Westfälische Wilhelms-Universität Münster

Bachelorarbeit

im Studiengang Geoinformatik

Visualisierung von Community-basierten Lärmmessungen in einer Android-basierten Augmented Reality Umgebung

Erstgutachter: Dr. Theodor Foerster

Zweitgutachter: Prof. Dr. Christian Kray

Institut für Geoinformatik

Eingereicht von

Holger Hopmann

Matrikelnummer: 358417

E-Mail: h.hopmann@wwu.de

Münster, 21. September 2011

Inhaltsverzeichnis

1 Einleitung 1
1.1 Zielsetzung 1
1.2 Strukturierung 1
2 Grundlagen 2
2.1 Augmented Reality 2
2.1.1 Abgrenzung 2
2.1.2 Struktur 3
2.2 Android Plattform 4
2.3 OpenGL for Embedded Systems 5
3 Sensorbasiertes Tracking 6
3.1 Definition der Koordinatensysteme 8
3.1.1 Sensorkoordinaten 8
3.1.2 Gerätekoordinaten 9
3.1.3 Weltkoordinatensystem 9
3.2 Transformation zwischen Welt- und Gerätekoordinaten 10
3.2.1 Bestimmung der Koordinatensystemtransformation 10
3.2.1.1 Beschleunigungssensor 11
3.2.1.2 Magnetfeldsensor 11
3.2.2 Transformation 12
3.2.3 Positionsbestimmung 13
3.2.4 Überführung in Gerätekoordinaten 14
4 Lärminterpolation 17
4.1 Datengrundlage 18
4.2 Rasterisierung 20
4.3 Clusteranalyse 20
4.4 Bestimmung des Lärmpegels 25
4.5 Positionsbestimmung 26
4.6 Modellierung der Lärmausbreitung 27
4.7 Zusammenfügen der Interpolation 29
4.8 Implementierungsentscheidungen 31
4.8.1 Datentyp 31
4.8.2 Datenstruktur 32
4.8.3 Speicherressourcen 35
4.8.3.1 Parameterabschätzung 36
4.8.3.2 Gesamtabschätzung 39
4.8.4 Bewertung 41

5 Lärmvisualisierung in einer Augmented Reality Umgebung 42
5.1 Darstellung der Realen Umgebung 42
5.2 Dreidimensionale Überlagerung der Schallimmissionsdaten 44
5.2.1 Extrinsische Parameter 45
5.2.2 Intrinsische Parameter 46
5.2.3 Darstellung 47
5.3 Zweidimensionale Überlagerung 47
5.3.1 Bestimmung der sichtbaren Interpolationsfläche 48
5.3.2 Zuordnung der Sichtstreifen 50
5.3.3 Darstellung 52
6 Referenzanwendung „NoiseAR“ 53
6.1 Aufbau der Anwendung 53
6.2 Programmarchitektur 54
6.2.1 Anzeigehierarchie 55
6.2.2 Datenquellen 56
6.2.3 Lärminterpolation 57
6.2.4 Lärmanalyse im Sichtfeld 57
6.2.5 Asynchronität 58
7 Zusammenfassung 59
Literaturverzeichnis VII

Abbildungsverzeichnis

Abbildung 1: Realitäts-Virtualitätskontinuum 2
Abbildung 2: Koordinatensysteme und Beziehungen innerhalb eines typischen visuellen Augmented Reality Systems 3
Abbildung 3: Android Architektur 4
Abbildung 4: Verteilung der OpenGL ES Versionsunterstützung aktiver Android Geräte basierend auf Daten aus dem Android Market vom 26.8. bis 2.9.2011 6
Abbildung 5: Sensorkoordinatensystem gemäß `SensorEvent` API 8
Abbildung 6: Darstellung des Weltkoordinatensystems 9
Abbildung 7: Darstellung des Weltkoordinatensystems bezüglich der Sensorkoordinaten 11
Abbildung 8: Gegenüberstellung der „column-major“ und „row-major“ Linearisierungsweisen an einem Beispiel 13
Abbildung 9: Schematische Darstellung des Interpolationsprozesses 18
Abbildung 10: Aktivitätsdiagramme k-Means Algorithmus (links) und Erweiterung (rechts) mit Aufteilung in Ablaufphasen 21
Abbildung 11: Laufzeitverhalten des trivialen Clusteringalgorithmus 23

Abbildung 12: Vergleich der Optimierungen des Clusteringalgorithmus' durch Sortierung nach Y bzw. nach X und Y Koordinate anhand der Laufzeiten 24

Abbildung 13: Überführung der Messpositionen anhand ihrer Genauigkeitsangaben in das Zielraster..... 27

Abbildung 14: Beispielhafte Berechnung der Lärmabschwächung in der Nachbarschaft einer Lärmkartierung. Annahme für Messungen/Geräuschwahrnehmung in je 1 m Entfernung 28

Abbildung 15: Acht Wege Symmetrie in der Lärmabschwächungsmodellierung. 29

Abbildung 16: Zusammenfügen der Lärmabschwächungs- und Positionsbestimmung 30

Abbildung 17: Ermittlung der Nachbarschaftsverhältnisse der bei der Positionsbestimmung identifizierten Zellen..... 31

Abbildung 18: Ressourcenbedarf des Lärmabschwächungsmodells in Abhängigkeit des Ausbreitungsradius, bei einer Interpolationsauflösung von 1 m pro Pixel..... 37

Abbildung 19: Gesamtabschätzung des Speicherressourcenbedarfs der verschiedenen Implementierungsweisen am Beispiel einer sphärischen Mercator Projektion als Rasterisierung 39

Abbildung 20: Aktivitätsdiagramm zur Verwendung einer `SurfaceView` zur Kameravorschau..... 43

Abbildung 21: Transformationssequenz der OpenGL Fixed Function Grafikpipeline..... 45

Abbildung 22: Darstellung des Grundkonzeptes der zweidimensionalen Überlagerung 48

Abbildung 23: Ablauf zur Findung der Schnittpunkte mit der Interpolationsebene ausgehend von Displayeckpunkten 50

Abbildung 24: Darstellung des entwickelten Algorithmus zur Immissionsmittelung der Sichtstreifen. Zeigt Zustand während Erzeugung einer Kantenliste..... 51

Abbildung 25: Augmented Reality Sicht 53

Abbildung 26: Kartenbasierte Lärmauswertung am Beispiel der "NoiseDroid" Datenquelle 54

Abbildung 27: Darstellung der zentralen Informationsleiste zur Anzeige von Status- und Fortschrittsmeldungen 55

Abbildung 28: Sequenzdiagramm zur typischen Interaktion bei der Anforderung einer Interpolation, im Beispiel ausgehend von einer Positionsänderung..... 57

Abbildung 29: Sequenzdiagramm zur Interpolation 58

Tabellenverzeichnis

Tabelle 1: Tracking Technologien für virtuelle Umgebungen mit Genauigkeitsangaben 7
Tabelle 2: Beispielhafte Darstellung des Gravitationsvektors definiert durch die Accelerometerdaten bei aufrechter Position des Gerätes in unterschiedlichen Konstellationen 14
Tabelle 3: Gegenüberstellung der Konstanten zur Beschreibung der Displayausrichtungen. 15
Tabelle 4: Anpassung der Sensorkoordinatenbasis bei Displayrotation 16
Tabelle 5: Konfigurationsmöglichkeiten für die Instanziierung eines `android.graphics.Bitmap` Objektes über eine statische `createBitmap` Methodenüberladung 33
Tabelle 6: Verwendungsszenarien der hervorgehobenen Datenstrukturen innerhalb des Interpolationsverfahrens 35
Tabelle 7: Abschätzung des Speicherbedarfs verschiedener Implementierungsweisen des Interpolationsprozesses 40
Tabelle 8: Zusammenfassung der Implementationsbewertung 41
Tabelle 9: Veranschaulichung der Verfahrensschritte zur Sichtstreifenzuordnung 52

Codelistings

Listing 1: Signatur der `getSystemService` Methode zum Erhalt einer Systemdienstinstanz 10
Listing 2: Signatur der `getRotationMatrix` Methode zur Bestimmung der Transformationsmatrix vom Sensorkoodinatensystem in das Weltkoordinatensystem 12
Listing 3: Signatur der `remapCoordinateSystem` Methode der `SensorManager` API..... 15
Listing 4: Bestimmung der aktuellen Displayrotation in einer `Activity` Klasse 16
Listing 5: Verwendung eines Java `byte` Typen ohne Vorzeichen 32
Listing 6: Exemplarische Verwendung einer Farbmatrix zur Transformation eines Alphakanals in einen Rot-Grün Farbverlauf 34
Listing 7: Signaturen der Abfragemethoden der `Camera.Parameters` Klasse der Android Hardware API zum Auslesen intrinsischer Kameraparameter 44
Listing 8 Setzen der extrinsischen Parameter im OpenGL ES Kontext zur Überlagerung mit Kamerabild 46
Listing 9: Signatur der gluPerspective Methode aus der `opengl.GLU` Klasse der Android API 46
Listing 10: Setzen der intrinsischen Parameter im OpenGL ES Kontext zur Überlagerung mit Kamerabild 47

1 Einleitung

Umgebungslärm ist eine der umfassendsten Umweltbelastungen, da der Mensch ihr über seinem gesamten Tagesablauf hinweg ungeschützt ausgesetzt ist. Der Einfluss geht dabei über eine einfache Belästigung des Menschen hinaus und umfasst beispielsweise das Auslösen von Schlafstörungen, die Förderung kognitiver Beeinträchtigung sowie die Hervorbringung von Herz-Kreislauf-Erkrankungen, weshalb Lärm inzwischen auch als entscheidende Krankheitsbelastung gewertet wird (vgl. World Health Organization 2011).

Mit dem Aufkommen des nutzerzentrierten „Web 2.0“ und mobiler Computer, sowie der Zukunft des ubiquitären Computings stehen gleichsam immer mehr Möglichkeiten zur Verfügung, diesen Umwelteinfluss direkt an seinem Immissionsort durch den Menschen einzuschätzen und diese Daten anderen Nutzern zur Verfügung zu stellen.

1.1 Zielsetzung

Mit dieser Bachelorarbeit soll eine Prozesskette entwickelt werden, die es auf der mobilen Android Plattform ermöglicht, von Nutzern zur Verfügung gestellte Lautstärkekartierungen in einer Augmented Reality Umgebung in Echtzeit darzustellen. Durch visuelle Überlagerung der real sichtbaren Eindrücke mit einer Lärminterpolation soll eine zielgerichtete nutzerzentrierte Analyse der Lärmbelastung der jeweiligen Umgebung ermöglicht werden, im Gegensatz zu einer klassischen statischen Lärmkarte.

Hauptaufgabe ist dabei die explizite Bezugnahme sowohl auf den speziellen Charakter nutzerbasierter Datenquellen, als auch auf die Eigenschaften ubiquitärer Phänomene wie des Schalls unter Beachtung der innerhalb der Android Plattform beschränkten Ressourcen.

In einer Referenzanwendung „NoiseAR“ sollen die erarbeiteten Konzepte praktisch in einer mobilen Applikation umgesetzt werden.

1.2 Strukturierung

Aufbauend auf einigen Grundlagenbeschreibungen in Kapitel zwei zur verwendeten Technologie und Theorie beschreibt Kapitel drei das Vorgehen zur Ausrichtung der realen Eindrücke an virtuelle Informationen ausgehend von auf der Android Plattform verfügbaren Sensoren in unbekannter Umgebung. Das vierte Kapitel widmet sich der Interpolation Community-basierter Lärmmessungen und erörtert insbesondere zu treffende Implementierungsentscheidungen. Darauffolgend werden in Kapitel fünf die Ergebnisse der Ausrichtung und Interpolation verknüpft und zwei Visualisierungsweisen der Schallimmission vorgestellt.

2 Grundlagen

2.1 Augmented Reality

In einem Augmented Reality System, zu Deutsch etwa „Erweiterte Realität“, wird die real existierende Welt um computergenerierte, virtuelle Eindrücke ergänzt. Bei diesen virtuellen Eindrücken handelt es sich bei aktuellen Applikationen zumeist um visuelle Objekte, welche ein Abbild der Realität erweitern, d. h. Bilder oder Videos realer Objekte werden um computergenerierte Informationen und Objekte vervollständigt, oder es werden Informationen unmittelbar in dem Sichtfeld des Nutzers positioniert, beispielweise bei der Darstellung von Routeninformationen auf einer Fahrzeugwindschutzscheibe. Die Definition der Erweiterten Realität beschränkt sich hierbei jedoch nicht ausschließlich auf visuelle Reize, sondern umfasst alle die Realitätswahrnehmung beeinflussenden Medien.

2.1.1 Abgrenzung

Abgegrenzt wurde das Feld der Erweiterten Realität durch das sogenannte „Virtualitätskontinuum“ (Milgram und Kishino 1994), welches gänzlich reale mit virtuellen Umgebungen verbindet. Hier steht sich die reale durch physikalische Gesetze beschränkte Welt einer synthetischen, ausschließlich aus virtuellen Objekten bestehenden Umgebung gegenüber. Dazwischenliegende Systeme, die Elemente beider Kontinuumsgrenzen stufenlos kombinieren, lassen sich in diesem Schema der sogenannten „Mixed Reality“, also der Vermischten Realität, zuordnen. Die Erweiterte Realität liegt in dem Konzept des Virtualitätskontinuums vor, wenn ein solches System vorwiegend durch eine reale Umgebung charakterisiert wird. Falls es überwiegend auf Seiten des anderen Extremums liegt, wird es als Erweiterte Virtualität bezeichnet.

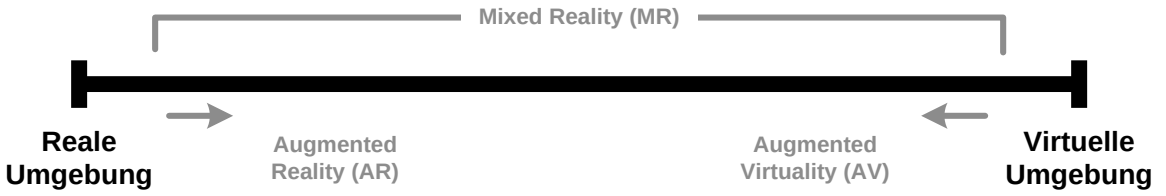


Abbildung 1: Realitäts-Virtualitätskontinuum
(Nach Milgram und Kishino 1994, 3)

Die nötigen Eigenschaften die ein Augmented Reality System erfüllen muss, werden dabei wie folgt zusammengefasst (Azuma, Baillot, et al. 2001, 34):

- Reale und virtuelle Objekte werden in realer Umgebung kombiniert
- Reale und virtuelle Objekte sind einander ausgerichtet
- Das System arbeitet interaktiv und in Echtzeit

In der Literatur wird meist auf diese Definition verwiesen (vgl. Mehler-Bicher, Reiß und Steiger 2011, 10), jedoch zusätzlich vorausgesetzt, dass die Verschmelzung der virtuellen Umgebung mit der Realität im dreidimensionalen Raum stattfinden muss. Dies rührt daher, dass in der ursprünglichen Zusammenfassung nach (Azuma 1997) zweidimensionale Überlagerungen noch explizit ausgeschlossen wurden.

2.1.2 Struktur

Augmented Reality Systeme basieren typischerweise auf verschiedenen Referenzsystemdefinitionen und Beziehungen, die eine reale Umgebung, darin platzierte Objekte sowie den Nutzer unabhängig voneinander beschreiben (Abbildung 2). Ziel innerhalb der Struktur ist die fortwährende Erfassung und Ausrichtung dieser Rahmen, sodass zwischen ihnen transformiert werden kann. Zur visuellen Überlagerung der virtuellen und realen Umgebung kommen dabei typischerweise ein Objektkoordinatensystem O, ein Weltkoordinatensystem W sowie ein Kamerakoordinatensystem C zur Anwendung (vgl. Vallino 1998). Im Sinne der Kamerageometrie wird zwischen diesem Weltbezugsrahmen und dem kameraeigenen System mittels der sogenannten extrinsischen Kameraparameter transformiert. Sie spiegeln die Ausrichtung (Rotationsmatrix R) und Position (Translation t) des Kamerazentrums bezüglich der Weltkoordinaten wider. Die Projektion des Kamerabildes auf die Bildebene U wird durch die intrinsischen Kameraparameter approximiert, die unter anderem die Brennweite, das Bildzentrum und Bildausmaße umfassen und durch eine Matrix A dargestellt werden.

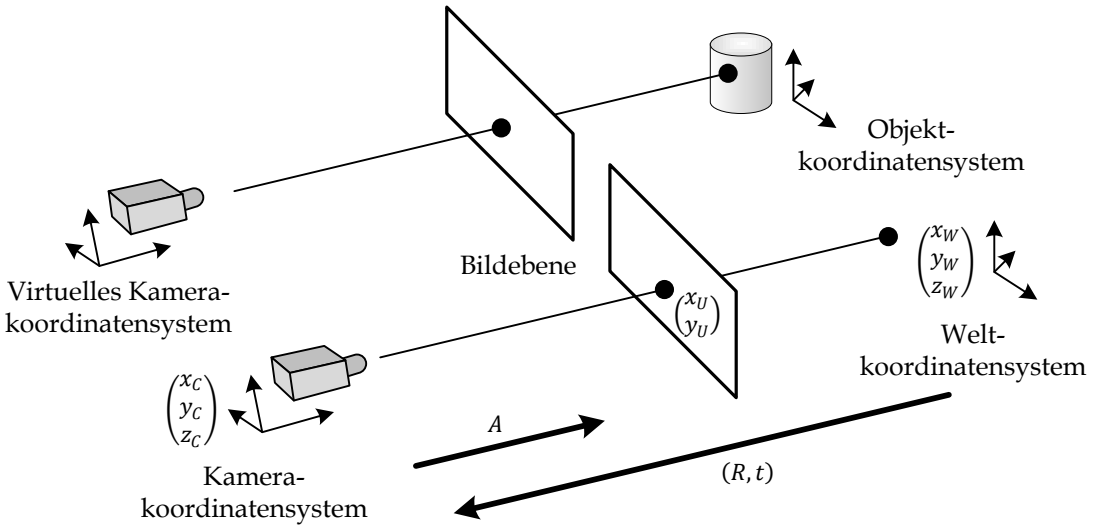


Abbildung 2: Koordinatensysteme und Beziehungen innerhalb eines typischen visuellen Augmented Reality Systems

(verändert nach Vallino 1998, 2)

2.2 Android Plattform

Die Android Plattform ist eine quelloffene Software Infrastruktur für mobile Geräte, die seit 2007 durch die Open Handset Alliance (OHA), einem Zusammenschluss von insgesamt 84 Konzernen vor allem aus dem Bereich der mobilen Technologien, Softwareanbietern und Werbefirmen, entwickelt wird (vgl. Open Handset Allinance 2011). Sie umfasst dabei neben eines auf dem Linux 2.6 Kernel basierenden Betriebssystems eine umfassende Middleware sowie grundlegende Applikationen für mobile Geräte (vgl. Google Inc. 2011f).

Die Gesamtarchitektur (Abbildung 3) beruht dabei auf dem Linux Kernel, der die Aufgaben eines klassischen Betriebssystems übernimmt, etwa die Hardware- und Ressourcenverwaltung. Basierend auf einer Standard C Implementierung werden Funktionen der nativen Systembibliotheken in einer ausführlichen Middleware, dem Android Framework, über die „Dalvik Virtual Machine“ zur Verfügung gestellt. Dies ist eine speziell angepasste ressourcenoptimierte Java Laufzeitumgebung, auf der die Anwendungen fußen (vgl. Komatineni, MacLean und Hashimi 2011, 7).

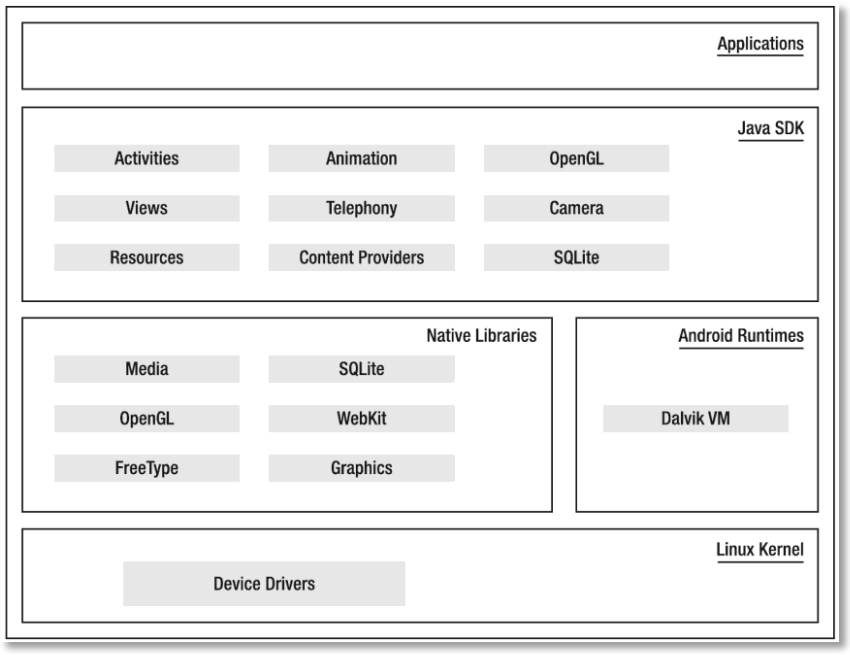


Abbildung 3: Android Architektur
(Komatineni, MacLean und Hashimi 2011, 7)

Der Grundsatz bei der Entwicklung und das Alleinstellungsmerkmal dieser Plattform ist laut Open Handset Alliance „Offenheit“, die sich zuvorderst in der quelloffenen Bereitstellung des gesamten Android Programmcodes zeigt. Innerhalb der Architektur zeigt sie sich dadurch, dass einer Anwendung alle Funktionen des jeweiligen mobilen Gerätes zur Ver-

wendung und zur Verfügung stehen, sowohl bezüglich der Hardware als auch des von allen Komponenten gleichsam genutzten Anwendungsrahmens.

Aktuell ist die Android Version 2.3.4 für Smartphones, sowie 3.2 für Tablets. Für diese Arbeit wird das Android System in ihrer Version 2.2 zugrunde gelegt, da mit ihr derzeit über 80% der androidfähigen Geräte adressiert werden können (Google Inc. 2011c) und sie gleichzeitig als Mindestmaß für einige hier verwendete Funktionen vorausgesetzt werden muss.

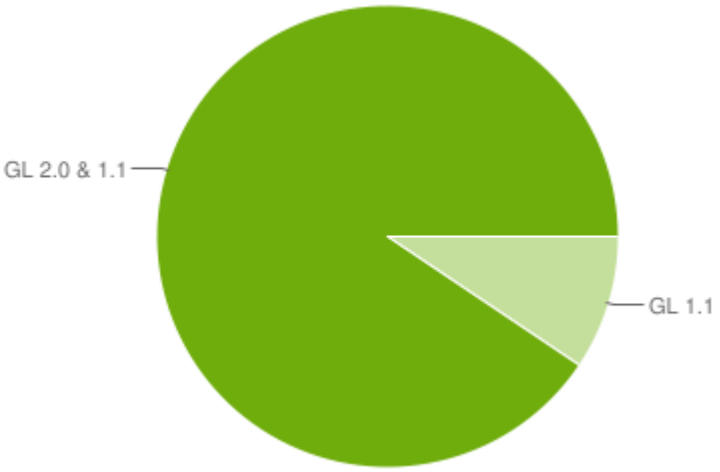
2.3 OpenGL for Embedded Systems

OpenGL for Embedded Systems (OpenGL ES) ist eine offene Softwareschnittstelle zur Interaktion mit der Grafikhardware, speziell auf die Bedürfnisse mobiler und eingebetteter Systeme zugeschnitten. Sie basiert dabei auf der OpenGL Spezifikation, die 1992 erstmals durch Silicon Graphics International veröffentlicht wurde. Seit 2003 wird dieser Schnittstellenstandard durch das Khronos Group Konsortium geleitet und weiterentwickelt (Khronos Group 2010). Für mobile und eingebettete Systeme wurde die Grafikbibliothek verkleinert, indem Redundante Funktionen entfernt wurden (The Khronos Group Inc. 2008, 2).

Aktuell sind die OpenGL ES Schnittstellen in der Version 1.1 bzw. 2.0. Die Versionsreihe 1.x fußt dabei auf OpenGL 1.3 und verwendet somit die sogenannte „fixed function pipeline“, in der eine feste Abfolge an Operationen zur Erstellung „hochwertiger Grafiken“ (The Khronos Group Inc. 2008) ausgehend von Objektdefinitionen vorgegeben ist. OpenGL ES 2.x basiert hingegen auf dem modernen Prinzip voll programmierbarer Shader.

Die OpenGL ES Grafikschnittstellen sind plattformunabhängig und werden von einer Vielzahl verschiedener Systeme und Anwendungen implementiert. So existiert auch seit der ersten Version der Android Plattform hier eine OpenGL 1.x Implementierung als einzige Grafikbibliothek zur Erzeugung von dreidimensionalen Bildern. Seit Android 2.2 wird auch OpenGL 2.0 unterstützt. Die aktuelle Verteilung der Versionsunterstützung ist Abbildung 4 zu entnehmen.

Da für diese Arbeit nur eine simple Darstellung unter Nutzung der OpenGL API notwendig ist, wird die Spezifikation in der Version 1.1 verwendet. Sie basiert auf der OpenGL 1.5 Spezifikation und führt unter anderem Funktionen zur Abfrage der aktuellen Transformationsmatrizen ein, die für die folgende Arbeit beansprucht werden.



OpenGL ES Version	Verbreitung
1.1	9,4 %
2.0	90,6 %

Abbildung 4: Verteilung der OpenGL ES Versionsunterstützung aktiver Android Geräte basierend auf Daten aus dem Android Market vom 26.8. bis 2.9.2011

(Google Inc. 2011b)

3 Sensorbasiertes Tracking

Für eine Augmented Reality Anwendung ist es nach (Azuma, Baillot, et al. 2001) obligatorisch, die virtuellen Zusatzinformationen mit der realen Welt bzw. dessen Abbildung in direkten Bezug zu stellen. Hierfür ist es erforderlich, zunächst eine Erfassung der realen Welt durchzuführen, die es ermöglicht die Position und Ausrichtung des Nutzers bzw. des zur Verschmelzung der realen und virtuellen Umgebung genutzten Gerätes innerhalb des virtuellen Weltkoordinatensystems (siehe Abbildung 2) akkurat und in Echtzeit darzustellen. Diese Erfassung wird dabei als das sogenannte Tracking bezeichnet. Es existieren dabei verschiedenste Methoden um eine solche Zuordnung in sechs Freiheitsgraden zu ermöglichen. (Rolland, Baillot und Goon 2001) geben eine Übersicht über die dafür entwickelten Technologien, zu denen unter anderem die in Tabelle 1 aufgeführten zählen.

Insbesondere bei Systemen, die auf visuellen Methoden basieren, ist es notwendig, dass die Umgebung, in der die Erweiterung der Realität verfügbar sein soll, speziell vorbereitet sein muss. Sie muss beispielweise mit bekannten Markierungssymbolen ausgestatten werden, die von einem optischen Tracker mittels Methoden aus der Mustererkennung identifiziert werden. Trotz hoher Genauigkeit und Bestimmung aller Freiheitsgrade können Methoden wie Ultraschallentfernungsmessung auf der Android Plattform nicht direkt umgesetzt werden, da sie spezielle Referenzstationen und Empfangshardware benötigen. Andere Systeme, wie die auf Trägheit beruhenden Methoden zur Ausrichtungsbestimmung, sind ohne weitere Vorkehrungen direkt auf das Android System übertragbar.

Tabelle 1: Tracking Technologien für virtuelle Umgebungen mit Genauigkeitsangaben
(Angaben nach Rolland, Baillot und Goon 2001, Google Inc. 2010)

Methode	Position	Ausrichtung
Auf Signallaufzeit beruhend		
Positionsbestimmung mittels GPS	absolut	
Entfernungsmessung mit Ultraschall	absolut (0,5 – 6 mm)	absolut (0,1 – 0,6 °)
Entfernungsmessung mit Laserimpulsen	absolut (0,1 mm)	absolut (0,1 °)
Auf Trägheit beruhend		
Bewegungsanalyse mit Accelerometer	relativ	Absolut entlang Gravitationsrichtung
Neigungsmessung mit Gyroskop		relativ (0,2 °)
Auf Magnetfeld beruhend		
Ausrichtungsbestimmung über Kompass		Absolut entlang mag. Nordrichtung (1 – 3 °)
Visuelle Methoden		
Erkennung eines optischen Musters	absolut	absolut

Sensorik in Android verfügbar
Sensorik in Android 2.2 vorgeschrieben

Da die bei dieser Bachelorarbeit behandelten Daten ubiquitär gemessen werden können, sowie eine mobile Augmented Reality Anwendung entwickelt werden soll, müssen zur Bestimmung des virtuellen Referenzsystems also raumunabhängige Technologien verwendet werden. So ist es beispielweise „unpraktisch“ (Azuma, Baillot, et al. 2001) die gesamte reale Umgebung mit künstlichen optischen Markern auszustatten, bzw. vorhandene reale Objekte mittels einer Bilddatenbank zuzuordnen, um eine visuelle Bildregistrierung anzuwenden.

Weil die zuzufügenden Zusatzinformationen sich in ihrer ubiquitären Natur nicht auf spezielle reale Objekte beziehen, sondern einzig in Bezug zur aktuellen Position des mobilen Gerätes stehen, muss das Trackingsystem darüber hinaus nur eine Ausrichtung des Weltkoordinatensystems ermöglichen und keine Verortung realer Objekte. Somit muss das Trackingverfahren keine typische Objekttransformation durchführen können.

Es ist folglich lediglich eine Kombination des Global Positioning System (GPS) mit dem integriertem Kompass und Accelerometer erforderlich, um Sensorinformationen über die aktuelle geografische Position und die Ausrichtung des mobilen Endgeräts zu erhalten, sodass die virtuelle und reale Umgebung erfolgreich überlagert werden können. Durch die verwendete Android 2.2 Version ist hierbei garantiert, dass alle nötigen Sensoren für die Positions- und

Ausrichtungsbestimmung auf allen entsprechenden androidfähigen Geräten verfügbar sind, da diese Sensorik nach (Google Inc. 2010) als Mindestanforderung für die Verwendung dieses Android Betriebssystems spezifiziert sind (siehe Tabelle 1).

Ausgehend von einer Definition der für diese Anwendung verwendeten Koordinatensysteme und deren Abhängigkeiten werden in den folgenden Abschnitten die Methoden und Prozesse beschrieben, die eine Abbildung des mobilen Android Systems in den virtuellen Raum ermöglichen.

3.1 Definition der Koordinatensysteme

Weil die tatsächliche Displaydarstellung in einem Android System nicht direkt mit der physischen Ausrichtung des mobilen Gerätes zusammenhängt, kann die gesuchte Kameratransformation C nicht unmittelbar hieraus bestimmt werden. Sie muss daher unter anderem aus weiteren virtuellen Referenzsystemen abgeleitet werden, die in der Android Plattform dynamisch die Nutzersicht anpassen.

3.1.1 Sensorkoordinaten

Mittels des Sensorkoordinatensystems werden innerhalb der Android Plattform die Ausgaben der integrierten Sensoren dargestellt. Diese Koordinatensystemdefinition ist unveränderbar und immer relativ zur sogenannten Standardausrichtung eines Gerätes definiert. Über diesen Standard entscheiden die Hersteller eines androidfähigen Gerätes selber, da er von der tatsächlichen Bauform, Displaygröße und angestrebten Verwendungsweise durch den Nutzer abhängt. Nach (Google Inc. 2011d) gilt für das Sensorkoordinatensystem folgende Definition:

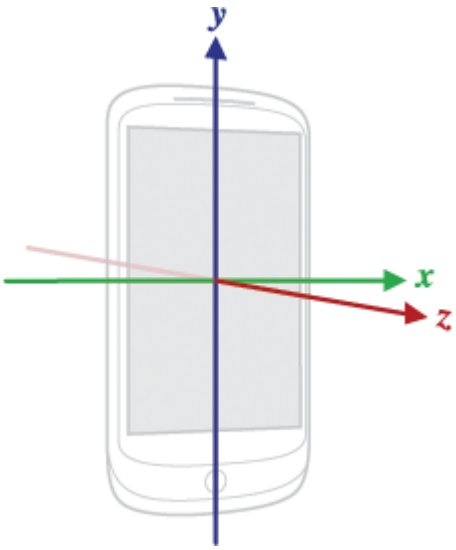


Abbildung 5: Sensorkoordinatensystem gemäß `SensorEvent` API
(Google Inc. 2011d)

- Die X Achse liegt horizontal zur Standardausrichtung nach rechts ausgerichtet
- Die Y Achse ist vertikal nach oben ausgerichtet
- Die Z Achse ist das Kreuzprodukt der X und Y Achse, also senkrecht zu diesen beiden aus dem Display hinaus zeigend definiert. (Rechtshändiges Koordinatensystem)

Bei den meisten androidfähigen Endgeräten entspricht die Standardausrichtung einem Hochformat des Displays (Abbildung 5). Mit der immer größer werdenden Anzahl verschiedenster Android Endgeräte existieren inzwischen auch Modelle, die keine Smartphone-typischen Ausmaße aufweisen und auch nicht von einer hochformatigen Standardorientierung des Displays ausgehen.

3.1.2 Gerätekoordinaten

Basierend auf dem Gerätekoordinatensystem findet die Displaydarstellung statt. Diese wird in der Android Plattform generell an die physische Rotation des Displays angepasst, da ein mobiles Endgerät naturgemäß auf verschiedenste Weisen bedient werden kann und sich die Displayanzeige entsprechend optimal hinsichtlich des Nutzers anpassen soll. Demgemäß wird bei erkannten physischen Rotationen das Gerätekoordinatensystem ausgehend von dem Sensorkoordinatensystem angepasst und aktualisiert. Dies ist ein für die eigentlichen Anwendungen transparenter Prozess und muss bei der Anwendungsdarstellung nicht explizit berücksichtigt werden.

Wegen möglicher unterschiedlicher Auffassung der Standardausrichtung eines jeden androidfähigen mobilen Gerätes, ist der Zusammenhang zwischen physischer Rotation und resultierendem Gerätekoordinatensystem nicht auf jeder Plattform identisch und bedarf spezieller Berücksichtigung.

3.1.3 Weltkoordinatensystem

Das Weltkoordinatensystem soll bei dieser Arbeit als eigentlicher virtueller Raum fungieren, in dem die virtuelle mit der tatsächlichen Realität kombiniert werden kann. Es ist ebenfalls in der Android API beschrieben um Sensorereignisse bezüglich der Erdoberfläche abzubilden. Hierfür gilt nach (Google Inc. 2011e) für einen Standort:

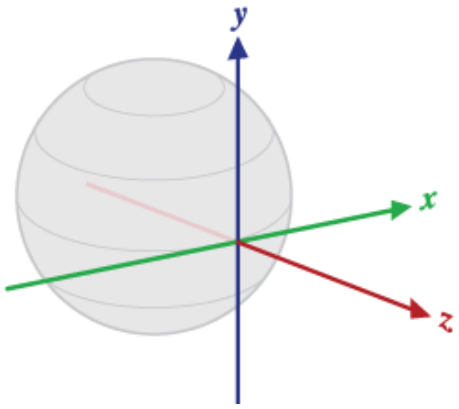


Abbildung 6: Darstellung des Weltkoordinatensystems
(Google Inc. 2011e)

- Die Z Achse steht senkrecht zur Erdoberfläche und zeigt in Richtung des Himmels
- Die Y Achse ist tangential zur Erdoberfläche ausgerichtet und zeigt zum magnetischen Nordpol
- Die X Achse ist das Kreuzprodukt der Y und Z Achse, liegt also senkrecht zu diesen beiden und zeigt ungefähr Richtung Osten

3.2 Transformation zwischen Welt- und Gerätekoordinaten

Über die Sensordaten muss es ermöglicht werden, zwischen dem Geräte- und Weltkoordinatensystem zu transformieren, sodass sowohl die virtuellen gerätebezogenen Informationen als auch die realen Informationen im selben Raum behandelt werden können. Dazu wird im Folgenden das Weltkoordinatensystem bezüglich des Sensorkoordinatensystems entwickelt sowie dessen Ursprung bestimmt. Anschließend wird die Abbildung bezüglich des aktuellen Gerätekoordinatensystems erläutert.

3.2.1 Bestimmung der Koordinatensystemtransformation

Alle für die Ausrichtung des Weltkoordinatensystems nötigen Parameter lassen sich über die Kombination der Daten der in Android 2.2 obligatorischen Beschleunigungs- und Magnetfeldsensoren (Google Inc. 2010, 16) ermitteln, da sie die Ausrichtungen zweier linear unabhängiger Achsen des Weltkoordinatensystems ausdrücken können (Abbildung 7). Die für den Zugriff auf diese Sensordaten nötigen Schnittstellen sind im `android.hardware` Paket der Android Plattform API definiert. Ausgangspunkt ist eine Instanz des „Sensor Service" Systemdienstes, welche über den aktuellen Kontext der jeweiligen Aktivität oder der gesamten Anwendung abgerufen werden kann. Dieser Kontext stellt die `getSystemService` Methode bereit (Listing 1), die hierzu mit der Konstante `Context.SENSOR_SERVICE` als Parameter aufgerufen werden muss.

```
public abstract Object getSystemService (String name)
```

Listing 1: Signatur der `getSystemService` Methode zum Erhalt einer Systemdienstinstanz

Die Sensordaten können mit der so erhaltenen Instanz des `SensorManager` auf einfache Weise über das Beobachtermuster mit dem Registrieren von Rückrufmethoden in Form einer `SensorEventListener` Implementierung mithilfe der `registerListener` Methoden ermittelt werden. Der jeweilige angeforderte Sensor muss dabei über ein `Sensor` Objekt identifiziert werden, welches beispielsweise über die `getDefaultSensor` Methode des `SensorManager` unter Angabe einer Sensortyp Konstante referenziert werden kann.

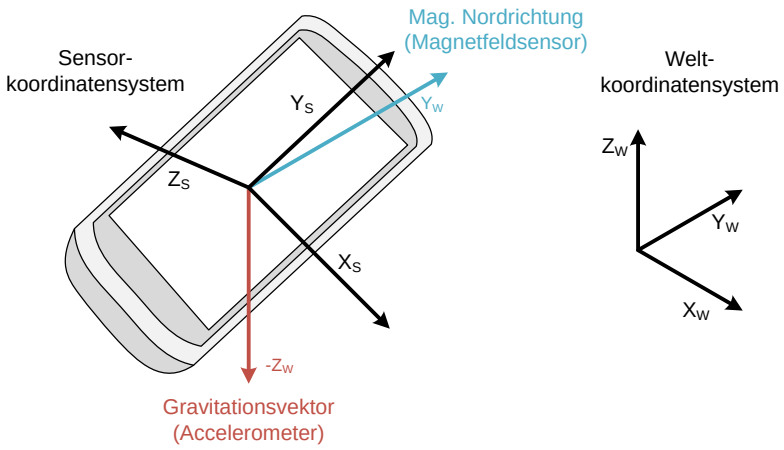


Abbildung 7: Darstellung des Weltkoordinatensystems bezüglich der Sensorkoordinaten
(Eigene Darstellung)

3.2.1.1 Beschleunigungssensor

Mithilfe der Daten des Beschleunigungsmessers oder Accelerometers (Typ `Sensor.TYPE_ACCELEROMETER`) lässt sich zur Ausrichtungsbestimmung die Richtung der Erdanziehung relativ zum mobilen Gerät feststellen, was der Z Achse des Weltkoordinatensystems entspricht (siehe Abbildung 6). Bei diesem Sensor werden jeweils die Beschleunigungen ermittelt, die das mobile Gerät in seinen drei Richtungsachsen erfährt und in der Einheit $\left[\frac{m}{s^2}\right]$ zur Verfügung gestellt. Diese Beschleunigungen sind durch die Gravitation beeinflusst, entsprechen daher ungefähr der Differenz der tatsächlichen Beschleunigung und der Erdanziehung. Wird keine zusätzliche Beschleunigung durchgeführt, bilden die gemessenen Beschleunigungskomponenten also den Vektor bezogen auf das Sensorkoordinatensystem, der in die Richtung der Erdanziehungskraft zeigt.

Liegt das Android Gerät unter diesen Voraussetzungen also flach auf einer ebenen Fläche, zeigt die API eine Gesamtbeschleunigung in Bezug auf die Z Komponente von ungefähr $9{,}81\frac{m}{s^2}$, was der tatsächlichen Beschleunigung ($0\ \frac{m}{s^2}$) minus der mittleren Erdbeschleunigung (hier $-9{,}81\frac{m}{s^2}$ entsprechend der Achsendefinition) entspricht (Google Inc. 2011d).

3.2.1.2 Magnetfeldsensor

Die Y Achse des Weltkoordinatensystems lässt sich durch den Magnetfeldsensor (Typ `Sensor.TYPE_MAGNETIC_FIELD`) bestimmen. Hierdurch kann das Koordinatensystem wie gefordert strikt in Richtung des magnetischen Nordpols ausgerichtet werden. Die mit diesem Sensortyp bestimmten Sensorwerte sind analog zu denen des Beschleunigungsmessers im Sensorkoordinatensystem angegeben, stellen jedoch die Stärke des umgebenen magnetischen Feldes in Mikrotesla [µT] entlang der Richtungsachsen dar. Somit bilden die drei Rich-

tungskomponenten hier einen Vektor bezüglich der Sensorkoordinaten der zum magnetischen Nordpol zeigt.

Für beide Sensortypen sollten zur weiteren Verwendung Filterungen durchgeführt werden, da die Sensordaten Schwankungen und Rauschen beinhalten. Eine Dämpfung dieser Einflüsse kann eine Tiefpassfilterung ermöglichen.

3.2.2 Transformation

Für die eigentliche Bestimmung der Transformationsmatrix zwischen dem Sensor- und Weltkoordinatensystem stellt die Android API die `getRotationMatrix` der `SensorManager` Klasse zur Verfügung, welche direkt für die Verwendung der beschriebenen Sensordaten vorgesehen ist:

```
public static boolean getRotationMatrix (float[] R, float[] I,
     float[] gravity, float[] geomagnetic)
```

Listing 2: Signatur der `getRotationMatrix` Methode zur Bestimmung der Transformationsmatrix vom Sensorkoodinatensystem in das Weltkoordinatensystem

Diese Methode überführt die beiden Vektoren der Y und Z Achse des Weltkoordinatensystems bezüglich der Sensorkoordinaten in eine entsprechende Transformationsmatrix bzw. Rotationsmatrix zwischen beiden Koordinatensystemen, d. h. sie bildet die entsprechende Orthonormalbasis bezüglich der Sensorkoordinaten. Die beiden Parameter `R` und `I` werden hierbei durch einen erfolgreichen Methodenaufruf verändert und stellen danach die gesuchte Rotationsmatrix zur Transformation zwischen den Koordinatensystemen sowie die für diese Arbeit unnötige Inklinationsmatrix als linearisierte 4×4 Matrizen dar. Die weiteren erwarteten Parameter `gravity` und `geomagnetic` sind der Erdanziehungsvektor wie er durch den Beschleunigungsmesser erhalten wird (Z Achse) und der Magnetfeldvektor wie er durch den Magnetfeldsensor erlangt wird (Y Achse).

In der eigentlichen Intention dieser Methode konvertiert die resultierende Transformationsmatrix konkret einen Vektor vom Sensorkoordinatensystem in das Weltkoordinatensystem. Diese Matrix ist jedoch als Array von `float` Werten realisiert worden, welches gewöhnlich auf zwei verschiedene Weisen in eine Matrixdarstellung überführt werden kann. Hierbei unterscheidet man zwischen der sogenannten „row-major“ und „column-major“ Linearisierung von Matrizen. Die Methoden innerhalb der Android eigenen API nutzen die „row-major“ Linearisierung, d. h. in einem entsprechenden Array sind nacheinander die Elemente pro Zeile aufeinanderfolgend aufgelistet. Wird diese Transformationsmatrix allerdings in der OpenGL ES API verwendet, wird sie bezogen auf dessen eigener Spezifikationen als „co-

lumn-major“ lineariserte 4 × 4 Matrix behandelt. Entsprechend sind hier die Elemente der einzelnen Spalten aufeinanderfolgend abgebildet und die eigentliche Matrix wird demzufolge als dessen Transponierte interpretiert.

```
short[] eingabeArray = new short[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

Row-major Linearisierung in eine 3 × 3 Matrix:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Column-major Linearisierung in eine 3 × 3 Matrix:

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

Abbildung 8: Gegenüberstellung der „column-major“ und „row-major“ Linearisierungsweisen an einem Beispiel

Aufgrund der Tatsache, dass es sich hierbei auch um eine Rotationsmatrix handelt, ist die Transponierte der Matrix gleich der Inversen der Matrix, weshalb die via `getRotationMatrix` bestimmte Rotationsmatrix direkt in der OpenGL API verwendet werden kann, um umgekehrt vom Weltkoordinatensystem in das Sensorkoordinatensystem abzubilden.

3.2.3 Positionsbestimmung

Neben der Ausrichtung des Augmented Reality Systems zum Weltkoordinatensystem muss auch dessen Positionierung bestimmt werden, also der Ursprung des Weltkoordinatensystems bezüglich der Gerätekoordinaten. Die Position in der Ebene lässt sich dabei auf einfache Weise über das Android API periodisch über das GPS abfragen. Hierzu muss zunächst analog zur Abfrage der Sensordaten eine Instanz des „Location Service“ Systemdienstes abgerufen werden. Über den so erhaltenen `LocationManager` lassen sich abschließend mit seiner `requestLocationUpdates` Methode die erforderlichen Positionsdaten abrufen. Dessen Überladungen ermöglichen es unter Angabe der geforderten Positionsbestimmungsmethode, Positionsdaten über Rückruffunktionen entsprechend des Beobachter Verhaltensmusters zu erhalten oder die Aktualisierungen über das Senden eines `Intent` zu realisieren. Außerdem bestehen die Möglichkeiten, ein Mindestzeitintervall sowie eine Mindestdistanz zwischen den Positionsmitteilungen festzulegen. Da GPS Funktionalität durch die Mindestanforderungen von Android 2.2 abgedeckt ist (Google Inc. 2010, 16), kann als Positionsbestimmungsmethode problemlos die Konstante `LocationManager.GPS_PROVIDER` verwendet werden, ohne dessen Existenz und Gültigkeit im Vorfeld prüfen zu müssen.

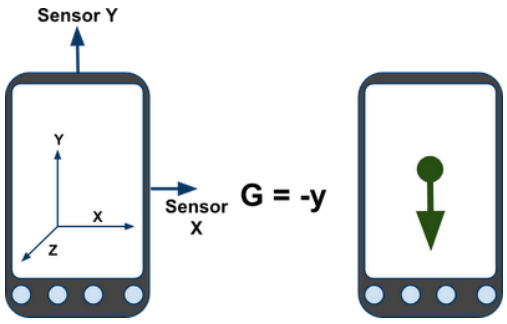
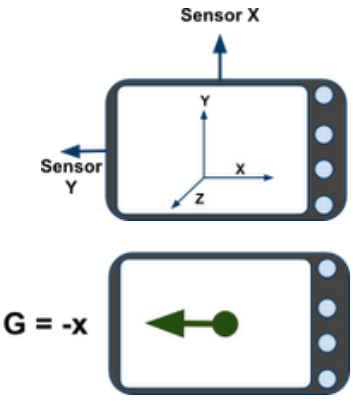
Für eine vollständige Anpassung der Welt- und Gerätekoordinaten ist zusätzlich noch eine Höheninformation bezüglich des mobilen Gerätes über Grund notwendig. Diese ist weder mit dem GPS noch mit anderen in der Android Plattform verfügbaren Sensoren ausreichend genau bestimmbar. Mit der GPS Sensorik wäre lediglich eine Höhe entsprechend des zugrundeliegenden Ellipsoids zu bestimmen die mit einer lokalen oder als Webservice verfügbaren Anpassung des Ellipsoides auf Bodenniveau verglichen werden müsste. Da die GPS Bestimmung schon Fehlertoleranzen von zwei Größenordnungen (vgl. de Lange 2005, 221) aufweist, ist sie für die Bestimmung der Gerätehöhe nicht zu verwenden. Entsprechend muss die Höhenangabe im Vorfeld geschätzt oder vom Nutzer manuell angegeben werden.

3.2.4 Überführung in Gerätekoordinaten

Wie zum Beginn dieses Abschnittes beschrieben, wird auf der Android Plattform die Darstellung auf dem Display generell an dessen physischer Rotation angepasst, da ein mobiles Endgerät naturgemäß auf verschiedenste Weise bedient werden kann und sich die Displaydarstellung entsprechend optimal anpassen soll. Diese dynamische Anpassung muss bei der Transformation zwischen Welt- und Gerätekoordinaten explizit berücksichtigt werden, da diese dabei auf dem statischen Sensorkoordinatensystem fußt.

Tabelle 2: Beispielhafte Darstellung des Gravitationsvektors definiert durch die Accelerometerdaten bei aufrechter Position des Gerätes in unterschiedlichen Konstellationen

(Nach Morrill 2010)

Standardausrichtung	Displayrotation 90 °
Sensor- und Gerätekoordinatensystem stimmen überein.	Dynamische Anpassung der Gerätekoordinaten, Sensor- und Gerätekoordinatensystem weichen voneinander ab
	

Zur Abbildung der physischen Rotation auf die Displaydarstellung existieren in der Android Plattform zwei sich ergänzende Konzepte. Zum einen wird das zur Anzeige verwendete Display stets einer Rotationskonstante zugeordnet. Hierbei wird in 90 ° Schritten zwischen Drehungen von 0 ° bis 270 ° unterschieden, wobei 0 ° der Standardausrichtung des jeweili-

gen Gerätes entspricht. Als weiteres Konzept wird entsprechend des Seitenverhältnisses der aktuellen Anzeige unterschieden, der sogenannten Displayorientierung. Tabelle 3 stellt die in der API zur Beschreibung der verschiedenen Zustände verwendeten Konstanten gegenüber.

Obwohl bei den meisten androidfähigen Endgeräten die Standardausrichtung einem Hochformat entspricht, kann wegen der uneinheitlichen Definition des Sensorkoordinatensystems der Zusammenhang zwischen Displayrotation und Displayorientierung nicht auf jedem Gerät gleichermaßen vorausgesetzt werden. Folglich muss auch bei Applikationen, die beispielsweise eine feste Displayorientierung vorgeben und deshalb nicht der dynamischen Anpassung des Android Systems unterliegen, die Displayrotation bei der Bestimmung der Transformationsmatrix für einen Augmented Reality Eindruck dennoch stets beachtet werden.

Tabelle 3: Gegenüberstellung der Konstanten zur Beschreibung der Displayausrichtungen

Displayrotation	Displayorientierung
android.view. Surface.ROTATION_0 Surface.ROTATION_90 Surface.ROTATION_180 Surface.ROTATION_270	*android.content.res.* Configuration.ORIENTATION_LANDSCAPE Configuration.ORIENTATION_PORTRAIT Configuration.ORIENTATION_SQUARE Configuration.ORIENTATION_UNDEFINED
Beschreibt die aktuell nötige Rotation der Displaydarstellung ausgehend von der Standardorientierung. Die tatsächliche physische Rotation des Gerätes ist daher genau die jeweils hierzu gegensätzliche Rotation.	Diese Konstanten beschreiben das Ansichtsverhältnis der Displaydarstellung. Sie wird beispielsweise verwendet um die für eine Applikation erlaubten Ansichten zu bestimmen.

Zu diesem Zweck bietet das Android API seit Version 1.5 in der `SensorManager` Klasse eine Methode zum Anpassen der Basis bezüglich derer eine Rotationsmatrix angegeben wurde:

```
public static boolean remapCoordinateSystem (float[] inR, int X,
      int Y, float[] outR)
```

Listing 3: Signatur der `remapCoordinateSystem` Methode der `SensorManager` API

Hierbei wird über den Parameter `inR` die row-major linearisierte Rotationsmatrix angegeben, die weiter transformiert werden soll. Das Ergebnis wird in das über den Parameter `outR` referenzierte Array geschrieben. Die Parameter `X` und `Y` bestimmen anhand von Kon-

stanten welcher Koordinatenachse die X bzw. Y Achse des Koordinatensystems der aktuellen Transformation durch diese Methode zugeordnet werden soll. Die Z Achse wird hierbei automatisch orthogonal zu den anderen Achsen ausgerichtet, sodass ein Orthonormalsystem entsteht.

Tabelle 4: Anpassung der Sensorkoordinatenbasis bei Displayrotation

Displayrotation	Neuzuweisung der Koordinatenbasis	
Surface.	X	Y
ROTATION_0	SensorManager.AXIS_X	SensorManager.AXIS_Y
ROTATION_90	SensorManager.AXIS_Y	SensorManager.AXIS_MINUS_X
ROTATION_180	SensorManager.AXIS_MINUS_X	SensorManager.AXIS_MINUS_Y
ROTATION_270	SensorManager.AXIS_MINUS_Y	SensorManager.AXIS_X
	X und Y beziehen sich auf die Parameterbezeichnungen der SensorManager.remapCoordinateSystem Funktion	

Die korrekte Anpassung der erarbeiteten Transformation zur Welt-Gerätekoordinaten Transformation basierend auf dieser Methode zeigt Tabelle 4 ausgehend von den verschiedenen Displayrotationen. Die aktuelle Rotation des Displays lässt sich dabei erneut mittels eines Android Systemdienstes abfragen, dem „Window Service“. Ein Vorgehen hierfür wird in Listing 4 exemplarisch demonstriert.

```
// Systemdienstinstanz erhalten
WindowManager windowManager = (WindowManager) getContext().
     getSystemService(Context.WINDOW_SERVICE);

// Rotationskonstante Surface.ROTATION_...
int rotation = windowManager.getDefaultDisplay().getRotation();
```

Listing 4: Bestimmung der aktuellen Displayrotation in einer `Activity` Klasse

4 Lärminterpolation

Durch die Verwendung Community-basierter Messungen kann potentiell die Qualität und der Gehalt von Lärmkarten bzw. -interpolationen erheblich verbessert werden. In der Praxis basieren sie derzeit hauptsächlich auf Simulationen ausgehend von einigen statischen Lautheitsmessungen und Nutzungsstatistiken diverser Verkehrswege (vgl. Europäisches Parlament und Rat 2002). Durch die ubiquitär von Nutzern einer Lärmmessungscommunity erhebbaren Lärmdaten können derartige Kartierungen sowohl bezüglich ihrer Aktualität als auch ihrer Genauigkeit verbessert werden. Es können durch die direkten Messungen der effektiv bei den Nutzern wirkenden Schalleinflüsse die Unsicherheiten solcher Simulationen reduziert und die tatsächlichen Einflüsse des Schalls auf die Menschen besser quantifiziert werden.

Ziel der Lärminterpolation ist es in dieser Arbeit, eine räumliche Darstellung des Maßes der an einem Punkt wirkenden Geräusche ausgehend von nutzerbasierten Lärmmessungen ohne zusätzlicher Eingabedaten rasterbasiert zu erzeugen. Dazu sollen die Charakteristika dieser Datenquellen und die Limitierungen der Android Plattform berücksichtig werden. Um die Interpolation einer weiteren Visualisierung zur Verfügung zu stellen, sollen zudem explizit Datenstrukturen gewahrt werden, die eine effektive Darstellung sowohl bezüglich der OpenGL ES Komponente als auch des Android-eigenen Anzeigesystems erlauben.

Das hierzu entwickelte Vorgehen, dessen Teilschritte in den folgenden Abschnitten näher erläutert werden, zeigt Abbildung 9 in einem schematischen Überblick. Neben einer theoretischen und praktischen Beschreibung des algorithmischen Vorgehens fokussiert sich jeweils das Ende der Abschnitte auf die Darstellung von Optimierungen, die eine Ausführung im mobilen Kontext sicherstellen. Im letzten Abschnitt werden die hierfür verwendbaren Datenstrukturen unter dem Aspekt der limitierten Ressourcen und weiteren Verwendungen diskutiert.

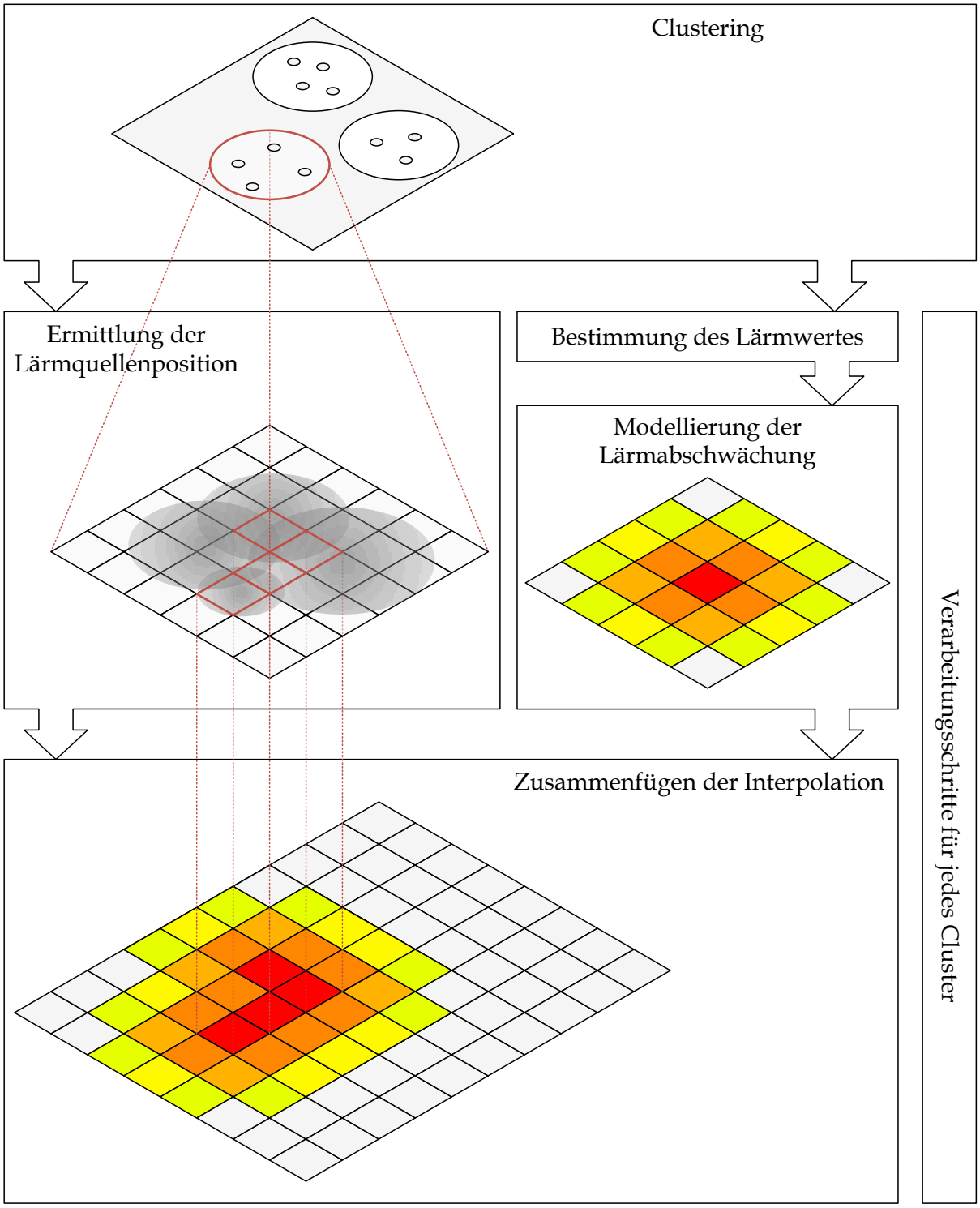


Abbildung 9: Schematische Darstellung des Interpolationsprozesses

(Eigene Darstellung)

4.1 Datengrundlage

Die hier entwickelten Prozesse zur Interpolation und Visualisierung von Lärmdaten sind keinen speziellen Lärmkartierungsprojekten zugeordnet. Sie umfassen allgemein Methoden und Praktiken, um communitybasierte, d. h. auf freiwilliger Basis durch ungeübte Nutzer

durchgeführte Lärmmessungen zu verarbeiten. Für ein Messereignis werden dabei lediglich folgende Merkmale vorausgesetzt:

- Angabe über die Messposition, ohne Höhenangabe
- Angabe über die Genauigkeit der Positionsbestimmung
- Angabe über den gemessenen Schalldruckpegel

Beispiele für bestehende Projekte, die das sogenannte Crowdsourcing nutzen um mobile personenbezogene Schallsensoren zu etablieren sind unter anderem das NoiseTube Projekt[1] und dessen Abkömmling BrusSense[2], welche verschiedene mobile Plattformen adressieren, sowie WideNoise[3] für Apple iOS. Diese nutzen integrierte Mikrofone zur Lärmmessung und teilen diese in einem Onlineangebot. Einzig NoiseTube stellt dabei die Messdatenbank über eine öffentliche API Entwicklern zur Verfügung.

Allen aktuellen Projekten zur nutzerbasierten Lärmkartierung ist derzeit gemein, dass sie zwar behaupten Lärmmessungen durchzuführen und bereitzustellen, es sich dabei jedoch lediglich um Angaben von objektiven Schalldruckpegeln handelt. Lärm wird jedoch allgemein als ein subjektives Phänomen beschrieben. Laut (Europäisches Parlament und Rat 2002) handelt es sich dabei bei Umgebungslärm genauer um „unerwünschte oder gesundheitsschädliche Geräusche im Freien, die durch Aktivitäten von Menschen verursacht werden". Diese subjektive Wirkung des Schalls auf den Menschen hängt beispielsweise „vom Charakter des Geräusches und von der Einwirkdauer" (Möser 2009) ab. Um solche Faktoren in der Bestimmung des Schallpegels zu berücksichtigen, definiert die Technische Anleitung zum Schutz gegen Lärm (Umweltbundesamt 1998) einige Korrekturterme, die diese Charakteristik eines Geräusches abbilden sollen und in einen objektiv gemessenen Schalldruckpegel einberechnet eine Lärmbewertung ergeben. In den aktuellen Lärmkartierungsprojekten konnte keine Anwendung solcher Korrekturmaßnahmen zur Annäherung an den subjektiven Lärmbegriff gefunden werden. Das NoiseTube Projekt beinhaltet jedoch die Möglichkeit Messungen mit persönlichen Schlagworten zu annotieren, dem sogenannten „Social Tagging" und verwendet nach eigenen Angaben auch Methoden zur automatischen Lärmklassifizierung (Maisonneuve, Stevens und Ochab 2010). Eine Möglichkeit auf diese Daten zuzugreifen scheint allerdings nicht zu existieren.

[1] NoiseTube Internetpräsenz: http://www.noisetube.net/
[2] BrusSense Internetpräsenz: http://www.brussense.be/
[3] WideNoise Internetpräsenz: http://www.widetag.com/widenoise/

Ein weiterer Aspekt zur Verarbeitung solcher Nutzermessungen ist, dass stets robuste Verfahren verwendet werden müssen, da die Qualität derartiger Daten wegen ihrer unsicheren Herkunft nicht gewährleistet werden kann und mit hohen Ausreißerquoten zu rechnen ist. Hierbei zeigen zudem auch die zur Lärmkartierung verwendbaren Methoden selbst große Unsicherheitsfaktoren, beispielsweise kann bei der Positionsbestimmung mit Systemen für den privaten Gebrauch keine exakte Position erwartet werden. Während die Einflüsse auf die Positionsbestimmung nur bedingt menschlicher Natur sind, etwa bei Abschattung des GPS Empfängers durch den Nutzer, hat das Verhalten des Nutzers bei der eigentlichen Lärmmessung einen weitaus größeren Einfluss. Diese Messung wird in den relevanten Projekten über das integrierte Mikrofon des mobilen Endgerätes durchgeführt und wird sehr stark von der Haltung und Ausrichtung sowie des Zugangs zum Mikrofon beeinflusst. Hinzu kommt auch hierbei eine systembedingte Unsicherheit, hervorgerufen durch die zumeist verwendeten Mikrofone als Schallmessinstrument. Für die Android Plattform, als Beispiel, werden zur Audiohardware der mobilen Geräte zwar Vorgaben in den betreffenden Spezifikationen bezüglich ihrer Sensibilität und einem standardisierten Aufnahmeverhalten gemacht, diese sind allerdings nicht als verbindlich definiert und gelten derzeit nur als Empfehlung für die jeweiligen Gerätehersteller (Google Inc. 2010, 13).

4.2 Rasterisierung

Da das Ergebnis wegen der Flächenhaftigkeit des zugrundeliegenden Lärmphänomens eine rasterbasierte Darstellung der Lärmimmissionen sein soll, muss dem Verfahren eine Kartenabbildung der meist geografischen Koordinaten der Eingangsdaten in ein kartesisches Koordinatensystem zugrunde gelegt werden. Um möglichst Ergebnisse räumlicher Operationen, wie etwa der Modellierung einer räumlichen Lärmdämpfung, mit einfachen und effizienten Rasteroperationen auf verschiedenen Position im Ergebnisraster anwenden zu können, sollte die Abbildung konform sein, sowie geringe kleinräumliche Verzerrungen aufweisen.

4.3 Clusteranalyse

Bei von Nutzern selbst erstellen Lärmmessungen existiert neben der systembedingten Ungenauigkeit der Positionsermittlung während des Messens auch eine Ungenauigkeit bezüglich der relativen Position zu der vorherrschenden Lärmquelle. Es ist davon auszugehen, dass dieselbe Lärmquelle in den Daten einer Lärmmessungscommunity von verschiedenen Nutzern an variierenden Positionen bezüglich Ort und Zeit mehrfach referenziert sein wird.

Um diese Ungenauigkeit bezüglich der Lärmquelle und ihrer Position bei der Interpolation von Lärmwerten zu berücksichtigen, aber auch um den Aufwand für weitere Prozessschritte

im Hinblick auf die einschränkten Systemressourcen zu reduzieren, bietet es sich an die Messungsdaten zu bündeln, sodass jede kleinräumliche Anhäufung von Messungen als ein gemeinsames Lärmereignis interpretiert und deren Messwerte entsprechend zusammengefasst werden. Dieser Bündelungsvorgang wird als Clustering oder Clusteranalyse bezeichnet, wobei einzelne Elemente eines Datensatzes zu sogenannten Clustern gruppiert werden.

Wegen der vergleichsweise hohen Geschwindigkeit und einfachen Verwendung wird zu diesem Zweck ein k-Means Algorithmus für diese Arbeit verwendet. Er gilt als die „gebräuchlichste Methode zum Auffinden von Clustern in Daten“ (Elkan 2003), da hier nicht alle Daten paarweise verglichen werden müssen um Gruppierungen aufzufinden. Ausgehend von einer bestimmten bekannten oder ermittelten Anzahl an Clustern mit zunächst zufälligen Mittelpunkten werden iterativ alle Elemente dem jeweils bezüglich eines Merkmals nahestem Cluster zugeordnet. Anschließend wird der Mittelpunkt jedes Clusters berechnet und die Zuordnung zu den hiernach nahesten Clustern wiederholt, bis Konvergenz eintritt. Abbildung 10 stellt diesen Arbeitsablauf schematisch dar.

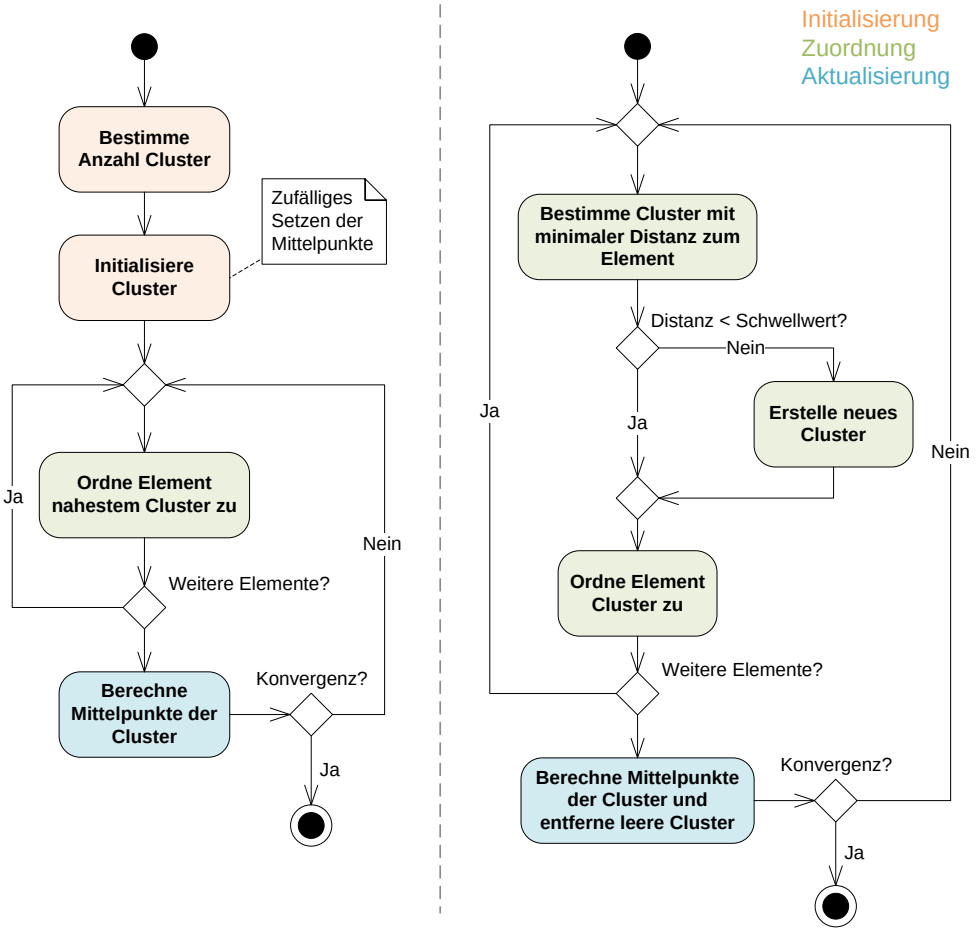


Abbildung 10: Aktivitätsdiagramme k-Means Algorithmus (links) und Erweiterung (rechts) mit Aufteilung in Ablaufphasen

(Eigene Darstellung)

Hieraus ergeben sich jedoch für diese Anwendung insbesondere folgende Probleme, die durch eine Erweiterung dieser Vorgehensweise korrigiert werden müssen:

- *Die Anzahl an Clustern muss bekannt sein*
  Da eine Gruppe jeweils ein spezielles Lärmereignis repräsentieren soll, ist zuvor nicht bekannt wie viele räumlich verschiedene Lärmereignisse von den Nutzern einer Lärmcommunity in einem Bereich tatsächlich erfasst wurden.
- *Die Cluster werden mit zufälligen Werten initialisiert, weshalb der Algorithmus unterschiedliche Ergebnisse bei gleichen Eingabedaten liefern kann*
  Es sollten bei einer Lärminterpolation keine unerwarteten Resultate berechnet werden.
- *Das Verfahren ist nicht robust, denn Elemente werden immer dem jeweils nahestem Cluster zugeordnet, weshalb hier ein Ausreißer großen Einfluss auf die Mittelpunkte der Cluster haben kann*
  Wegen der großen Unsicherheitsfaktoren bei Community-basierten Daten muss der Algorithmus jedoch robust bezüglich der Lärmmessungsposition sein.

All diese Einschränkungen können behoben werden, sobald man eine maximale Clustergröße annimmt und für jedes Element des Datensatz ein neues Cluster erstellt, sobald die Entfernung zum nahesten diese maximale Größe überschreitet. Somit wird die Anzahl an Clustern durch den Algorithmus dynamisch und deterministisch bestimmt, sowie der Einfluss von Ausreißern auf das Clustering reduziert, da diese automatisch als ein eigenes Cluster repräsentiert werden und gegebenenfalls im Anschluss entfernt werden können. Dieses speziell angepasste Clusteringschema stellt Abbildung 10 gegenüber dem ursprünglichen Ablauf dar. Als Ähnlichkeitsmaß wird das dem Raster zugrundeliegende Distanzmaß bezüglich der jeweils zur Messung kartierten Koordinaten verwendet.

Der größte Berechnungsaufwand bei diesem Verfahren ist die Bestimmung der jeweiligen Gruppierungen, die einer Lärmmessung am nahesten liegen, da diese in jeder Iteration für jede Messung durchgeführt werden muss. Hierbei kann sich eine Komplexität von $0(c \cdot n^2)$ einstellen, wobei c die maximale Anzahl an Iterationen darstellt. Die Laufzeit eines derart trivialen Algorithmus‘, welcher im Zuordnungsschritt jeweils die Distanzen eines Messereignisses zu jedem bereits bekannten Cluster vergleicht, zeigt Abbildung 11. Zur Darstellung des Verhaltens wurde das Verfahren mit verschiedenen Anzahlen zufällig gleichverteilt erzeugten Lärmereignissen für ein etwa 0,5 km² großes Gebiet auf einem HTC Legend Mobil-

telefon mit der Android Version 2.2 getestet, wobei jeweils 15 Iterationen entsprechend des k-Means Musters durchgeführt wurden. In den für diese Arbeit herangezogenen Datenquellen basierend auf einem Lärmkartierungsprojekt aus der Lehrveranstaltung „Geosoftware II“ des Wintersemesters 2010/2011 existiert für ein solches Gebiet zum Vergleich ein Maximum von 413 Lärmmessungsdaten.

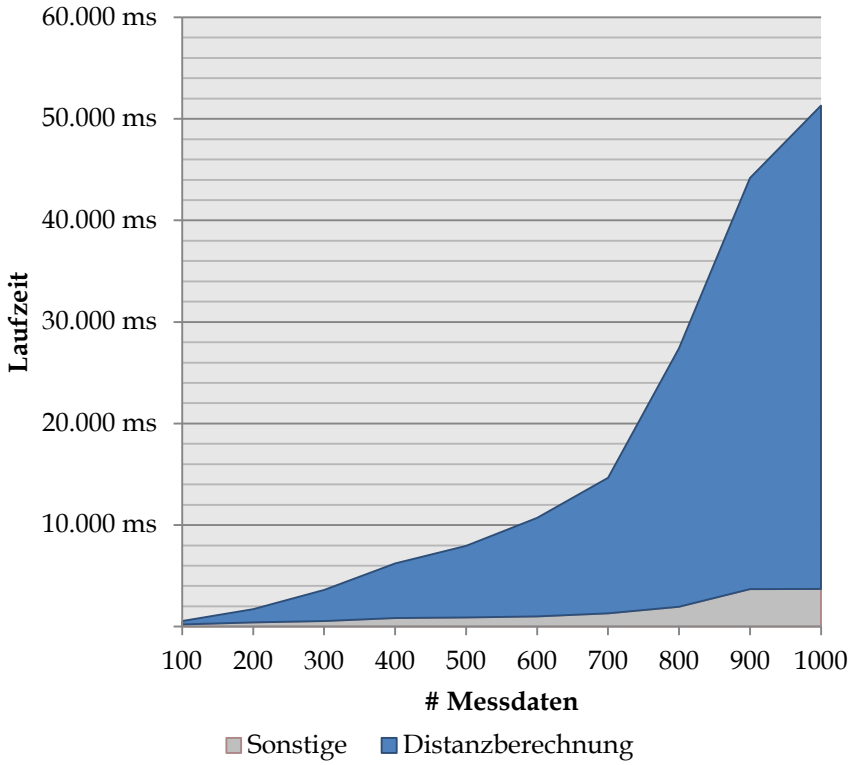


Abbildung 11: Laufzeitverhalten des trivialen Clusteringalgorithmus

Optimiert kann ein solcher Algorithmus durch geschicktes Sortieren der jeweils verwendeten Datenstrukturen werden. Für diese Arbeit werden die Cluster entsprechend ihrer Y-Positionskomponente mittels binärer Suchverfahren sortiert gespeichert, sodass die Suche nach den nächsten Nachbarn effektiv auf einen Auszug der Daten bezüglich ihrer Y-Komponente reduziert werden kann. Ebenfalls mittels binärer Suche lässt sich so die Menge an bereits identifizierten Clustern für jede Messung derart einschränken, dass die eigentliche Distanzberechnung nur noch für Cluster ausgeführt wird, die sich bezüglich ihrer Y-Koordinaten noch in der maximalen Clustergröße befinden.

Hierzu können die in der Java 1.2 Spezifikation definierten Suchmethoden `Collections.binarySearch` verwendet werden, wodurch jeweils höchste plattformunabhängige Leistung erwartet werden kann. Das Verhalten des nach diesen Maßstäben implementierten Algorithmus‘ zeigt Abbildung 12 (links). Die Einschränkung der in jeder Iteration zu betrachtenden Menge an Clustern ist deutlich an der bis zum Faktor 54 reduzierten Ausfüh-

rungszeit der Distanzberechnungen zu erkennen. Der Schritt „Cluster Einsortierung“ bedeutet hierbei die Zeit, die während des Vorgangs gebraucht wird, um ein neu erstelltes Cluster entsprechend der Y-Koordinate sortiert zu speichern bzw. um die Cluster im Aktualisierungsschritt neu anzuordnen.

Verfahren mit gleichzeitiger Sortierung nach Y Koordinate

Verfahren mit zusätzlicher Sortierung nach X Koordinate

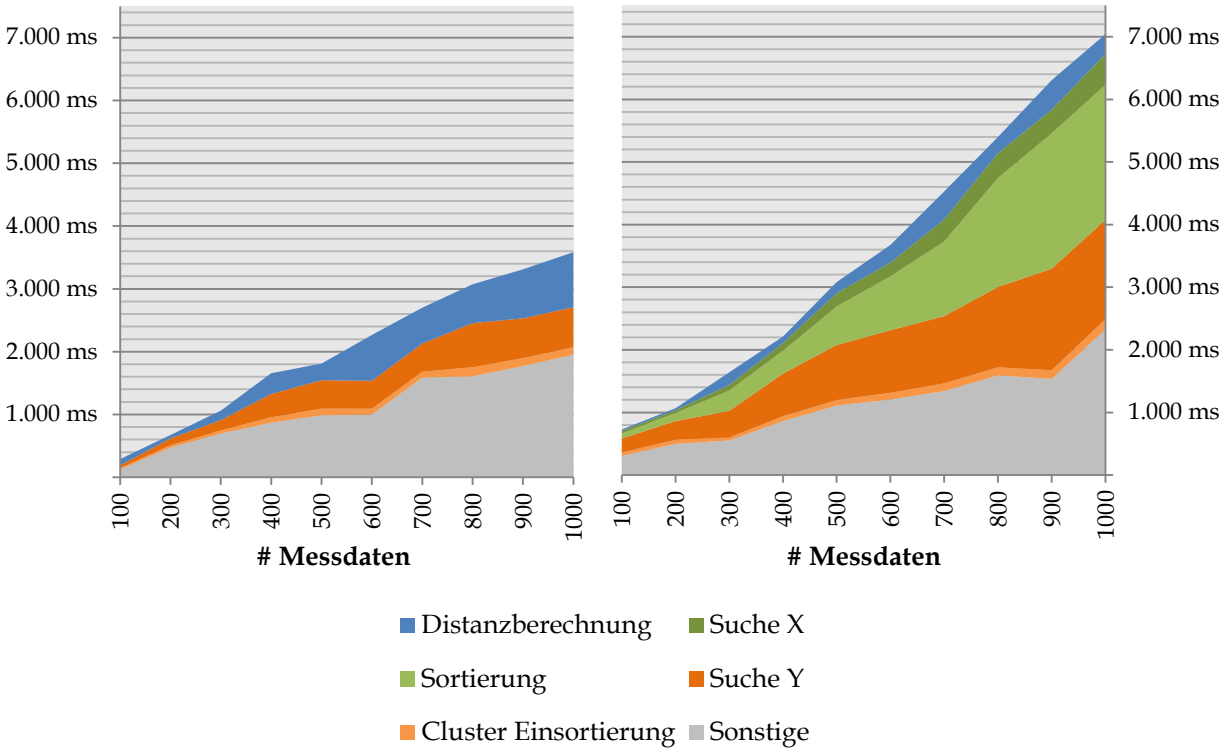


Abbildung 12: Vergleich der Optimierungen des Clusteringalgorithmus' durch Sortierung nach Y bzw. nach X und Y Koordinate anhand der Laufzeiten

Eine zusätzliche Einschränkung der beim Distanzvergleich jeweils zu betrachtenden Menge an Clustern entsprechend ihrer X-Koordinaten erwies sich in diesem Simulationsszenario als unvorteilhaft (Abbildung 12, rechts). Hierdurch zeigte sich zwar eine angestrebte zusätzliche Reduzierung der Distanzberechnungszeiten um den Faktor 2,7, die Gesamtdauer des Verfahrens wurde dabei jedoch nahezu verdoppelt. Die zum Clustering nötigen Berechnungen sind nicht von ausreichend hoher Komplexität, als dass dessen Reduzierung den Mehraufwand des zusätzlichen Sortierens nach der X Koordinate rechtfertigen kann. Zusätzlich zum vermehrten Aufwand durch die exemplarisch verwendete in der Java 1.2 Spezifikation definierten Mergesort-Variante mit $0(n \log n)$ Komplexität müssen hierbei auch erhöhte Ausführungszeiten bei der eigentlichen Indizierung nach der Y-Komponente hingenommen werden. Anders als zuvor kann nicht länger Gebrauch eines Auszuges der Clusterliste gemacht werden, wie er mittels `Collections.subList` auf schnelle Weise zur Verfügung steht.

Stattdessen muss für diesen Auszug ein eigener Speicherbereich alloziert werden, um das notwendige Sortieren zur Einschränkung bezüglich der X-Koordinate mit hoher Geschwindigkeit zu ermöglichen. Das Ergebnisobjekt dieser `subList` Methode delegiert lediglich elementspezifische Aufrufe an die eigentliche Listeninstanz weiter und würde die Integrität der Sortierung nach der Y-Komponente verletzen.

4.4 Bestimmung des Lärmpegels

In den bekannten Community-basierten Lärmdatenbanken existiert für einen Messpunkt nur Auskunft über die dort gemessene Schallstärke unter Angabe des Schalldruckpegels in der Einheit Dezibel [dB]. Es muss dabei angenommen werden, dass es sich hierbei um Immissionsmessungen handelt, d. h. die gemessene Stärke ist die Summe verschiedener an einem Ort wirkender Lärmquellen und beschreibt nicht die Schallausstrahlung einer spezifischen Lärmquelle.

Die angegebene Stärke resultiert dabei genauer aus den im jeweils verwendeten Mikrofon bestimmten Schalldruckdifferenzen innerhalb eines Messzeitraums. Für diesen sogenannten Effektivwert des Schalldrucks (p_{eff}) „gilt allgemein die Wurzel aus dem quadratischen Mittelwert […] des Schalldrucks“ (Möser 2009, 56), welcher wie folgt definiert ist:

$$p_{\text{eff}} = \sqrt{\frac{1}{T}\int_0^T p^2(t)\,dt} \qquad \text{[Pa]}$$

mit $p^2(t)$ quadrierter Schalldruck in [Pa²] an Zeitpunkt t
T Beobachtungszeit

Gleichung 1: Berechnung des Effektivwertes des Schalldrucks p_{eff}
(nach Möser 2009, 56)

Aus diesem Effektivwert lässt sich unmittelbar der „allgemein bekannt[e]“ (Möser 2009, 56) logarithmische Schalldruckpegel (L_p) als Verhältnis zu einem Bezugsschalldruck berechnen, denn es gilt:

$$L_p = 10 \log_{10} \frac{p_{\text{eff}}^2}{p_0^2} = 20 \log_{10} \frac{p_{\text{eff}}}{p_0} \qquad \text{[dB]}$$

mit p_0 Bezugsschalldruck (standardisiert auf $2 \cdot 10^{-5}$ Pa
p_{eff} Effektivwert des Schalldrucks

Gleichung 2: Berechnung des Schalldruckpegels L_p

(nach Möser 2009, 56)

Mittels dieser Zusammenhänge lässt sich abschließend ein durchschnittlicher Lärmpegel für jedes Messungscluster berechnen, welcher somit sowohl die Messposition als auch den Messzeitraum mittelt, aus dem die Messungen bezogen werden. Wegen der logarithmischen Skalierung des Lärmpegels müssen diese zur Mittelwertbildung zunächst in ein Schalldruckverhältnis umgerechnet werden. Hierbei folgt aus der Definition des Schalldruckpegels und des Logarithmus:

$$\begin{aligned} & L_p = 10 \log_{10} \frac{p_{\text{eff}}^2}{p_0^2} \\ \Leftrightarrow \quad & \frac{L_p}{10} = \log_{10} \left(\frac{p_{\text{eff}}}{p_0} \right)^2 \\ \Leftrightarrow \quad & \left(\frac{p_{\text{eff}}}{p_0} \right)^2 = 10^{\frac{L_p}{10}} \qquad \text{da } \log_b c = x \Leftrightarrow b^x = c \end{aligned}$$

Und somit gilt für das gesuchte arithmetische Mittel der Schalldruckpegel $L_{p,1}, \ldots, L_{p,n}$:

$$\bar{L}_p = 10 \log_{10} \left(\frac{1}{n} \sum_{i=1}^{n} 10^{\frac{L_{p,i}}{10}} \right) \qquad \text{[dB]}$$

Gleichung 3: Formel für das arithmetische Mittel von Schalldruckpegeln

(nach Möser 2009, 62)

4.5 Positionsbestimmung

Zur genaueren Bestimmung der Lärmquellenposition innerhalb eines jeden ermittelten Clusters, sowie zum Zweck der Überführung der Messdaten in das zur Interpolation und Visualisierung verwendete Rastersystem muss für jede Rasterzelle die Wahrscheinlichkeit bekannt sein, mit der sich in dieser schlussendlich eine kartierte Lärmquelle befindet. Hierzu wird in dieser Arbeit das durch den Genauigkeitsradius bestimmte Gebiet als eine bivariate Normalverteilung interpretiert. Im Zentrum des Genauigkeitsradius herrscht somit die höchste Wahrscheinlichkeitsdichte, die entsprechend der Normalverteilung zum Rand hin abnimmt. Mit dieser Annahme kann für jede Rasterzelle die Wahrscheinlichkeit des Auftritts bezüglich

eines einzelnen Messereignisses modelliert werden. Um die Rasterzellen mit den höchsten Auftrittswahrscheinlichkeiten bezüglich aller Messereignisse innerhalb eines Clusters zu bestimmen, kann das Prinzip eines Akkumulators basierend auf dem finalen Raster herangezogen werden, welcher die Wahrscheinlichkeiten aller Messdaten für jede Rasterzelle aufaddiert. Aus diesem Akkumulator können anschließend die Rasterzellen mit den höchsten aufaddierten Werten als Positionen für das kartierte Lärmereignis herangezogen werden. Dieses Vorgehen stellt Abbildung 13 als Überblick dar. Durch das flexible stochastische System können auch weitere Modellierungen eingebunden werden, wie z. B. eine Abbildung der Linienhaftigkeit der Messorte durch eine Hough Transformation.

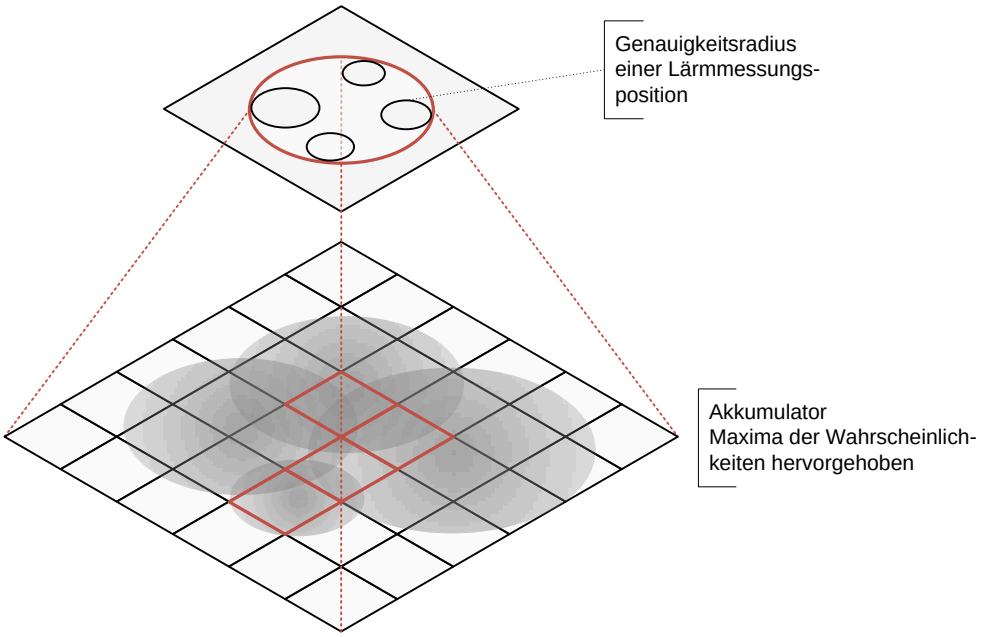


Abbildung 13: Überführung der Messpositionen anhand ihrer Genauigkeitsangaben in das Zielraster
(Eigene Darstellung)

Das durch die Vielzahl der dabei zu bestimmenden distanzabhängigen Wahrscheinlichkeiten auftretende Optimierungsproblem kann zum Beispiel mittels Dynamischer Programmierung gelöst werden. Die zu akkumulierenden Auftrittswahrscheinlichkeiten stellen dabei die Teilprobleme dar, die zur Effektivitätssteigerung als Zwischenresultate gespeichert werden können.

4.6 Modellierung der Lärmausbreitung

Um eine Visualisierung der Lärmdaten zu ermöglichen muss zuvor als letzte Vorbereitung eine Interpolation basierend auf den vorherigen Schritten durchgeführt werden. Hierbei stehen nur die in den vorherigen Schritten aus den Lärmkartierungsdaten extrahierten Schallpegelinformationen zur Verfügung. Entsprechend können keine komplexen Schallausbrei-

tungsmodelle angewandt werden um aus diesen Daten eine Extrapolation zu entwickeln, da nur Informationen über die effektiv wirkenden Schalldrücke, jedoch keinerlei gesicherte Daten über die tatsächlichen Emissionsquellen vorhanden sind. Auch weitere, den Schall und seine Ausbreitung bestimmenden Faktoren können mangels Information nicht in der Modellierung berücksichtig werden, wie z. B. der Einfluss der Temperatur, des Windes oder der dämpfenden Eigenschaften der Luftzusammensetzung (vgl. Müller 2003).

Die bekannten Lärmimmissionen können dagegen nur entsprechend des physikalischen Abstandsgesetzes in ihrer Ausbreitung modelliert werden, da es sich bei den vorhandenen Schalldruckpegelwerten um sogenannte lineare Feldgrößen handelt. Hiernach gilt für zwei Lärmpegel bezogen auf dieselbe Lärmquelle, aber gemessen in verschiedenen Entfernungen folgender Zusammenhang:

$$L_{p,2} = L_{p,1} - 20 \log_{10} \frac{r_2}{r_1} \qquad \text{[dB]}$$

mit $L_{p,i}$ Schalldruckpegel an Messpunkt i
r_i Messabstand von Messpunkt i zur Lärmquelle

Gleichung 4: Schalldruckpegelabschwächung in Abhängigkeit zur Distanz einer Schallquelle
(Sengpiel 2011)

Somit kann im Zielraster um jeden identifizierten Messpunkt für eine bestimmte Nachbarschaft an Rasterzellen die exakte, entsprechend des Abstandsgesetzes abgeschwächte Lärmimmission berechnet werden.

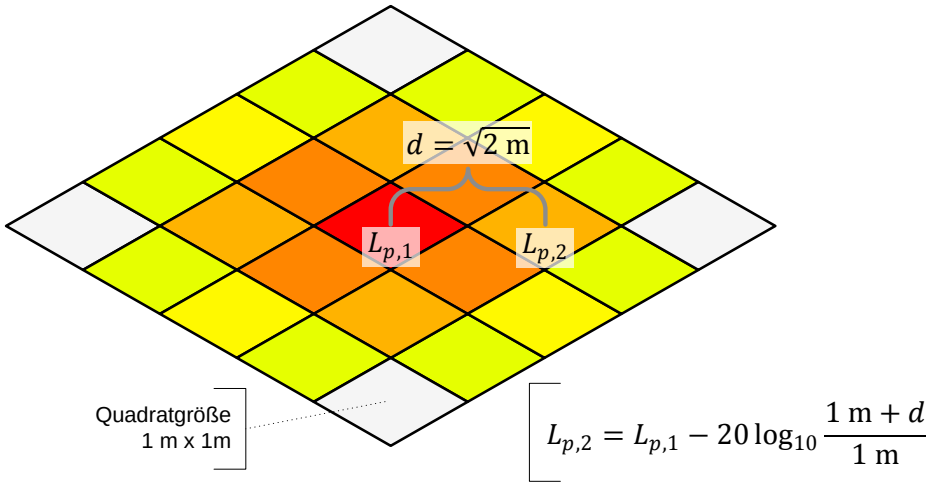


Abbildung 14: Beispielhafte Berechnung der Lärmabschwächung in der Nachbarschaft einer Lärmkartierung. Annahme für Messungen/Geräuschwahrnehmung in je 1 m Entfernung
(Eigene Darstellung)

Weil eine gleichmäßige Ausbreitung des Schalls in alle Richtungen angenommen wird, herrscht für ein hierbei entstehendes Lärmabschwächungsraster eine acht Wege Symmetrie (Abbildung 15). Dadurch können ausgehend von einer Rasterzelle sieben weitere ohne erneuten arithmetischen Aufwand gesetzt werden, d. h. es müssen 87,5 % weniger Lärmwerte interpoliert werden.

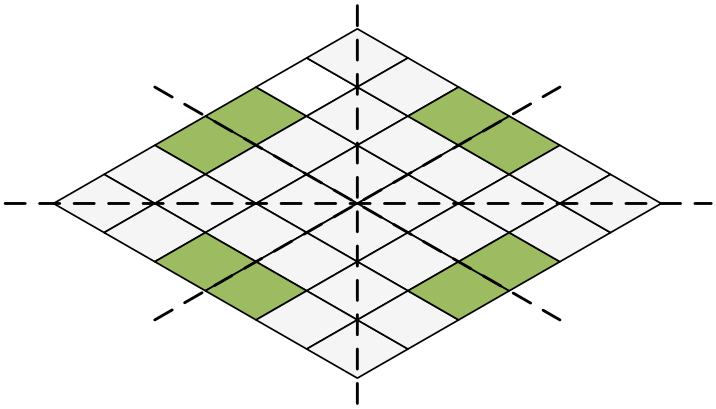


Abbildung 15: Acht Wege Symmetrie in der Lärmabschwächungsmodellierung.
(Eigene Darstellung)

Darüber hinaus kann das nötige Speichervolumen zur Beschreibung einer solchen Lärmabschwächung für einen Schallwert auf ein Viertel der trivialen Größe reduziert werden, wenn jeweils nur ein Quadrant des entstehenden Rasters gespeichert wird und dieses entsprechend der Symmetrieeigenschaften bei weiterer Verwendung gespiegelt angewandt wird.

4.7 Zusammenfügen der Interpolation

Um die abschließende Interpolation basierend auf den gemittelten Immissionswerten zu erstellen, müssen die modellierten Schallausbreitungen aller einzelnen Cluster in ein gemeinsames Raster überführt werden. Bei der Konzeption dieses Schrittes müssen insbesondere die Eigenschaften der Akustik berücksichtigt werden, da die möglichen Überlappungen der modellierten Schallausbreitungen gesondert zu behandeln sind.

Zum einen treten bei diesem Schritt Überschneidungen der Ausbreitungen innerhalb der Cluster hervor, da die zuvor durchzuführende Positionsermittlung eine Menge von Standpunkten ermittelt, an denen die tatsächlich von den Nutzern gemessene Immission vorzufinden ist. Zum anderen können hierbei auch Überlappungen zwischen den Ausbreitungen verschiedener Clustern auftreten. Wenn es sich hierbei gesichert um unabhängige Lärmereignisse handelt, müssen die Schallpegel unter Annahme inkohärenter Schallquellen im Überlappungsbereich energetisch addiert werden (vgl. Möser 2009, 62). Da aufgrund der Datengrundlage dies das Maß gesicherter Information überschreitet, wird im Fall der Überschneidung von Ausbreitungen lediglich eine Maximumsfunktion angewandt. Existiert also

beim Zusammenfügen für eine Zelle bereits eine berechnete Immission, muss aufgrund des mangelnden Informationsgehalts der Ausgangsdaten diese durch die neue ersetzt werden, falls sie einen höheren Schallwert beschreibt.

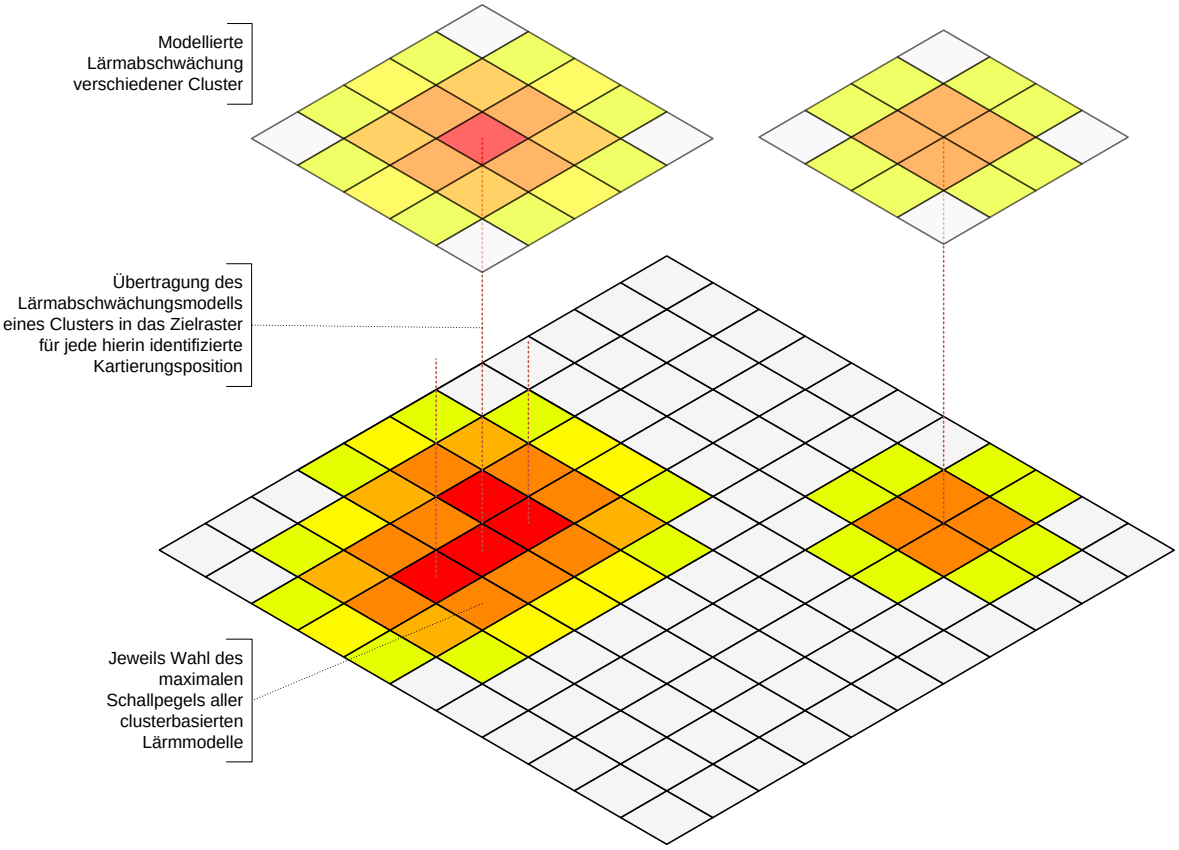


Abbildung 16: Zusammenfügen der Lärmabschwächungs- und Positionsbestimmung
(Eigene Darstellung)

Entsprechend dieser Regel kann schrittweise für jedes Cluster die ermittelte Lärmausbreitung elementweise ausgehend von jeder als Immissionsort identifizierten Rasterzelle in das Zielraster kopiert werden um die abschließende Interpolation für einen definierten Ausschnitt zu erhalten.

Vor allem bei Bereichen, die sich aus vielen Lärmkartierungen oder Lärmkartierungen mit ungenauen Positionsangaben zusammensetzen, werden im Positionsbestimmungs-Teilschritt eine Vielzahl an finalen Rasterzellen identifiziert, für die in diesem Schritt jeweils die modellierte Lärmausbreitung übertragen werden muss. Um dabei Redundanzen zu reduzieren oder zu vermeiden, können zuvor die Nachbarschaftsverhältnisse der ermittelten Punkte analysiert werden und hiervon abgeleitet nur die jeweils nötigen Quadranten der Lärmausbreitung pro Rasterzelle übertragen werden. Dazu wird der in der Positionsbestimmung (4.5) erstellte Akkumulator nicht direkt nach Zellwerten oberhalb eines Schwellwertes durchsucht, sondern zeilenweise in eine Indikatormatrix überführt, die anzeigt, ob

eine Zelle den Schwellwert überschreitet. Hier müssen zur Ermittlung einer 3 × 3 Nachbarschaft nur jeweils 3 aufeinanderfolgende Zeilen dieser Matrix zwischengespeichert werden (siehe Abbildung 17) und beim Überführen in die Indikatoren aus der jeweils zuvor bearbeiteten Zeile die Nachbarschaftsverhältnisse abgelesen werden.

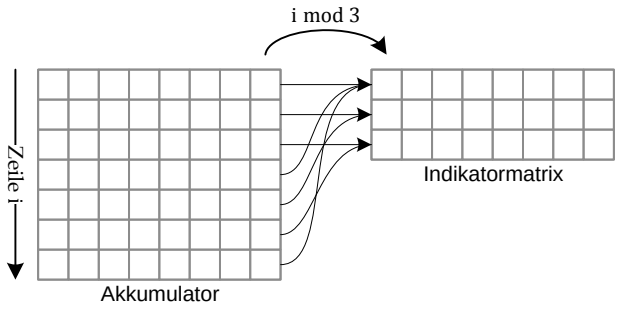


Abbildung 17: Ermittlung der Nachbarschaftsverhältnisse der bei der Positionsbestimmung identifizierten Zellen

(Eigene Darstellung)

4.8 Implementierungsentscheidungen

Diese Verfahrenskette muss mehreren Bedarfen zugleich genügen. Um überhaupt anwendbar zu sein, muss sie vor allem mit dem Speicher auskommen, der innerhalb des ausführenden Prozesses in der Android Architektur zur Verfügung steht. Hierbei sollte der Verbrauch so gering wie möglich gehalten werden, damit das Verfahren auch im Kontext komplexerer Anwendungen implementiert werden kann. Gleichzeitig müssen die verwendeten Datenstrukturen eine ausreichend große Interpolationsgenauigkeit erlauben. Darüber hinaus muss die Ausführungszeit aus Nutzersicht möglichst gering gehalten werden, um ein gutes Ansprechverhalten der mobilen Applikation zu ermöglichen. Hierzu zählen auch die nötigen Zugriffszeiten bei der weiteren Verwendung der Interpolationsdaten, etwa für eine Visualisierung.

4.8.1 Datentyp

Der nötige Wertebereich zum Abbilden von Schalldruckpegeln wird durch die Wahrnehmungsgrenzen des menschlichen Ohres bestimmt. Er erstreckt sich von der Hörgrenze mit ca. 10^{-5} Pa bis zur Schmerzgrenze mit einem Schalldruck von ca. 20 Pa (vgl. Lerch, Sessler und Wolf 2008, 9). Dies entspricht für den Schalldruckpegel L_p nach Gleichung 2 einem Wertebereich von ca. 0 dB bis ca. 120 dB.

Dieser Bereich lässt sich direkt mit dem primitiven Java Datentyp `byte` abbilden, der einen Werteberich von $[-2^7, 2^7 - 1]$ bzw. $[-128, 127]$ erlaubt, womit eine Speichergenauigkeit von ± 1 dB erzielt werden kann. Der Ressourcenverbrauch beträgt hiermit also 8 Bit bzw. 1 Byte pro interpoliertem Immissionswert.

Da in Java primitive Datentypen als Zweierkomplementdarstellung implementiert sind, kann bei Zugriff auf die `byte` Werte durch jeweiliges Überführen in einen größeren Datentyp unter Missachtung des Vorzeichenbits hiermit auch der Wertebereich $[0, 255]$ abgebildet werden (Listing 5). Die Genauigkeit kann also durch vorgehende Verdoppelung des Schallwertes auf $\pm 0{,}5$ dB erhöht werden, wobei der Berechnungsaufwand beim Zugriff auf die Daten leicht steigen kann, der Speicherbedarf aber kleinstmöglich bei 8 Bit pro Schallpegel bleibt.

```
// Vorliegender Lärmwert ([0,120]), Beispiel
float noise = 119.5f;
// Umwandlung in byte, Verdoppelung nutzt Wertebereich
// fast vollständig aus ([0,240]) -> Optimale Genauigkeit
byte noiseByte = (byte) (noise * 2);

// Zurückwandlung in ursprünglichen Wert, logische
// Verknüpfung maskiert Vorzeichenbit
noise = (noiseByte & 0xFF) / 2f;
```

Listing 5: Verwendung eines Java `byte` Typen ohne Vorzeichen

Der bei der Positionsbestimmung innerhalb der Cluster verwendete Akkumulator benötigt wegen des unvorhersehbaren Wertebereichs einen größeren Datentyp sowie Fließkommaarithmetik. Hierzu wird bei den folgenden Betrachtungen ein Java `float` Datentyp mit 4 Byte Verbrauch pro Wert angenommen.

4.8.2 Datenstruktur

Eine Maßgabe, die der Interpolationsprozess erfüllen soll, ist eine hohe Ausführungsgeschwindigkeit sowie eine einfache und schnelle Weiterverwendbarkeit des Ergebnisfeldes.

Sowohl für eine weitere Verwendung als Textur im OpenGL Kontext als auch für die direkte Anzeige in einer Applikation, zum Beispiel als Kartenoverlay, zeigt sich zum Zweck der schnellen und einfachen Weiterverwendung als optimale Datenstruktur die `android.graphics.Bitmap` Klasse. Diese stellt Bildinformationen mit bis zu vier Farbkanälen effizient dar, da sie direkt auf nativen Systembibliotheken fußt (siehe Abbildung 3) und für beide adressierten Darstellungskontexte unmittelbar verwendbar ist. Für eine solche `Bitmap` Klasse existieren vier verschiedene Konfigurationsmöglichkeiten in der Android API, welche in Tabelle 5 näher beschrieben werden.

Tabelle 5: Konfigurationsmöglichkeiten für die Instanziierung eines `android.graphics.Bitmap` Objektes über eine statische `createBitmap` Methodenüberladung

Bitmapkonfiguration	Beschreibung	Speicherbedarf pro Pixel
android.graphics.Bitmap.		
Config.ALPHA_8	Nur Transparenzkanal, 8 Bit Genauigkeit pro Pixel	1 Byte
Config.RGB_565	Drei Farbkanäle, Rot und Blau mit 5 Bit Genauigkeit, Grün mit 6 Bit Genauigkeit	2 Byte
Config.ARGB_4444	Drei Farbkanäle plus Transparenzkanal, jeweils 4 Bit Genauigkeit pro Pixel	2 Byte
Config.ARGB_8888	Drei Farbkanäle plus Transparenzkanal, jeweils 8 Bit Genauigkeit pro Pixel	4 Byte

Da für eine spätere Darstellung ein Transparenzkanal unabdingbar ist, sowohl um Bereiche ohne Immissionsdaten auszublenden, als auch um im Sinne der Erweiterten Realität die Daten semitransparent überblenden zu können, stehen als Struktur für die finale Interpolation nur die Konfigurationen `ARGB_4444` und `ARGB_8888` zur näheren Betrachtung zur Verfügung.

Mit der geringen Wortbreite von nur 4 Bit (16 mögliche Werte) pro Kanal kann mit der `ARGB_4444` Konfiguration nur eine Interpolationsgenauigkeit von $\pm$7,5 dB erzielt werden. Da sie darüber hinaus in der API inzwischen wegen der geringen Qualität als „deprecated“ markiert wurde, ergibt sich als einzig mögliche Darstellungsstruktur bezüglich der Weiterverwendbarkeit zur Visualisierung eine `Bitmap` in `ARGB_8888` Konfiguration.

Theoretisch besteht ferner die Möglichkeit, die Visualisierung nicht in einem individuellen `Bitmap` Objekt zu erzeugen und zur Anzeige zu speichern, sondern sie direkt innerhalb der Zeichenroutinen der Android Architektur mittels bereitgestellter Bildoperationen auf dem Display erscheinen zu lassen. Es bietet sich dabei an, die modellierten Lärmwerte in einer `ALPHA_8` konfigurierten `Bitmap` zu speichern, welche den geringstmöglichen Ressourcenbedarf darstellt. Ausgehend hiervon könnte dieses `Bitmap` mittels einer Farbtransformation mit den von den Darstellungsroutinen bereitgestellten Zeichenmethoden direkt zur Anzeige gebracht werden, ohne hierbei ein zusätzliches `Bitmap`, welches den zuvor beschriebenen Anforderungen genügt, allozieren zu müssen. Dabei wird beim Zeichnen des Bitmap Objektes die Farbe eines jeden Pixels entsprechend der Matrix nach Gleichung 5 transformiert.

$$\underbrace{\begin{pmatrix} a & b & c & d & e \\ f & g & h & i & j \\ k & l & m & n & o \\ p & q & r & s & t \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{Farbmatrix}} \cdot \underbrace{\begin{pmatrix} R_s \\ G_s \\ B_s \\ A_s \\ 1 \end{pmatrix}}_{\text{Eingangsfarbe}} = \begin{pmatrix} a\,R_s + b\,G_s + c\,B_s + d\,A_s + e \\ f\,R_s + g\,G_s + h\,B_s + i\,A_s + j \\ k\,R_s + l\,G_s + m\,B_s + n\,A_s + o \\ p\,R_s + q\,G_s + r\,B_s + s\,A_s + t \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} R_d \\ G_d \\ B_d \\ A_d \\ 1 \end{pmatrix}}_{\text{Ausgangsfarbe}}$$

Gleichung 5: Pixelweise Farbtransformation wie in der `ColorMatrix` Klasse der Android API beschrieben

Die beispielhafte praktische Anwendung dieses Verwendungsschemas innerhalb einer Android Zeichenroutine zeigt Listing 6.

Bei zusätzlicher Verwendung der Interpolationsresultate für Zwecke, die über die direkte Anzeige hinausgehen, muss allgemein beachtet werden, dass der Zugriff auf die in einem `Bitmap`-Objekt dargestellten Werte nur um ein Vielfaches langsamer möglich ist als beispielsweise der direkte Zugriff auf ein entsprechendes Java `byte` Array der Immissionsinterpolationen.

Die verschiedenen hiernach als sinnvoll angesehenen Verwendungsszenarien der einzelnen Datenstrukturen für das Interpolationsverfahren fasst Tabelle 6 zusammen.

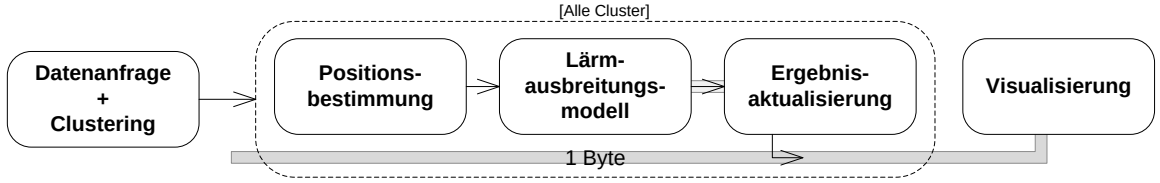
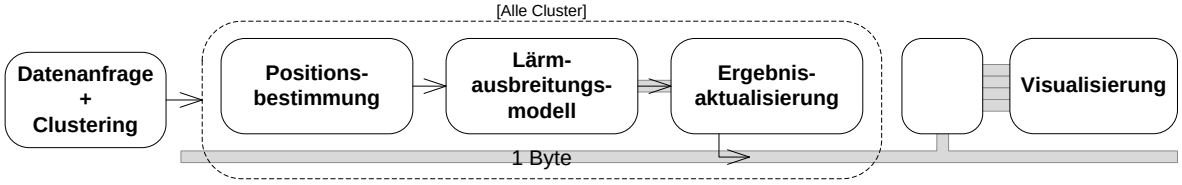
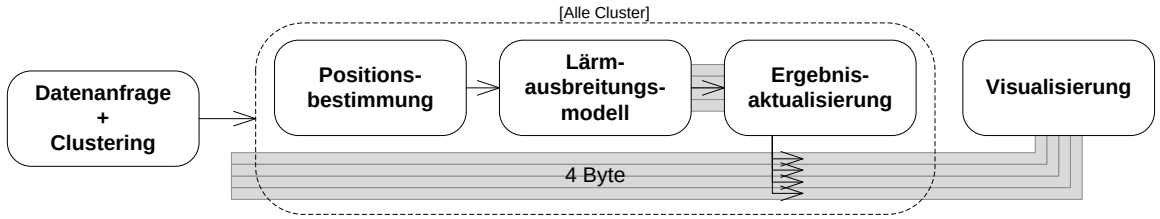
```
// Annahme, dass interpolierte Immission in einer ALPHA_8
// konfigurierten Bitmap „alpha_8_noise_bitmap“ mit
// Wertebereich [0,255] vorliegt.

// row-major linearisierte Matrix
float[] matrix = new float[] {
      0, 0, 0, 1, 0,
      0, 0, 0, -1, 255,
      0, 0, 0, 0, 0,
      0, 0, 0, 0, 255 };

// Klassen aus android.graphics Package

// Sollte einmalig instanziiert werden
Paint noiseGradientPaint = new Paint();
noiseGradientPaint.setColorFilter(
      new ColorMatrixColorFilter(matrix));

// canvas wird von Zeichenroutinen zur Anzeigemanipulation
// zur Verfügung gestellt
canvas.drawBitmap(alpha_8_noise_bitmap, 0, 0, noiseGradientPaint);
```

$$\underbrace{\begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 255 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 255 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}}_{\texttt{matrix}} \Rightarrow \begin{aligned} R_d &= A_s \\ G_d &= 255 - A_s \\ B_d &= 0 \\ A_d &= 255 \end{aligned}$$

Wobei A_s den in [0,255] skalierten Schalldruckpegel (L_p [dB]) darstellt

Listing 6: Exemplarische Verwendung einer Farbmatrix zur Transformation eines Alphakanals in einen Rot-Grün Farbverlauf

Tabelle 6: Verwendungsszenarien der hervorgehobenen Datenstrukturen innerhalb des Interpolationsverfahrens

Szenario	Beschreibung
8 Bit Bitmap, Transformation mit Farbmatrix	 - Verwendung einer 8 Bit Bitmap zur Speicherung der Interpolation - Darstellung als Farbverlauf durch Transformation mittels Farbmatrix während des Anzeigevorgangs - Transparenzkanal durch Maskierungsoperationen während des Anzeigevorgangs
8 Bit Interpolationsdaten + 32 Bit Bitmap	 - Bereitstellung der Immissionsmodellierung als `byte` Array - Zusätzliche Bereitstellung einer 32 Bit Bitmap zur Visualisierung - Einmalige Überführung der Modellierung in die Visualisierung - Direkte Nutzung der Interpolationswerte möglich
32 Bit Bitmap, Direktzugriff	 - Direkte Erstellung der Visualisierung innerhalb einer 32 Bit Bitmap Struktur - Keine Ablegung der unverarbeiteten Immissionswerte, nur der Bilddaten zur Visualisierung

4.8.3 Speicherressourcen

Dieses Interpolationsverfahren ist vor allem bezüglich der Speicherressourcen, die die Android Plattform zur Verfügung stellt, beschränkt. Die Virtuelle Maschine erlaubt es hier einer Anwendung als mindeste Obergrenze 16 MB Systemspeicher zu belegen. Falls ein Android

System über eine hochauflösende Anzeige verfügt, wird dieses Speicherlimit auf mindestens 24 MB erhöht (Google Inc. 2010, 9), da auch die Ressourcen für `Bitmap` Objekte, bzw. Rastergrafiken mit diesem Limit zusammengefasst werden, obwohl sie außerhalb des Java Heap verwaltet werden. Somit sollte unabhängig von den Displaygrößen ausreichend Speicher für unter Umständen größer dimensionierte Grafiken auf einem androidfähigem Gerät bereitgestellt sein.

In diesem Abschnitt werden die zuvor identifizierten Datenstrukturen bezüglich ihres Speicherressourcenbedarfs während des Interpolationsprozesses analysiert. Es werden die bei den entwickelten Anwendungsszenarien (siehe Tabelle 6) jeweils hinsichtlich verschiedener Modellierungsparameter zu allozierenden Speicherausmaße in theoretischer Weise approximativ erfasst und diskutiert. Hierbei sollen sich Einschränkungen und Bedingungen ergeben, die einen Einsatz des Verfahrens in der Android Architektur sicherstellen. Für diese Abschätzungen gilt:

- Es wird die Wortbreite an Elementen summiert, die in Arrays gespeichert werden, es wird also der Heap-Verbrauch approximiert
- Es wird angenommen, dass die für die Positionsbestimmung aufzuwendenden Speicherressourcen für das Lärmausbreitungsmodell wieder zur Verfügung stehen

4.8.3.1 Parameterabschätzung

Während der Bedarf zur Speicherung der Resultate in jedem Szenario konstant von der angeforderten Größe abhängt, sind die für die Positions- und Ausbreitungsbestimmung nacheinander benötigen Ressourcen von diversen Modellierungsparametern innerhalb des Verfahrens abhängig. Um die verschiedenen Strukturen bezüglich ihrer Eignung bewerten zu können, müssen zunächst die Modellierungsparameter, die in den verschiedenen Schritten Anwendung finden, abgegrenzt werden. Dazu werden die Extremfälle dieser Schritte hinsichtlich des Speicherverbrauchs bestimmt und die Parameter hierauf basierend angepasst. Für diese Betrachtung wird allgemein eine maximale Interpolationsgenauigkeit von 1 m pro Pixel angenommen.

Bezüglich der Lärmausbreitungsmodellierung zeigt Abbildung 18 die allgemeine Abhängigkeit von der zu modellierenden Lärmpegeldifferenz bzw. des räumlichen Ausbreitungsradius. Dabei wird jeweils von einem Ausgangspegel bis auf das Niveau von 40 dB, was einem Gespräch in 50 m Entfernung entspricht (vgl. Möser 2009, 3), der Radius der Ausbreitung sowie der nötige Speicher zum Abbilden der Abschwächungswerte bei 8 Bit pro Wert angegeben. Diese Zusammenhänge ergeben sich aus Gleichung 4 und werden in Gleichung 6

zusammengefasst. Aufgrund der logarithmischen Skalierung der Schallpegeleinheit nehmen diese Maße exponentiell zu. Es zeigt sich, dass bis zu einer Berechnung für eine Pegeldifferenz von 65 dB der nötige Speicherbedarf mit ungefähr 3000 KiB bei 1 Byte pro Lärmwert noch tragbar ist (Abbildung 18, links). Bei einer Differenz von 70 dB verdreifacht sich dieser Bedarf bereits und würde mindestens 60 % des minimal verfügbaren Speichers ausnutzen.

Auf der rechten Seite der Abbildung 18 sind die Kennlinien für hierauf angepasste Parameter bei einer Datenstruktur mit 8 Bit pro Lärmwert zu sehen. Hier wird die Lärmabschwächung ausgehend von einem Schallpegel bis zum halben Schallpegelwert berechnet. Somit liegt die maximal zugrundeliegende Pegeldifferenz bei 60 dB, was einem maximalen Ausbreitungsradius von 1000 m bei den nach Gleichung 4 definierten Zusammenhängen ergibt.

$$S = \left(\frac{1}{\lambda} \cdot 10^{\frac{\Delta L_p}{20}}\right)^2 \cdot b \qquad \text{[Byte]}$$

mit ΔL_p Schalldruckpegeldifferenz [dB]

$\lambda > 0$ Interpolationsauflösung $\left[\frac{\text{Pixel}}{\text{m}}\right]$

b Wortbreite pro Pixel [Byte]

Gleichung 6: Approximierter Speicherbedarf der Lärmausbreitung eines Clusters

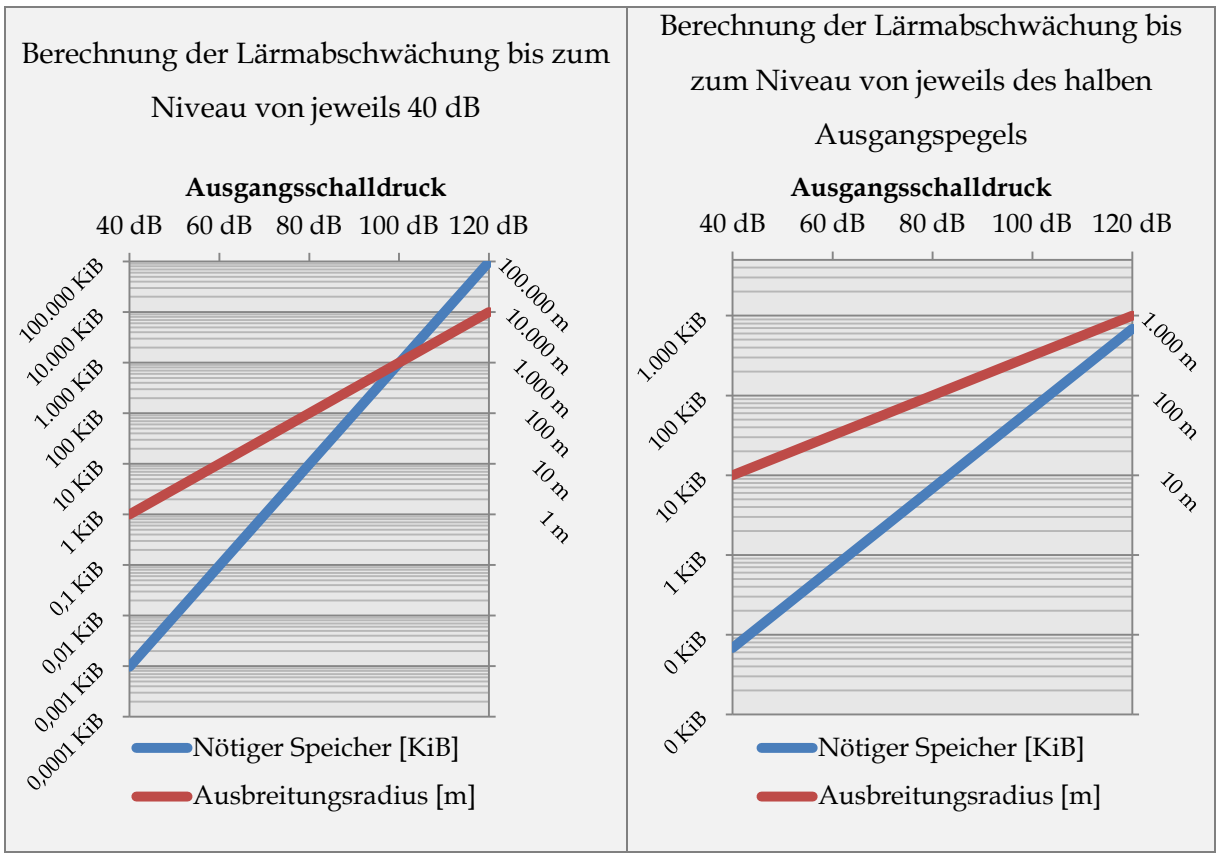


Abbildung 18: Ressourcenbedarf des Lärmabschwächungsmodells in Abhängigkeit des Ausbreitungsradius, bei einer Interpolationsauflösung von 1 m pro Pixel

Die Positionsbestimmung wird vor allem durch die Ausmaße des verwendeten Akkumulators limitiert. Dessen maximale Größe wird durch die maximal mögliche Clusterausbreitung definiert. Diese hängt zum einen von dem in der Clusteranalyse verwendeten Schwellwert des Zuordnungsschrittes, aber auch von den individuellen Genauigkeitsangaben der Positionsbestimmung der einzelnen Lärmkartierungen ab. Ein angemessener Schwellwert hängt von den tatsächlich verfügbaren Daten ab, da er die räumliche Variation und Unsicherheit in den Daten kompensiert. Diese sind abhängig von den in der jeweiligen Lärmcommunity verwendeten Techniken. Um eine maximale Speicherbelastung festlegen zu können, muss ein weiterer Schwellwert eingeführt werden, der anhand des Genauigkeitsradius einer Lärmkartierungsposition bestimmt, ob eine Messung überhaupt in diesem Verfahren verwendet werden soll. Hierdurch wird auch die Robustheit der Prozesse verbessert.

Es ergibt sich für den Ressourcenbedarf Gleichung 7 unter der Verwendung aller Optimierungen. Um auch hier ein Maximum von ungefähr 3000 KiB nicht zu überschreiten gilt folglich:

$$T_{\text{Clusteringdistanz}} + T_{\text{Genauigkeitsradius}} \leq 440\ \text{m}$$

Ansprechende Ergebnisse wurden für $T_{\text{Clusteringdistanz}}$ von 25 m und $T_{\text{Genauigkeitsradius}}$ von 200 m erzielt.

$$S = \underbrace{\left(\frac{2}{\lambda} \cdot \left(T_{\text{Clusteringdistanz}} + T_{\text{Genauigkeitsradius}}\right)\right)^2}_{\texttt{float[]}\ \text{Akkumulator}} \cdot 4\ \text{Byte} + \underbrace{\frac{2}{\lambda}\left(T_{\text{Clusteringdistanz}} + T_{\text{Genauigkeitsradius}}\right)}_{\text{Cache zur Nachbarschaftsermittlung}\ (\textit{siehe}\ \text{Abbildung 17})} \cdot \frac{3}{8}\ \text{Byte} \qquad [\text{Byte}]$$

mit $T_{\text{Clusteringdistanz}}$ beim Zuordnen der Cluster verwendeter Schwellwert [m]

$T_{\text{Genauigkeitsradius}}$ maximal mögliche Genauigkeitsangabe der Positionsbestimmung einer Lärmkartierung [m]

$\lambda > 0$ Interpolationsauflösung $\left[\frac{\text{Pixel}}{\text{m}}\right]$

Gleichung 7: Approximation des bei der Positionsbestimmung nötigen Speichers

4.8.3.2 *Gesamtabschätzung*

Um die einzelnen näher betrachteten Implementierungsmöglichkeiten vergleichen zu können, wurde ihr jeweiliger maximaler Speicherbedarf entsprechend Tabelle 7 summiert. Diese Auflistung berücksichtigt dabei die zu verwendenden Datentypen. Die als nicht kritisch zu bewertende Positionsbestimmung wurde in vereinfachter Weise bemessen.

Am praktischen Beispiel einer sphärischen Mercator Projektion wie sie beispielweise mit der Kartenvisualisierung der `android.maps.MapView` Klasse der Android Google Maps API verwendet wird, stellt Abbildung 19 den ungefähren Gesamtbedarf der verschiedenen Implementierungen dar. Dabei wird die Interpolation für ein 940 × 780 Pixel großes Gebiet simuliert. Dies entspricht der typischen Displayauflösung eines androidfähigen Gerätes von 640 × 480 Pixeln (vgl. Google Inc. 2011g) mit einem zusätzlichem Puffer von 150 Pixeln.

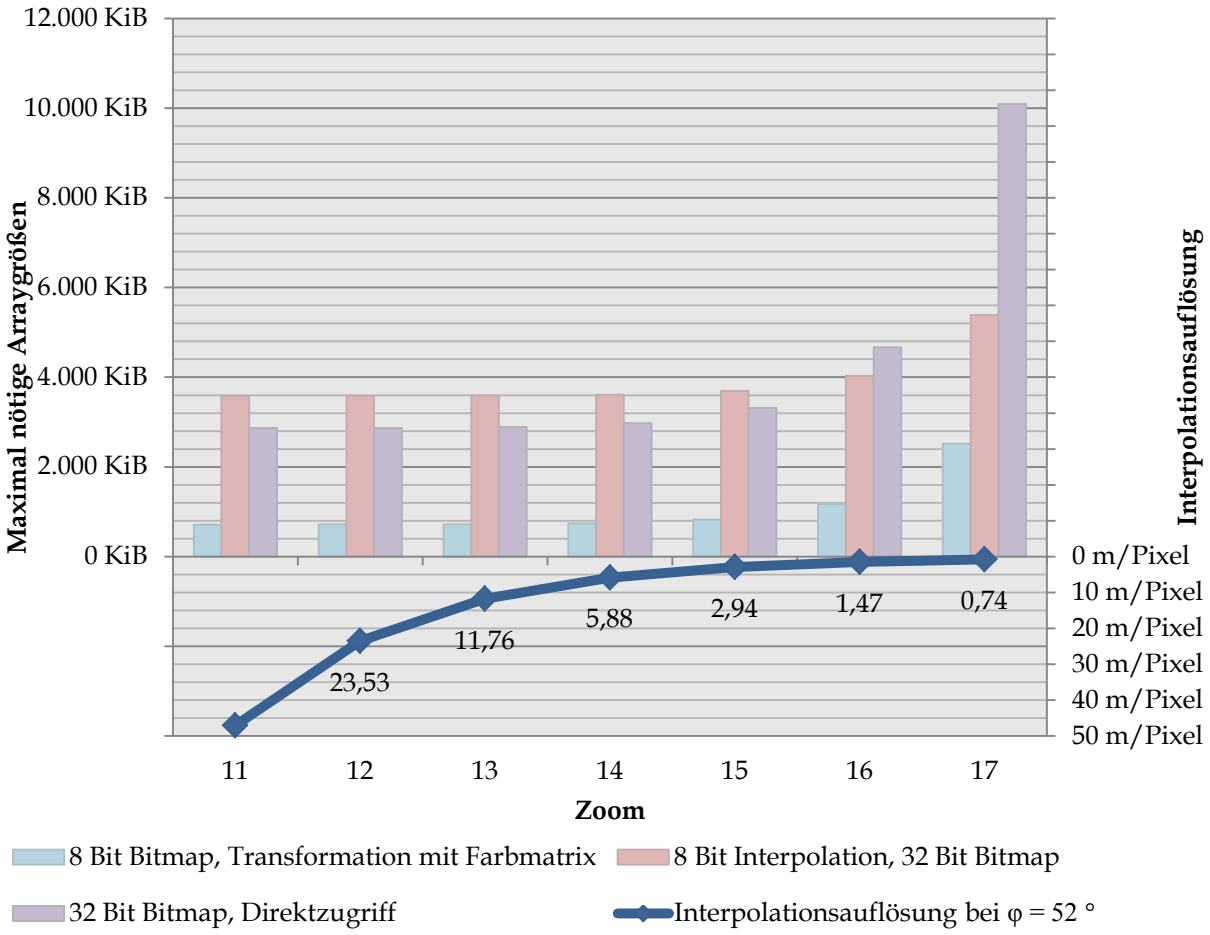
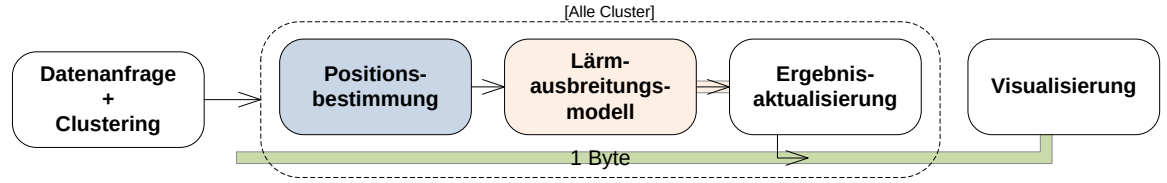
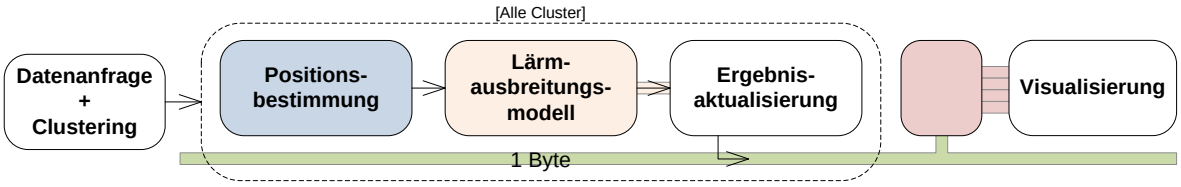
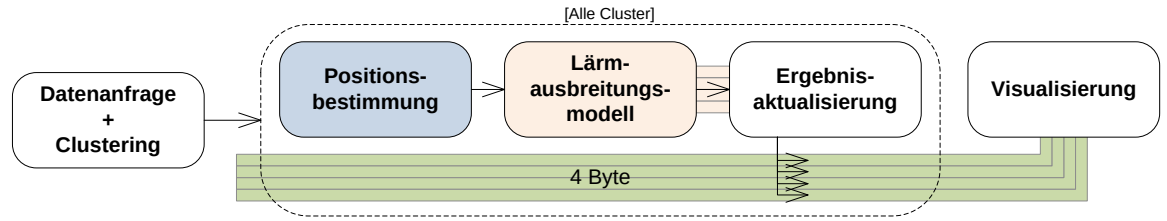


Abbildung 19: Gesamtabschätzung des Speicherressourcenbedarfs der verschiedenen Implementierungsweisen am Beispiel einer sphärischen Mercator Projektion als Rasterisierung

Tabelle 7: Abschätzung des Speicherbedarfs verschiedener Implementierungsweisen des Interpolationsprozesses

Szenario	Ungefährer Maximaler Speicherbedarf
8 Bit Bitmap, Transformation mit Farbmatrix	 $\underbrace{A_{\text{Interpolation}} \cdot 1\ \text{Byte}}_{\text{8 Bit Bitmap}} + \max\left(\underbrace{A_{\text{Max Clustergröße}} \cdot 4\ \text{Byte}}_{\text{Positionsbestimmung}}, \underbrace{\frac{A_{\text{Max Ausbreitung}}}{4} \cdot 1\ \text{Byte}}_{\text{Lärmausbreitung}}\right)$
8 Bit Interpolationsdaten + 32 Bit Bitmap	 $\underbrace{A_{\text{Interpolation}} \cdot 1\ \text{Byte}}_{\text{8 Bit Array (byte[])}} + \max\left(\underbrace{A_{\text{Max Clustergröße}} \cdot 4\ \text{Byte}}_{\text{Positionsbestimmung}}, \underbrace{\frac{A_{\text{Max Ausbreitung}}}{4} \cdot 1\ \text{Byte}}_{\text{Lärmausbreitung}}\right)$ $+ \underbrace{A_{\text{Interpolation}} \cdot 4\ \text{Byte}}_{\text{32 Bit Bitmap}}$
32 Bit Bitmap, Direktzugriff	 $\underbrace{A_{\text{Interpolation}} \cdot 4\ \text{Byte}}_{\text{32 Bit Bitmap}} + \max\left(\underbrace{A_{\text{Max Clustergröße}} \cdot 4\ \text{Byte}}_{\text{Positionsbestimmung}}, \underbrace{\frac{A_{\text{Max Ausbreitung}}}{4} \cdot 4\ \text{Byte}}_{\text{Lärmausbreitung}}\right)$ Bitmap benötigt 4 Byte Zugriff

mit

$$
\begin{aligned}
n &\mathrel{\widehat{=}} \text{Anzahl Cluster} \\
A_{\text{Cluster},i} &= \text{Fläche Cluster } i \text{ entsprechend Raster} \\
A_{\text{Max Clustergröße}} &= \max\left(A_{\text{Cluster},1}, \dots, A_{\text{Cluster},n}\right) \\
A_{\text{Max Ausbreitung}} &= \max\left(A_{\text{Ausbreitung},1}, \dots, A_{\text{Ausbreitung},n}\right) \\
A_{\text{Interpolation}} &= \text{Interpolationsfläche}
\end{aligned}
$$

4.8.4 Bewertung

Zur abschließenden Bewertung muss zwischen der möglichen Leistung, auch bezogen auf weitere Verwendungen in der Visualisierung, und dem Ressourcenbedarf abgewogen werden. Tabelle 8 fasst dazu die Implementierungsszenarien nach diesen Aspekten zusammen.

Am attraktivsten wirkt die Verwendung einer 8 Bit Bitmap, da sie wegen des geringen Speicherbedarfs innerhalb einer Applikation die Möglichkeiten offen hält, komplexere und umfangreiche Methoden zur Weiterverarbeitung der Interpolation zu entwickeln. Die zu erwartenden Latenzzeiten beim Zugriff auf die gespeicherten Daten wegen der Weiterdelegierung eines Aufrufs der virtuellen Java Maschine an native Methoden des zugrundeliegenden Systems könnten darüber hinaus wegen des geringen Speicherbedarfs durch zusammengefasste Datenmanipulationen kompensiert werden.

Aufgrund unvollendeter Implementierung[4] der hier vorgeschlagenen 8 Bit `Bitmap` Konfiguration im Android Framework, kann diese „on-the-fly" Darstellung der Interpolation derzeit noch nicht angewandt werden, weshalb die Variante der Datenerzeugung in einem kleinstmöglichem Array, welches anschließend zur Visualisierung in eine entsprechende Bitmap übertragen wird, zu bevorzugen ist.

Tabelle 8: Zusammenfassung der Implementationsbewertung

Szenario		Speicherbedarf	Geschwindigkeit		
			Interpolation	Visualisierung	Weitergehender Zugriff
8 Bit Bitmap		Sehr gering	Wegen nativen Zugriffs langsam	Sehr schnell	Wegen nativen Zugriffs langsam
32 Bit Bitmap	Datenerzeugung direkt in der Bitmap	Hoch, da Daten in 4 Byte Länge nötig	Wegen nativen Zugriffs langsam	Sehr schnell	Wegen nativen Zugriffs langsam, Keine Unverarbeiteten Lärmdaten
	Datenerzeugung in zusätzlichem Array, nachträgliche Überführung in Bitmap	Mittelhoch, da Zwischenresultate nicht in 4 Byte Länge vorgehalten werden müssen	Schnell	Sehr schnell Vorige Überführung in Bitmap nötig	Sehr schnell

[4] http://code.google.com/p/android/issues/detail?id=18877

5 Lärmvisualisierung in einer Augmented Reality Umgebung

Mit der Definition eines einheitlichen virtuellen Koordinatensystems und der Erarbeitung eines sensorbasierten Trackingverfahrens zur Transformation zwischen dem virtuellen und realen Raum, wurde der Rahmen zur Visualisierung von Daten ubiquitärer Phänomene in einer Augmented Reality Umgebung auf der Android Plattform gesichert. Mit den entwickelten Methoden zur Interpolation nutzerbasierter Schallimmissionsdaten wurde darüber hinaus die Datenbasis geschaffen, auf der im Folgenden Möglichkeiten zur Visualisierung von Community-basierten Lärmdaten im Sinne der Erweiterten Realität entwickelt werden können.

Milgram unterscheidet allgemein zwischen optischen und videobasierten Augmented Reality Systemen (Milgram, Takemura, et al. 1994). Optische Systeme zeichnen sich dadurch aus, dass der Nutzer durch ein Sichtmedium direkt die reale Umgebung wahrnimmt, während dieses die Zusatzinformationen im Sichtfeld platziert. Diese Systeme werden zumeist über sogenannte „Head-Mounted Displays“ realisiert. Videobasierte Systeme kombinieren hingegen ein Abbild der realen Welt mit der virtuellen Information in einer einzelnen Ausgabe, die vom Nutzer betrachtet wird. Dies wird als „window-on-the-world […] Augmented Reality“ (Milgram, Takemura, et al. 1994) bezeichnet.

Ein androidfähiges mobiles Gerät kann hierbei als ein solches „window-on-the-world“ System verwendet werden, da laut Spezifikation ein solches System sowohl über ein Display als auch über eine nach hinten, d. h. entgegen der Displayvorderseite, gerichtete Kamera verfügen muss (Google Inc. 2010, 15). Zur Realisierung eines solchen Systems wird in den folgenden Abschnitten zunächst die Darstellung einer Kamerasicht mittels des Android Systems verdeutlicht und die dabei angewandten Transformationen beschrieben. Im Anschluss werden zwei Visulisierungsmethoden aufgezeigt, sowie die konkrete Überlagerung der virtuellen und realen Information dargestellt.

5.1 Darstellung der Realen Umgebung

Die reale Umgebung soll mittels der im mobilen Gerät verbauten Kamera auf dessen Anzeigedisplay in Echtzeit abgebildet werden. Dabei soll ihre Ausrichtung bezüglich der Gerätekoordinaten berücksichtigt werden, sowie ihre Projektionseigenschaften ermittelt werden.

Den Zugriff auf aktuelle Videodaten der integrierten Kamera ermöglicht die Android API über die `android.hardware.Camera` Klasse. Sie stellt Methoden bereit, die die Darstellung des aktuellen Kamerabildes als Vorschau in ein sogenanntes `Surface` übertragen und

somit direkt zur Displayanzeige bringen. Ein `Surface` ist die oberste Ebene der Android Zeichenhierarchie und verfügt entsprechend über direkten und schnellen Zugriff auf die Grafikhardware. Zur speziellen Verwendung eines solches `Surface` Objektes innerhalb der Android Anzeigehierarchie existiert die Hilfsklasse `android.view.SurfaceView`, die ein zugrundeliegendes `Surface` erstellt und verwaltet. Eine exemplarische Vorgehensweise zur derartigen Nutzung der Kamerahardware zeigt Abbildung 20.

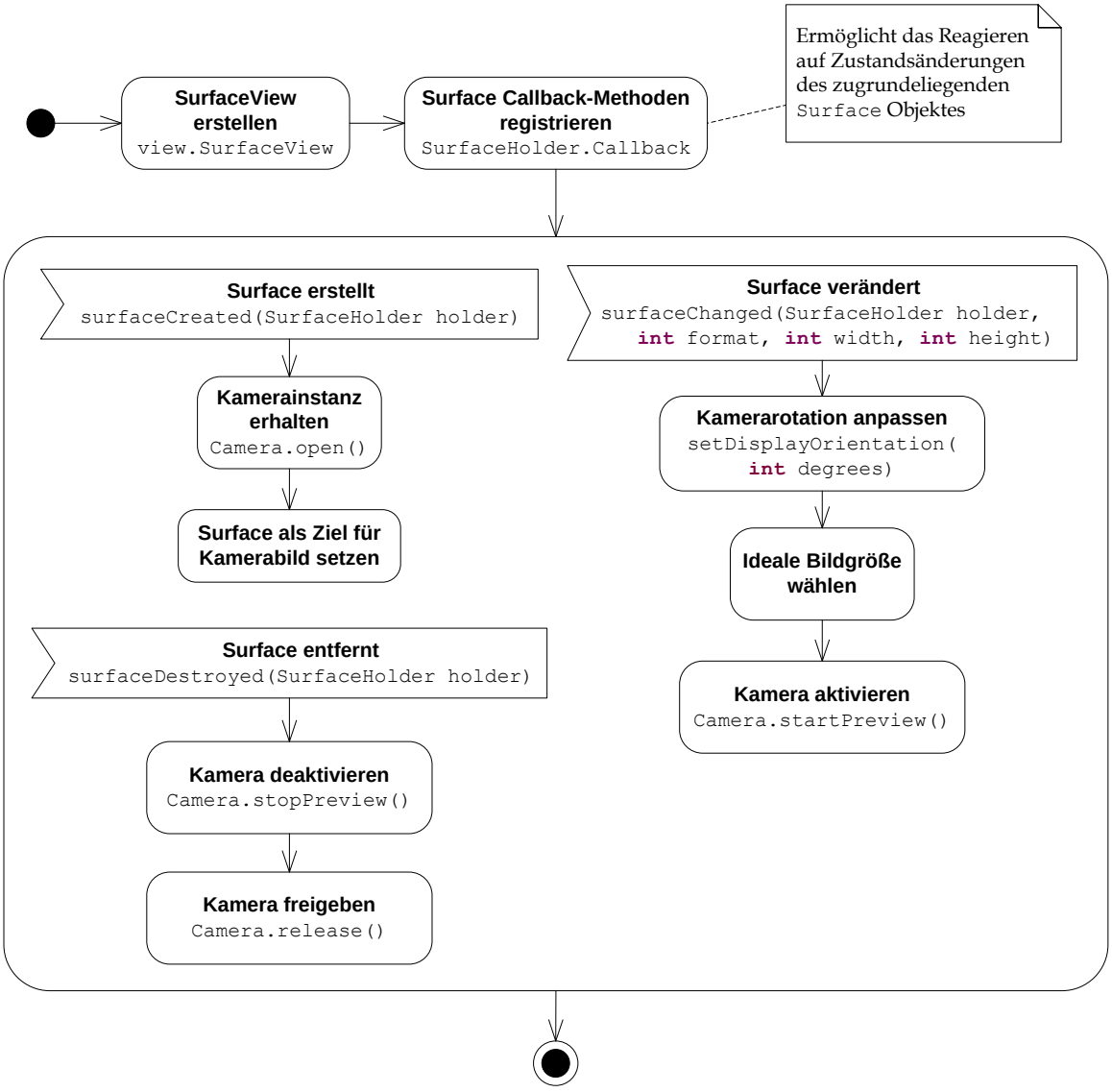


Abbildung 20: Aktivitätsdiagramm zur Verwendung einer `SurfaceView` zur Kameravorschau
(Eigene Darstellung, nach Google Inc. 2011a)

Es wird angenommen, dass die integrierte Kamera parallel zum Gerätekoordinatensystem aufnimmt. Ihre Position und Ausrichtung kann also durch dieselben Transformationen mit dem Weltkoordinatensystem in Bezug gebracht werden, wie sie in Abschnitt 3.2 bezüglich der Gerätekoordinaten dargestellt werden. Dabei handelt es sich also direkt um die soge-

nannten extrinsischen Kameraparameter, welche in das in Augmented Reality Systemen typische Kamerakoordinatensystem überführen (siehe 2.1.2).

Es muss dabei allerdings gegebenenfalls neben der physischen Rotation des mobilen Gerätes zusätzlich eine fixe Rotation der integrierten Kamera bezüglich der Standardausrichtung bzw. des Sensorkoordinatensystems einbezogen werden, um zu der Darstellung in Gerätekoordinaten zu finden, da die Kameraausrichtung in der Android Spezifikation nicht genau bestimmt ist. (Google Inc. 2011a) zeigt hierzu Programmausschnitte um diese Drehung bei der Anzeige zu kompensieren.

Zur Bestimmung der intrinsischen Parameter der verwendeten Kamera, d. h. ihrer grundlegenden Projektionseigenschaften der Abbildung auf die Bildebene, wurden mit Android 2.2 Methoden zur Abfrage des horizontalen und vertikalen Blickwinkels sowie ihrer Brennweite zugefügt (siehe Listing 7).

```
public float getHorizontalViewAngle()

public float getVerticalViewAngle()

public float getFocalLength()
```

Listing 7: Signaturen der Abfragemethoden der `Camera.Parameters` Klasse der Android Hardware API zum Auslesen intrinsischer Kameraparameter

5.2 Dreidimensionale Überlagerung der Schallimmissionsdaten

Für eine flächenhafte dreidimensionale Überlagerung der Lärminterpolation über das Kamerabild wird die OpenGL ES 1.1 Grafikschnittstelle der Android Plattform verwendet. Hierfür wird das Verfahren aus Kapitel 4 auf den direkten Bereich um die aktuelle geografische Position des Nutzers bzw. mobilen Gerätes (siehe 3.2.3) angewandt und das dabei erzeugte `Bitmap` Objekt direkt für die Darstellung im OpenGL ES Kontext verwendet. Durch eine genaue dreidimensionale Überlagerung dieses Interpolationsresultates mit dem Weltkoordinatensystem und einer halbtransparente Darstellung kann der Nutzer eine direkte Zuordnung zwischen Punkten der realen Welt und des dort wirkenden Schalls herstellen.

Zur einfachen Verwendung der Grafikbibliothek innerhalb der Anzeigehierarchie existiert in der Android API die `android.opengl.GLSurfaceView` Klasse. Sie ist von der zur Kameradarstellung verwendeten `SurfaceView` Klasse abgeleitet und verwaltet neben eines `Surface` Objektes zum nahezu direkten Zugriff auf den Bildspeicher auch die zum Erzeugen von Grafiken mittels der OpenGL Bibliotheken nötigen Ressourcen. Um mit der Grafik-

bibliothek zu interagieren, muss hier eine eigene Implementierung des `GLSurfaceView.Renderer` Interfaces der `GLSurfaceView` zugefügt werden.

Für das Zeichnen dreidimensionaler Objekte erwartet OpenGL als Eingabe eine Beschreibung ihrer Vertices. Diese werden strikt nach der „fixed function pipeline" zunächst in den zweidimensionalen Bildraum übertragen und danach auf Rasterbasis zur schlussendlichen Anzeige verarbeitet. Um eine korrekte referenzierte Überlagerung mit dem Kamerabild zu erhalten, müssen die sogenannten „Per-Vertex Operationen", also die Übertragungen in die Bildebene, auf gleiche Weise geschehen wie bei der integrierten Kamerahardware. Es soll realisiert werden, dass Objektbeschreibungen, die dreidimensional im Weltkoordinatensystem vorliegen, im Gerätekoordinatensystem dargestellt und entsprechend des Kamerabildes projiziert werden. Die genauen Zusammenhänge zwischen dem virtuellen Abbildungsprozess im OpenGL Kontext und dem realen in einer einfachen Kamerageometrie fasst (Ming 2001) detailliert zusammen.

5.2.1 Extrinsische Parameter

Abbildung 21 zeigt die Transformationssequenz wie sie auf jeden Eingabevertex der OpenGL Grafikpipeline angewandt wird. Im ersten Schritt dieser „Per-Vertex Operationen" werden die darzustellenden Objekte mit Hilfe der Model-View Matrix in ein Kamerakoordinatensystem überführt. Dieser Schritt gleicht dabei der Anwendung der extrinsischen Kameraparameter im Abbildungsprozess der realen Kamera, entsprechend muss hier die Transformation in die Gerätekoordinaten stattfinden.

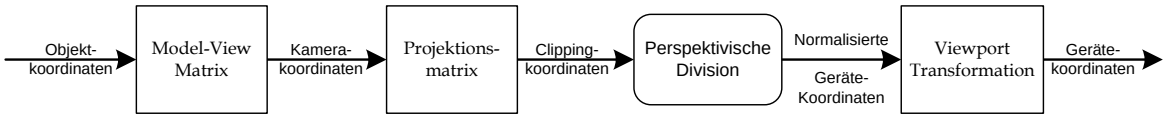


Abbildung 21: Transformationssequenz der OpenGL Fixed Function Grafikpipeline
(nach The Khronos Group Inc. 2008, 27)

Auf einen im Weltkoordinatensystem gegebenen Vektor muss folglich die Inversen der in Abschnitt 3.2 entwickelten Transformationen angewandt werden um einen Eingabevertex entsprechend des Gerätekoordinatensystems darzustellen. Listing 8 stellt dar, wie die Model-View Matrix entsprechend manipuliert werden kann.

```
// javax.microedition.khronos.opengles.GL10
gl.glMatrixMode(GL10.GL_MODELVIEW);

// Zusätzliche Verschiebung zur Simulation einer virtuellen
// erhobenen Kamera
gl.glTranslatex(0, -5, 0);

// Rotationsmatrix transformiert von Geräte- in Weltkoordinaten und
// liegt als row-major linearisiertes float[] Array vor.
// OpenGL interpretiert es als column-major, folglich als
// transponierte Matrix, folglich als Inverse und überführt
// von Welt- in Gerätekoordinaten
gl.glMultMatrixf(rotMatrix, 0);

// posX, posY Nutzerposition bezüglich gewählter Darstellung
gl.glTranslatex(-posX, 0, -posY);
```

Listing 8 Setzen der extrinsischen Parameter im OpenGL ES Kontext zur Überlagerung mit Kamerabild

5.2.2 Intrinsische Parameter

In der Projektionsmatrix, die in der OpenGL Transformationssequenz als nächstes angewandt wird, müssen die spezifischen Parameter der verwendeten integrierten Kamerahardware aufgehen. Nach (Ming 2001) handelt es sich bei der realen Kameraprojektion um eine perspektivische Projektion, die mit OpenGL direkt unter Missachtung möglicher Verzerrungseigenschaften nachempfunden werden kann. Dazu kann die in Listing 9 angegebene `gluPerspective` Methode der OpenGL Utility Library verwendet werden, mit der in der OpenGL Pipeline ein perspektivisches Sichtvolumen erzeugt wird. Der für eine genaue Überblendung nötige Parameter `fovy` entspricht hier dem aus dem `Camera` Objekt abrufbarem vertikalen Blickwinkel (siehe 5.1). Der Parameter `aspect` gibt das Seitenverhältnis der virtuellen Kameraprojektion an und muss dem Verhältnis des horizontalem zum vertikalen Blickwinkel entsprechen. Die letzten Parameter sind für die Korrespondenz der realen und virtuellen Darstellung unerheblich und beschrieben die Positionen der Clippingebenen zwischen denen OpenGL Objekte darstellen wird.

```
public static void gluPerspective (GL10 gl, float fovy,
     float aspect, float zNear, float zFar)
```

Listing 9: Signatur der gluPerspective Methode aus der `opengl.GLU` Klasse der Android API

Zur Übereinanderführung der Bildebenen müssen lediglich sowohl bei der Kameravorschau als auch der OpenGL Grafik dieselben Positionen und Ausmaße bezüglich der Einbettung in die Displayanzeige gewählt werden. Dazu muss die OpenGL Viewport Transformation in dieselbe Bildgröße überführen wie die des Kameravorschaubilds. Listing 10 zeigt die praktische Handhabung der intrinsischen Parameter.

```
// gl11 als opengles.GL11 Instanz

// Viewport Matrix
// Ausmaße entsprechend Vorschaubild der realen Kamera
gl11.glViewport(0, 0, NoiseCamera.glViewportWidth,
     NoiseCamera.glViewportHeight);

// Projektionsmatrix
gl11.glMatrixMode(GL11.GL_PROJECTION);
gl11.glLoadIdentity();
// Projektionsparameter aus Camera Objekt
GLU.gluPerspective(gl11, NoiseCamera.fovY, NoiseCamera.aspect,
     1, 100);
```

Listing 10: Setzen der intrinsischen Parameter im OpenGL ES Kontext zur Überlagerung mit Kamerabild

5.2.3 Darstellung

Durch das übereinstimmende Setzen der OpenGL Koordinatensysteme können nun Objektdefinitionen die in Weltkoordinaten gegeben sind auf direkte Weise im OpenGL Kontext gezeichnet werden. Ein einfaches mit dem Interpolationsergebnis texturiertes Quadrat reicht dabei aus, um eine räumliche Überlagerung der Lärmwerte, sowie eine Immersion des Nutzers zu erzeugen.

5.3 Zweidimensionale Überlagerung

Neben der Visualisierung der räumlichen Ausbreitung der Immission durch exakte dreidimensionale Überlagerung mit dem Kamerabild, kann auch eine lediglich richtungsabhängige Visualisierung und Quantifizierung der Schallimmission bestimmt werden. Ziel dieser Darstellungsweise ist es, dem Nutzer einen quantitativen Überblick über die gemessenen Schallimmissionen in seiner direkten Umgebung bzw. Sichtrichtung zu geben.

Dazu soll der auf dem Display sichtbare Ausschnitt der realen Umgebung in gleichmäßig verteilte Streifen eingeteilt werden, die jeweils ein Gebiet im virtuellen Weltkoordinatensystem abbilden, für welches die interpolierte Lärmimmission gemittelt werden kann (Abbildung 22). Die so für jeden Streifen der aktuellen Displaydarstellung ermittelten Lärmwerte können dem Nutzer anschließend in verschiedensten Formen angezeigt werden, beispielsweise als überlagerte Balkendiagramme oder als konkrete Zahlenwerte für jeden Streifen. So wird die reale Umgebung mit einer unmittelbaren Quantifizierung des im Blickfeld wirkenden Schalls erweitert.

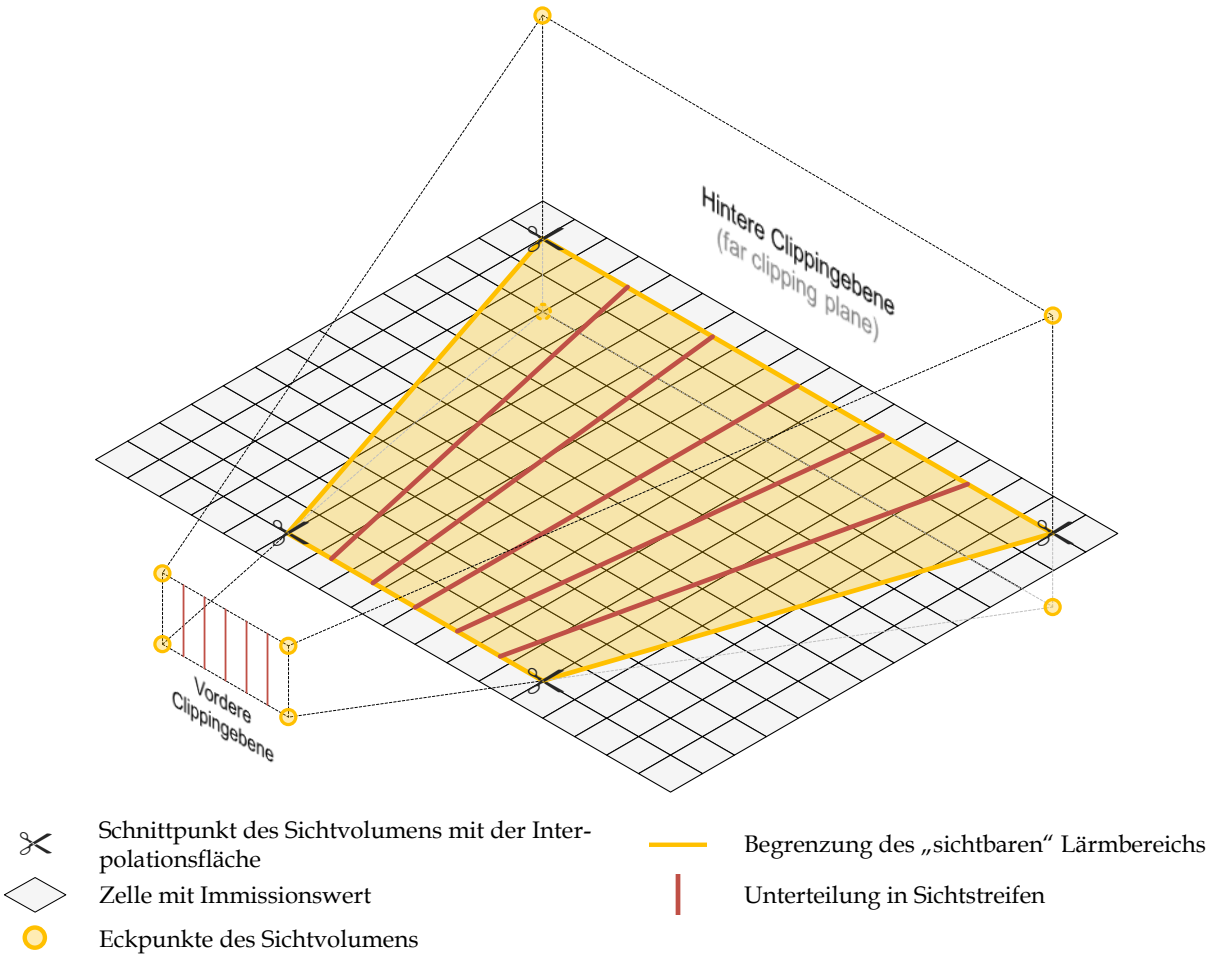


Abbildung 22: Darstellung des Grundkonzeptes der zweidimensionalen Überlagerung
(Eigene Darstellung)

5.3.1 Bestimmung der sichtbaren Interpolationsfläche

Im vorigen Abschnitt wurden die nötigen Transformationen entwickelt, die es mit der OpenGL ES Bibliothek erlauben, einen Vektor bezüglich des Weltkoordinatensystems auf die Vorschau der integrierten Kamera eines androidfähigen Gerätes abzubilden.

Um den tatsächlich sichtbaren Bereich in Weltkoordinaten zu bestimmen, muss diese Transformation Rückgängig gemacht werden, d. h. durch Anwenden der inversen Transformationssequenz (siehe Abbildung 21) wird ein Punkt in Fensterkoordinaten auf seine ursprüngliche Repräsentation im Objektsystem zurückgeführt, wobei angenommen wird, dass dies zugleich dem virtuellen Weltkoordinatensystem entspricht. Für eine solche Rückprojektion bietet die OpenGL Utility Library die Methode `glGluUnproject()`, die in der Android API in der `android.opengl.GLU` Klasse implementiert wurde. Sie erwartet als Eingabeparameter neben den „column-major“ linearisierten Transformationsmatrizen aus der Vertex Transformationssequenz den zu transformierenden Vektor bezüglich der Geräte- bzw. Fensterkoordinaten. Um das Resultat eindeutig zu gestalten, muss für den zu projizierenden Punkt auch eine Tiefeninformation angegeben werden. Hierbei bedeutet ein Wert von 0, dass

die Objektkoordinaten an der vorderen Clippingebene angefordert werden, wohingegen ein Tiefenwert von 1 die hintere Ebene des Sichtvolumens beschreibt (Neider, Davis und Woo 1994).

Für jede Ecke des Anzeigebereichs kann somit der korrespondierende Punkt sowohl an der vorderen als auch an der hinteren Clippingebene in Objektkoordinaten gefunden werden, wodurch die Eckpunkte dieses Körpers in Objektkoordinaten dargestellt werden (siehe Abbildung 22). Der tatsächlich mit der virtuellen Kamera sichtbare Bereich ist folglich durch die Schnittmenge dieses Sichtkörpers mit der Interpolationsebene gegeben. Dieser Schnitt ist dabei ein Polygon mit drei bis sechs Eckpunkten (Lara, Flores und Calderon 2009, 1) in Weltkoordinaten. Zur Zuordnung der Lärmwerte zu den auf das Display bezogenen Richtungsstreifen muss dabei eindeutig festgelegt sein, welche Polygonkanten die untere bzw. obere Displaykante abbilden. Um die nachfolgende Zuordnung der Lärmwerte zu den Richtungsstreifen zu vereinfachen, wird deshalb ein auf den Kanten des Sichtvolumens basierender Algorithmus angewandt, der immer einen durch vier Punkte bestimmten Bereich liefert. Bei üblicher Verwendung des Augmented Reality Systems sollte dies keine Einschränkung darstellen.

Dazu werden die seitlichen Kanten des Sichtvolumens nacheinander jeweils von einem Displayeckpunkt aus auf Schnittpunkte mit der Interpolationsebene geprüft. Ein Schnittpunkt lässt sich dabei über Einsetzen der Geradenbeschreibung einer Sichtvolumenkante in die Definition einer Ebene ermitteln:

$$\left.\begin{array}{l}\underbrace{(p - p_0) \cdot n = 0}_{\text{Ebene}} \\ \underbrace{p = tl + l_0}_{\text{Gerade}}\end{array}\right\} \Rightarrow \quad (dl + l_0 - p_0) \cdot n = 0$$

$$\Leftrightarrow dl \cdot n + (l_0 - p_0) \cdot n = 0$$

$$\Leftrightarrow \quad \frac{(p_0 - l_0) \cdot n}{l \cdot n} = d$$

mit $l_0, l_1 \in \mathbb{R}^3$ Start- und Endpunkt einer Sichtvolumenkante
$l = l_1 - l_0$ Vektor in Richtung der Kante
$p_0 \in \mathbb{R}^3$ Punkt auf der Ebene
$n \in \mathbb{R}^3$ Normale der Ebene

Gleichung 8: Berechnung des Schnittpunktes einer Geraden mit einer Ebene

Dabei stellt d den Abstand des Schnittpunktes von l_0 aus dar. Für $d \in [0,1]$ schneidet die Linie die Fläche zwischen den Punkten l_0 und l_1. Angewandt auf die Interpolationsfläche in der XY-Ebene des Weltkoordinatensystems gilt $n = (0,0,1)^T$ und für d folglich:

$$d = \frac{p_{0z} - l_{0z}}{l_{1z} - l_{0z}}$$

Für einen gegebenen Eckpunkt des Displays und dem jeweils gegenüberliegenden lässt sich der Schnittpunkt mit der Interpolationsebene mit dieser Formel nach dem Schema aus Abbildung 23 schrittweise mit jedem Eckpunkt bestimmen.

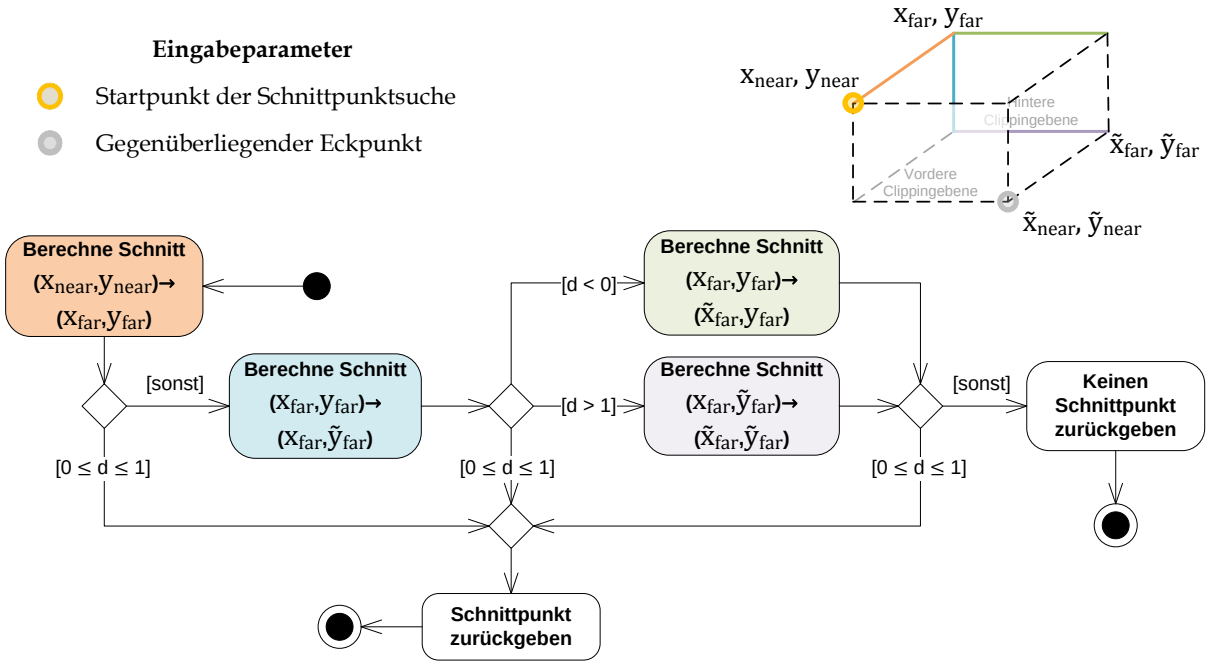


Abbildung 23: Ablauf zur Findung der Schnittpunkte mit der Interpolationsebene ausgehend von Displayeckpunkten

(Eigene Darstellung)

5.3.2 Zuordnung der Sichtstreifen

Es wurden bereits vier Ecken des sichtbaren Bereichs der Interpolationsebene in Weltkoordinaten ausgewiesen, außerdem sind dabei die der Darstellungsebene zu- und abgewandten Kanten bekannt. Die benötigten Trennlinien können folglich entlang dieser Kanten gleichmäßig in der Interpolation verteilt gesetzt werden und die jeweiligen Flächen in ihren Immissionswerten gemittelt werden.

Wegen der potenziell großen Datenmengen, die eine Schallinterpolation umfassen kann, muss die Zuordnung zu den Streifen auf effizientem Weg durchgeführt werden, um dem Nutzer ein interaktives Erlebnis sowie ein ungestörtes Ansprechverhalten der Applikation zu ermöglichen. Dazu werden die einzelnen Abgrenzungen mit einer einfachen Linienrasterisierung basierend auf dem klassischen Bresenham-Algorithmus (vgl. Bresenham 1965) im Interpolationsraster erfasst. Durch Differenzanalyse benötigt dieser Algorithmus nur einfache Additions- und Vergleichsoperationen um eine stetige Liniendefinition zu rasterisieren.

Ziel der Linienrasterisierung ist hierbei jedoch nicht das Zeichnen von Geraden, sondern das Setzen von Schnittpunktmarkierungen zur Kombination mit einem vereinfachten Kantenlistenalgorithmus zum Füllen von Polygonen. Basierend auf einzelnen Bildzeilen, den sogenannten Scanlines, werden dabei allgemein ihre Schnittpunkte mit den Polygonkanten ermittelt, diese anschließend geeignet sortiert und zum effektiven zeilenweisen Füllen des Polygons verwendet. Für die spezielle Anwendung der Immissionszuordnungen ist in der Kantenliste nur eine Aufführung der jeweiligen horizontalen Kantenextrema bezüglich einer Bildzeile vonnöten (siehe Abbildung 24), da hier nur jeweils vier Kanten beteiligt sind, von denen keine konkave Formgebung erwartet wird, bei der mehrere Schnittpunkte pro Bildzeile auftreten würden. Bei der eigentlichen Füllung der Polygone werden statt Zeichenoperationen die einzelnen Immissionswerte entsprechend Gleichung 3 summiert und gemittelt. Das Resultat ist ein gemittelter Schalldruckpegel für jeden einzelnen Sichtstreifen. Tabelle 9 fasst die Schritte des Vorgehens anschaulich zusammen.

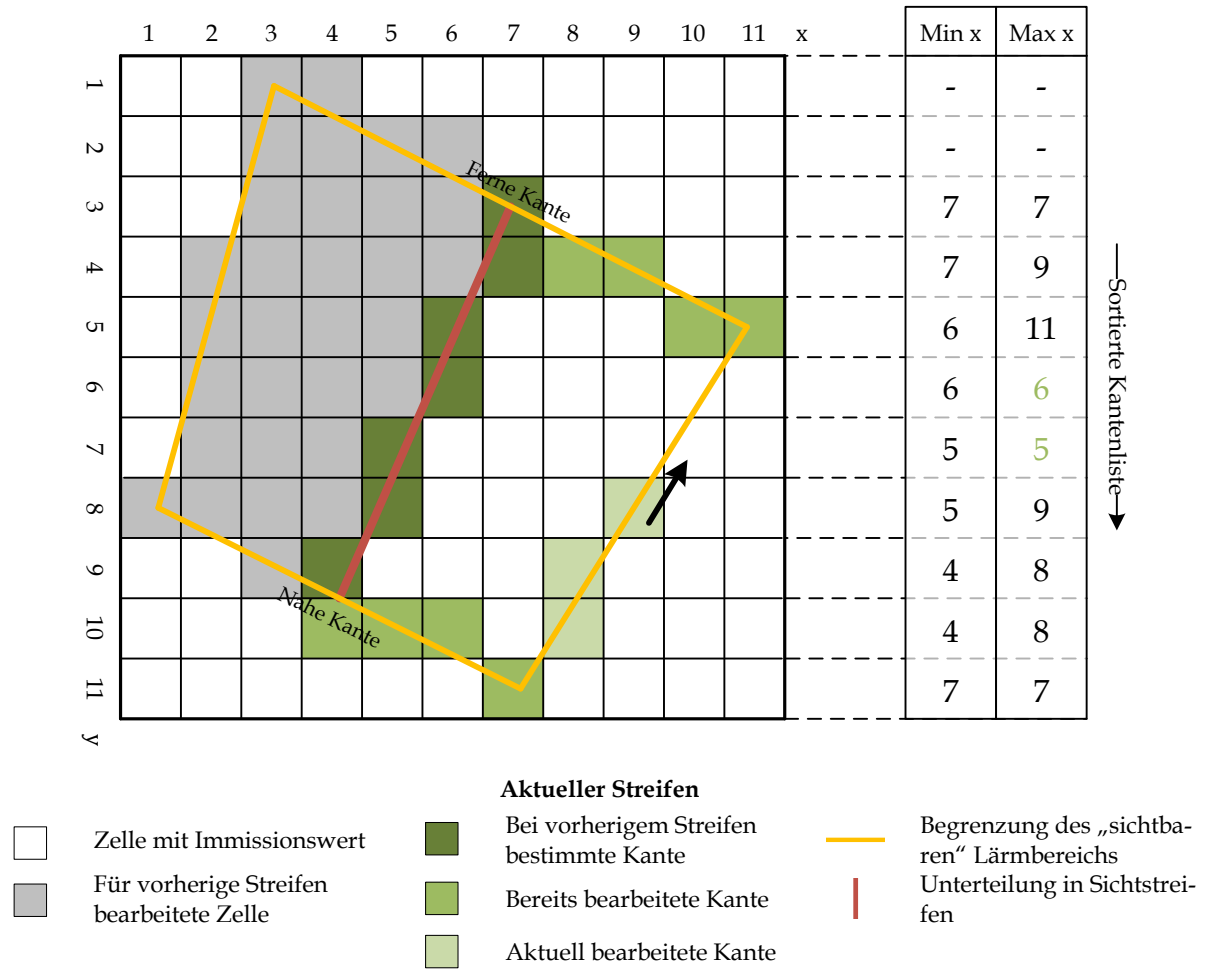
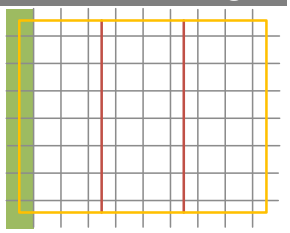
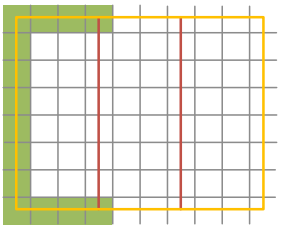
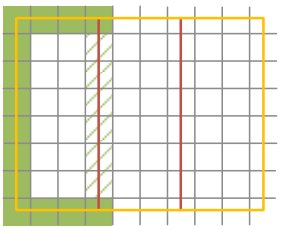
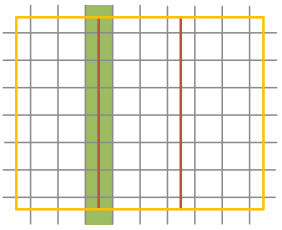


Abbildung 24: Darstellung des entwickelten Algorithmus zur Immissionsmittelung der Sichtstreifen. Zeigt Zustand während Erzeugung einer Kantenliste

(Eigene Darstellung)

Tabelle 9: Veranschaulichung der Verfahrensschritte zur Sichtstreifenzuordnung

Beschreibung	Veranschauung
1. *Initialisierung* a. Erstellen zweier Scanlinecaches entsprechend y-Ausdehnung, einen aktiven und inaktiven b. Aktualisierung der aktiven Scanlinebegrenzungen durch erste seitliche Streifenbegrenzung	
2. *Für alle Streifen* a. Schalldruckakkumulator erstellen *b.* Aktive Scanlinebegrenzungen durch vordere Begrenzung aktualisieren *c.* Aktive Scanlinebegrenzungen durch hintere Begrenzung aktualisieren d. Inaktiven Scanlinecache zurücksetzen *e.* Inaktive Scanlinebegrenzungen durch seitliche Begrenzung aktualisieren	
f. Inaktive für aktive Scanlinebegrenzungen übernehmen i. Vorhandene aktive Scanlinebegrenzung aktualisieren ii. Neue Scanlinebegrenzung zufügen	
g. Für alle Scanlines *i. Für alle Interpolationswerte in Begrenzung* 1. Schalldruckpegel in Schalldruck umrechnen 2. Schalldruck akkumulieren *h.* Gemittelten Schalldruckpegel aus Akkumulator berechnen und speichern	
i. Aktive und Inaktive Scanlinebegrenzungen tauschen *Das Verfahren ist nun für den nächsten Schleifendurchlauf vorbereitet*	
3. *Gemittelte Schalldruckpegel zurückgeben*	

5.3.3 Darstellung

Zur Anzeige der Resultate im „window on the world“ System können die ermittelten Schallmittel als einfache zweidimensionale Grafik entsprechend der Sichtstreifeneinteilung auf dem Kamerabild dargestellt werden. Es ist lediglich eine genaue horizontale Ausrichtung mit dem Kamerabild erforderlich um die korrekte Überlagerung zu wahren. Hierzu bietet es sich an eine eigene Implementierung der android.view Klasse zu erstellen.

6 Referenzanwendung „NoiseAR“

Im Zuge dieser wissenschaftlichen Ausarbeitung wurden die erarbeiteten Algorithmen und Strukturen zur Lärminterpolation und sensorbasierten Erweiterung der Realität in einer entsprechenden Android Applikation zusammengefasst und erprobt. Das Ergebnis hiervon befindet sich im Anhang dieser Arbeit sowohl als Quelltext mitsamt nötiger Ressourcen für eine Android 2.2 Entwicklungsumgebung mit Google API Paket zur Verwendung der Google Maps API, als auch als kompilierte Android Package Datei. Zur Benutzung der MapView Klasse muss dabei beachtet werden, dass der verwendete Google Maps API Schlüssel zu den von der Entwicklungsumgebung zur Anwendungssignierung verwendeten Zertifikaten passt[5].

6.1 Aufbau der Anwendung

Die mobile Applikation setzt alle in dieser Arbeit entwickelten Algorithmen und Prozesse praktisch um. Ausgehend von einer Kartendarstellung, in der die Schalleinwirkung sowohl auf eine Straßenkarte als auch auf Satellitenbilder überblendet werden kann, eröffnet sie einem Nutzer auf seinem Handydisplay eine korrekte Überlagerung eines Kameravorschaubildes mit interpolierten Lärmwerten. Durch Zielen mit seinem Handy bekommt er Einsicht in die in seiner Umgebung gemessenen Schallimmissionen (siehe Abbildung 25).

In einigen einfach und übersichtlich gestalteten Menüs kann dabei sowohl ein Zeitraum als auch eine Tageszeit angegeben werden, worauf die Datenbasis basieren soll. Auch kann hier zwischen verschiedenen Datenquellen und Lärmvisualisierungen gewechselt werden. Es ist ferner Möglich, in der Kartenansicht zusätzlich aktuelle Verkehrsdaten der Google Dienste einzublenden (siehe Abbildung 26).

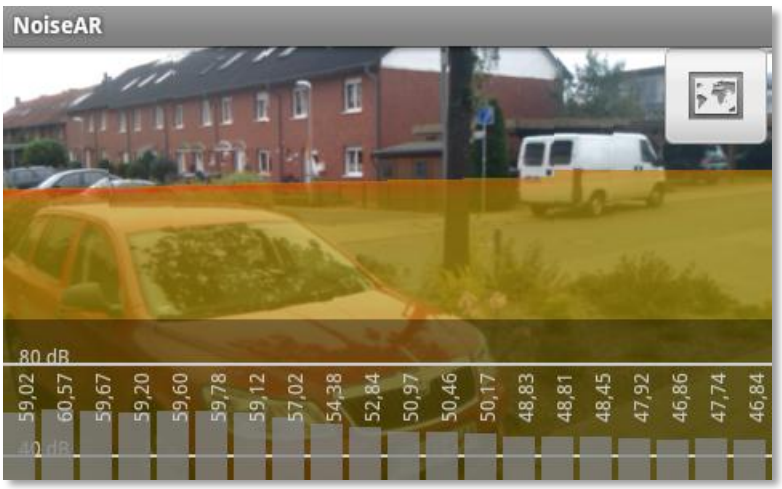


Abbildung 25: Augmented Reality Sicht

[5] Siehe auch http://code.google.com/intl/de-DE/android/add-ons/google-apis/mapkey.html

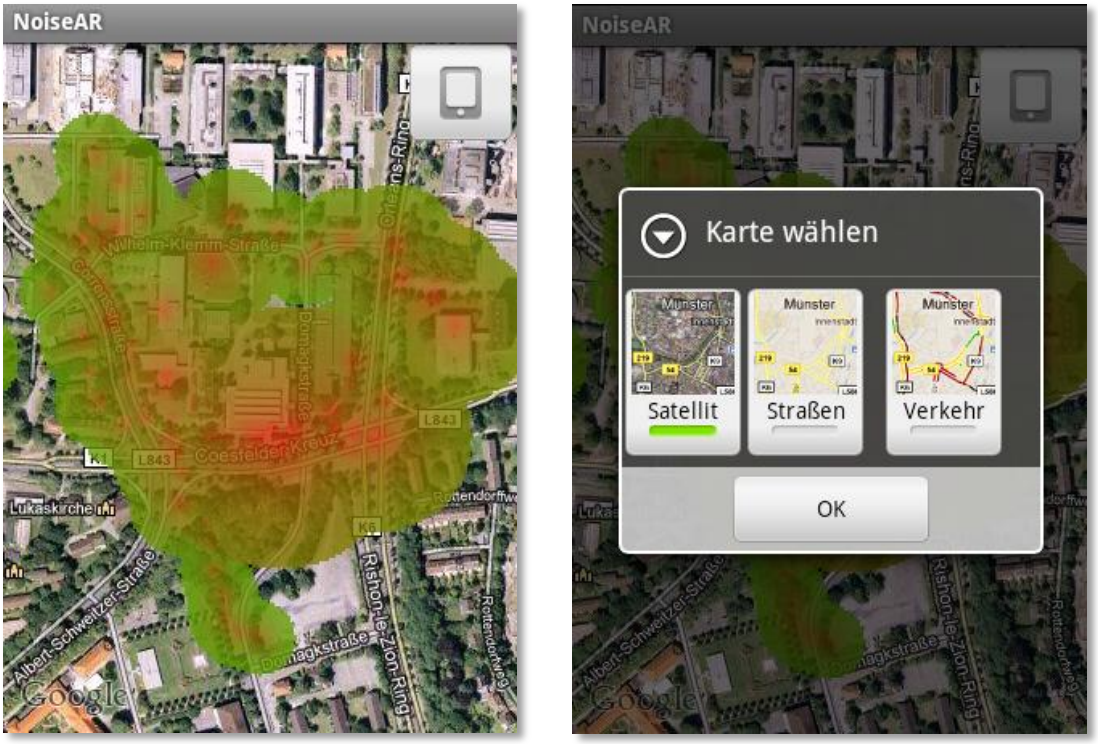


Abbildung 26: Kartenbasierte Lärmauswertung am Beispiel der "NoiseDroid" Datenquelle

6.2 Programmarchitektur

Der Programmcode ist grundsätzlich in drei Pakete eingeteilt. Das Paket `view` umfasst alle zur Präsentation nötigen Klassen, darunter eine zusammenfassende Klasse zur Anzeige des Augmented Reality Systems sowie eine Klasse zum Anzeigen der speziellen Kartenansicht.

Im Paket `datasource` werden die Komponenten zum Messdatenzugriff aufgeführt. Implementiert wurde dabei ein Zugriff auf die Lärmkartierungsanwendung „NoiseDroid“ der Lehrveranstaltung „Geosoftware II“ des Wintersemesters 2010/2011. Er ermöglicht sowohl Zugang zu der entstandenen Internetplattform als auch auf lokal vorhandene Messdaten, falls die dabei entwickelte Applikation „NoiseDroid“ auf demselben Android System installiert wurde. Ausfälle und hohe Übertragungszeiten erlaubten keine Implementierung eines NoiseTube Datenzugriffs.

Alle in dieser Arbeit entwickelten Algorithmen und Prozesse wurden im Paket `algorithms` zusammengefasst. Kern sind hier die Klasse `Interpolation` in welcher der gesamte Prozess aus Kapitel vier umgesetzt wird sowie die Klasse `NoiseView`, welche der Umsetzung der zweidimensionalen richtungsorientierten Visualisierung dient (siehe Kapitel 5.3).

Eine Verwendung eines vorhandenen Augmented Reality Frameworks für das Android System war nicht möglich. Zum einen erlaubt kein bekanntes Produkt eine genaue flächenhafte Registrierung der virtuellen Eindrücke in allen drei Raumdimensionen in unbekannter Um-

gebung. So bieten bekannte Augmented Reality Browser wie mixare[6] (mix Augmented Reality Engine) nur eine Richtungsabhängige Registrierung an. Zum anderen fehlen nötige Dokumentationen und offene Schnittstellen, um eine wie in Kapitel 5.3 erläuterte Schallimmissionsanalyse durchzuführen.

Einzig auf optischen Methoden beruhende Frameworks, beispielsweise auf dem ARToolkit[7] basierend, erlauben eine flächenhafte Zuordnung. Wegen der ungebundenen Umgebung bei Lärmmessungen ist dies, wie bereits in Kapitel drei ausgeführt, hier nicht anwendbar.

6.2.1 Anzeigehierarchie

Die gesamte Applikation umfasst lediglich eine sogenannte `Activity`. Um sowohl die Kartenansicht als auch die Sicht zur Erweiterten Realität gleichsam in der Anwendung zu platzieren, wurde eine zusätzliche Anzeigeverwaltung eingefügt. Wegen der engen Verzahnung beider Sichten war es nicht möglich, sie in separate Anzeigefenster aufzuteilen. Auch eine Verwendung einer sogenannten `ActivityGroup` erschien zu unflexibel.

Die zentrale `NoiseARActivity` verwaltet mehrere `NoiseARView` Implementierungen, die dabei jeweils eine Präsentations- und Kontrollschicht umfassen. Sie synchronisieren sich über gemeinsame Schnittstellen, die die Aktivität ihnen zur Verfügung stellt. So haben beispielsweise alle dieser `NoiseARView` Zugang zur einer zentralen Positionsbestimmungsverwaltung und einer gemeinsamen Informationsleiste (siehe Abbildung 27) und können diese unabhängig voneinander verwenden. Darüber hinaus bietet die `NoiseARView` Schnittstelle die Möglichkeit, an dem Android Optionsmenü teilzuhaben. Objekte können eigene Menüeinträge bereitstellen, dessen Ereignisse an sie zurück delegiert werden.

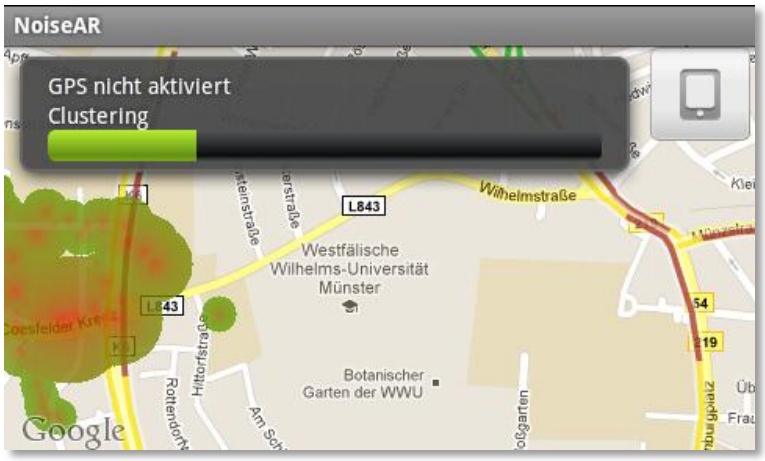


Abbildung 27: Darstellung der zentralen Informationsleiste zur Anzeige von Status- und Fortschrittsmeldungen

[6] Mixare Internetpräsenz: http://www.mixare.org/de/
[7] Internetpräsenz der ARToolworks: http://www.artoolworks.com/

6.2.2 Datenquellen

Zur flexiblen Einbindung von verschiedenen Datenquellen, sowie zur einfachen Erweiterung wird die Schnittstelle `DataSource` zum Datenzugriff im Sinne des Entwurfsmusters der Abstrakten Fabrik bereitgestellt. Eine konkrete Datenanfrage wird dabei durch den `MeasurementManager` gestellt. Hierbei wird ein `Tile` Objekt sowie eine `MeasurementFilter` Instanz übergeben.

Über das `Tile` Objekt wird die räumliche Ausdehnung bestimmt, für die Daten angefordert werden. Wegen möglicherweise hohem Datenaufkommens wurde ein räumlicher Index implementiert, der Messungen nur in einzelnen Kacheln anfordert und diese Ergebnisse zwischenspeichert (siehe Abbildung 28). Die räumliche Aufteilung basiert dabei auf der sphärischen Mercator Projektion (EPSG:3857), die von dem Google Maps Dienst verwendet wird[8]. Somit wird eine unmittelbare Anzeige der Lärminterpolation in der Kartenansicht der Applikation ermöglicht. Es wird durch die konkret verwendete `DataSource` Implementierung entschieden, wie lange eine Zwischenspeicherung gültig bleibt. Darüber hinaus verfügt der Index über eine automatische Speicherverwaltung, die gänzlich auf den Anforderungen der virtuellen Java Laufzeitumgebung basiert. Sämtliche zwischengespeicherte Ergebnisse liegen lediglich als `SoftReference` vor, die es der Automatischen Speicherbereinigung (Garbage Collector) der Umgebung erlauben, selbstständig Messungen aus dem Index zu löschen, falls Bedarf nach Speicherressourcen besteht. Die Vorgaben für dieses Verfahren sehen vor, dass selten verwendete Referenzen zuerst freigegeben werden, sie jedoch so lange wie möglich gehalten werden (vgl. Oracle 2011), wodurch eine ideale Messungszwischenspeicherung zugelassen wird.

Ein `MeasurementFilter` Objekt beschreibt bei der Datenanfrage die geforderten Einschränkungen bezüglich der Zeit, die von den `DataSource` Implementierungen individuell umgesetzt werden müssen.

[8] Siehe auch http://code.google.com/intl/de-DE/apis/maps/documentation/javascript/v2/overlays.html#Google_Maps_Coordinates

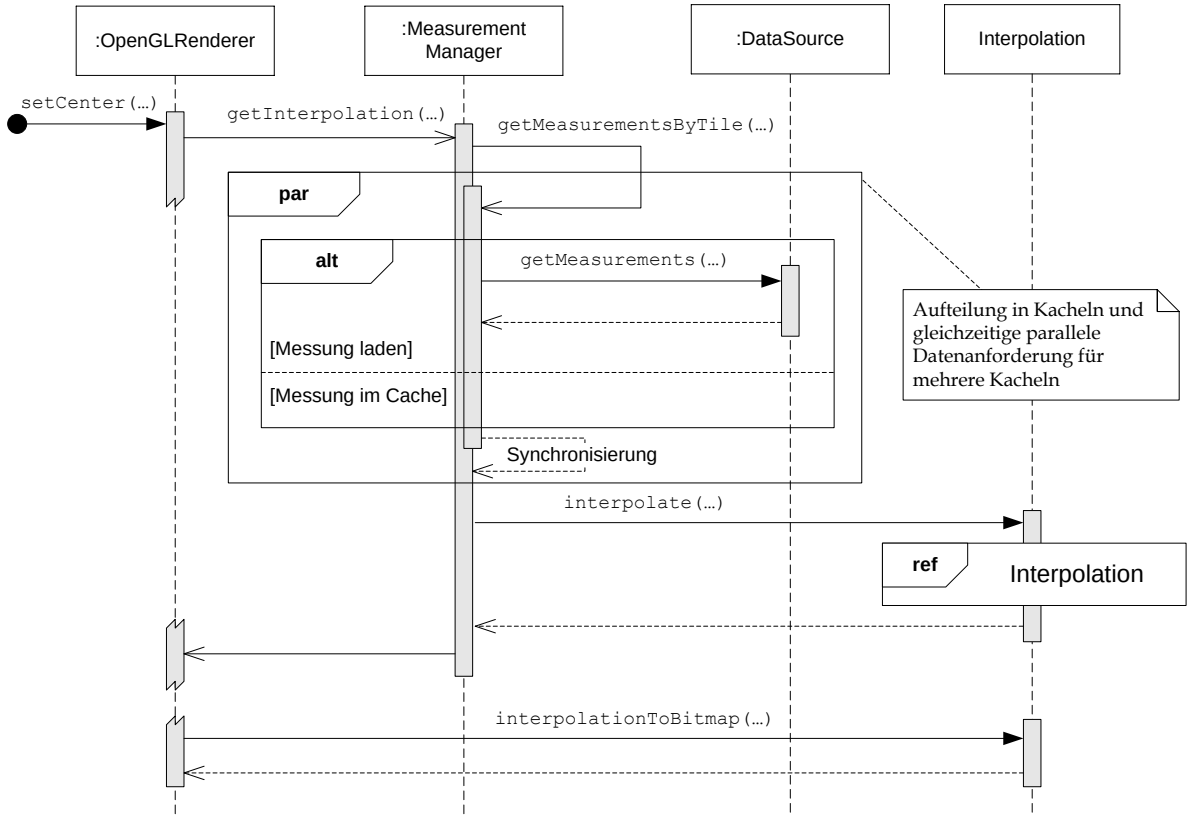


Abbildung 28: Sequenzdiagramm zur typischen Interaktion bei der Anforderung einer Interpolation, im Beispiel ausgehend von einer Positionsänderung

(Eigene Darstellung)

6.2.3 Lärminterpolation

Schnittstelle zur Lärminterpolation ist die `algorithms.Interpolation` Klasse. Sie führt ausgehend von einer Liste an Messungsobjekten die im Kapitel vier beschriebenen Prozesse durch. Die Methoden sind dabei so konzipiert, dass sie zur Speicheroptimierung grundsätzlich das Ergebnisobjekt einer zuvor durchgeführten Operation erwarten, sodass dieses wieder verwendet und aktualisiert werden kann, ohne neue Ressourcen anzulegen. Wird keines angegeben, so wird für die Rückgabe neuer Speicher alloziert. Den Interpolationsvorgang im Detail zeigt Abbildung 29.

6.2.4 Lärmanalyse im Sichtfeld

Zur Analyse der im Blickfeld der Kamera befindlichen Interpolationszellen wurden zwei Schnittstellen beschrieben. Als `NoiseViewChangedListener` können von der Augmented Reality Ansicht laufend die Projektionen der vier Displayeckpunkte auf die Interpolationsebene erhalten werden. Als Implementierung der `NoiseGridValueProvider` Schnittstelle bietet die Ansicht zudem direkte Zugriffsmethoden auf die entsprechenden Interpolationswerte an. Die `algorithms.NoiseView` Klasse führt hierauf basierend alle in 5.3.2 beschriebenen Prozesse durch.

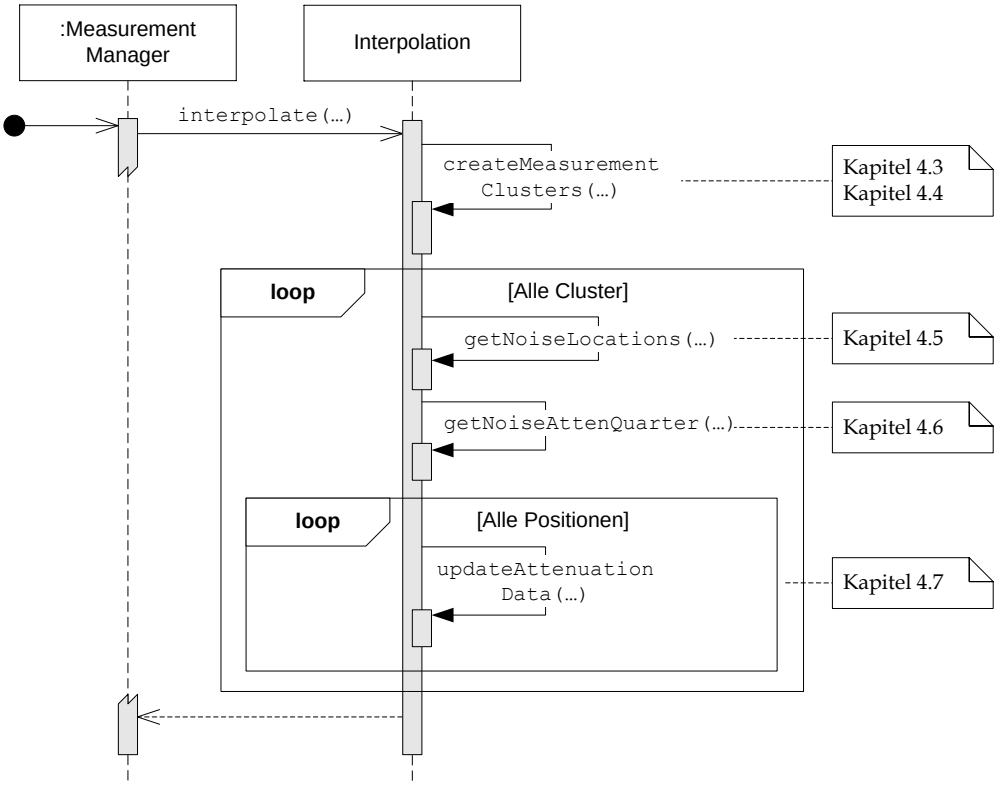


Abbildung 29: Sequenzdiagramm zur Interpolation

(Eigene Darstellung)

6.2.5 Asynchronität

Alle das Interpolieren und Anfragen von Daten betreffende Operationen sind asynchron implementiert. Im Sinne des Beobachter Entwurfsmusters werden für die jeweiligen Operationen Rückrufmethoden bzw. –objekte übergeben, die zu einem späteren Zeitpunkt mit dem Resultat der Operation aufgerufen werden, bzw. im Fehlerfall über den Abbruch informiert werden. Außerdem verfügen die Rückrufobjekte über Methoden zur Mitteilung des aktuellen Arbeitsfortschrittes, sodass dem Nutzer eine Rückmeldung hierzu gegeben werden kann.

Als zusätzliche direkte Rückgabe der asynchronen Funktionsaufrufe wird ein `MeasurementManager.RequestHolder` Objekt zurückgegeben, welches ein sofortiges Beenden der jeweiligen Operation ermöglicht, ähnlich zu der Java-eigenen `Future` Schnittstelle.

Im Falle des Datenzugriffs und der Interpolation werden die Operationen durch einen `ThreadPoolExecutor` mit drei Arbeitselementen parallel ausgeführt. Hierdurch wird einerseits das Reaktionsvermögen der Anwendung gewahrt und andererseits werden Wartezeiten innerhalb der Operationen durch die Parallelisierung ausgeglichen. Durch die Möglichkeit des nachträglichen Abbruchs einer Operation kann das Ansprechverhalten weiter optimiert werden, da beispielsweise Messdaten nicht weiter angefordert werden, wenn der Nutzer die Kartenansicht ausblendet oder den Ausschnitt verändert.

7 Zusammenfassung

Diese Arbeit stellt eine Prozesskette zur dynamischen Bereitstellung einer nutzerzentrieren Lärmanalyse vor. Durch Crowdsourcing gewonnene Lärmdaten, die den tatsächlich bei den Menschen bzw. Nutzern wirkenden Lärmeinfluss abbilden, werden verwendet, um im Gegenzug den Nutzern individuelle raumbezogene Einsicht in die gesammelten Daten zu ermöglichen.

Ausgehend von Schallmessdaten, die sich wegen ihrer Datengrundlage sowohl durch systembedingte als auch operationelle Fehler kennzeichnen, werden hierbei Verfahren entwickelt, bei denen der Fokus explizit auf hohe Robustheit gelegt wird. Mittels stochastischer Methoden werden Unsicherheiten hinsichtlich der räumlichen Lage gemessener Schallimmissionen erfasst. Messungenauigkeiten sowie tatsächliche Variationen in den Schallmessungen innerhalb eines Messzeitraumes werden auf Grundlage der akustischen Zusammenhänge gemittelt und interpoliert, die bei dem unsicheren Charakter der Datenquellen noch angenommen werden können. Dabei wurde erkannt, dass zur Erfassung und Behandlung des tatsächlichen subjektiven Aspektes des Umgebungslärms weitere über den resultierenden Schalldruck hinausgehende Erhebungen nötig sind, die eine Beschreibung des Lärmereignisses durch den Nutzer bzw. durch automatische Klassifikationen voraussetzen. Zur Einarbeitung subjektiven Empfindens wäre folgend eine einheitliche Kontextualisierung und Referenzierung der subjektiven Beschreibung im semantischen Raum obligatorisch. Hier wäre auch eine Vorverarbeitung der Lärmdaten in einem zentralen System von Vorteil, da eine zusätzliche Abbildung subjektiver Eindrücke und Lärmmuster einen komplexeren Grad an Modellierung benötigt.

Zentraler Gegenstand der Verfahrensentwicklung ist außerdem die Optimierung der Verwendung der im Android System verfügbaren Ressourcen. Es werden verschiedene Implementierungsentscheidungen unter dem Aspekt der Speicherauslastung und Laufzeit diskutiert. Durch analytische Untersuchungen werden limitierende Faktoren erkannt und infolge dessen Datenstrukturen und –typen sowie Verfahrensparameter identifiziert, die eine möglichst schnelle und speicherschonende Ausführung innerhalb der im Android System wirkenden Dalvik Java Laufzeitumgebung garantieren. Dominiert werden die Implementierungsentscheidungen durch eine Abwägung zwischen schnellem nativem Zugriff auf die interpolierten Lärmdaten und schneller Erzeugung der Interpolation innerhalb der Java Umgebung. Gerade wegen der häufig nötigen Aufrufe zwischen der nativen und virtuellen Laufzeitumgebung, bietet sich als zukünftige Arbeit eine Portierung des Interpolationspro-

zesses in eine nativ in Applikationen eingebundene C-Umgebung an, die auch den hohen Grad an Parallelisierbarkeit des Verfahrens ausnutzt.

Zur Überführung der Lärmdaten in ein visuelles Augmented Reality System werden zunächst die typischen und nötigen Referenzrahmen definiert und anschließend Methoden vorgestellt, die eine fortwährende Registrierung der Bezugssysteme ausgehend von der geografischen Position und Ausrichtung des verwendeten androidfähigen Gerätes ermöglichen. Hierbei wird die praktische Verwendung eines hybriden sensorbasierten Trackingverfahrens in der Android Plattform in Bezug auf dessen theoretischer Basis erläutert, sowie dessen Einschränkungen bezüglich der Höhenermittlung des mobilen Gerätes hervorgehoben. Gerade bei Darstellung flächenhafter Phänomene ist diese wichtig, um eine korrekte Zuordnung der virtuellen Information in allen Freiheitsgraden zu ermöglichen, anders als bei lediglich punkthaften Darstellungen wobei nur eine genaue Richtungsermittlung ausreicht. Neben dieser Schwäche zeigt das hybride Trackingsystem auch negative Einflüsse durch Ausreißer und Rauschen in den Sensordaten. In der Anwendung „NoiseAR“ wurde diesem mit einem Tiefpassfilter entgegnet. Die Sensorfilterung bietet jedoch weitere Themenschwerpunkte, wie der Einbeziehung der magnetischen Deklination oder der Verwendung von optischen, auf Mustererkennung beruhenden Korrekturen, die ein genaueres Tracking ermöglichen.

Die durch das Tracking bestimmbaren Transformationen vom virtuellen Bezugssystem in die Kameraabbildung werden verwendet, um zwei unterschiedliche Lärmvisualisierungen in der Erweiterten Realität innerhalb des Android Systems zu realisieren. Beide spiegeln die dynamische auf den Standort und Sichtbereich eines Nutzers zugeschnittene Lärmanalyse wider. Die praktische Umsetzung „NoiseAR“ zeigt dessen Nutzen.

Literaturverzeichnis

Azuma, Ronald. „A Survey of Augmented Reality." *Presence: Teleoperators and Virtual Environments*, August 1997: 355-385.

Azuma, Ronald, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier und Blair MacIntyre. „Recent Advances in Augmented Reality." *IEEE Computer Graphics and Applications*, November/Dezember 2001: 34-47.

Bresenham, J. E. „Algorithm for computer control of a digital plotter." *IBM Systems Journal*, 1965: 25-30.

de Lange, Norbert. *Geoinformatik in Theorie und Praxis.* Springer, 2005.

Elkan, Charles. „Using the Triangle Inequality to Accelerate k-Means." *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

Europäisches Parlament und Rat. „Richtlinie 2002/49/EG vom 25. Juni 2002 über die Bewertung und Bekämpfung von Umgebungslärm." *Amtsblatt der Europäischen Gemeinschaften*, 18. Juli 2002: 12-25.

Google Inc. „Android 2.2 Compatibility Definition." *Android Open Source Project.* 2010. http://static.googleusercontent.com/external_content/untrusted_dlcp/source.android.com/de//compatibility/2.2/android-2.2-cdd.pdf (Zugriff am 1. August 2011).

—. *Camera | Android Developers.* 2011a. http://developer.android.com/reference/android/hardware/Camera.html (Zugriff am 14. August 2011).

—. *OpenGL ES Versions | Android Developers.* 2011b. http://developer.android.com/resources/dashboard/platform-versions.html (Zugriff am 5. September 2011).

—. *Platform Versions | Android Developers.* 2011c. http://developer.android.com/resources/dashboard/platform-versions.html (Zugriff am 14. September 2011).

—. *Screen Sizes and Densities | Android Developers.* 2011g. http://developer.android.com/resources/dashboard/screens.html (Zugriff am 15. September 2011).

—. *SensorEvent | Android Developers.* 2011d. http://developer.android.com/reference/android/hardware/SensorEvent.html (Zugriff am 22. Juli 2011).

—. *SensorManager | Android Developers.* 2011e. http://developer.android.com/reference/android/hardware/SensorManager.html (Zugriff am 23. Juli 2011).

—. *What is Android? | Android Developers.* 2011f. http://developer.android.com/guide/basics/what-is-android.html (Zugriff am 2. September 2011).

Khronos Group. *About OpenGL.* 2010. http://www.opengl.org/about/overview/ (Zugriff am 2. September 2011).

Komatineni, Satya, Dave MacLean, und Sayed Hashimi. *Pro Android 3.* Apress, 2011.

Lara, Carlos, Juan J. Flores und Felix Calderon. „On the Hyperbox – Hyperplane Intersection Problem." *InfoComp*, Dezember 2009: 21-27.

Lerch, Reinhard, Gerhard Sessler und Dietrich Wolf. *Technische Akustik: Grundlagen und Anwendungen.* Springer, 2008.

Maisonneuve, Nicolas, Matthias Stevens und Bartek Ochab. „Participatory noise pollution monitoring using mobile phones." *Information Polity*, August 2010: 51-71.

Mehler-Bicher, Anett, Michael Reiß und Lothar Steiger. *Augmented Reality: Theorie und Praxis.* München: Oldenbourg, 2011.

Milgram, Paul, Haruo Takemura, Akira Utsumi und Fumio Kishino. „Augmented Reality: A class of displays on the reality-virtuality continuum." *SPIE Vol. 2351, Telemanipulator and Telepresence Technologies*, 1994: 282-292.

Milgram, Paul und Fumio Kishino. „A Taxonomy of Mixed Reality Visual Displays." *IEICE Transactions on Information Systems, Vol E77-D*, Dezember 1994.

Ming, Li. „Correspondence Analysis Between The Image Formation Pipelines of Graphics and Vision." *Proceedings of the IX Spanish Symposium on Pattern Recognition and Image Analysis*, Mai 2001: 187-192.

Morrill, Dan. *Android Developers Blog*. 9. September 2010. http://android-developers.blogspot.com/2010/09/one-screen-turn-deserves-another.html (Zugriff am 28. Juli 2011).

Möser, Michael. *Messtechnik der Akustik.* Springer, 2009.

Müller, Gerhard. *Taschenbuch Der Technischen Akustik.* Springer, 2003.

Neider, Jackie, Tom Davis und Mason Woo. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R).* Addison-Wesley Publishing Company, 1994.

Open Handset Allinance. *Alliance Members.* 2011. http://www.openhandsetalliance.com/oha_members.html (Zugriff am 2. September 2011).

Oracle . *SoftReference (Java Platform SE 6).* 2011. http://download.oracle.com/javase/6/docs/api/java/lang/ref/SoftReference.html (Zugriff am 2. September 2011).

Rolland, Jannick P., Yohan Baillot und Alexei A. Goon. *A Survey of Tracking Technology for Virtual Environments.* 2001.

Sengpiel, Eberhard. *Abnahme des Schallpegels in dB mit Entfernung [...] - sengpielaudio Sengpiel Berlin.* 1. September 2011. http://www.sengpielaudio.com/Rechner-entfernung.htm (Zugriff am 4. September 2011).

The Khronos Group Inc. *OpenGL(R) ES Common/Common-Lite Profile Specification, Version 1.1.12 (Full Specification).* 24. April 2008.

Umweltbundesamt. *Sechste Allgemeine Verwaltungsvorschrift zum Bundes-Immissionsschutzgesetz (Technische Anleitung zum Schutz gegen Lärm - TA Lärm).* 1998.

Vallino, James. *Interactive Augmented Reality.* 1998.

World Health Organization. *Burden of disease from environmental noise. Quantification of healthy life years lost in Europe.* 2011.

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Thema

„Visualisierung von Community-basierten Lärmmessungen in einer Android-basierten Augmented Reality Umgebung“

selbstständig und ohne fremder Hilfe ausschließlich unter Verwendung der im Literatur- und Quellenverzeichnis angegebenen Quellen angefertigt habe und, dass ich alle wörtlich übernommenen Stellen besonders gekennzeichnet und zitiert habe.

Münster, den 21. September 2011 _____

(Holger Hopmann)