Setzen des Debug-Attributs hat hier lediglich Auswirkungen darauf, welche Standarddaten initialisiert werden, also welcher Datensatz geladen wird und wie die initiale Definition des verteilten Systems aussieht.

Verwendungsbeispiel: `-remote` oder keine Kommandozeilen-Parameter

**Rendering-Client** Ein Rendering-Client, wie er auf den Powerwall-Rechnern zu starten ist, wird durch die Option `-client` bzw. `-c` angezeigt. Dabei muss auch der Endpunkt (des Remote-Rechners) angegeben werden, zu dem sich der Client verbinden soll. Dieser wird an einer beliebigen Stelle in den Kommandozeilen-Parametern mitübergeben und ist nach dem Muster `endpoint={IP-Adresse}:{Port}` bzw. `ep={IP-Adresse}:{Port}` angegeben.

Wird die Anwendung im Debug-Modus ausgeführt, kann ein einzelner Rechner als Simulation eines verteilten Systems verwendet werden. Statt sich bildschirmfüllend am oberen, linken Rand zu platzieren, wird sich das Fenster der Anwendung entsprechend seiner Position im verteilten System und der Größe des zugehörigen Rechteckausschnitts auf dem Bildschirm zeigen. Das ermöglicht das Ausführen mehrerer Clients auf einem Rechner gleichzeitig, sodass alle sichtbar sind und nebeneinander die Anordnung des verteilten Systems widerspiegeln. Außerdem werden durch bestimmte Tastatureingaben auf dem `ClientForm` eigene, nicht im Netzwerk synchronisierte Transformationen ermöglicht.

Verwendungsbeispiel: `-client endpoint=127.0.0.1:22100 -debug`

**Tablet** Auf den Tablets wird beim Start der Anwendung die `-device`- bzw. `-i`-Option verwendet. Wie bei den Rendering-Clients muss dabei der Endpunkt angegeben werden, zu dem sich das Tablet verbinden soll. Auch hier wird die Syntax `endpoint={IP-Adresse}:{Port}` bzw. `ep={IP-Adresse}:{Port}` verwendet.

Im Debug-Modus wird das `DeviceForm` nicht bildschirmfüllend, sondern in einer dem verteilten System entsprechenden Größe angezeigt.

Verwendungsbeispiel: `-device endpoint=169.220.194.122:22200`

# Literaturverzeichnis


[BBM+en] F. Beck, M. Burch, T. Munz, L. Di Silvestro, D. Weiskopf. Generalized Pythagoras Trees for Visualizing Hierarchies. In *IVAPP '14*. 2014 (noch nicht erschienen). (Zitiert auf den Seiten 12 und 22)

[BDL05] M. Balzer, O. Deussen, C. Lewerentz. Voronoi Treemaps for the Visualization of Software Metrics. In *Proceedings of the 2005 ACM symposium on Software visualization*, S. 165–172. ACM, 2005. (Zitiert auf den Seiten 18 und 20)

[BH94] B. B. Bederson, J. D. Hollan. Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, S. 17–26. ACM, 1994. (Zitiert auf Seite 32)

[BN05] R. Ball, C. North. Effects of Tiled High-Hesolution Display on Basic Visualization and Navigation Tasks. In *CHI'05 extended abstracts on Human factors in computing systems*, S. 1196–1199. ACM, 2005. (Zitiert auf Seite 27)

[BP90] K.-F. Böhringer, F. N. Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, S. 43–51. ACM, New York, NY, USA, 1990. doi: 10.1145/97243.97250. URL http://doi.acm.org/10.1145/97243.97250. (Zitiert auf Seite 53)

[BRW10] M. Burch, M. Raschke, D. Weiskopf. Indented Pixel Tree Plots. In *Advances in Visual Computing*, S. 338–349. Springer, 2010. (Zitiert auf Seite 18)

[BSW11] M. Burch, H. Schmauder, D. Weiskopf. Indented Pixel Tree Browser for Exploring Huge Hierarchies. In *Advances in Visual Computing*, S. 301–312. Springer, 2011. (Zitiert auf Seite 19)

[CGW13] K. Cook, G. Grinstein, M. Whiting. The VAST Challenge: History, Scope, and Outcomes: An Introduction to the Special Issue. *Information Visualization*, 2013. (Zitiert auf Seite 23)

[CRM+06] M. Czerwinski, G. Robertson, B. Meyers, G. Smith, D. Robbins, D. Tan. Large Display Research Overview. In *CHI'06 extended abstracts on Human factors in computing systems*, S. 69–74. ACM, 2006. (Zitiert auf Seite 27)

[CSR+03] M. Czerwinski, G. Smith, T. Regan, B. Meyers, G. Robertson, G. Starkweather. Toward Characterizing the Productivity Benefits of Very Large Displays. In *Interact*, Band 3, S. 9–16. IOS Press, 2003. (Zitiert auf Seite 27)

[DBB10] S. Diehl, F. Beck, M. Burch. Uncovering Strengths and Weaknesses of Radial Visualizations—an Empirical Approach. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):935–942, 2010. doi:http://doi.ieeecomputersociety.org/10.1109/TVCG.2010.209. (Zitiert auf Seite 54)

[EKE08] M. Eissele, M. Kreiser, T. Ertl. Context-Controlled Flow Visualization in Augmented Reality. In *Proceedings of graphics interface 2008*, S. 89–96. Canadian Information Processing Society, 2008. (Zitiert auf Seite 33)

[GFC04] M. Ghoniem, J.-D. Fekete, P. Castagliola. A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations. In *IEEE Symposium on Information Visualization, 2004.*, S. 17–24. IEEE, 2004. (Zitiert auf Seite 18)

[GR12] J. F. Gantz, D. Reinsel. The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. IDC, 2012. (Zitiert auf den Seiten 22 und 23)

[HFM07] N. Henry, J.-D. Fekete, M. J. McGuffin. NodeTrix: A Hybrid Visualization of Social Networks. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1302–1309, 2007. (Zitiert auf Seite 18)

[HMM00] I. Herman, G. Melançon, M. S. Marshall. Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000. (Zitiert auf Seite 88)

[Hol06] D. Holten. Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006. (Zitiert auf Seite 18)

[HSH+04] S. Hotelling, J. A. Strickon, B. Q. Huppi, I. Chaudhri, G. Christie, B. Ording, D. R. Kerr, J. P. Ive. Gestures for Touch Sensitive Input Devices, 2004. US Patent App. 10/903,964. (Zitiert auf Seite 25)

[IC07] P. Isenberg, S. Carpendale. Interactive Tree Comparison for Co-located Collaborative Information Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1232–1239, 2007. (Zitiert auf Seite 26)

[JS91] B. Johnson, B. Shneiderman. Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures. In *IEEE Conference on Visualization, 1991*, S. 284–291. IEEE, 1991. (Zitiert auf Seite 18)

[Kei02] D. A. Keim. Information Visualization and Visual Data Mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002. (Zitiert auf Seite 23)

[KL83] J. B. Kruskal, J. M. Landwehr. Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37(2):162–168, 1983. (Zitiert auf Seite 20)

[KY93] H. Koike, H. Yoshihara. Fractal Approaches for Visualizing Huge Hierarchies. In *Proceedings of the IEEE Symposium on Visual Languages, 1993.*, S. 55–60. IEEE, 1993. (Zitiert auf Seite 21)

[LSST11] A. Lehmann, H. Schumann, O. Staadt, C. Tominski. Physical Navigation to Support Graph Exploration on a Large High-Resolution Display. In *Advances in Visual Computing*, S. 496–507. Springer, 2011. (Zitiert auf Seite 28)

[Man91] B. B. Mandelbrot. *Die fraktale Geometrie der Natur*. Birkhäuser Verlag, 1991. (Zitiert auf Seite 20)

[Nor02] D. A. Norman. *The Design of Everyday Things*. Basic Books, 2002. (Zitiert auf den Seiten 26 und 32)

[PKS+08] P. Peltonen, E. Kurvinen, A. Salovaara, G. Jacucci, T. Ilmonen, J. Evans, A. Oulasvirta, P. Saarikko. "It's Mine, Don't Touch!": interactions at a large multi-touch display in a city centre. In *Proceedings of the SIGCHI conference on human factors in computing systems*, S. 1285–1294. ACM, 2008. (Zitiert auf Seite 26)

[RLN07] R. Rosenholtz, Y. Li, L. Nakano. Measuring Visual Clutter. *Journal of Vision*, 7(2), 2007. (Zitiert auf Seite 18)

[Sam84] H. Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984. (Zitiert auf Seite 81)

[Shn96] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the IEEE Symposium on Visual Languages, 1996.*, S. 336–343. IEEE, 1996. (Zitiert auf Seite 24)

[Stu09] S. Stusak. Collaboration in Information Visualization. *Trends in Information Visualization*, S. 46, 2009. (Zitiert auf Seite 26)

[TC05] J. J. Thomas, K. A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Computer Society Press, 2005. (Zitiert auf Seite 24)

[Tou10] S. Toub. *Patterns of Parallel Programming: Understanding and Applying Parallel Patterns with the .NET Framework 4 and Visual C#*. Microsoft Corporation, 2010. URL http://www.microsoft.com/en-us/download/details.aspx?id=19222. (Zitiert auf Seite 52)

[Wat05] M. Wattenberg. A Note on Space-Filling Visualizations and Space-Filling Curves. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, S. 181–186. IEEE, 2005. (Zitiert auf den Seiten 8, 20 und 21)

[WW99] J. J. van Wijk, H. van de Wetering. Cushion Treemaps: Visualization of Hierarchical Information. In *Information Visualization, 1999.(Info Vis' 99) Proceedings. 1999 IEEE Symposium on*, S. 73–78. IEEE, 1999. (Zitiert auf Seite 18)

[YKSJ07] J. S. Yi, Y. ah Kang, J. T. Stasko, J. A. Jacko. Toward a Deeper Understanding of the Role of Interaction in Information Visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1224–1231, 2007. (Zitiert auf Seite 24)

[YWR02] J. Yang, M. O. Ward, E. A. Rundensteiner. Interring: An interactive tool for visually navigating and manipulating hierarchical structures. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, S. 77–84. IEEE, 2002. (Zitiert auf Seite 20)

Alle URLs wurden zuletzt am 12. 11. 2013 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 Exemplaren überein.

---

Ort, Datum, Unterschrift