TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK
INSTITUT FÜR SOFTWARE- UND MULTIMEDIATECHNIK
PROFESSUR FÜR COMPUTERGRAPHIK UND VISUALISIERUNG
PROF. DR. STEFAN GUMHOLD

Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Medieninformatiker

Open Display Environment Configuration Language

Ronald Großmann
(Geboren am 6. April 1990 in Sebnitz, Mat.-Nr.: 3507432)

Betreuer: Dr. rer. nat. Sebastian Grottel

Dresden, 6. August 2015

Aufgabenstellung

Die Anzahl der Multi-Display-Installationen nimmt in beinahe jeder Umgebung zu: angefangen von Desktopsystemen mit mehreren Monitoren, über Projektionsflächen mit mehreren Projektoren (aka Powerwalls) bis hin zu komplexen VR-Installationen wie Caves. Oft benötigen solche Display-Systeme leistungsstarke, parallel GPU-Cluster zur Bilderzeugung, allein um der notwendigen Pixelfüllrate gerecht werden zu können. Zusätzlich kommen in solchen Anlagen üblicherweise auch Tracking-Systeme zum Einsatz, die den Benutzer, also die physische Welt, mit den dargestellten Szenen, der künstlichen Welt, verbinden; auch im Desktop-Bereich, z.B. durch Windows Kinect oder Leap Motion. Um solche Systeme zu betreiben bedarf es komplexer Software. Diese ist oft im akademischen Umfeld entwickelt. Solche Software muss umfassend konfiguriert werden, einerseits was die verfügbare Rechnerinfrastruktur betrifft (welche Computer sind mit welchen Ausgabegeräten verbunden, welche Computer dienen rein zur entfernten Bilderzeugung und wie sind die Rechner miteinander vernetzt), andererseits auch was die logischen und physikalischen Parameter der Ausgabegeräte betrifft (virtuelle Desktop-Größen und Teile einzelner Beamer, physikalische Anordnung von Display oder Projektoren und Abgleich mit den Raumkoordinaten eines Benutzer-Trackings). Allerdings hat sich für diese Konfigurationen bisher kein Standard entwickelt.

In dieser Arbeit soll ein Vorschlag für so ein standardisiertes Konfigurationsformat auf Basis von XML entwickelt werden. Mittels XSLT soll es möglich sein, aus einer XML-Datei Konfigurationsdateien für unterschiedliche Programme zu erzeugen. Ein graphischer interaktiver Editor soll das Erstellen und Bearbeiten der XML-Dateien, sowohl der strukturellen Eigenschaften (Computer-Cluster-Architektur, inklusive GPUs und Display-Anschlüssen) als auch der 3D physikalischen Eigenschaften (Display-, Projekt-Setup) anschaulich und einfach ermöglichen. Das XML-Format muss sauber durch Namespaces aufgeteilt und erweiterbar sein, was auch durch entsprechende Funktionen im graphischen Editor reflektiert werden muss (z.B. muss es im Editor möglich sein, eigentlich nicht unterstütze Tags editieren und beim Abspeichern erhalten zu können). Die Erstellung von XSLT-Dateien muss durch den Editor NICHT unterstützt werden, ihre Anwendung zum Export der Konfiguration in entsprechende andere Formate jedoch schon.

Folgende Hardware-Installationen müssen unterstützt werden:

1. Desktop-Computer mit mehreren Monitoren (mindestens zwei) die nicht in einer gemeinsamen Ebene stehen.
2. Stereo-Powerwall durch zwei Beamer betrieben an einem Rechner (Powerwall an der Professur

CGV)

3. Großfläche Displaywand mit mehreren Panels (Displaywand an der Professur Multimedia-Technologie)
4. Fünf-Wand-CAVE mit zehn Beamer (im VR-Labor des Lehrstuhls Konstruktionstechnik / CAD)

Hierbei müssen die physikalischen Display- und Projektionsanordnungen unterstützt werden, und zusätzliche Infrastruktur, wie z.B. Computer, GPUs, Tracking-Systeme etc., sollen so weit wie möglich unterstützt werden.

Die Hardwareinstallationen sollen in ihrem physikalischen Raum, in Metern, frei definierbar sein. Ist kein Benutzertracking vorhanden, so muss eine Standardposition für den Benutzer (Blickpunkt) konfigurierbar sein.

Die Konfiguration der Rechnerinfrastruktur muss mindestens die Computer enthalten, die direkt an die Ausgabegeräte angeschlossen sind. Ihre Netzwerkverbindungen untereinander sollten enthalten sein. Eventuelle Compute-Cluster zur Bilderzeugung und ihre Netzwerkverbindungen untereinander, sowie zu den Ausgaberechnern sollten ebenfalls konfigurierbar sein.

Folgende Software muss unterstützt werden, indem ihre Konfigurationsdateien, mindestens aber der entsprechend dieser Arbeit relevante Teil der Konfigurationsdateien, erzeugt werden kann:

a, MegaMol, bzw. mittels einer kleinen Bibliothek jede an der TUD selbst entwickelte Software

b, Paraview

c, Equalizer (optional)

Weitere Software soll nach Absprache mit dem Betreuer ebenfalls unterstützt werden.

Die Bearbeitung erfolgt mit diesen Teilzielen:

- Anforderungsanalyse auf Basis der vorgegebenen Display-Hardware und Konfigurationsspezifikationen der einzusetzenden Software
- Literaturrecherche zur Large-Display- und VR-Software-Middleware und Konfigurationen. Auch zu allgemeinen Arbeiten zur Kalibrieren und Konfiguration solcher Hardware-System
- Spezifikation der XML-basierten Konfiguration
- Umsetzung des geforderten Editor-Prototypens inklusive XSLT-basiertem Export der Konfigurationen

- Evaluierung im Kontext der vorgegebenen Display-Systeme durch Darlegung und Durchführung des kompletten Konfigurationsprozesses anhand der vorgegebene Software
- Optional: Erweiterung des Editors um weitere Funktionalitäten zur semi-automatischen Erzeugung der Konfigurationsdateien
- Optional: Code-Bibliothek zur direkten Nutzung der Konfigurations-Xml-Datei
- Optional: Untersuchung weiterer Display-Konfigurationen (z.B. gekrümmter Projektionen) und weiterer Visualisierungs- und VR-Software

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tag dem Prüfungsausschuss der Fakultät Informatik eingereichte Arbeit zum Thema:

Open Display Environment Configuration Language

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den 6. August 2015

Ronald Großmann

Kurzfassung

Zusammenfassung Text Deutsch

Abstract

abstract text english

Inhaltsverzeichnis

1 Einführung

Tiled Displays beziehungsweise Large High Resolution Displays (LHRD) im Sinne dieser Arbeit, beschreibt Installationen, bei denen mehrere Displays zu einem größeren Display zusammengesetzt werden. Im Falle von Monitoren, besteht das LHRD aus mehreren einzelnen Monitoren die ein gemeinsames Display darstellen. Dieses muss nicht in einer Ebene liegen, sondern kann auch eine Krümmung aufweisen. Im Falle von Projektoren besteht das LHRD aus mehreren Projektionsflächen für die Projektoren. Dabei können auch mehrere Projektoren auf ein und die selbe Fläche projizieren um beispielsweise eine stereoskopische Projektion zur erzeugen, beziehungsweise muss auch bei diesen LHRD das Display nicht zwangsläufig eine flache Ebene sein.
LHRD bieten den Vorteil eine große Pixeldichte auf einer großen Fläche darzustellen. Dadurch lassen sich Details lokal noch hochauflösend darstellen, während gleichzeitig das gesamt Bild und Kontext dargestellt werden kann. Diese Eigenschaften machen LHRD besonders attraktiv für Anwendungsgebiete, in denen viele Informationen gleichzeitig dargestellt werden oder der Kontext einer Information wichtig ist. Ein solches Anwendungsgebiet findet sich in der Wissenschaftlichen Visualisierung. Hier lassen sich sich dank der Größe des Displays sehr viele Daten gleichzeitig darstellen. Auch für Kommando und Kontrollaufgaben finden LHRD Anwendung, beispielsweise im Militär oder in der Luftfahrt. Besonders konstruierte LHRD, wie beispielsweise die CAVE [CNSD93], bieten dem Nutzer eine besonders immersive Umgebung. Diese kann in der Industrie zur Unterstützung beim Design benutzt werden, beispielsweise bei der Entwicklung in der Automobilindustrie.
Der Aufbau und Betrieb eines LHRD ist Aufwändig, weshalb diese Technologie noch nicht so weit verbreitet ist. Die Komponenten für ein LHRD müssen meist individuell ausgewählt und konfiguriert werden, da es einige Probleme beim Betrieb eines LHRD gibt, auf die im folgenden Eingegangen wird.

1.1 Problemfelder

Durch die hohe Anzahl der Pixel in LHRD ist es nötig mehrere GPUs für das Rendern zu verwenden um in annehmbarer Zeit ein Bilder zu erzeugen. Gerade bei interaktiven Visualisierungen ist ein möglichst kurze Reaktionszeit und eine möglichst flüssiger Ablauf der Visualisierung erforderlich. Für gewöhnlich

wird dafür ein Cluster aus mehreren Rechnern zur Bilderzeugung verwendet, wodurch sich weitere organisatorische Probleme ergeben. Zum einem kann man das Berechnen der Visualisierung auf einige, dafür spezialisierte Rechner verlagern. Die Ergebnisse, in diesem Fall die Bilddaten, müssen dann entsprechend auf Rechner verteilt werden, die mit dem LHRD verbunden sind. In diesem Fall handelt es sich also um das Problem des Display Data Streamings. Der zweite Ansatz ist, dass entsprechende Rendering auf den Rechnern auszuführen, die an das LHRD angeschlossen sind. Für dieses verteilte Rendern stellt sich das Problem der Verteilung der Daten, die für das Rendering nötig sind. Prinzipiell muss jeder Rechner nur die Daten Rendern, die den entsprechenden Teil des Bildes darstellen, dessen Teil im LHRD der Rechner bedient. Zusätzlich gibt es bei diesem Ansatz noch das Problem der Synchronisation, dass also sichergestellt sein muss, dass alle Rechner des verteilten Renderings im selben Takt arbeiten, also synchron gleichzeitig die Frames für das LHRD ausgeben.

Ein weiteres Problemfeld für LHRD ist die Kalibrierung und Einrichtung an sich. Gerade bei Projektor basierenden LHRDs muss sehr genau kalibriert werden, um ein optimales Bild zu erhalten. Da die Geometrie der Projektion aus technischen Gründen nicht perfekt rechtwinkelig sein kann, ist es schwer eine möglichst verzerrungsfreie Abbildung zu erhalten. Wenn man mehrere Projektoren benutzt um ein LHRD zu erzeugen ist vor allem die Helligkeit der Projektion ein Problem. Diese ist je nach Projektor ungleich über die Projektionsfläche verteilt und gerade am Rand kommt es zu Problemen, wenn man einen möglichsten unauffälligen Übergang von einer in die anliegende Projektion erhalten möchte. Oftmals werden bei solchen Systemen Spiegel benutzt, um die Projektionsstrahlen um zu lenken und somit räumlich kompaktere Installationen zu erhalten. Durch die zusätzliche Umlenkung durch Spiegel jedoch, ergibt sich natürlich eine weitere Fehler- und Problemquelle. Bei Monitor basierenden LHRD ist es etwas einfacher die Geometrie beziehungsweise Form des LHRD zu erhalten, da es einfacher ist die Monitore in einem regelmäßigen Gitter anzuordnen. Das größte Problem bei solchen LHRDs jedoch sind die Rahmen, die Zwangsläufig an jedem Monitor vorhanden sind. Auch wenn es inzwischen Monitor Modelle mit sehr schmalen Rahmen gibt, verhindern diese Rahmen nach wie vor den Eindruck, dass es sich um eine große, ununterbrochene Bildfläche handelt. Auch bei Monitoren gibt es Probleme mit der Helligkeitsverteilung, vor allem Rand. Diese ist auch bei Monitoren vom gleichen Modell von Exemplar zu Exemplar unterschiedlich.

Ein drittes Problemfeld für LHRDs betrifft die Anwendungen, die auf dieses Systemen betrieben werden sollen. Da viele LHRD Installationen individuell gebaut und betrieben werden, ergibt sich eine sehr Heterogene Landschaft aller LHRDs. Dies hat zur Folge, dass Anwendungen oft individuell für einzelne LHRD Installationen entwickelt werden beziehungsweise bestehende Anwendungen umfangreich angepasst werden müssen. Dies schränkt die Portabilität dieser Anwendungen ein, da es nur wenige Standards oder portable Frameworks für die Entwicklung gibt.

2 Verwandte Arbeiten

TODO

Allgemeines zu LHRD, vielleicht auch n bisschen was Richtung MCI

A Survey of Large High Resolution Display Technologies, Techniques and Application 2006 [NSS+06]

Designing LHRD Workspaces 2012 [EBZ+12]

2.1 Betrieb von Large High Resolution Displays

Mit Chromium wird in [HHN+02] ein Framework für interaktives Rendering auf Cluster vorgestellt. Dabei wird ein Stream aus Grafik API Befehlen (zum Beispiel OpenGL) durch die einzelnen Rechner im Cluster manipuliert. Dabei kann ein Rechner einen oder mehrere Streams empfangen, verarbeiten und an einen oder mehrere Rechner im Cluster weiterleiten. Während der Verarbeitung können verschiedene Filter und Transformationen angewendet werden, wodurch eine komplexe parallele Grafikarchitektur auch mit moderaten GPUs erreicht werden kann. Somit kann mit Chromium jeder auf Cluster basierender paralleler Render Algorithmus implementiert werden, da die Streamverarbeitung in Chromium sehr allgemein gehalten ist.

In [JJR+] wird mit SAGE (Scalable Adaptive Graphics Environment) eine Schnittstelle zwischen Applikation und LHRD vorgestellt. Mit SAGE ist es möglich mehrere Bilderzeugungsquellen, beispielsweise Simulationen oder Visualisierungen, zusammen zu führen und auf einem LHRD aus zu geben. Die Anzeige der einzelnen Quellen auf dem LHRD lassen sich beliebig skalieren und positionieren. Damit mit die Quellen mit SAGE verarbeitet werden können, müssen diese für das SAGE Framework angepasst werden. Dies besteht darin, dass bestimmte Befehle für die SAGE Application Interface Library hinzugefügt werden. Diese regelt die Kommunikation und den Transport des Outputs der Applikation an SAGE.

2.2 Kalibrierung von Large High Resolution Displays

PixelFlex [YGH+01] ist ein Projektor basierendes LHRD, welches aus bis zu 8 Projektoren besteht. Diese projizieren über individuell neig- und schwenkbare Spiegel auf eine Projektionsfläche. Die Spiegel können sich zusammen mit den Fokussier- und Zoomfunktionen der Projektoren von einem Rechner aus Steuern lassen. Zusätzlich gibt es eine Kamera zur Kalibrierung. Ziel des PixelFlex Systems ist es, automatisch eine Abbildung von Projektorkoordinaten in Weltkoordinaten zu finden, so dass eine verzerrungsfreie und gleichmäßig helle Projektion von allen Projektoren auf die Projektionsfläche stattfindet. Dies geschieht über die Kamera, wobei zunächst die Abbildung von Weltkoordinaten in Kamerakoordinaten ermittelt wird. Über diese wird dann die Abbildung von Weltkoordinaten in Projektorkoordinaten ermittelt für jeden einzelnen Projektor. Mit dieser Abbildung lassen sich nun die Ausgaben an die Projektoren so konfigurieren, dass ein LHRD entsteht. Zur Anpassung der Helligkeit an den Übergängen, wo sich mehrere Projektionen überlappen, werden Alphamasken benutzt, um eine gleichmäßige Helligkeit über das LHRD zu erreichen.

2.3 Middleware für Anwendungen auf Large High Resolution Displays

In [RFC+03] wird ein Ansatz vorgestellt, wie verschiedene Heterogene Applikationen zu einer VR-Anwendung verknüpft werden können. Dies geschieht, in dem man die einzelnen Applikationen zu Modulen weg kapselt, die nur noch über entsprechend spezifizierte input und output ports kommunizieren. Die Kommunikation erfolgt über Messages in denen die Daten zur Weiterverarbeitung an das entsprechende Modul gesendet wird. Dieser Ansatz einer Message Oriented Middleware bietet den Vorteil, dass die einzelnen Module, in denen die verschiedenen Applikationen weg gekapselt sind, austauschbar sind, solange die input und ouput ports gleich spezifiziert sind. Der Message Manager der Middleware steuert dann den Event basierenden Ablauf einer Anwendung.

Einen ähnlichen Ansatz verfolgt FlowVR [AGL+04] welches eine Middleware darstellt die vor allem für verteilte VR-Anwendungen auf Cluster konzipiert ist. Auch hier sollen Heterogene Einzelkomponenten zu einer gesamt VR-Anwendung verknüpft werden, ohne den bestehenden Code der Einzelkomponenten zu stark zu verändern. Das Hauptaugenmerk liegt dabei auf der Kommunikation und Synchronisation innerhalb des Clusters. Ähnlich zu [RFC+03] haben die einzelnen Module input und output ports, die mit anderen Modulen verbunden sind. Die Kommunikation läuft auch hier über Messages. Dabei gibt es spezielle Messsages zur Synchronisation, womit beispielsweise eine gleichmäßige Bildrate für die Aus-

gabe auf ein LHRD erzielt wird, wenn dieses LHRD von mehreren Rechnern des Cluster betrieben wird.

Im Falle der CAVE handelt es sich um ein besonderes LHRD. Da der Nutzer sich in einem Raum befindet, der von mehreren Seiten projiziert wird, bekommt der Nutzer eine besonders immersive Erfahrung. Die Entwicklung von VR-Anwendungen für solche Installation ist durch die Besonderheit nicht einfach. In [BJH+01] wird mit dem VR Juggler eine virtuelle Platform zu Entwicklung und Ausführung von VR-Anwendungen in CAVE Installationen vorgestellt. Das Ziel hierbei war, den Entwicklern der VR-Anwendungen einfache Software Werkzeuge zu geben und gleichzeitig die Komplexität der Hardware Konfiguration weg zu kapseln. Damit sollen sich Entwickler voll auf die eigentliche VR-Anwendung konzentrieren können und eine einfache Basis zum Entwickeln, Testen und Ausführen haben. Dabei agiert VR-Juggler ähnlich der Virtuellen Maschine bei JAVA, wie eine zusätzliche Schnittstelle zwischen Anwendung und Hardware. Eine Anwendung, die einmal für den VR-Juggler entwickelt wurde, läuft also auf allen Installationen, die VR-Juggler unterstützt.

2.4 Einordnung meiner Arbeit

Das Problemfeld welches diese Arbeit abdeckt, ist Problemfeld drei. Es stellt eine Middleware für Anwendungen auf LHRD dar. Dabei verfolgt die Arbeit einen Ansatz der in den von mir recherchierten Arbeiten noch nicht behandelt wurde. Diese Arbeit konzentriert sich auf die Konfiguration und Aufbau des LHRD an sich. Die Infrastruktur zum Betrieb steht eher im Hintergrund, wird jedoch auch mit aufgegriffen. Diese Arbeit stellt eine Middleware zur Verfügung, mit der bestehende Anwendungen auf Basis einer Beschreibung des LHRD, individuell konfiguriert werden können.

3 Anforderungsanalyse

Im Zuge der Anforderungsanalyse, habe ich die in der Aufgabenstellung genannte Hard- und Software analysiert. Dabei habe ich mich im Falle der Hardware vor allem auf die Anordnung und Zusammensetzung der Displayelemente konzentriert, welche das LHRD bilden. Dabei ging es einerseits um die rein physischen Maße und Gegebenheiten, aber auch die Infrastruktur dahinter. Wie die Displayelemente angesteuert und abgebildet werden ist ein wichtiger Aspekt.
In der Softwareanalyse habe ich mich auf die Konfiguration der Anwendungen konzentriert. Insbesondere die Konfigurierbarkeit der Bildausgabe war für meine Arbeit wichtig. Dies betraf die genaue Positionierung der Ausgabefenster als auch die Eigenschaften des virtuellen Viewports. Ich habe versucht jede der zu unterstützenden Softwarepakete auf den verschiedenen Hardwareinstallationen auszuführen.

3.1 Hardwareanalyse

Im folgenden werden die verschiedenen Hardware Installationen beschrieben, welche ich in meiner Arbeit analysiert und getestet habe. Für drei der vier Installationen gibt es eine schematische Abbildung (Abb. 3.1, Abb. 3.2, Abb. 3.3) welche auf der linken Seite die Infrastruktur darstellt und auf der rechten Seite den Aufbau des entsprechenden LHRD.

Desktop Systeme

Eine der Installationen ist ein Desktop System mit mindestens zwei Displays, die nicht zwangsläufig in einer Ebene liegen müssen. Da es viele Möglichkeiten gibt ein solches Desktop System einzurichten, habe ich mich dafür entschieden ein mir Verfügbares System als Referenzsystem zu benutzen. Grundsätzlich betrachte ich ein solches Desktop System als LHRD, wenn der Desktop auf die Displays erweitert wird. Dabei hat jedes Pixel der Displays eine eindeutige Desktop Pixel Koordinate. Das Referenz System, was mir zur Verfügung stand, bestand aus einem Notebook mit einem LCD Display. Dieses hatte eine Auflösung von 1280 Pixel horizontal und 800 Pixel vertikal, sowie eine physikalische Größe von 331 mm horizontal und 207 mm vertikal. An dieses Notebook ist ein LED Monitor angeschlossen und

steht rechts neben dem Notebook, so dass die Oberkanten beider Displays auf gleicher Höhe sind. Der Monitor hat eine Auflösung von 1920 Pixel horizontal und 1080 Pixel Vertikal, sowie eine physikalische Größe von 464 mm horizontal und 260 mm vertikal. Da beide Displays einen Rahmen haben, können sie nicht lückenlos nebeneinander stehen.

Als Betriebssystem lag Windows 7 vor, welches so eingerichtet wurde, dass der LCD Bildschirm des Notebooks der Hauptbildschirm war und somit in dessen oberen linken Ecke der Ursprung des Pixelkoordinatensystem lag. Der Desktop wurde dann nach rechts auf den LED Monitor erweitert. Daraus folgte, dass die linke obere Ecke des LED Monitors die Pixelkoordinaten (1280;0) hatte.

CGV Stereowall

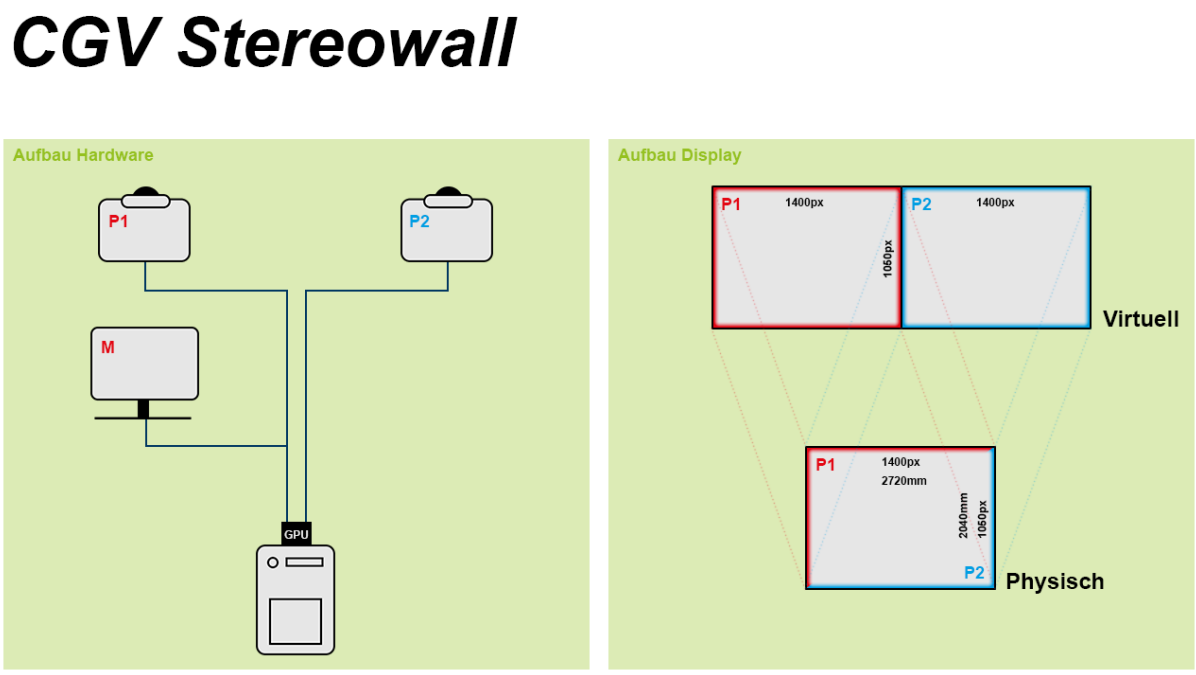

Abbildung 3.1: Schematische Darstellung der Stereopowerwall des Computergraphik Lehrstuhls

Bei der Stereopowerwall des Computergraphik Lehrstuhls handelt es sich um ein Projektor basierendes LHRD. Dabei sind zwei Projektoren an eine GPU eines Rechners angeschlossen (Abb. 3.1 links). Das Bildsignal an einen der beiden Projektoren wird noch einmal gedoppelt um es an einem Monitor anzuzeigen (Abb. 3.1 P1 und M). Dieser Monitor dient zur vereinfachten Bedienung des Rechners. Die Besonderheit dieser Installation besteht darin, dass beide Projektoren auf ein und die Selbe Projektionsfläche strahlen, und die Projektionen möglichst deckungsgleich aufeinander liegen (Abb. 3.1 rechts). Dazu wurden die beiden Projektoren direkt übereinander montiert und entsprechend eingestellt. Um Platz zu sparen werden die Strahlen über einen Spiegel umgelenkt, bevor sie von Hinten auf die Projek-

tionsfläche treffen. Vor beiden Projektoren sind Polarisationsfilter so montiert, dass sich die Polarisation der Strahlen beider Projektoren um 90° unterscheidet. Die Projektionsfläche besteht aus einem Stoff, der diese Polarisation erhält und somit zusammen mit einer entsprechenden Polarisation Stereobrille ein getrenntes Bild für das linke und rechte Auge darstellt. Die Projektionsfläche hat eine Größe von 2720 mm horizontal und 2040 mm vertikal. Die beiden Projektoren haben eine Auflösung von 1400 Pixel horizontal und 1050 Pixel Vertikal.

Der Rechner an den die Projektoren und der Monitor angeschlossen waren, hatte unter anderem Windows 7 als Betriebssystem, welches ich genutzt habe. Dabei war der Desktop über beide Projektoren erweitert, die gesamt größe des Desktops betrug also 2800 Pixel horizontal und 1050 Pixel vertikal. Dabei war der Projektor, der für das linke Auge projizierte, auch der linke Teil des Desktops. Dies war auch der Teil, der auf dem zusätzlichen Monitor angezeigt wurde. Dementsprechend wurde der rechte Teil des Desktop von dem Projektor dargestellt, welche für das rechte Auge projizierte.

MT Powerwall

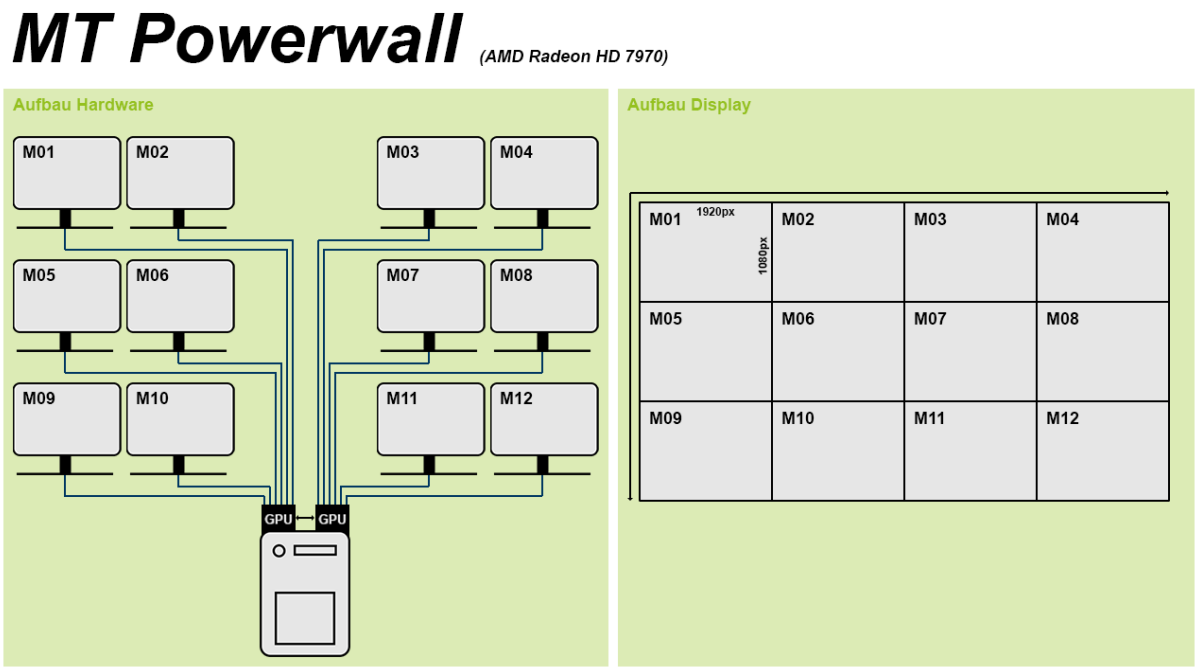

Abbildung 3.2: Schematische Darstellung der Powerwall des Lehrstuhl für Multimedia-Technologie

Bei der Powerwall des Lehrstuhl für Multimedia-Technologie handelt es sich um ein LHRD bestehend aus zwölf Monitoren die in einem vier mal drei Gitter angeordnet sind (Abb. 3.2 rechts). Jeder dieser Monitore hat eine Auflösung von 1920 Pixel horizontal und 1080 Pixel vertikal, sowie eine physikalische Größe von 1210 mm horizontal und 680 mm vertikal. Das gesamte LHRD hat also eine Auflösung von

7680 Pixel horizontal und 3240 Pixel vertikal, sowie eine metrische Größe von 4840 mm horizontal und 2040 mm vertikal. Die Rahmen um die Monitore sind mit 2,5 mm äußerst schmal.

All diese Monitore werden von einem Rechner mit zwei GPUs betrieben. An jeder GPU sind sechs Monitore angeschlossen (Abb. 3.2 links). Auch auf diesem Rechner stand unter anderem Windows 7 als Betriebssystem zur Verfügung, welches ich benutzt habe. Der Desktop war über alle zwölf Monitore erweitert mit dem Monitor in der linken oberen Ecke als Hauptmonitor. Dadurch lag der Ursprung des Pixelkoordinatensystem für den gesamten Desktop auch in der linken oberen Ecke.

Trotz der zwei leistungsstarken GPUs, hatte das System unter Windows manchmal Probleme mit der großen Pixelzahl. Gerade bei Testen der Softwarepakete kam es des öfteren zu Abstürzen oder Bildfehler für einige Monitore, welche das Testen erschwerte.

CAVE

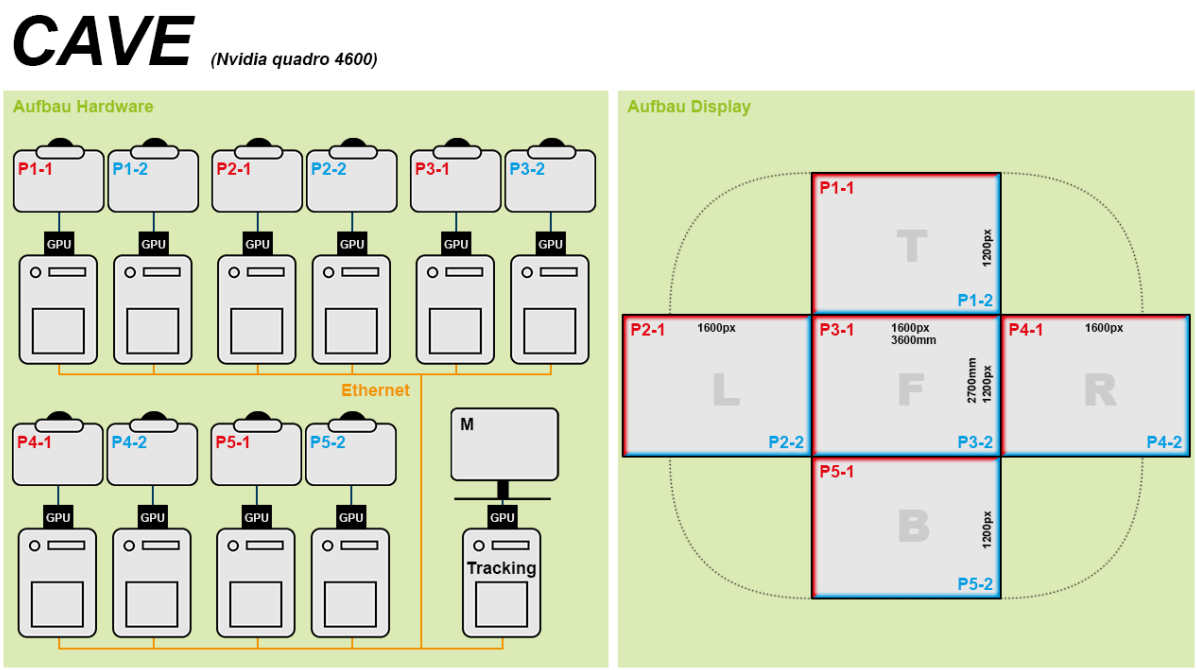

Abbildung 3.3: Schematische Darstellung der Fünf-Seiten-CAVE des Lehrstuhl Konstruktionstechnik/CAD

Bei der CAVE des Lehrstuhl Konstruktionstechnik/CAD handelt es sich um ein Projektor basierendes LHRD, welches Anfang der 1990er Jahre von [CNSD93] vorgestellt wurde. In diesem Fall handelt es sich um eine Fünf-Seiten-CAVE, bei der nur die Seite nicht für die Projektion benutzt wird, durch die man die CAVE betritt. Die restlichen fünf Seiten werden von hinten jeweils von zwei Projektoren deckungsgleich bestrahlt. Jede der Seiten hat eine Größe von 3600 mm horizontal und 2700 mm vertikal (Abb. 3.3

rechts). Da die Seiten nicht quadratisch sondern rechteckig mit einem Verhältnis von vier zu drei sind, kommt es zu einer Diskrepanz zwischen den Seiten links und rechts (Abb. 3.3 rechts L und R) und den Seiten oben und unten (Abb. 3.3 rechts T und B). Dadurch reichen die Seiten oben und unten nicht ganz bis zum Eingang der CAVE. Jeder der zehn Projektoren hat eine Auflösung von 1600 Pixel horizontal und 1200 Pixel vertikal. Im Gegensatz zur Stereopowerwall des Computergraphik Lehrstuhls, werden hier die Bilder für linkes und rechts Auge durch einen Interferenzfilter getrennt. Dabei werden mehrere Wellenlängen der Strahlen jeweils gefiltert, sodass mit einer entsprechenden Brille die Bilder für die Augen wieder getrennt wahrgenommen werden. Auch hier werden mittels Spiegel die Stahlen umgelenkt um Platz zu sparen.
Jeder der zehn Projektoren ist mit jeweils einem separaten Rechner verbunden. Diese zehn Rechner und ein zusätzlicher Rechner sind durch ein Ethernet Netzwerk verbunden (Abb. 3.3 links). Der zusätzliche Rechner hat einen eigenen Monitor außerhalb der CAVE. Dieser spezielle Rechner dient zum Steuern der anderen 10 Rechner via Remote Control. Zusätzlich verarbeitet dieser Rechner die Tracking Daten und sendet sie an die 10 Rechner mit den Projektoren. Auf allen Rechnern ist Windows XP als Betriebssystem installiert. Aufgrund der speziellen Treibersoftware für die GPUs, konnte noch nicht auf ein aktuelleres Betriebssystem umgestiegen werden.

3.2 Softwareanalyse

In der Softwareanalyse habe ich mich auf die Konfigurierbarkeit der einzelnen Anwendungen konzentriert. Die betrifft hauptsächlich die Konfiguration der Bildausgabe. Dabei geht es einerseits um die Positionierung der Ausgabe, also wie kann ich das Fenster der Ausgabe den Anforderungen entsprechend positionieren. Auch die Anpassung des Viewports, zum Beispiel bezüglich des FOV oder der Blickrichtung, sollte konfigurierbar sein. Bei Stereoprojektionen ist es zudem wichtig, die einzelnen Ausgaben für linkes und rechtes Auge zu konfigurieren.
Zusätzlich habe ich zwei der drei hier analysierten Anwendungen auf möglichst allen passenden Hardware Installationen ausgeführt und getestet.

MegaMol

Bei MegaMol handelt es sich um eine Middleware zur Visualisierung von Punkt basierenden Molekular Datensätzen [GKM+15].
MegaMol-Projekte bestehen aus einer Pipeline von verschiedenen Filtern, die auf den Datensatz angewandt werden. Dabei können Projekte auch in mehreren Instanzen gestartet werden, welche dann je-

weils auch ein eigenes Ausgabefenster besitzen. Für das Rendern auf mehrere verteilte Displays bietet MegaMol einen konfigurierbaren TileView-Filter an. In diesem lassen sich auch die Stereoeigenschaften konfigrueren.
Die Konfiguration erfolgt an zwei Stellen. Zum einem in der Datei *megamol.cfg* wo für verschiedene Fenster die Größe und Position festgelegt werden kann. Der Name des Fensters ist dabei der Name der Instanz, den man beim Starten des Projektes angibt, gefolgt von "-window" (Abb. TODO). Die Konfiguration der TileView erfolgt dann über eine Parameter-Datei, die beim Starten des Projektes mit übergeben wird. Darin werden die Eigenschaften der einzelnen Filter für jede Instanz gespeichert (Abb. TODO). Für TileView sind das die Stereoeigenschaften (Abb. TODO a), Die Position und Größe des Tiles innerhalb des Viewports (Abb. TODO b) sowie die Gesamtgröße des Viewports (Abb. TODO c). Mit diesen Konfigurationen lassen sich beliebig Tiles positionieren und anpassen.
Mit diesen Konfigurationen habe ich MegaMol erfolgreich auf dem Desktop System und der Powerwall des Lehrstuhl für Multimedia-Technologie getestet. Auf der Stereopowerwall des Computergraphik Lehrstuhls habe ich MegaMol auch erfolgreich mit der Stereoprojektion getestet. Da MegaMol noch keinen Filter besitzt, mit dem man mehrere Viewports erzeugen kann, die verschiedene Blickrichtungen haben, war MegaMol ungeeignet für einen Test in der CAVE des Lehrstuhl Konstruktionstechnik/CAD.

ParaView

ParaView [AGL05] ist eine open-source Plattform auf Basis von VTK [SLM04] für die Analyse und Visualisierung von Daten. Die von mir benutzte Version von ParaView ist die vor kompilierte Desktop-Version 4.3 für Windows. Diese besitzt ein auf QT basierenden Interface.
Leider lässt sich die Ausgabe in diesem Interface nicht konfigurieren. ParaView bietet jedoch die Möglichkeit für Remote Rendering. Dafür wird ein ParaView Render Server mit Hilfe von MPI gestartet mit dem sich dann der ParaView Client verbinden kann. Dies funktioniert auch wenn der Server auf dem gleichen Rechner wie der Client läuft, mittels localhost. Beim Starten des Servers kann man diesen mittels Startparameter konfigurieren (Abb. TODO). Dabei gibt es die Möglichkeit, die Ausgabe auf dem Rechner des Servers zu erzeugen. Diese Ausgabe lässt sich dann mittels einer Konfigurationsdatei genauer einstellen. Die Konfigurationsdatei wird auch als Startparameter beim Starten des Servers übergeben und ist XML Formatiert. Diese PVX-Dateien (Abb. TODO) konfigurieren die Ausgabe für die Render Server. *Machine* beschreibt den Rechner auf dem der Server läuft. Dabei ist das Attribute *Name* der Name des Rechners. Im Falle eines Clusters, lassen sich also auch die Ausgaben auf mehreren Rechnern konfigurieren. Die weiteren Attribute konfigurieren die Ausgabe. *Geometry* konfiguriert die Größe und Position des Ausgabefenster. *LowerLeft, LowerRight* und *UpperRight* sind drei Raumvektoren, die

die entsprechenden Ecken des Displays im Raum beschreiben. Um mehrere Ausgabefenster auf dem gleichen Rechner zu erzeugen, genügt ein weiteres *Machine* Element mit dem gleichen *Name* Attribut. Für jedes Ausgabefenster muss jedoch beim Starten des Servers wie MPI mindestens ein Prozess zur Verfügung stehen (Abb. TODO).
Mit diesen ParaView Renderserver habe ich Paraview erfolgreich auf dem Desktop System und der Powerwall des Lehrstuhl für Multimedia-Technologie getestet. Leider lässt sich nie das ganze LHRD zur Ausgabe benutzen, da die Interaktion mit Paraview weiterhin nur über den Client funktioniert. Somit muss dieser immer mit angezeigt werden. Da ParaView keine Konfiguration zur Verfügung stellt, mit der man Stereoprojektion in separaten Fenster ausgeben kann, habe ich auf einen Test auf der Stereopowerwall des Computergraphik Lehrstuhls verzichtet.
Durch die Möglichkeit des Remote Rendering auf einem Cluster sollte ParaView auch sehr gut in der CAVE des Lehrstuhl Konstruktionstechnik/CAD laufen. Die durch die PVX Konfigurationsdateien gegebene Möglichkeit mehrere Viewports mit verschiedenen Blickrichtungen zu konfigurieren, ermöglicht eine immersive Visualisierung in der CAVE. Die Kommunikation zwischen den Servern im Cluster wird mittel MPI ermöglicht. Leider konnte ich ParaView nicht erfolgreich in der CAVE des Lehrstuhl Konstruktionstechnik/CAD ausführen und testen. Mir ist es nicht gelungen MPI so auszuführen und zu testen, dass es die ParaView Render Server auf dem Cluster ausführt. Deshalb sind die Konfigurationen von ParaView bezüglich der CAVE des Lehrstuhl Konstruktionstechnik/CAD nur theoretischer Natur.

Equalizer

Equalizer [EMP09] ist eine Middleware für die Entwicklung und Ausführung von OpenGL Anwendungen. Mit Equalizer können diese Anwendungen auf unterschiedlich skalierten Systemem, bezüglich Renderleistung oder Displaygröße, ausgeführt werden ohne diese zu verändern. Die Anpassung an die verschiedenen System erfolgt über die Konfiguration.
Für die Konfiguration für Equalizer Anwendungen gibt es eqc-Textdateien. In diesen sind die Einstellungen für die Server gespeichert. Darunter auch das Layout für die Ausgabe. Darin können mehrere View definiert und konfiguriert werden (Abb. TODO). Auch die verschiedenen Stereoeigenschaften lassen sich im Layout konfigurieren.

4 Konzeption

Das Ziel hinter Open Display Enviroment Configuration Language (OpenDECL) ist es, eine adäquate aber dennoch knappe Beschreibung für LHRD zu entwickeln. Diese Beschreibung sollte all nötigen Informationen enthalten, die für die Konfiguration der Anwendung, die auf dem beschriebenen LHRD ausgeführt werden soll, benötigt werden. Aus der Softwareanalyse ergab sich, dass für eine Konfiguration im wesentlichen zwei Aspekte eine Rolle spielen. Zum einem der Aufbau des LHRD aus einzelnen Displays und der Abbildung der jeweiligen Pixelkoordinaten darauf. Und zum anderem die Infrastruktur die das LHRD betreibt.

Display Beschreibung

TODO hier kommt noch ein tolles Bild ähnlich zu dem mit den Vektoren aus meiner Zwischenpräsentation.
Zur Beschreibung des Displays ist zum einem die physische Größe und Anordnung der Display Elemente an sich nötig, als auch die Abbildung der Pixelkoordinaten auf diese Displays.
Unter der Annahme, dass es sich bei den Displays um Rechteckige Flächen handelt, lässt sich die Breite und Höhe einfach angeben. Für die Positionierung der Displays im Raum ist ein Referenzkoordinatensystem notwendig. Wenn man dieses definiert hat, lassen sich alle Displays positionieren in dem Beispielsweise ein Vektor die Position der linken oberen Ecke des Displays beschreibt. Damit hat man das Display positioniert aber noch nicht orientiert. Dafür sind zwei weitere Vektoren notwendig. Mit drei Vektoren, die Beispielsweise die drei der vier Ecken des Displays beschreiben, lässt sich ein Display eindeutig im Raum platzieren. Damit lässt sich die Anordnung der einzelnen Displays in einem LHRD beschreiben.
Die Frage wo man das Referenzkoordinatensystem ansetzt, also wohin man den Ursprung legt, kann auf zwei verschiedenen Wegen geklärt werden. Wenn man kein anderes Koordinatensystem als Anhaltspunkt hat, zum Beispiel durch ein Trackingsystem oder ähnliches, kann man den Benutzer als Ursprung des Referenzkoordinatensystems benutzen. Dazu sucht man sich die bevorzugte Position des Benutzers vor dem LHRD und beschreibt dann ausgehend vom Kopf des Benutzers als Koordinaten Ursprung die Positionierung und Orientierung der einzelnen Displays.

Im Fall eines bereits vorhanden Koordinatensystems, beispielsweise durch ein Trackingsystem, können die einzelnen Displays in diesem System positioniert und orientiert werden. In diesem Fall wäre jedoch angebracht auch den Benutzer zu positionieren, da sich dessen Position nicht mehr durch den Koordinatenursprung, wie in dem ersten Fall, ergibt. Der Benutzer kann mit zwei Vektoren beschrieben werden, die die Position und Orientierung seines Kopfes beschreiben.
Besonders bei LHRD, die einen erweiterten Desktop abbilden, ist es wichtig auch die Zuordnung der Pixelkoordinaten zu den einzelnen Displays zu beschreiben. Diese Beschreibung ist nötig, da die Ausgabefenster der Anwendungen für das LHRD nur in diesem Pixelraum positioniert werden können. Somit wird eine Abbildung von Pixelkoordinaten auf die einzelnen Displays nötig. Die naheliegende Lösung dafür sind drei weitere Vektoren zur Beschreibung der einzelnen Displays. Diese Vektoren sind keine Raumvektoren, sondern Pixelvektoren und beschreiben die entsprechenden Ecken des Displays in Pixelkoordinaten.
Mit diesen beiden Beschreibungen der Ecken der einzelnen Displays, also räumliche Koordinaten und Pixelkoordinaten, lässt sich das LHRD eindeutig beschreiben und die Anwendungen dafür konfigurieren.

Infrastruktur Beschreibung

Die Beschreibung der Infrastruktur ist besonders dann wichtig, wenn mehrere Rechner das LHRD betreiben. Dann ist es wichtig zu wissen, welcher Teil des LHRD von welchem Rechner betrieben wird. Dies lässt sich beschreiben, indem man für jeden Rechner angibt, mit welchen Displays er verbunden ist. Zusammen mit der Beschreibung der Displays, lässt sich so einfach beschreiben, welcher Rechner für welchen Teil des LHRD zuständig ist.
Bezüglich der Verbindung der Rechner untereinander, gehe ich von einem Datennetzwerk aus, was von allen Rechnern geteilt wird, ähnlich zu Ethernet. Da es in einem solchen Netzwerken keine gerichteten Verbindungen gibt, genügt es zu beschreiben, in welchen Netzwerken sich die Rechner befinden und welche Adressen sie darin haben.

5 OpenDECL Spezifikation

TODO fancy Infografik mit Überblick und Referenzpfeilen
Die aus meiner Konzeption hervorgehende benötigten Beschreibungen für LHRD, habe ich in die Spezifikation der OpenDECL Dokumente umgesetzt. Diese Spezifikation liegt als XML Schema Document vor (Anhang?) und besteht im grob aus zwei Teilen. Zum einem die Beschreibung der Infrastruktur mit den *node* und *network* Elementen und der Beschreibung des LHRD in den *display-setup* Elementen. Ein gültiges OpenDECL Dokument benötigt dabei beliebig viele aber mindestens ein *node* Element, beliebig viele *network* Elemente und beliebig viele aber mindestens ein *display-setup* Element. Abbildung TODO zeigt einen Überblick, wie die einzelnen Elemente verschachtelt sind und auf welche Elemente die Referenzen verweisen. Im folgenden werden die einzelnen Elemente und deren Attribute der Spezifikation näher beschrieben und welche Aspekte der Konzeption damit umgesetzt werden.

5.1 node

Das *node* Element beschreibt einen Rechner. Würde das LHRD also von einem Cluster betrieben, gäbe es in der OpenDECL Beschreibung dafür auch mehrere *node* Elemente. Das *id* Attribut beschreibt einen eindeutigen Bezeichner für den Rechner und ist ein Pflichtattribut. Dies kann Beispielsweise der Name des Rechners im Netzwerk sein. Das *purpose* Attribut ist optional und kann die Funktion des Rechners beschreiben. Solche Funktionen können zum Beispiel “Rendern“, “Verarbeitung“ oder “Berechnung“ sein.
Ein *node* Element und damit ein Rechner, kann beliebig viele *graphics-device* und *network-device* Elemente enthalten.

5.1.1 graphics-device

Das *graphics-device* Element beschreibt eine Grafikkarte in einem Rechner. Das *id* Attribute dient als eindeutiger Bezeichner für diese Grafikkarte und ist ein Pflichtattribut. Das *gpu-count*, *vram* und *model-name* Attribut sind optionale Attribute um die Grafikkarte bezüglich Anzahl der GPUs, Speicher und Model Namen näher zu beschreiben.

Ein *graphics-device* Element enthält beliebig viele aber mindestens ein *port* Element.

port

Das *port* Element beschreibt einen Anschluss an der Grafikkarte, an den ein Display angeschlossen werden kann. Das *id* Attribut dient als eindeutiger Bezeichner dieses Anschlusses und dient als Referenz für die Zuordnung von Displays an diesen Anschluss. Das *type* und *slot* sind optionale Attribute um diesen Anschluss näher zu beschreiben, beispielsweise ob es sich um einen HDMI oder VGA Anschluss handelt oder welcher Anschluss bei Grafikkarten mit mehreren Anschlüssen beschrieben wird.

5.1.2 network-device

Was soll * darstellen

Welche Atrribute beschreiben was + optional vs required

5.2 network

Was soll * darstellen

Welche Atrribute beschreiben was + optional vs required

5.3 display-setup

Was soll * darstellen

Welche Atrribute beschreiben was + optional vs required

vector

Was soll * darstellen

Welche Atrribute beschreiben was + optional vs required

5.3.1 user

Was soll * darstellen

Welche Atrribute beschreiben was + optional vs required

5.3.2 display

Was soll * darstellen

Welche Atrribute beschreiben was + optional vs required

physical

Was soll * darstellen

Welche Atrribute beschreiben was + optional vs required

virtual

Was soll * darstellen

Welche Atrribute beschreiben was + optional vs required

6 XSLT Konfigurationsgenerierung

was ist zu beachten

wie viel logik kann man in XSLT reinstecken, wo liegen die Grenzen?

6.1 Beispiel

TODO

einfaches Beispiel CGV Stereowall mit Megamol

komplexeres Beispiel MT Powerwall mit Paraview

hypothetisches Beispiel für CAVE mit Paraview

7 Editor

7.1 Funktionen

Was kann Editor

Wo liegen Grenzen des Editors

kleines Manual

7.2 Umsetzung

Grober Aufbau

Datenhaltung im Hintergrund

Zugriff via Interface

8 Evaluation

Wie gut Erfüllt Ergebnis die Aufgabenstellung

Wie gut lassen sich die XSLTs erstellen

Wie ist die Qualität der Konfigurationen die hinten raus kommen

9 Diskussion

Vergleich Ergebnis mit Vorgaben, Ideen, Konzeption

10 Ausblick

Mögliche Erweiterungen

Ausgelassene oder vereinfachte Aspekte, die noch Ausbaufähig sind

weitere Optionen für Editor

Abbildung 10.1: beschriftung

Literaturverzeichnis

[AGL+04] ALLARD, Jérémie ; GOURANTON, Valérie ; LECOINTRE, Loïck ; LIMET, Sébastien ; RAFFIN, Bruno ; ROBERT, Sophie: *FlowVR: A Middleware for Large Scale Virtual Reality Applications*. 2004. – 497–505 S

[AGL05] AHRENS, James ; GEVECI, Berk ; LAW, Charles: 36 ParaView: An End-User Tool for Large-Data Visualization. In: *The Visualization Handbook* (2005), S. 717

[BJH+01] BIERBAUM, Allen ; JUST, Christopher ; HARTLING, Patrick ; MEINERT, Kevin ; BAKER, Albert ; CRUZ-NEIRA, Carolina: VR Juggler: A Virtual Platform for Virtual Reality Application Development. In: *Virtual Reality*, 2001, S. 89–96

[CNSD93] CRUZ-NEIRA, Carolina ; SANDIN, Daniel J. ; DEFANTI, Thomas A.: Surround-screen projection-based virtual reality: the design and implementation of the CAVE. In: *Annual Conference on Computer Graphics*, 1993, S. 135–142

[EBZ+12] ENDERT, Alex ; BRADEL, Lauren ; ZEITZ, Jessica ; ANDREWS, Christopher ; NORTH, Chris: Designing large high-resolution display workspaces. (2012), S. 58–65

[EMP09] EILEMANN, Stefan ; MAKHINYA, Maxim ; PAJAROLA, Renato: Equalizer: A Scalable Parallel Rendering Framework. In: *IEEE Transactions on Visualization and Computer Graphics* 15 (2009), May/June, Nr. 3, S. 436–452

[GKM+15] GROTTEL, S. ; KRONE, M. ; MULLER, C. ; REINA, G. ; ERTL, T.: MegaMol – A Prototyping Framework for Particle-based Visualization. In: *Visualization and Computer Graphics, IEEE Transactions on* 21 (2015), Feb, Nr. 2, S. 201–214. – ISSN 1077–2626

[HHN+02] HUMPHREYS, Greg ; HOUSTON, Mike ; NG, Ren ; FRANK, Randall J. ; AHERN, Sean ; KIRCHNER, Peter D. ; KLOSOWSKI, James T.: Chromium: a stream-processing framework for interactive rendering on clusters. In: *ACM Transactions on Graphics* 21 (2002), S. 693–702

[JJR+] JEONG, Byungil ; JAGODIC, Ratko ; RENAMBOT, Luc ; SINGH, Rajvikram ; JOHNSON, Andrew ; LEIGH, Jason: Scalable Graphics Architecture for High Resolution Displays. In:

Presented at IEEE Information Visualization Workshop 2005, S. 10–24

[NSS+06] NI, Tao ; SCHMIDT, Greg S. ; STAADT, Oliver G. ; LIVINGSTON, Mark A. ; BALL, Robert ; MAY, Richard A.: A Survey of Large High-Resolution Display Technologies, Techniques, and Applications. In: *Virtual Reality, IEEE Annual International Symposium*, 2006, S. 223–236

[RFC+03] RODRIGUES, Fábio ; FERRAZ, Rodrigo ; CABRAL, Márcio ; TEUBL, Fernando ; BELLOC, Olavo ; KONDO, Marcia ; ZUFFO, Marcelo ; LOPES, Roseli: Coupling Virtual Reality Open Source Software Using Message Oriented Middleware. (2003)

[SLM04] SCHROEDER, Will J. ; LORENSEN, Bill ; MARTIN, Ken: *The visualization toolkit*. Kitware, 2004

[YGH+01] YANG, Ruigang ; GOTZ, David ; HENSLEY, Justin ; TOWLES, Herman ; BROWN, Michael S.: PixelFlex: a reconfigurable multi-projector display system. In: *IEEE Visualization*, 2001, S. 167–174

Danksagung

Danke Dr. rer. nat. Sebastian Grottel für die Betreuung

Danke Dipl.-Inf. Ulrich von Zadow für Betreuung an der Powerwall des Lehrstuhl für Multimedia-Technologie

Danke Dr.-Ing. Wolfgang Steger für die Betreung an der CAVE des Lehrstuhl Konstruktionstechnik/CAD

Erklärungen zum Urheberrecht

Hier soll jeder Autor die von ihm eingeholten Zustimmungen der Copyright-Besitzer angeben bzw. die in Web Press Rooms angegebenen generellen Konditionen seiner Text- und Bildübernahmen zitieren.