Fakultät für Informatik
und Wirtschaftsinformatik

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Master Thesis

Name:	Andreas Suchanek
Thema:	Konzeption und Entwicklung einer Android-App zur Berechnung von Radarbildern auf Grundlage 3-dimensionaler-Objekte
Arbeitsplatz:	Fraunhofer (IOSB), Karlsruhe
Referent:	Prof. Dr. Vogelsang
Korreferent:	Prof. Dr. Fuchß
Abgabetermin:	31.08.2011

Karlsruhe, den 01.03.2011

Der Vorsitzende des
Prüfungsausschusses

Prof. Dr. Ditzinger

Eidesstattliche Erklärung

Ich versichere, dass ich diese Masterthesis ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Diese Masterthesis wurde ferner noch nirgends zur Anerkennung als Prüfungsleistung vorgelegt.

Karlsruhe, den 30. 08. 2011

_____

Unterschrift

Zusammenfassung

Das E-Learning gewann in den vergangenen zwei Jahrzehnten zunehmend an Bedeutung. Vom anfänglichen, auf Computer zentrierten, Lernen, ging die Entwicklung hin zu webbasierten Kursen. Mit der zunehmenden Verbreitung von Smartphones, ist ein neuer Trend zu erkennen: E-Learning für unterwegs – mobile learning. Lektionen und ganze Kurse, sind direkt auf dem Smartphone abrufbar. Das Fraunhofer Institut IOSB entwickelte in der Vergangenheit bereits einen E-Learning Kurs für die Bundeswehr, der über den Browser absolviert werden kann. Im nächsten Schritt soll die Entwicklung und Programmierung einer App erfolgen, die es ermöglicht, einfache 3-D-Objekte auf dem Smartphone in ein Radarbild zu konvertieren. Dies ist der erste Schritt zum mobilen Lernen.

Abstract

In the last thirty years, the importance of e-learning increased rapidly. In the beginning, computer based training dominated the market. But over the years, and because of the new born Internet, web based training become state of the art. Today the evolution takes place again. More and more people have a smartphone, so there is a new trend: mobile learning. Lectures and courses, are available every time and every place on the smartphone. The Fraunhofer Institut IOSB developed e-learning-courses for the Bundeswehr, available via Browser. The next step is the development of an App, which transform a 3-D-Object in a Radar image, at the smartphone. This is the first step to mobile learning.

Inhaltsverzeichnis

1 Einleitung

Der Marktanteil an Smartphones wächst beständig. Die kleinen Geräte halten Einzug in den Alltag und unterstützen Anwender durch eine umfangreiche Sammlung an Applikationen. Die Absatzzahlen von Android-Smartphones lagen im vierten Quartal 2010 bei 33,3 Millionen verkauften Geräten für das laufende Jahr [And2010].

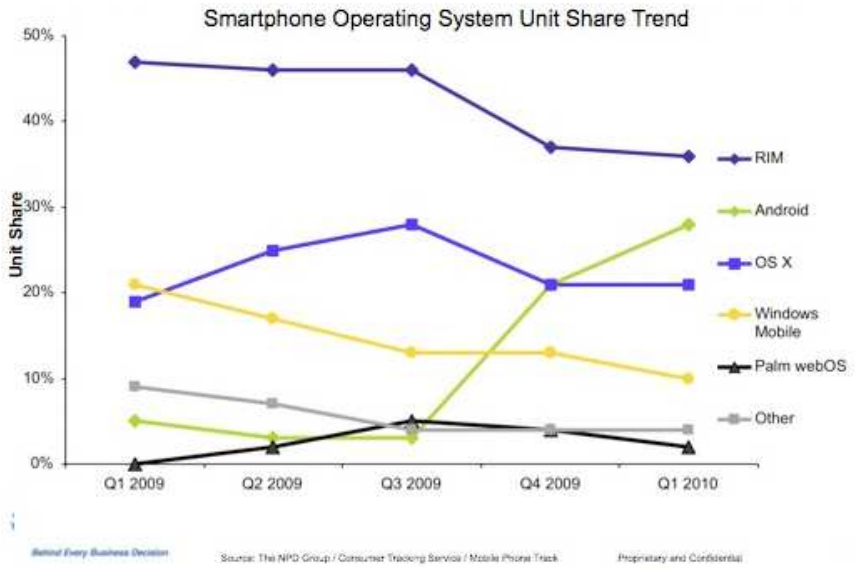


Abbildung 1: Verbreitung von Smartphones[1]

Doch nicht nur im Alltag sind die Geräte von Nutzen. Auch zur Unterstützung des E-Learning-Marktes, durch die mobile Präsentation von digitalem Lernmaterial, lassen sich die Geräte zum Lehren und Lernen einsetzen. Im Folgenden wird die Motivation hinter dieser Masterthesis und das zu erreichende Ziel vorgestellt.

1.1 Motivation und Zielsetzung

Smartphones ermöglichen dem Anwender, durch auf dem Gerät ausführbare Applikationen (Apps) die Verwendung von vielfältigen Inhalten. Eines dieser Anwendungsgebiete ist das E-Learning. Um Kurse zu unterstützen kann eine solche App multimediale Inhalte präsentieren.

[1] http://mattabad.com/smartphone-operating-system-unit-share-trend

Am Fraunhofer Institut IOSB (Institut für Optronik, Systemtechnik und Bildauswertung) wurde das Programm ViSAR (Visualization of Geometric SAR Effects) entwickelt; das anhand eines vorhandenen 3-D-Objektes ein Radarbild generiert. Aus einem einfachen Gebäude oder primitiven Objekt (beispielsweise ein Würfel) wird, durch mehrere Verarbeitungsschritte, ein Radarbild des Objekts erstellt. Bisher findet ViSAR in E-Learning Kursen der Bundeswehr Anwendung, wo die korrekte Interpretierung von Radarbildern und auftretenden Störeffekten vermittelt wird. Um den Anwendern auch das mobile Lernen zu ermöglichen, soll die Anwendung auch auf Android-Smartphones möglich werden.

Im Zentrum dieser Masterthesis steht die Entwicklung einer prototypischen Implementierung für ein solches 3-D-Radar-Simulationssystem auf Android-Smartphones. Unter Ausnutzung der gerätespezifischen Sensoren sollen 3-D-Objekte dargestellt und manipuliert werden können. Einzoomen und Auszoomen durch Fingerberührung sowie Rotation des Objekts durch die Gravitationssensoren und Fingergesten soll möglich sein. Im nächsten Schritt soll die ViSAR-Software integriert werden, um aus dem dargestellten 3-D-Objekt ein Radarbild zu erzeugen.

1.2 Struktur der Arbeit

Kapitel 1: Am Anfang steht die Einleitung, Motivation und Zielsetzung sowie die Struktur der Arbeit. Das Arbeitsumfeld (die Fraunhofer Gesellschaft) wird näher beschrieben.

Kapitel 2: Im nächsten Schritt wird die wichtigste Literatur vorgestellt, die für die Arbeit relevant war, sowie der aktuelle Stand der Technik auf verschiedenen, diese Thesis betreffenden Themengebieten, erläutert.

Kapitel 3: Die Grundlagen folgen in diesem Kapitel. Die Theorie zu Android-Smartphones, die notwendigen Programmierstrukturen für die Android-Entwicklung sowie das Visualisieren mit Android (2D und 3D) werden beschrieben.

Kapitel 4: Die 3-D-Modellierung und die verschiedenen Modelle, die dabei zum Einsatz gelangen werden vorgestellt.

Kapitel 5: Details zum ViSAR-Programm werden vorgestellt, die für die Arbeit von zentraler Bedeutung sind. Der technischen Aufbau, die verwendete Version sowie eine Desktop-Version werden hier beschrieben.

Kapitel 6: In diesem Kapitel wird der Prototyp konzipiert und die Umsetzung mittels Text und UML[2]-Diagrammen dargestellt. Skizzen (Mock-ups) stellen die Oberfläche als Konzept dar. Den Abschluss macht ein Experiment, dass den Einsatz des Prototypen darstellt und eine Diskussion mit anschliessendem Fazit und Ausblick.

Kapitel 7: Ein Anhang fasst Quellenverzeichnis und Abbildungsverzeichnis zusammen.

1.3 Die Fraunhofer-Gesellschaft und das Fraunhofer IOSB

Bei der Fraunhofer Gesellschaft handelt es sich um eine europaweit agierende Institution, mit Sitz in der Bundesrepublik Deutschland. Ziel ist eine anwendungsorientierte Forschung, zum direkten Nutzen für den Endanwender (die Gesellschaft im Allgemeinen und andere Unternehmen).

Die Forschungsbereiche der Fraunhofer Gesellschaft decken ein breites Spektrum an verschiedenen Bereichen ab. Im Folgenden ein Überblick über die thematischen Verbünde innerhalb der Gesellschaft:

- Informations- und Kommunikationstechnik
- Mikroelektronik
- Oberflächentechnik und Photonik
- Produktion
- Werkstoffe & Bauteile
- Verteidigungs- und Sicherheitsforschung

Im Laufe der vergangenen Jahrzehnte, ist die Zahl der Standorte des Unternehmens beständig angewachsen. So ist Fraunhofer mittlerweile deutschlandweit über 50-mal vertreten.

Das Fraunhofer-Institut für Optronik, Systemtechnik und Bildauswertung hat seinen Sitz in Karlsruhe und beschäftigt sich mit Systemtechnik, Optronik und Bildauswertung[3]. Das Aufgabenfeld der Abteilung Interoperabilität und Assistenzsysteme (IAS), in der auch diese Masterthesis angefertigt wird, erstreckt sich über die Vernetzung von Systemen, die Arbeit mit ontologiebasierten Informationssystemen bis hin zu Web-Services.

---

[2] Unified Modeling Language. Eine grafische Modellierungssprache zur Konzeption, Konstruktion und Dokumentation.

[3] http://www.iosb.fraunhofer.de/servlet/is/2225

Im Detail bedeutet dies: Konzipierung, Realisierung und Bewertung von interaktiven Assistenz-, Ausbildungs- und Informationsmanagementsystemen. Hierbei liegt der Schwerpunkt in der bildgestützten Aufklärung und Überwachung. Folgende Produkte sind Teil der IAS-Entwicklung:

- Assistenzsysteme
  Unter anderem in der Augmented Reality, werden vom Fraunhofer IOSB verschiedene Assistenzsysteme entwickelt.

- Synthetische Apertur Radar (SAR) -Tutor
  Ausgehend von einer Lehrbuchmetapher bedient sich SAR-Tutor bei verschiedener Multimedia und Hypertexttechniken zur Unterstützung des Lernprozesses.

- Crayons®
  Die E-Learning-Software ermöglicht das Erstellen von elektronischen Lerneinheiten. Editoren, die an bestehende Software angelehnt sind, erlauben das schnelle generieren von Content.

Besonders die Arbeit im Bereich des E-Learnings und der Objekterkennung ist für die aktuelle Thesis Grundlage, da die Schulung in der Radarbildauswertung von der Bundeswehr eingesetzt wird.

2 Related Work

Die eingesetzte Technik im E-Learning-Sektor, die verwendeten Frameworks und Content Management Systeme, haben in den letzten Jahrzehnten eine vielfältige Entwicklung durchgemacht. Im Folgenden wird der aktuelle Stand der Technik vorgestellt.

2.1 State of the Art

Im Verlauf der letzten Jahrzehnte hat das E-Learning eine umfangreiche Entwicklung absolviert, und mittlerweile eine hohe Bedeutung im Bildungssektor erlangt. Angefangen mit dem reinen Computer Based Training (CBT), das mit zunehmender Verbreitung des PCs in den 80er Jahren erfolgte; gefolgt vom Web Based Training (WBT), das durch das Internet möglich wurde, expandierte die Branche.

Aktuell gibt es auf dem Markt der E-Learning-Systeme eine ganze Reihe von Plattformen, die auch beständig weiterentwickelt werden. Einige der bekanntesten Systeme sind die ILIAS-Plattform, Moodle, Blackboard und das Fraunhofer Autoren- und Lernsystem Crayons®. Alle Systeme erlauben es dem Autor Kurse zu erstellen, die von einem Anwender über ein Web-Interface bearbeitet werden können. Dabei unterscheiden sich die Systeme durch Gestaltung des Interfaces, zuschaltbare Spezialfunktionen und unterstützte Formate. [CampS]

Mit dem Aufkommen von Smartphones entwickelte sich mittlerweile eine neue Form, des E-Learnings - das mobile learning. Die kleinen Mini-Computer ermöglichen es an jedem Ort E-Learning-Kurse zu absolvieren. Meist ist hierfür eine aktive Internetverbindung notwendig, um die eingegebenen Daten an den Server zu senden, der die eigentlich Logik enthält.

Bisher sind die auf Smartphones ausgerichteten E-Learning-Angebote noch in der Minderheit, doch nach und nach ändert sich das.

Anfang 2011 startete die Firma Click&Learn einen Weiterbildungskurs für die Fremdsprachen Englisch und Tschechisch, für alle Bewohner in und um die Österreichische Stadt Gmünd, die ein Smartphone besitzen. [C&L2011] Auch das Fraunhofer Institut für Systeme der Kommunikationstechnik ESK hat verschiedene Projekte in der Entwicklung.

Der eCoach erweitert bestehende E-Learning-Systeme um einen elektronischen Trainer, der autonom und dynamisch mit dem Anwender kommuniziert. Dabei bilden dynamische, psychologische Modelle die Grundlage, und das Verhalten und die Persönlichkeit des Nutzers beeinflussen so den eCoach. Durch das adaptive Feedback-

System entsteht ein Lernkreislauf, der durch mobile Geräte flexibel und effizient gestaltet ist. [Comp2007]

Der Mobile MENTOR wird ebenfalls aktuell vom Fraunhofer ESK entwickelt. Hierbei handelt es sich um ein Projekt für Entwicklungsländer. Per Handy oder Smartphone wählen sich die Lernenden in eine E-Learning-Plattform ein. Dabei ist keine Anbindung an das Internet erforderlich, die Verbindung zwischen mobilem Gerät und Server erfolgt über das Handy-Netz. So soll die Bevölkerung in diesen Ländern gerade im Bereich Umweltschutz geschult werden. [ESK2011]

Mit der zunehmend leistungsfähigeren Hardware der Smartphones, ist es auch möglich 3-dimensionale Objekte zu generieren und als Anwender zu manipulieren. Dies bietet auch im Zusammenhang mit dem E-Learning neue Möglichkeiten, gerade für die Simulation von Radarbildeffekten.

Am Fraunhofer Institut IOSB wurde mit dem Learning Management System Crayons® ein System geschaffen, das bei der Bundeswehr eingesetzt wird, um die Interpretation von Radarbildern und Radarbildeffekten zu lehren. Dabei bietet Crayons® für den jeweiligen Tutor die Möglichkeit Kurse zu implementieren, verschiedene Aufgaben zu erstellen und diese den Schülern zur Verfügung zu stellen. Bisher erfolgt der Zugriff auf das jeweilige Lernangebot über ein Web-Interface im Browser.

Für Smartphones gibt es dergleichen bisher jedoch noch nicht.

2.2 Literatur

Im Verlauf dieser Thesis wurde es notwendig, Fachliteratur aus verschiedenen Bereichen durchzuarbeiten. Um die Grundlagen der Android-Entwicklung zu verstehen, leistete das Buch „Android: Grundlagen und Programmierung“ von Arno Becker und Marcus Pant einen großen Beitrag, das 2010 im dPunkt Verlag erschien. Auch eine große Hilfe war „Pro Android 2“ von Sayed Y. Hashimi, Satya Komatineni und Dav MacLean, das 2010 bei Apress erschienen ist.

Eine gute Einführung in OpenGL ES gab der Vortrag von Matthias Braun, den dieser im Wintersemester 2009 an der Hochschule Karlsruhe hielt.

Ergänzend zu dieser Literatur wurde Artikel, Webseiten und Paper zu Rate gezogen, auf die im Quellenverzeichnis referenziert wird.

3 Grundlagen

Gerade im Bereich der Software-Entwicklung auf Smartphones ist das Verständnis der einschränkenden Rahmenbedingungen von essenzieller Bedeutung. So stehen auf einem Handheld-Gerät deutlich weniger Hardware-Ressourcen zur Verfügung, als auf einem PC. Die Programmierung erfolgt über spezielle, vom Hersteller zur Verfügung gestellte Software Development Kits (SDKs). Im folgenden Kapitel werden die Grundlagen zu diesen Elementen auf Software- und Hardwareebene erläutert.

3.1 Android-Smartphones

Durch die ständige Weiterentwicklung der herkömmlichen Handys, entstanden in den vergangenen Jahren Geräte mit einem deutlich erweiterten Umfang an Ressourcen (Hardware) und Anwendungen (Software). Die Geräte erhielten im Zuge dieser Verbesserungen und dem deutlich erweiterten Umfang an Anwendungsmöglichkeiten die Bezeichnung: Smartphones. Deren hauptsächliche Ausrichtung ist dabei nicht länger das Telefonieren, sondern die komfortable Bedienung einer breiten Palette von Anwendungen. Ein wichtiges Element, über das Smartphones verfügen, sind zudem die unterschiedliche Sensoren: Bewegungs-, Magnetfeld-, Licht-, Lage-, und Näherungssensoren, sind in den aktuellen Geräten Integriert.

Abbildung 2: Das Google Nexus S

Für diese Thesis stand ein Android-Smartphone als Plattform zur Verfügung. Das Google Nexus S (siehe: „Abbildung 2: Das Google Nexus S“), bot alle notwendigen Sensoren (Bewegungs- und Lagesensor) und ermöglichte zudem die Entwicklung und Darstellung von 3-D-Objekten. Die Bedienung erfolgte über eine Touch-Oberfläche.

Im Folgenden werden die Details zur Entwicklung von Applikationen für Android-Geräte, das verwendete Programmier-Framework und Grundlagen zum Code vorgestellt. [CT2003]

3.2 App-Entwicklung für Android

Die Programmierung von Software für Android-Geräte erfolgt auf der Basis von Java. Die von Google zur Verfügung gestellte Bibliothek unterscheidet sich aber in den zur Verfügung gestellten Klassen. Einige Elemente, wie Java3D, könnten unter Android nicht verwendet werden. Im Folgenden Details zum Android-SDK.

3.2.1 Das Android-SDK

Um für Android-Smartphones Anwendungen zu programmieren, stellt Google Entwicklern ein SDK zur Verfügung. Mittlerweile ist das Framework, für die App-Entwicklung auf Basis von Java, in der Version 3.1 verfügbar[4]. Ebenfalls zur Verfügung, steht das Android Development Tools (ADT) Plug-In, in der Version 11.0.0. Dieses ermöglicht die Programmierung unter der Entwicklungsumgebung Eclipse, die als Open Source Software zur Verfügung steht. Durch die enthaltene Dalvik Virtual Machine, eine für mobile Geräte entwickelte virtuelle Registermaschine, lassen sich virtuell Geräte simulieren. Das integrierte Debugging-System ermöglicht die Fehlersuche direkt aus der Entwicklungsumgebung heraus.

Neben dem SDK, ist auch das Native Development Kit (NDK) erhältlich. Dies ermöglicht es, Android-Anwendungen in nativen Codesprachen, wie C und C++, zu entwickeln. Enthalten sind Build-Dateien zum erzeugen nativer Code-Bibliotheken. Zudem erlaubt das NDK die Verwendung von Header und Bibliotheken, die den OpenGL ES-Standard unterstützen. Somit ist auch die Entwicklung von 3-D-Inhalten mit dem NDK möglich. [PROA2010]

Im Vergleich zu den Desktop-Anwendungen, spielen bei der Entwicklung von Apps für Android-Smartphones verschiedene Mechanismen des SDK eine besondere Rolle. Für die

[4] http://developer.android.com

Interaktion mit dem Anwender stehen zahlreiche Mechanismen zur Verfügung. Im Folgenden werden diese Elemente des Frameworks, die auch für die Entwicklung der späteren Prototypen notwendig sind, einer gesonderten Betrachtung unterzogen.

3.2.1.1 Die Bestandteile des User Interface

Eine Activity ist zentraler Bestandteil des User-Interface-Konzepts von Android. Sie repräsentiert einen einzelnen Bildschirm innerhalb der Anwendung.

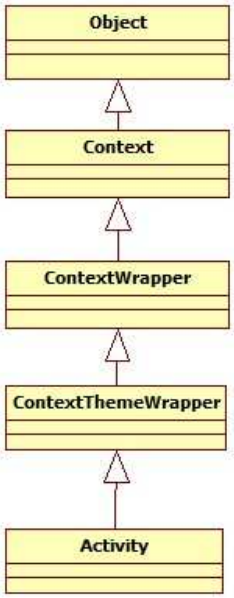


Abbildung 3: Klassen des User-Interface für eine Activity

Dieser wiederum enthält – in der Regel - eine oder mehrere Ansichten (Views). Views wiederum sind Container für bestimmte Arten von Inhalten (Contents). Es existieren jedoch auch Anwendungen, die zwar mit Activitys, jedoch ohne Views arbeiten. Wichtig für die Verwendung von Activitys, ist daher der Intent.

Ein Intent steht generell für das ausführen einer Aufgabe (die Intention). Ein Intent dient somit der Kapselung einer Aufgabe. Im Detail werden Intents für folgende Abläufe verwendet:

- Broadcast einer Nachricht
- Starten eines Service
- Ausführen einer Activity
- Eine Website oder Liste von Kontakten darstellen
- Ein Anruf beantworten oder ausführen

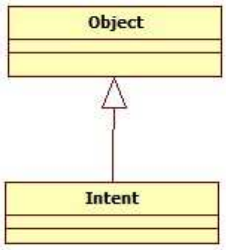


Abbildung 4: Klassen des User-Interface für einen Intent

Das Entwickeln von User Interfaces (UIs) für Android-Smartphones wurde mit der Entwicklung des Frameworks von Google stark vereinfacht. Bisher bestand oftmals das Problem, dass unter anderem Entwickler von Web Anwendungen, die ein bestimmtes Framework verwendeten, auch die darunter angesiedelten Bibliotheken kennen mussten – auf diese setzte das verwendete Framework immerhin auf.

Ein Problem, das bei der Entwicklung mit dem Android-SDK nicht länger besteht. Das Framework wurde simpel gehalten, die User-Interface-Controls auf das wichtigste beschränkt. Die Verfügbaren Views wurden limitiert. Das Konzept wurde komplett auf die Intents ausgerichtet. Ein User möchte genau eine Aktion ausführen, die an genau einen Intent gebunden wird. Dies vereinfacht die Ausrichtung des User-Interfaces auf spezifische Aktionen.

Genau wie die übrigen SDKs, bietet auch das Android-Framework unter anderem die grundlegenden Elemente wie:

- Textfelder
- Buttons
- Listen
- Tabellen

Die wichtigsten Klassen für die User Interfaces sind: android.view.View und android.view.ViewGroup. Alle übrigen Klassen zum Erstellen der unterschiedlichen Views, erben von diesen Basisklassen.

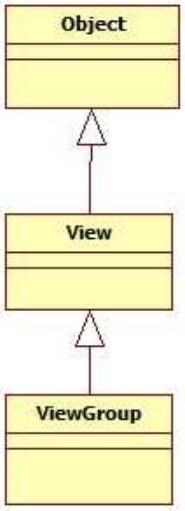


Abbildung 5: Klassen des User-Interface für Views

Während die android.view.View-Klasse zum Erstellen der unterschiedlichen Views dient, enthält die android.view.ViewGroup-Klasse unterschiedliche Untergruppen von Views. Damit ist ViewGroup die Basisklasse für eine umfangreiche Liste an Layout-Klassen. Das Android-Framework verwendet das Konzept von unterschiedlichen Layouts, um festzulegen wie die Kontrollen außerhalb des View-Containers weiter verarbeitet werden. Durch die Verwendung der unterschiedlichen Layouts ist es – wie aus Java bekannt – simpel, die Position der Elemente auf dem User-Interface zu kontrollieren.

Es gibt verschiedene Möglichkeiten, in Android User-Interfaces zu erstellen. Zum einen kann dieses komplett per Code geschehen, zum anderen ist es per XML möglich. Ein Mix aus beidem ist ebenfalls möglich. Die Definition des User-Interface erfolgt in XML, dort wird auf den Code referenziert. [PROA2010]

3.3 Visualisieren mit Android

Neben verschiedenen Layouts und textuellen Darstellung, sind es meist Grafiken, die in einer App die Oberfläche dominieren. Hierbei kann es sich um Bilder, einfache 2-D oder 3-D-Zeichnugnen handeln. Nachfolgend werden die Grundlagen beider Darstellungsarten erläutert.

3.3.1 2-D-Zeichnungen

Das Implementieren von 2-D-Zeichnungen wird in Android durch eine Bibliothek ermöglicht. Hierfür stehen folgende Packages zur Verfügung:

- android.graphics.drawable
- android.view.animation

Drawables bezeichnen generell Objekte, die gezeichnet werden können. Die Klasse Drawable leitet wiederum zahlreiche Unterklassen ab, die spezifische Objekte, die gezeichnet werden können, repräsentieren.

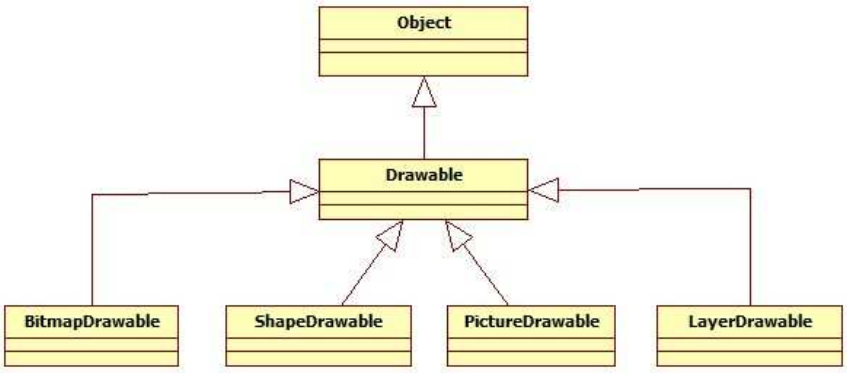


Abbildung 6: Die abgeleiteten Drawable-Objekte

Um eigene Objekte zu definieren, kann der Anwender von den oben genannten Klassen ableiten. Eigene Drawables können über eine Bilddatei referenziert, über eine XML-Datei definiert, oder über den Standardkonstruktor definiert werden.

Nach den 2-D-Zeichnungen, gehören 3-D-Grafiken zu den komplexeren Elementen der App-Entwicklung. Gerade für die Umwandlung von 3-D-Objekten in Radarbilder, müssen diese erst generiert werden. Im Folgenden weitere Details dazu.

3.3.2 3-D-Zeichnungen mit OpenGL ES

Um unter Android 3-D-Zeichnungen zu programmieren, steht die OpenGL ES API zur Verfügung. Die Bibliothek ist ein Abkömmling des OpenGL-Frameworks. Nachfolgend werden das Framework und seine Entwicklungsstufen erläutert.

OpenGL ES basiert auf OpenGL, der bekannten 3D-Grafik-API. Einsatz findet OpenGL auf dem MacOS, Linux, Unix, sowie Windows. Das Framework entstammt ursprünglich der IRIS GL API und ist die Basis einiger weiterer High-Level APIS wie: Open Inventor5, VRML6 und Java 3D 7. OpenGL selbst ist eine low-level-API, die das grafische Rendering in eine Hardware Pipeline auslagert, die wiederum auf verschiedene Hardware-Rümpfe aufsetzt. Die einfach gehaltene Architektur hält das Framework kompakt. Alle Komponenten sind modular aufgebaut. So ist es leicht, einzelne Features zu deaktivieren oder zu aktivieren.

OpenGL wurde allerdings nicht entwickelt, um die Größe des Frameworks und des Speichers gering zu halten. Stattdessen sollte eine Plattform- und Programmiersprachenunabhängige Systemschnittstelle entstehen, die komplexe Grafiken erstellen kann. Zudem beinhaltet die API einige Module, die zum Zeichnen von 2-D-Grafiken entwickelt wurden – wie User-Interfaces.

Um die Entwicklung von 3-D-Grafiken zu unterstützen, hat die Khronos Group auf der Grundlage von OpenGL ein leichteres, auf Smartphones ausgerichtetes Framework entwickelt: OpenGL ES. Hauptaugenmerk lag auf der Verkleinerung des Designs und der verringerten Inanspruchnahme von Ressourcen mobiler Geräte. Der API-Footprint sollte verkleinert werden, Redundanzen verschwinden. In Folge entstand, in mehreren Schritten, das OpenGL ES-Framework. [COSTE2009]

3.3.2.1 OpenGL ES 1.0

Die erste Version des neuen Frameworks ermöglichte eine kompakte und effiziente Implementierung auf mobilen Geräten wie Smartphones, die lediglich Hardware Support für Integer Zahlen unterstützten. Anfang 2002 wurde OpenGL ES standardisiert, die Spezifikation wurde im Sommer des Jahres 2003 publiziert. OpenGL ES 1.0 arbeitet mit Vertex-Arrays die einzelne Vertex-Positionen definieren, einfache Vektoren, Textur-Koordinaten, Farben und Material implementieren. Die Vertices werden hier durch Modelview- und Projektion-Matrizen transformiert.

Die einzelnen Vertices werden über Linien miteinander verbunden, wodurch einfache, auf Dreiecken basierende Objekte entstehen.

---

[5] http://oss.sgi.com/projects/inventor

[6] http://www.web3d.org/x3d/specifications/#vrml97

[7] http://java3d.java.net

Texture-Maps geben dem Entwickler die Möglichkeit, Farben zu verwenden und diese mit der Farbe des Grundlagenmodells zu kombinieren. Verschiedene Tests stehen zur Verfügung: auf Tiefe, Alpha-Wert und einige mehr, um die Farbe zu generieren.

Der erste Schritt in der Entwicklung, die Vereinfachung des ursprünglichen Frameworks, wurde durch Verschlankung von OpenGL 1.3 erreicht.

Es ist kein direkter Zugang zum Bildspeicher – dem Frame Buffer – des Video-RAM gegeben, der digitale Abbilder der Monitoranzeige enthält. OpenGL ES enthält daher alle Blending Modi und logischen Operationen, um die Problematik zu umgehen. Die Textur-API wurde an einigen Stellen vereinfacht, ihre Funktionalität aber kaum eingeschränkt.

Eine Menge Veränderungen gab es auch bei den Texturen. In OpenGL werden drei Arten von Texturen unterschieden. 1D-Textur messen immer einen Pixel in der Höhe, während die Breite frei wählbar ist (in Abhängigkeit von der Hardware jedoch auf 2^n+1 beschränkt). Die häufigste Umsetzung von 1-D-Texturen sind Lookup-Tabellen. Der Einsatz erfolgt in der Abschwächung von Lichtquellen (der Abschwächungsfaktor liegt hier in der 1-D-Textur) oder in der Abschwächung nicht fotorealistischer Effekte.

Die wohl bekannteste Art von Texturen sind 2-D-Texturen. Sowohl Breite als auch Höhe sind hier frei wählbar. Durch sie werden in der 3-D-Computergrafik Oberflächen von Modell angepasst - zur Simulation von Materialeigenschaften.

3-D-Texturen besitzen nun auch noch eine Tiefe. So können volumetrische Effekte erzeugt werden. Gerade im medizinischen Sektor finden sie weite Verbreitung, da das Volumen auch in Frames zerteilt einzeln abgerufen werden kann. 3-D-Texturen haben auf der anderen Seite aber einen recht hohen Speicherverbrauch.

In OpenGL ES wurden Texturen über 1-D und 3-D aus dem Framework entfernt, nur 2-D-Texturen sind geblieben. Diese sind wiederum in der Lage, 1-D-Texturen zu emulieren. Damit stehen die am häufigsten verwendeten Texturen noch immer zur Verfügung, während jene mit dem höchsten Speicherverbrauch, nicht länger angewendet werden können.

Das Entfernen multipler Redundanzen macht OpenGL ES gegenüber dem Desktop-Framework deutlich schlanker. Alle 3-D-Operationen von OpenGL können durch die verbliebene 2-D-Funktionalität emuliert werden – auch wenn ein großer Teil davon entfernt wurde. OpenGL ES setzt an vielerlei Stellen auf Emulation. Ein weiteres Beispiel hierfür ist das Zeichnen eines 2-D-Vierecks über die Emulation der Funktionen line stippling und drawpixels mittels Textur-Mapping.

Trotz der Vereinfachungen, die OpenGL ES in der Version 1.0 erfuhr, war es immer noch komplexe, einfache, primitive Typen zu erstellen. Daher erfuhr das Framework in der nächsten Version einige weitere Änderungen. [COSTE2009]

3.3.2.2 OpenGL ES 1.1

OpenGL ES 1.1 wurde ein Jahr nach der ersten Version fertiggestellt. Während die Vorgängerversion primär verschlankt wurde, ging es bei 1.1 darum, grafische Operationen besser zu unterstützen. Mehr neue Ressourcen wurden hinzugefügt, darunter eine bessere Unterstützung für Grafik-Hardware.

Insgesamt sieben Neuerungen wurden implementiert, fünf von diesen Pflicht, zwei Optional. Eine bessere Unterstützung für das Textur-Mapping, war eine der Verbesserungen. Multitexturing funktioniert nun nur noch auf Basis von zwei implementierten Textur-Einheiten.

Dot3-Texture-Mapping ist ebenfalls neu implementiert. Dieses ermöglicht das Rendering von Objekten, die nur durch eine geringe Anzahl an Polygonen dargestellt werden. Trotzdem erfolgt eine sehr detaillierte Darstellung.

In Version 1.1 können die Vertex-Daten in einem Vertex-Buffer-Objekt gespeichert werden. Point sprites erlaubt die Definition von Texture-Maps, die durch einen einzelnen Vertex-Punkt projiziert werden können (unter Verwendung von zwei Dreiecken und vier Vertices).

Version 1.1 des ES-Frameworks führte zudem erstmals Vertex-Arrays ein, die alle Informationen in einigen Arrays speichern. Diese Arrays werden durch einen einfachen Funktionsaufruf an die Grafik-API weitergegeben. Eine einfache und schnelle Implementierung war gegeben. Damit war es nur noch auf diese Art möglich, primitive OpenGL ES-Objekte zu implementieren.

Auch der Umfang, der primitiven Objekte wurde reduziert. Nur Punkte, Linien, und dreieckbasierte Objekte, wurden erhalten. Quadrate und weitere Polygone wurden entfernt. Um diese nach wie vor darzustellen, werden sie aus Dreiecken zusammengesetzt, und intern fürs Rendern wieder gesplittet. OpenGL verwendet für das Grafik-Rendering Floating-Point Zahlen, was mit einem typischen Handheld-Gerät nicht zu realisieren ist – ein Hardware-Support für IEEE-Floats besteht nicht. Die Arbeit mit reinen Ganzzahlen ist jedoch sehr langsam.

Ein neuer Datentyp wurde für das Framework erstellt: glFixed - eine 32-bit-fixed-point Zahl. Diese setzt sich zusammen aus einem 16-bit signed integer, gefolgt von einem 16-bit-Dezimalzahl Bereich. Jede Floating-Point Operation wird durch ein solches fixed-point-Argument ersetzt.

Im Folgenden kam es bis zur aktuellen Version 4.1 im Juli 2010 zu weiteren Anpassungen, die für die Umsetzung des Prototypen in dieser Masterthesis nicht von Bedeutung sind.

[COSTE2009]

Abschliessend ein kurzer Überblick über die Evolution von OpenGL ES über die vergangenen Jahre.

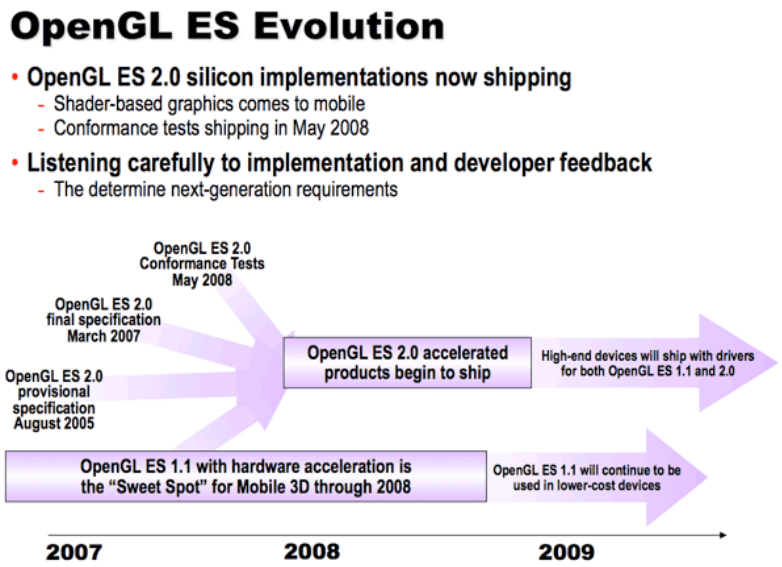


Abbildung 7: Die Entwicklung von OpenGL ES über die Jahre[8]

Für die Funktionen des geplanten Prototypen, der im Verlauf der Masterthesis entstehen soll, ist lediglich OpenGL ES 1.1 notwendig. Daher wird auf einen detaillierten Ausblick der späteren Versionen verzichtet.

[8] http://www.khronos.org/opengles

4 3-D-Modellierung

Mit Hilfe der Computergrafik können 3-Dimensionale Objekte dargestellt (modelliert) werden. Die Darstellung kann dabei in unterschiedlicher Form erfolgen, sehr häufig sind es Drahtgittermodelle, die erzeugt werden. Durch Manipulation der Oberflächeneigenschaften, wie dem hinzufügen einer Textur, wird die Darstellung weiter angepasst. Eine wichtige Rolle spielt zudem das eingesetzte geometrische Modell. Dieses beschreibt die Form geometrischer Objekte (im zweidimensionalen wie dreidimensionalen Bereich). Im Folgenden werden zwei Modelle vorgestellt, die für die Darstellung von Objekten in der Radarbildauswertung von Bedeutung sind.

4.1 Vektorbasierte Modelle

Vektorbasierte Modelle arbeiten mit der Darstellung linienhafter Objekte mittels Punkten und Linien. Während Rasterdaten (siehe hierzu „Kapitel 4.2: Rasterung“) als areale Modelle bezeichnet werden, handelt es sich bei Vektormodellen um lineare Modelle. Aus dem Blinkwinkel einer 3-dimensionalen Umgebung, wird auch von Drahtmodellen (Vektormodell) und Volumenmodellen (Rastermodell) gesprochen. Insbesondere für die Darstellung von einfachen, linienbasierten Objekten wie Straßen und Flüssen, ist ein Vektormodell gut geeignet. Dieses benötigt, im Vergleich zum Rastermodell, deutlich weniger Speicherplatz.

Verknüpfungen werden in einem solchen, georelationalen Modell, eindeutig definiert. Basiselemente können nun zu verzweigteren Strukturen verschmolzen werden, wodurch komplexe grafische Gefüge entstehen.

Modelle dieser Art werden in der Computergrafik unter Einsatz eines Rasterverfahrens in computerlesbare Bilder wie eine Bitmap umgewandelt. Wie eine solche Rasterung verläuft, wird im nächsten Kapitel erläutert. [Rob2010]

4.2 Rasterung

Die Rasterung ist in der Computergrafik unter dem Namen „Rendern“ bekannt und bezeichnet die Umwandlung einer Vektorgrafik in eine Rastergrafik (also ein Bild, das aus computerlesbaren Daten besteht). Eine solche Rastergrafik wird auch als Rastermodell bezeichnet. Bei einer solchen Umwandlung kommen verschiedene Arten von Algorithmen zur

Anwendung, je nach Art des grafischen Primitiv (Linie, Polygon, Kreis, Ellipse), das umgewandelt werden soll. Ausgangsbasis für alle Rasterungen ist jedoch die Linie.

Bei diesem Verfahren werden jene Punkt farbig markiert, deren Näherung sich möglichst exakt an der Linie befindet. Je nach Verfahren werden die Pixel nur einfarbig markiert (ältere Algorithmen) oder erhalten eine farbliche Abstufung (neuere Verfahren). Abbildung 2 zeigt die farbigen Markierungen, im Vergleich zur blau gezeichneten, durchgehenden Originallinie.

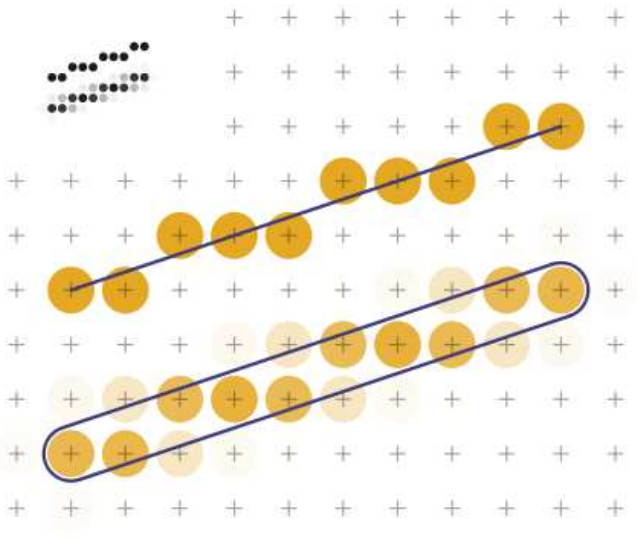


Abbildung 8: Scan-conversion of a line [PHROOD]

Die Rasterung von Linien erfolgt durch unterschiedliche Algorithmen:

- Der Bresenham-Algorithmus
- Pixelreihen
- Das N-Schritt-Verfahren
- Die Bidirektionale Rasterung

In der 3-D-Modellierung wird die Rasterung eingesetzt, wenn Vektorgrafiken als Ausgangsmaterial vorliegen. Da bei der Arbeit mit ViSAR jedoch ein Höhenmodell zum Einsatz kommt, und nicht mit Vektorgrafiken gearbeitet wird, soll die Rasterung hier lediglich kurz erläutert werden. Wie ein solches, von ViSAR verwendetes, Höhenmodell aussieht, wie genau

es erstellt wird und welche mathematischen Grundlagen diesem zugrunde liegen, wird im Folgenden erläutert. [Bond2002]

4.3 Digitale Höhenmodelle

Das Digitale Höhenmodell (DHM) dient als Oberbegriff für Digitales Geländemodell (DGM) und Digitales Oberflächenmodell (DOM). Während das DOM die Erdoberfläche mit allen darauf befindlichen Objekten darstellt, repräsentiert das DGM lediglich die Oberfläche. Das Digitale Höhenmodell ist also, je nach Fachdisziplin, beides.

Bereits seit vierzig Jahren werden Höhenmodell durch Luftbildfotogrammetrie erstellt. Dabei gibt es folgenden Ablauf:

Das Erstellen der Luftbildaufnahmen

Aus der Luft werden Aufnahmen der Geländepunkte angefertigt. Dabei ist zu beachten, dass die Aufnahmen einen Überdeckungsgrad von minimal 60 % besitzen müssen. Nur so ist eine spätere, lückenlose Rekonstruktion möglich. Nun wird die Position der Kamera gespeichert, zusammen mit den X- und Y-Koordinaten der Bildstrahlen. Es entsteht ein X-Y-Z-Triple.

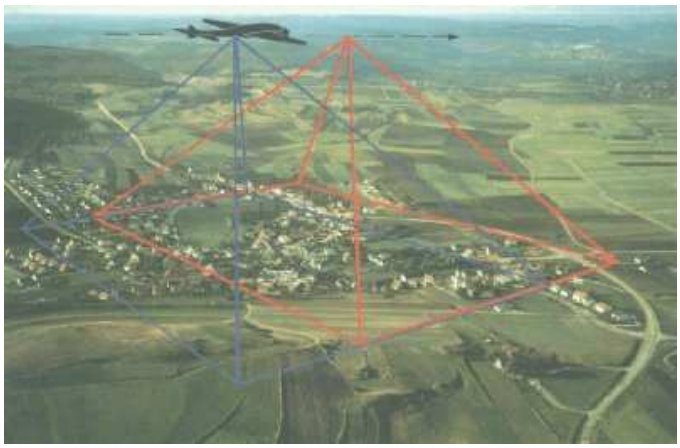


Abbildung 9: Luftbildaufnahmen mit Bildstrahlen[9]

[9] http://satgeo.zum.de/satgeo/methoden/DHM_Web/Bilder/Erstellung/Photogr_1.jpg

Registrierung: Koordinatenbestimmung von sich entsprechenden Punkten

Im nächsten Schritt beginnt die Rekonstruktion. Hierfür wird versucht, Deckungsgleiche Punkte auf den verschiedenen Fotos zu finden. X- und Y-Koordinaten werden dafür abgeglichen.

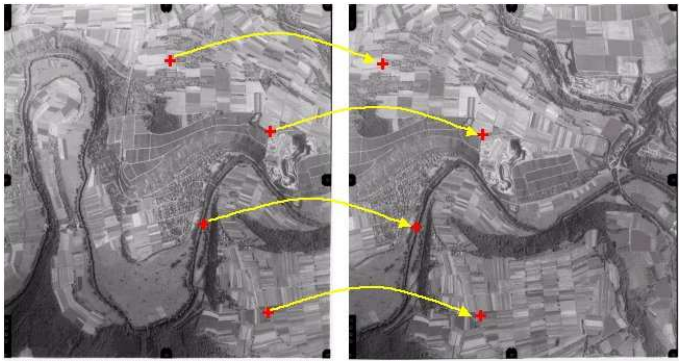
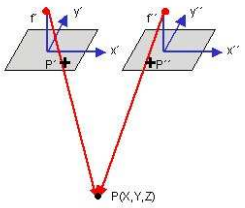


Abbildung 10: Abgleich der X- und Y-Koordinaten[10]

Durchführen der Rekonstruktion

Unter Verwendung mathematischer Methoden, die den gespeicherten Z-Punkt (also die Position der Kamera) und die gegebenen Winkel berücksichtigen, werden die Bildstrahlen nun rekonstruiert.

[10] http://satgeo.zum.de/satgeo/methoden/DHM_Web/Bilder/Erstellung/Photogr_2.jpg



Schritte:

1. Kamerastandpunkte bekannt
2. Bildkoordinaten von entsprechenden Punkten messen
3. Bildstrahlen rekonstruieren
4. Schnittpunkt der Bildstrahlen bestimmen

Abbildung 11: Die einzelnen Schritte zur Berechnung[11]

Es gibt zahlreiche, mathematische Verfahren, die zur Erstellung eines Digitalen Höhenmodells eingesetzt werden. [DigHoeh]

Ein solches Digitales Höhenmodell wird auch durch das am Fraunhofer Institut IOSB entwickelte ViSAR-Programm berechnet. Im Folgenden Kapitel weitere Details hierzu.

[11] http://satgeo.zum.de/satgeo/methoden/DHM_Web/Bilder/Erstellung/Photogr_3.jpg

5 ViSAR

Der SAR-Tutor, ein vom Fraunhofer IOSB entwickeltes E-Learning-System, ermöglicht die Präsentation und Aufbereitung von Wissen, über elektronische Lerninhalte. Ein solches System wird auch von der Bundeswehr eingesetzt, um Schulungen im Bereich der Radarbildauswertung durchzuführen. Für diese Auswertungen ist das Programm ViSAR (Visualization of Geometric SAR Effects) integriert. Das Programm generiert ein Radarbild, anhand eines vorhandenen 3-D-Objektes. Aus einem einfachen Gebäude oder Primitiv, wird ein Höhenmodell (siehe hierzu „Kapitel 4.3“), und in weiterer Folge das Radarbild erzeugt.

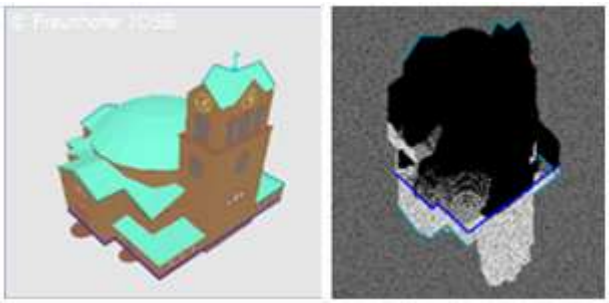


Abbildung 12: Ein 3-D-Objekt (links) wird zum Radarbild (rechts)

Für die Anfertigung des späteren Prototypen wird auch auf ViSAR zurückgegriffen. Dafür ist ein Verständnis des Klassenaufbaus und der Funktionalität wichtig, da diese auch in die App eingefügt werden. Im Folgenden ein Überblick zum Aufbau der Klassen des Programms.

5.1 Technischer Aufbau ViSAR

Für eine klare Strukturierung wurden die einzelnen Klassen in verschiedene Packages aufgeteilt. Dargestellt wird dies in „Abbildung 13: Übersicht der ViSAR-Pakete“. Während das DEM-Paket alle Dateien zur Generierung des Digitalen Höhenmodells beinhaltet, kümmern sich die Klassen im Algorithm-Package um die Erzeugung des Radarbildes. Die grafischen Klassen verwenden Swing, um das User Interface zu erzeugen, während die Input-Dateien (VRML- und PNG-Files) im Model-Package liegen.

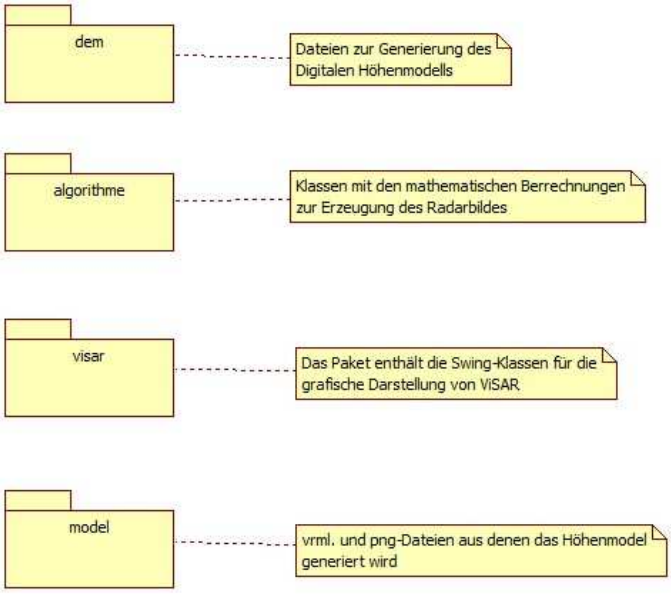


Abbildung 13: Übersicht der ViSAR-Pakete

Die in „Abbildung 14: Die Klassen des DEM-Paketes“ dargestellten Klassen dienen der Erstellung jener Datei, die das digitale Höhenmodell beinhaltet. Als Input wird das 3-D-Objekt in Form einer Virtual Reality Modeling Language (VRML)-Datei benötigt sowie eine Portable Network Graphic (PNG)-Datei, in dem über eine Graustufencodierung das eigentlich Bild erzeugt werden kann.

Das DEM-Interface wird durch die Klasse „DEM_impl“ realisiert.

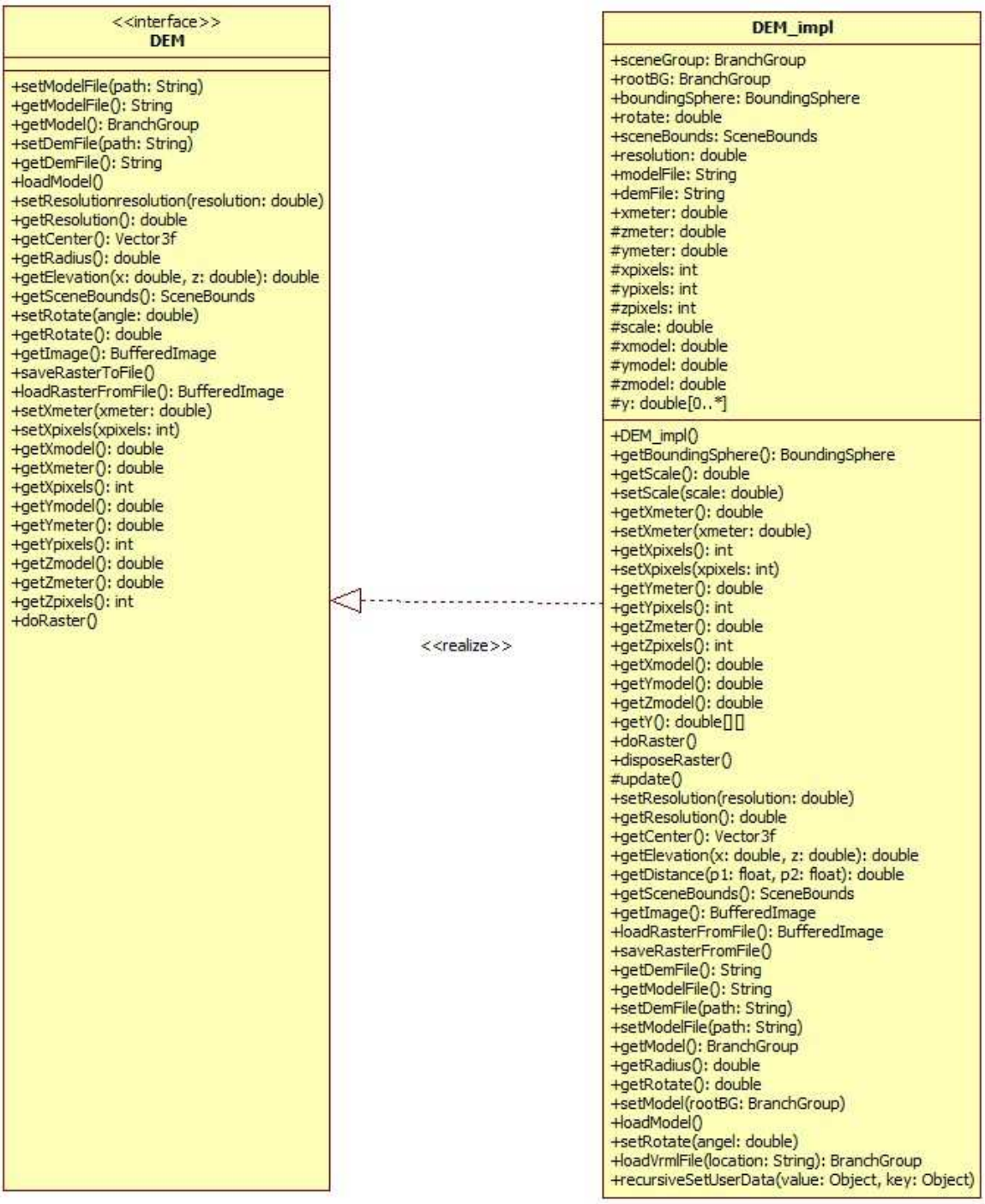


Abbildung 14: Die Klassen des DEM-Paketes

Die in „Abbildung 15: Die Processor-Klassen“ dargestellten Klassen dienen der Berechnung von Layover und Schatten, ohne dabei das Material oder den Aspektwinkel zu berücksichtigen. Die Klasse „SAR_Processor_impl“ realisiert die Methoden des „SAR_Processor“-Interface. Die zentrale Funktion beinhaltet die „run“-Methode, die den Großteil der Klasse ausmacht. In ihr erfolgt die eigentliche Berechnung von Layover und Schatten.

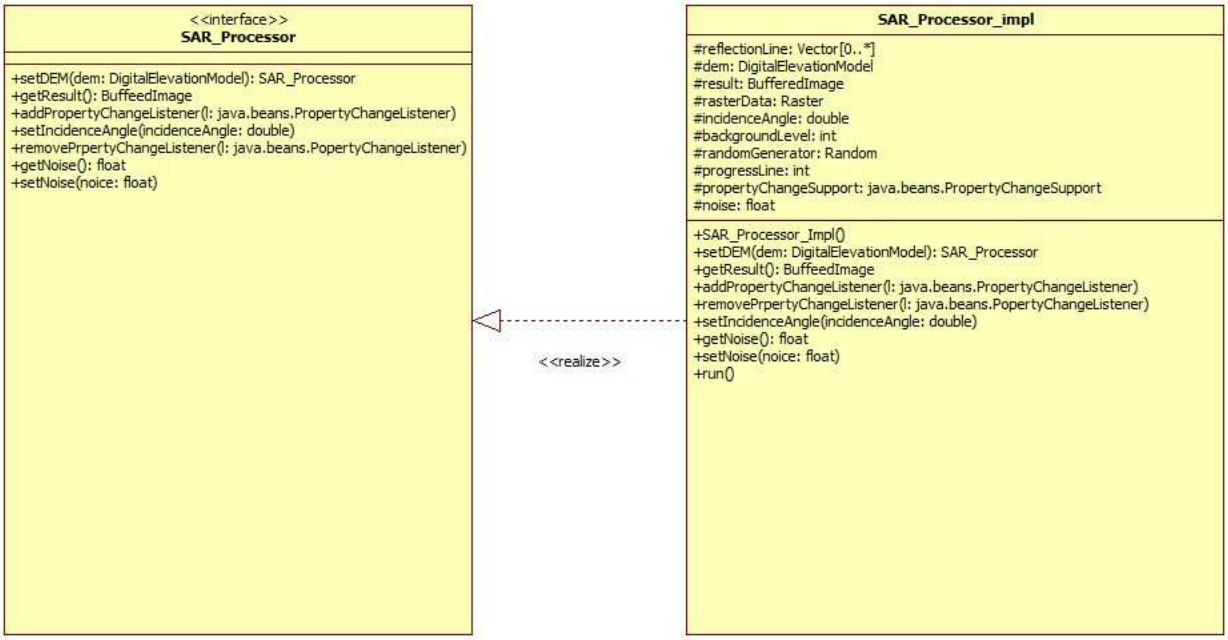


Abbildung 15: Die Processor-Klassen

Neben diesen Hauptklassen gibt es noch eine ganze Reihe weiterer Klassen, die mathematische Funktionen, Filterungen und Simulationen durchführen. Diese alle aufzuführen würde jedoch den Rahmen sprengen, weshalb lediglich die wichtigsten Klassen in einem Überblick aufgeführt wurden. Einen detaillierten Blick auf die Funktion der Klassen und deren Inhalt, erfolgt in: „Kapitel 6.3:

Realisierung“

5.2 Verwendete Version

Um die umfangreichen 3-D-Funktionen des Programms umzusetzen, wurde für die ursprüngliche ViSAR-Anwendung die Java 3D-Bibliothek eingesetzt. Diese kann auf der Android-Plattform nicht eingesetzt werden und muss daher bei der Portierung komplett durch Elemente der Android-Bibliothek ersetzt werden.

In neueren ViSAR-Versionen stehen deutlich mehr Funktionen zur Verfügung, gleichzeitig wird jedoch auch mehr Java3D eingesetzt. Komplexere Schattenberechnungen und diverse weitere Elemente sind für die Anwendung, die auf dem Android-Smartphone laufen soll, nicht erforderlich. Aus diesem Grund wurde eine ältere Version der Anwendung aus dem Jahr 2009 ausgewählt, die portiert werden soll.

5.3 ViSAR-Desktop-Standalone-Version

Da zur Anwendung auf dem Smartphone lediglich jene Methoden verwendet werden, die aus den übergebenen Winkeln und dem bereits generierten Höhenmodell das Radarbild erzeugen, kann das gesamte Package für die Darstellung entfernt werden.

Zudem steht bereits eine PNG-Datei und die 3-D-Objektdatei (Vrml-Datei) zur Verfügung. Es ergibt sich eine schlankere Anzahl an Paketen.

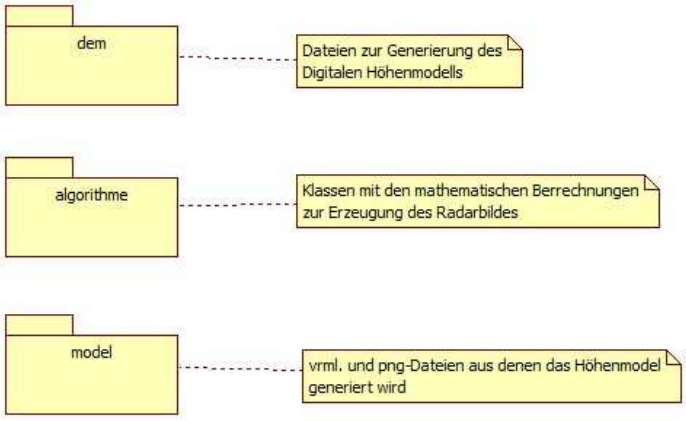


Abbildung 16: Die Packages der Standalone-Version von ViSAR

Die verbleibenden Klassen werden verwendet, um den Input zu verarbeiten, und das fertige Radarbild in einen separaten Ordner auszugeben. In einem späteren Schritt soll diese Version nun in eine Android-App umgewandelt werden, um im Prototypen des 3-D-Sensor-Simulators eingesetzt zu werden. In diesem Schritt geht es also lediglich darum, den ViSAR-Algorithmus bzw. die Klassen der 2009er-Version abzuspecken und auf das notwendigste zu beschränken. Wie die verwendeten Klassen Funktionieren, welche Funktion sie genau implementieren, und wie die neue Oberfläche aussieht, wird im Folgenden „Kapitel 6: 3-D-Sensor-Simulator“ von der Konzeption bis zur Implementierung ausführlich erläutert.

6 3-D-Sensor-Simulator

Auf Basis der erarbeiteten Grundlagen und dem bestehenden ViSAR-System wird im Folgenden ein Prototyp erstellt. Dieser soll ein bestehendes 3-D-Objekt, dargestellt auf dem Handy-Bildschirm, in ein Radarbild umwandeln. Hierfür wird das zuvor besprochene OpenGL ES verwendet, um das Grafikobjekt zu erstellen, und der ViSAR-Simulator, zur Umwandlung des 3-D-Objektes in ein Radarbild.

6.1 Konzeption

Zu Beginn steht die Frage, inwieweit der Anwender manipulierend auf das 3-D-Objekt zugreifen kann. Die Oberfläche soll in Form von Tabs eine Auswahl zwischen dem Objekt und dem Generieren des Radarbildes ermöglichen.

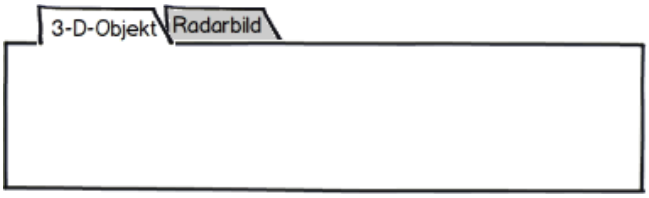


Abbildung 17: Mockup der Tab-Oberfläche

Beim Start der Anwendung gelangt der Anwender direkt in die Ansicht für den 3-D-Würfel.

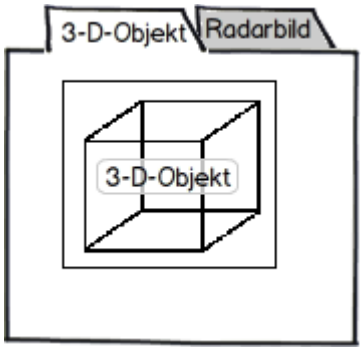


Abbildung 18: Start-Tab mit 3-D-Würfel

Nun soll der Anwender durch Berührung des Würfels diesen drehen, sowie das Objekt über durch eine Zoom-Funktion über Gesten (Multi-Touch) im Detail betrachten können. Unter Ausnutzung der gerätespezifischen Lagesensoren, soll der Würfel sich zudem bewegen, wenn das Smartphone die Lage ändert. Durch die im oberen Bereich des Bildschirms angeordneten Tabs kann zwischen der normalen Ansicht auf das Objekt und dem Radarbild hin und her geschaltet werden.

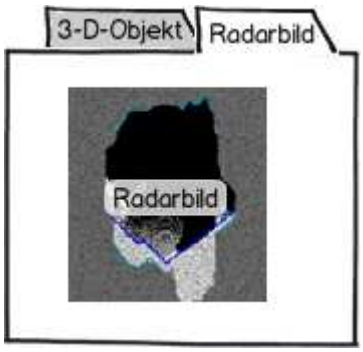


Abbildung 19: Generiertes Radarbild im zweiten Tab

Der Anwender kann also die Ansicht wechseln, sowie verschiedene Manipulationen des Würfels in der 3-D-Ansicht vornehmen. Abbildung 20: Ein Anwendungsfalldiagramm für den 3-D-Sensor-Simulator zeigt die Möglichkeiten des Anwenders anhand eines Anwendungsfalldiagramms (engl.: Use-Case-Diagramm) auf.

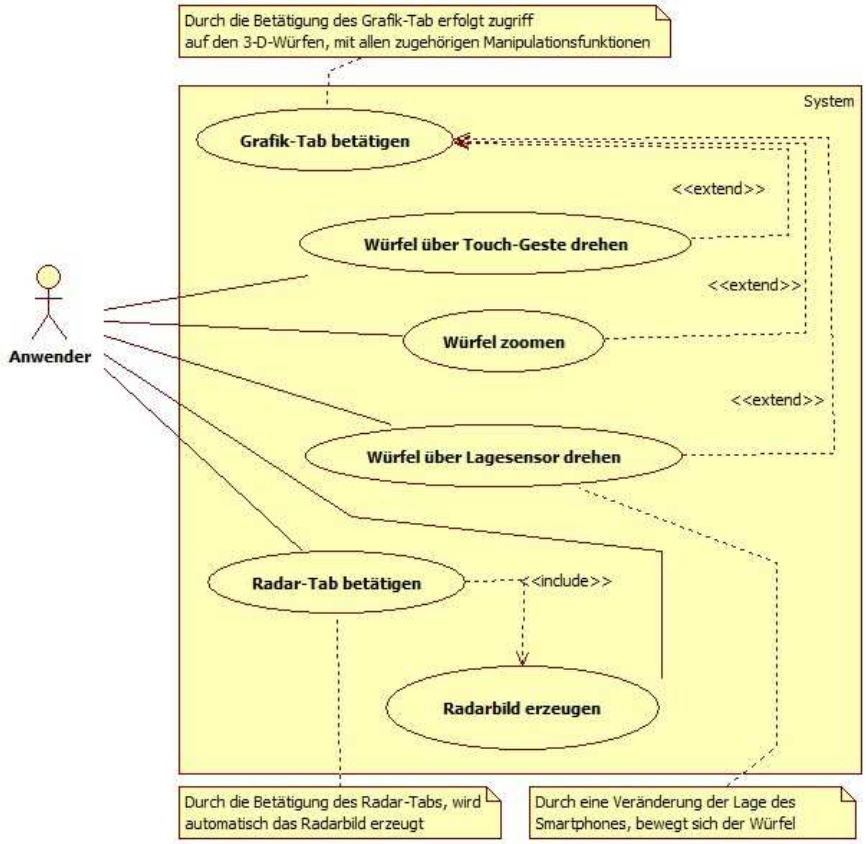


Abbildung 20: Ein Anwendungsfalldiagramm für den 3-D-Sensor-Simulator

Einzig das Radarbild kann der Anwender in keiner Form direkt manipulieren. Lediglich durch die Veränderung der Lage des ursprünglichen 3-D-Würfels, und des darauffolgenden Erzeugen des Radarbildes, kann dieses verändert werden. Neben der Eingabe des Anwenders, kann das 3-D-Objekt also auch durch den Einsatz des Lagesensors verändert werden. Das folgende Kapitel beleuchtet die Sensoren des Android-Smartphones.

6.2 Hardwarespezifischen Sensoren

Das in der Thesis verwendete Nexus S von Google enthält, wie nahezu alle aktuellen Smartphones eine ganze Reihe an Sensoren.

Die Messung der Lage, der Geschwindigkeit, die Kompassrichtung, das magnetische Feld und die Neigung, können über die internen Sensoren ausgelesen werden. Durch unterschiedliche Apps werden die Daten grafisch aufbereitet und auf dem Bildschirm ausgegeben. Im Folgenden werden zwei der Sensoren näher vorgestellt.

6.2.1 Näherungssensor (Näherungsschalter)

Die Messung der Annäherung eines Objektes an den Sensor erfolgt berührungsfrei, also ohne Kontakt zwischen Sensor und Objekt. Generell wird bei diesen Sensoren zwischen verschiedenen Arten unterschieden: induktiver, kapazitiver, magnetischer oder optischer Näherungsschalter. In aktuellen Smartphones kommen induktive und kapazitive Näherungssensoren zum Einsatz.

Ein induktiver Sensor dient zum berührungslosen Erkennen elektrisch leitfähiger Objekte. Aufgebaut ist ein solcher Sensor aus drei Bestandteilen: ein Oszillator, eine Auswerteeinheit und eine Ausgangsstufe. Mit dem Anlegen einer Spannung am Oszillator, beginnt dieser zu schwingen. So entsteht ein elektrisches Feld, das mittels eines Ferritkerns zur aktiven Fläche ausgerichtet wird. Nähert sich nun ein Objekt an, entzieht dieses dem ausgerichteten Schwingkreis Energie. In der Folge wird die Oszillatorspannung kleiner, was von der Auswerteeinheit erkannt wird.

Der kapazitive Näherungsschalter kann neben Leitenden auch nicht leitende Objekte detektieren. Im Zentrum steht erneut ein Oszillator. Im Gegensatz zu seinem induktiven Pedant, bildet sich die frequenzbestimmende Kapazität teilweise durch das zu detektierende Medium. Wird nun ein leitendes oder nicht leitendes Objekt angemessen, ändert sich diese Kapazität um einen materialspezifischen Wert. [IFM]

6.2.2 Beschleunigungssensor (Accelerometer)

Ein Beschleunigungssensor bestimmt die auf eine Testmasse einwirkende Trägheitskraft, wodurch eine ständige Überprüfung auf Zu- oder Abnahme der Geschwindigkeit stattfindet. Umgesetzt wird ein solcher Beschleunigungsmesser durch a) Piezoelektrische

Sensoren oder b) Mikrosysteme. In erstem Fall erfolgt die Messung der Geschwindigkeit durch Sensorplättchen, die Druckschwankungen in elektrische Signale wandeln. Es sind aber die Mikrosysteme, die in aktuellen Handys zum Einsatz gelangen. Hergestellt aus Silizium, bestehen die Sensoren aus Federn, die nur wenige Mikrometer Breit und mit einer festen Bezugselektrode verbunden sind. So kann eine Änderung in der elektrischen Kapazität angemessen werden.

In aktuellen Smartphones entstehen aus diesen Mikrosystemen 3-Achsen-Sensoren, die Beschleunigung und Lage auslesen können. Erkennt das Gerät, dass sich die Lage ändert, wird automatisch zwischen der vertikalen und der horizontalen Anzeige umgeschaltet. Die Anzeige auf dem Display „dreht“ sich.

Im Folgenden wird das Konzept realisiert. [BESCHL]

6.3 Realisierung

Die Realisierung der Radarkonverter-Funktionalität erfolgt in mehreren Schritten. Zu Beginn steht die Realisierung des sehr einfach gehaltenen User-Interfaces (siehe: „Kapitel 6.3.1: Umsetzung: Tab-Oberfläche“). Daraufhin wird der Würfel, gefolgt von der Umsetzung des Lagesensors und der Touch-Funktion implementiert. Den Abschluss macht ein Überblickt der gesamten Klassenstruktur, um das Zusammenspiel der einzelnen Elemente zu verdeutlichen, bevor die ViSAR-Integration zur Radarbildgenerierung erfolgt.

6.3.1 Umsetzung: Tab-Oberfläche

Mit dem Start der Applikation wird die Klasse „Radarkonverter“ aufgerufen.

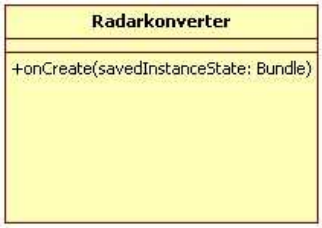


Abbildung 21: Die Radarkonverter-Klasse

Damit der Anwender im späteren Verlauf zwischen dem 3-D-Objekt und dem Radarbild navigieren kann, erzeugt diese Klasse direkt die Tab-Oberfläche. Initialisiert wird die Anwendung mit der Ansicht auf das 3-D-Objekt, dessen Umsetzung im nächsten Kapitel besprochen wird. Durch die Betätigung des „Radar“-Tabs, wechselt die Ansicht automatisch in den anderen Tab, das Radarbild wird generiert.

Innerhalb der Klasse wird zuerst ein TabHost erzeugt, über den die einzelnen Ansicht generiert und mit einer jeweils eigenen Klasse verknüpft werden. Pro Ansicht wird hier ein Intent verwendet.

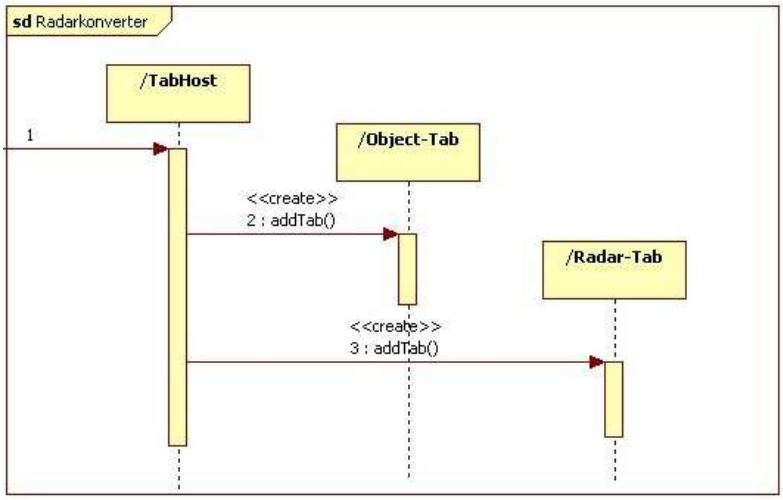


Abbildung 22: Erzeugung der Tab-Oberfläche

Daraus resultiert eine leere Tabelle, die der Anwender als Ausgabe auf dem Smartphone erhält (siehe: „Abbildung 23: Die Tab Oberfläche“).

Abbildung 23: Die Tab Oberfläche

Damit ist die Grundlage erzeugt. Im nächsten Schritt wird der Objekt-Tab mit einem 3-dimensionalen Objekt gefüllt.

6.3.2 Umsetzung: Der Würfel

Für den Prototypen wird nun ein fest vorgegebenes 3-D-Objekt generiert: ein Würfel. Hierzu wird das bereits erwähnte OpenGL ES verwendet, durch das grafische Elemente unter Android programmiert werden können. Die zentrale Klasse für die Umsetzung des Würfels, ist „OpenGLSensors“. Diese greift auf „GLSurfaceViewNoThread“ zu. „Cube“ dient der Festlegung aller Würfeleigenschaften.

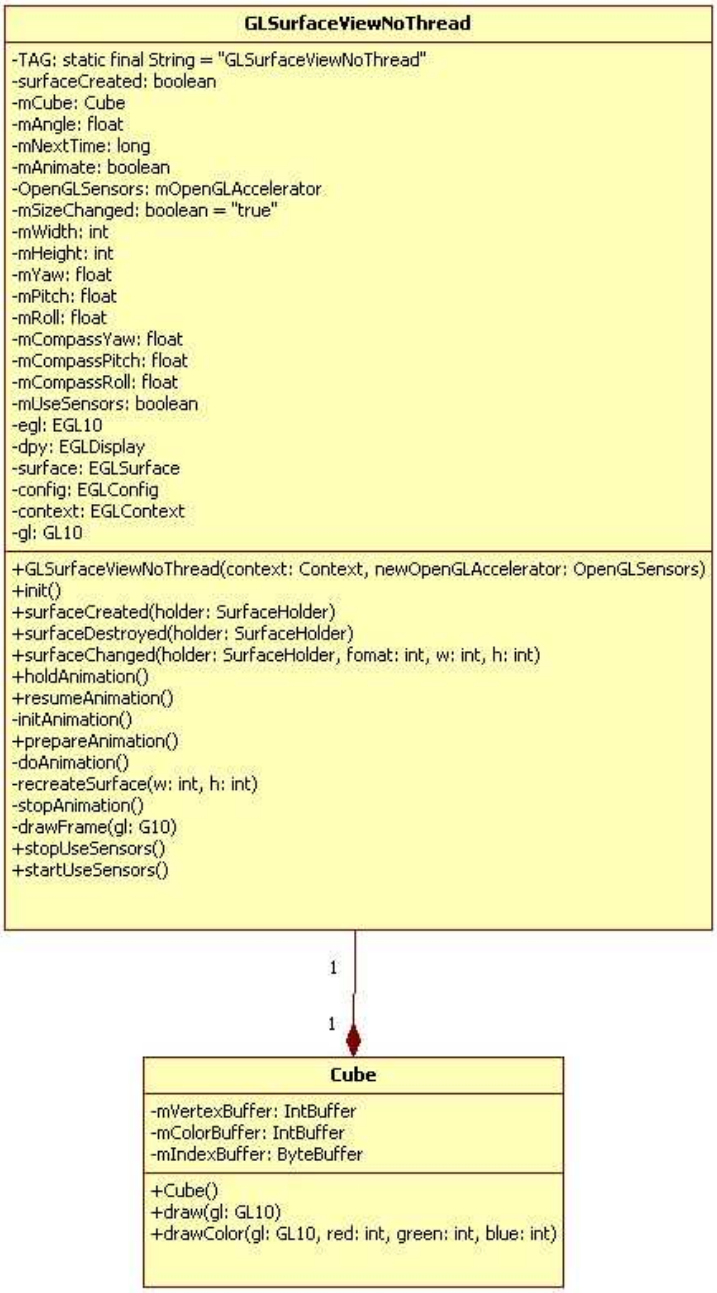


Abbildung 24: Die OpenGL ES-Klassen für die Umsetzung des Würfels

Die „OpenGLSensors“-Klasse dient der Auslesung des Lagesensors, um den Würfel zu drehen. Hierzu folgen weitere Details im nächsten Kapitel. Für die Zeichnung des Würfels ist „GLSurfaceNoThread“ zuständig. Hier werden die Eigenschaften des 3-D-Objekts, die in der Klasse „Cube“ festgelegt werden gezeichnet. Auf dem Handy entsteht so ein farbiger, 3-dimensionaler Würfel.

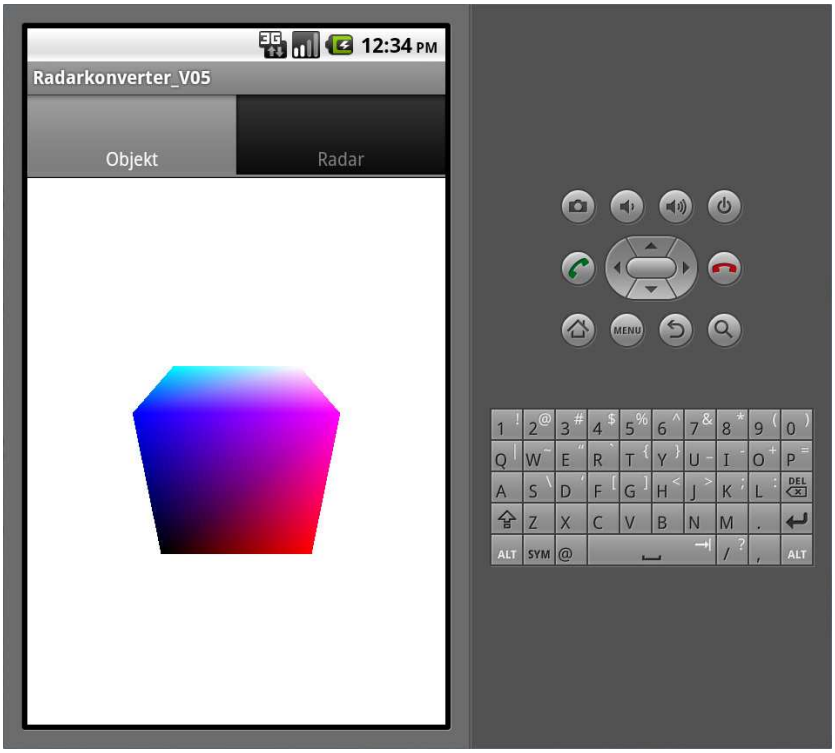


Abbildung 25: Der 3-D-Würfel im Tab

Bisher ist dieser jedoch noch nicht beweglich. Durch das Auslesen des Lagesensors kann dies geändert werden. Im nächsten Kapitel folgen Details hierzu.

6.3.3 Umsetzung: Lagesensor

Durch die Veränderung der Lage des Smartphones, soll der Würfel entsprechend reagieren und sich drehen. Möglich wird dies durch das Auslesen des Lagesensors, der in das Smartphone integriert ist. Zuständig für dessen Auslesen und einer damit einhergehende Anpassung des Würfels, ist die Klasse „OpenGLSensors“.

Abbildung 26: Die Klasse „OpenGLSensors“

Um den Würfel entsprechend zu manipulieren, greift „OpenGLSensors“ auch auf die Klassen „GLSurfaceViewNoThread“ und „Cube“ zu.

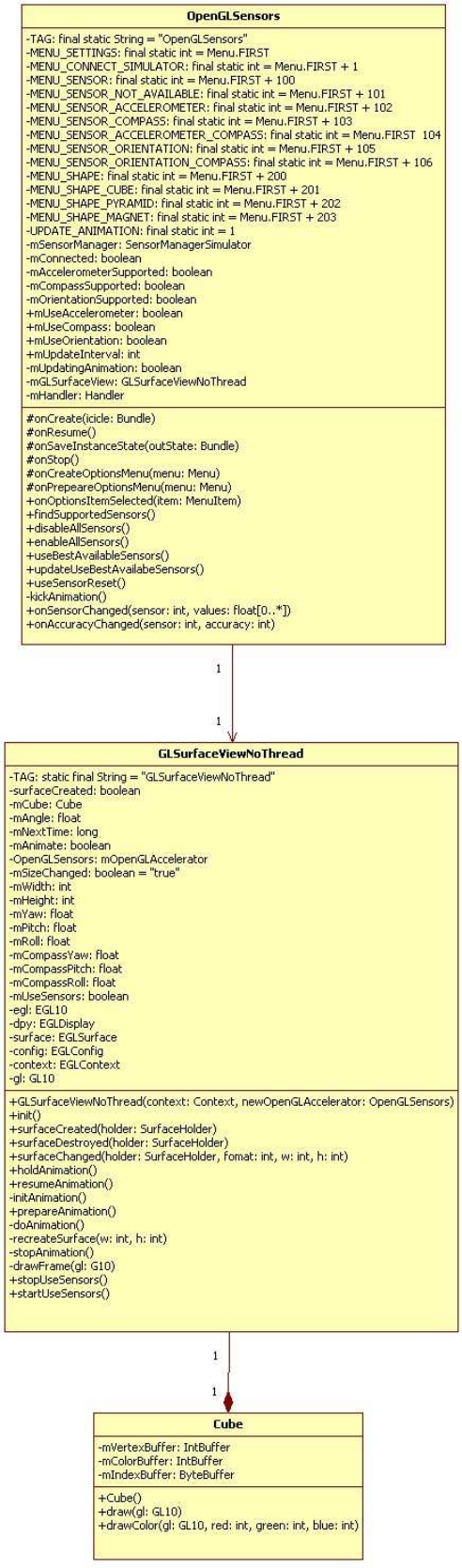

OpenGLSensors
-TAG: final static String = "OpenGLSensors"
-MENU_SETTINGS: final static int = Menu.FIRST
-MENU_CONNECT_SIMULATOR: final static int = Menu.FIRST + 1
-MENU_SENSOR: final static int = Menu.FIRST + 100
-MENU_SENSOR_NOT_AVAILABLE: final static int = Menu.FIRST + 101
-MENU_SENSOR_ACCELEROMETER: final static int = Menu.FIRST + 102
-MENU_SENSOR_COMPASS: final static int = Menu.FIRST + 103
-MENU_SENSOR_ACCELEROMETER_COMPASS: final static int = Menu.FIRST 104
-MENU_SENSOR_ORIENTATION: final static int = Menu.FIRST + 105
-MENU_SENSOR_ORIENTATION_COMPASS: final static int = Menu.FIRST + 106
-MENU_SHAPE: final static int = Menu.FIRST + 200
-MENU_SHAPE_CUBE: final static int = Menu.FIRST + 201
-MENU_SHAPE_PYRAMID: final static int = Menu.FIRST + 202
-MENU_SHAPE_MAGNET: final static int = Menu.FIRST + 203
-UPDATE_ANIMATION: final static int = 1
-mSensorManager: SensorManagerSimulator
-mConnected: boolean
-mAccelerometerSupported: boolean
-mCompassSupported: boolean
-mOrientationSupported: boolean
+mUseAccelerometer: boolean
+mUseCompass: boolean
+mUseOrientation: boolean
+mUpdateInterval: int
-mUpdatingAnimation: boolean
-mGLSurfaceView: GLSurfaceViewNoThread
-mHandler: Handler
#onCreate(icicle: Bundle)
#onResume()
#onSaveInstanceState(outState: Bundle)
#onStop()
#onCreateOptionsMenu(menu: Menu)
#onPrepeareOptionsMenu(menu: Menu)
+onOptionsItemSelected(item: MenuItem)
+findSupportedSensors()
+disableAllSensors()
+enableAllSensors()
+useBestAvailableSensors()
+updateUseBestAvailabeSensors()
+useSensorReset()
-kickAnimation()
+onSensorChanged(sensor: int, values: float[0..*])
+onAccuracyChanged(sensor: int, accuracy: int)
1
1
GLSurfaceViewNoThread
-TAG: static final String = "GLSurfaceViewNoThread"
-surfaceCreated: boolean
-mCube: Cube
-mAngle: float
-mNextTime: long
-mAnimate: boolean
-OpenGLSensors: mOpenGLAccelerator
-mSizeChanged: boolean = "true"
-mWidth: int
-mHeight: int
-mYaw: float
-mPitch: float
-mRoll: float
-mCompassYaw: float
-mCompassPitch: float
-mCompassRoll: float
-mUseSensors: boolean
-egl: EGL10
-dpy: EGLDisplay
-surface: EGLSurface
-config: EGLConfig
-context: EGLContext
-gl: GL10
+GLSurfaceViewNoThread(context: Context, newOpenGLAccelerator: OpenGLSensors)
+init()
+surfaceCreated(holder: SurfaceHolder)
+surfaceDestroyed(holder: SurfaceHolder)
+surfaceChanged(holder: SurfaceHolder, fomat: int, w: int, h: int)
+holdAnimation()
+resumeAnimation()
-initAnimation()
+prepareAnimation()
-doAnimation()
-recreateSurface(w: int, h: int)
-stopAnimation()
-drawFrame(gl: G10)
+stopUseSensors()
+startUseSensors()
1
1
Cube
-mVertexBuffer: IntBuffer
-mColorBuffer: IntBuffer
-mIndexBuffer: ByteBuffer
+Cube()
+draw(gl: GL10)
+drawColor(gl: GL10, red: int, green: int, blue: int)

Abbildung 27: Das Zusammenspiel aller drei Klassen zur Würfelerzeugung und Manipulation

Über ein Objekt der GLSurfaceViewNoThread-Klasse, wird die Manipulation des Würfels gewährleistet. Dieses wird erzeugt, und damit die Sensoren gestoppt. Danach wird ein Sensormanager angelegt, über den alle Sensorkonfigurationen eingestellt werden. Hierüber werden im Folgenden auch alle Sensorwerte ausgelesen.

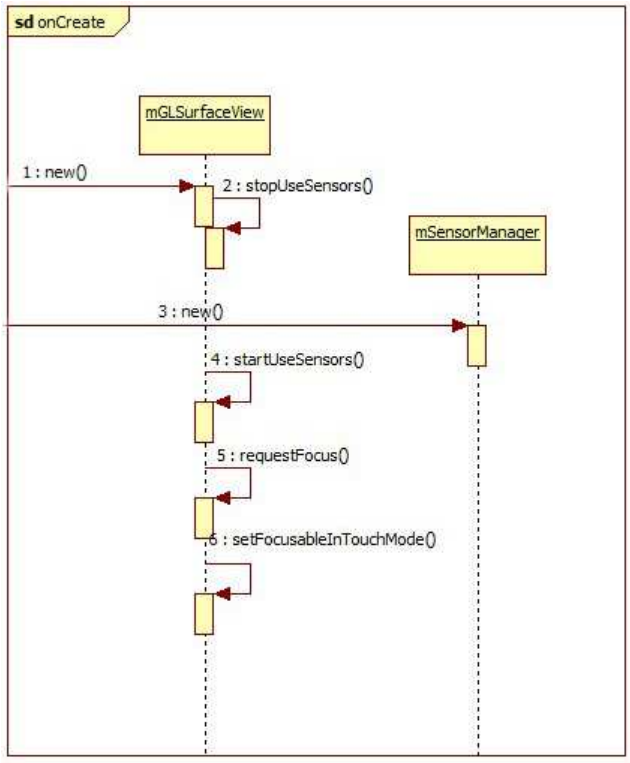


Abbildung 28: Die OnCreate-Methode der OpenGLSensors-Klasse

In der onResume-Methode wird der Sensormanager nun mit einem Listener registriert.

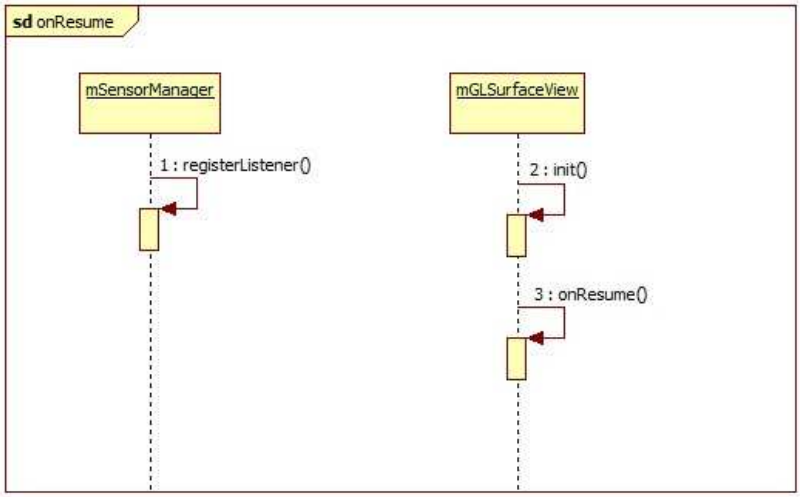


Abbildung 29: Die onResume-Methode der OpenGLSensors-Klasse

Liefert der 3-Achsen-Sensor eine Veränderung der Lage zurück, wird auf den Würfel zugegriffen. Es erfolgt eine Translation der Koordinaten, wodurch sich das Objekt auf dem Bildschirm dreht. Weitere Methoden sorgen für ein Stop der Drehbewegung, pausieren die Anwendung, suchen die aktiven Sensoren oder schalten alle Sensoren ein.

6.3.4 Umsetzung: Die bisherige Klassenstruktur

Die Gesamtstruktur der App-Klassen sieht aus wie in „Abbildung 30: Der bisherige Aufbau aller Klassen der App“ beschrieben. Dabei ist die Klasse „Visar_Activity“ bisher noch leer, zudem fehlt die Implementierung der Touch-Funktion. Trotzdem ist die App bereits funktionsfähig. Ein 3-dimensionales-Objekt wird dargestellt und kann über die Veränderung der Lage manipuliert werden. Über die Klasse „TouchRotateActivity“ erfolgt im nächsten Schritt die Einbindung der Touch-Funktion, die „Visar_Activity“-Klasse bindet die Funktionen der ViSAR-Klassen ein, die zuvor jedoch noch an die Beschränkungen der Android Spezifika angepasst werden müssen. Im Folgenden Kapitel wird genauer erläutert, wie die ViSAR-Integration funktioniert, welche Klassen verwendet werden und weshalb eine Anpassung erfolgen muss.

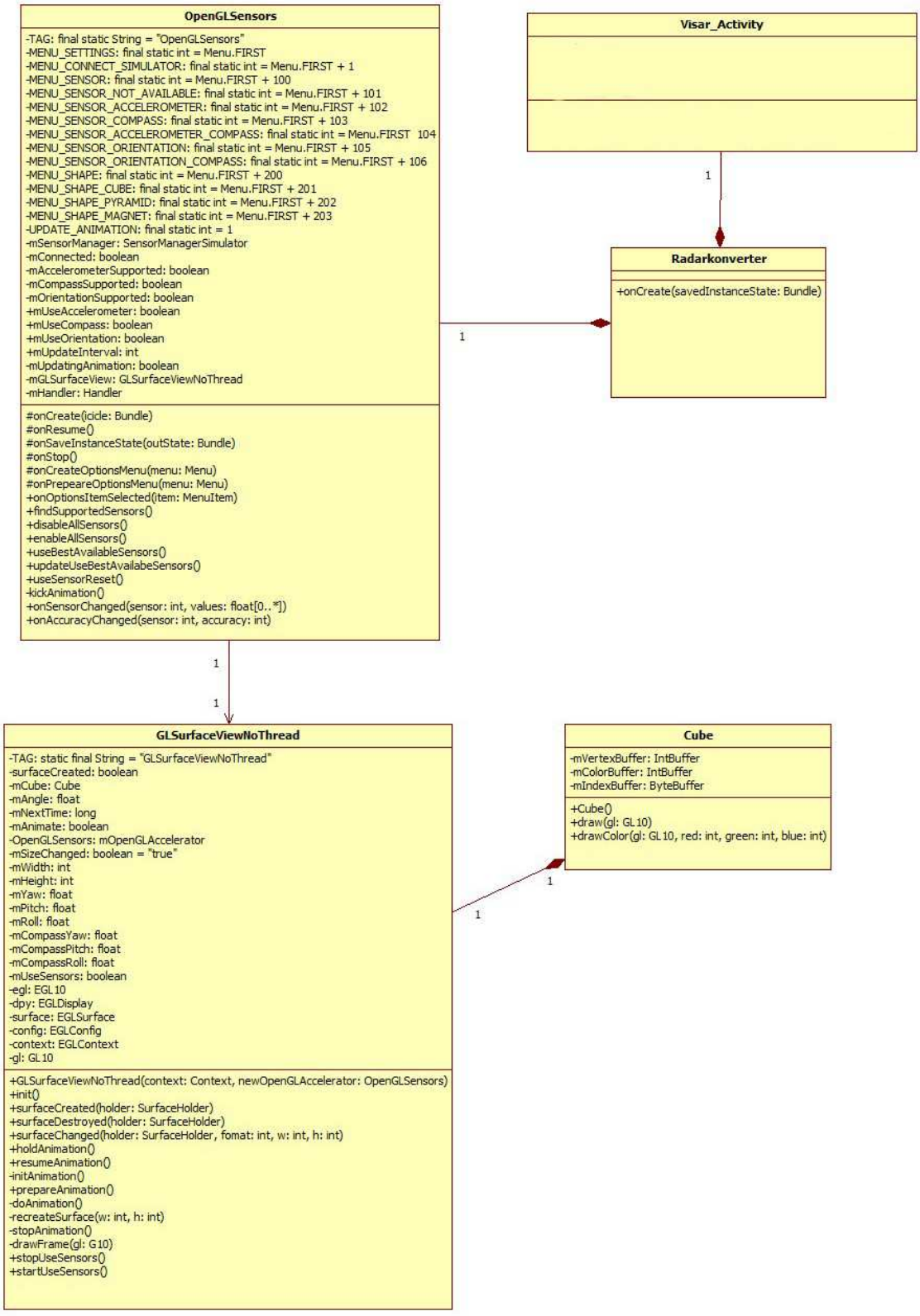


Abbildung 30: Der bisherige Aufbau aller Klassen der App

6.3.5 Umsetzung: Die Touch-Funktion

Neben der Manipulation des Würfels über den Lagesensor des Handys, ist dies auch über die Berührung des Würfels mit dem Zeigefinger – also über eine Touch-Geste möglich. Die benötigte Funktion hierfür, wird durch die Klasse „TouchRotateActivity“ bereitgestellt.

Innerhalb der Klasse werden, wie auch in der Umsetzung für den Sensor, die Methoden „onCreate“, „onResume“ und „onPause“ verwendet, um die Funktionalität der Activity in den verschiedenen Modi zu implementieren. Dabei wird direkt beim Start der Activity, also in der Methode „onCreate“, der Focus in den Touch-Modus versetzt und eine neue View für selbigen angelegt. In der „TouchSurfaceView“-Klasse werden dann alle mathematischen Manipulationen implementiert, die auf den Würfel durch eine Berührung wirken (also die Rotation).

Innerhalb einer weiteren, inneren Klasse, dem „CubeRenderer“, findet die Auswirkung der Manipulation statt. Hier werden die Auswirkungen, der angewendeten Rotation auf den Cube übertragen, wodurch dies für den Anwender sichtbar wird.

Die „TouchRotateActivity“-Klasse wird über die in „Abbildung 30: Der bisherige Aufbau aller Klassen der App“ bereits im Gesamtkontext aufgezeigte „Radarkonverter“-Klasse eingebunden. Sie ersetzt die „OpenGLSensors“-Klasse. Damit sind zu diesem Zeitpunkt zwar beide Funktionen (Sensorauslesung und Touch-Rotation) verwendbar, jedoch lediglich getrennt, nicht gleichzeitig.

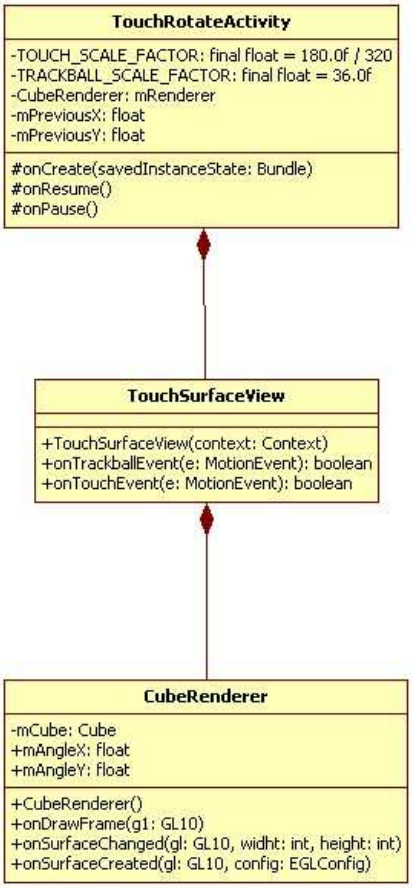


Abbildung 31: Zusammenspiel der Klassen für die Touch-Funktion

6.3.6 Die Visar-Integration

Nachdem eine Standalone-Version des ViSAR-Programms durch herausnehmen der bisher beschriebenen Basisklassen bereits erstellt wurde, muss diese nun weiter angepasst werden, um in die App eingepasst werden zu können. Hierzu werden alle Bestandteile der Java3D Library entfernt.

Bei einer Portierung des kompletten Funktionsumfangs wäre eine umfangreiche Anpassung der in Tabelle 1 aufgeführten Klassen notwendig.

Tabelle 1: Gegenüberstellung Java3D / Android

Java3D	Android
javax.media.j3d.BoundingSphere	**com.jme3.bounding.BoundingVolume**
BoundingSphere getBoundingSphere()	BoundingSphere getBoundingSphere()
setBoundingSphere(BoundingSphere)	setBoundingSphere(BoundingSphere)
getCenter(point3d)	getCenter(Vector3f)
getRadius()	getRadius()
BranchGroup	Keine direkte Entsprechung
javax.vecmath.Point3D	**com.jme3.math.Vector3f**
Point3D()	Vector3f()
Point3D(double, double, double)	Vector3f(float , float, float)
double getY()	float getY()
PickTool	Keine direkte Entsprechung
PickResult	Keine direkte Entsprechung
java.awt.image.BufferedImage	**android.graphics.Bitmap**
BufferedImage(int, int, int)	createBitmap(int, int, Bitmap.Config)
setRGB (int, int, int)	setPixel(int, int, int)
ImageIO **write(File, String, int)**	**android.graphics.BitmapFactory** Über Bitmap / BitmapFactory und Descriptor
Scene	Keine direkte Entsprechung
ObjectFile	Keine direkte Entsprechung
VrmlLoader	Keine direkte Entsprechung
SceneGraphLoader	Keine direkte Entsprechung
Shape3D	Keine direkte Entsprechung

Da für die App jedoch nicht alle Klassen notwendig sind, beschränken sich die Anpassungen auf 2 Methoden, die lediglich die in Tabelle 2 angegebenen Java3D-Elemente

beinhalten. Diese Methoden werden in die App portiert und die entsprechenden Java3D Elemente durch Methoden des Android-Frameworks ersetzt.

Tabelle 2: Die Java3D-Elemente, die in der App ersetzt werden müsen

Java3D	Android
java.awt.image.BufferedImage	**android.graphics.Bitmap**
BufferedImage(int, int, int)	createBitmap(int, int, Bitmap.Config)
setRGB (int, int, int)	setPixel(int, int, int)
ImageIO	**android.graphics.BitmapFactory**
write(File, String, int)	Über Bitmap / BitmapFactory und Descriptor

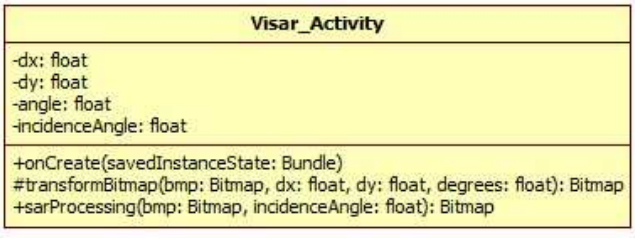


Abbildung 32: Die Visar_Activity-Klasse mit Methoden

Die ursprünglichen Klassen der ViSAR-Standalone-Awendung werden komplett entfernt. Um die bisherige Funktion, reduziert auf die Umwandlung von 3-D-Objekt zu Radarbild, weiter zu gewährleisten, werden zwei Methoden eingefügt. Die zuvor der Klasse „DigitalElevationModel“ angehörige Methode „doit()“ wird zur Methode „transformBitmap“ und in der Klasse „Visar_activity“ eingefügt. Gleiches geschieht mit der Methode „run()“ der Klasse „SAR_Processor_Impl“.

Die Methode „OnCreate“ greift auf den Asset-Ordner des Android-Projektes zu, holt von dort die benötigte PNG-Datei und wandelt diese zur Bitmap um. Im Folgenden wird diese Bitmap nacheinander an zwei Methoden übergeben, die diese weiterbearbeiten. „Abbildung 33: Auszug der OnCreate-Methode der Klasse VisarApp“, zeigt den Ablauf innerhalb dieser Methode, bis zu diesem Punkt.

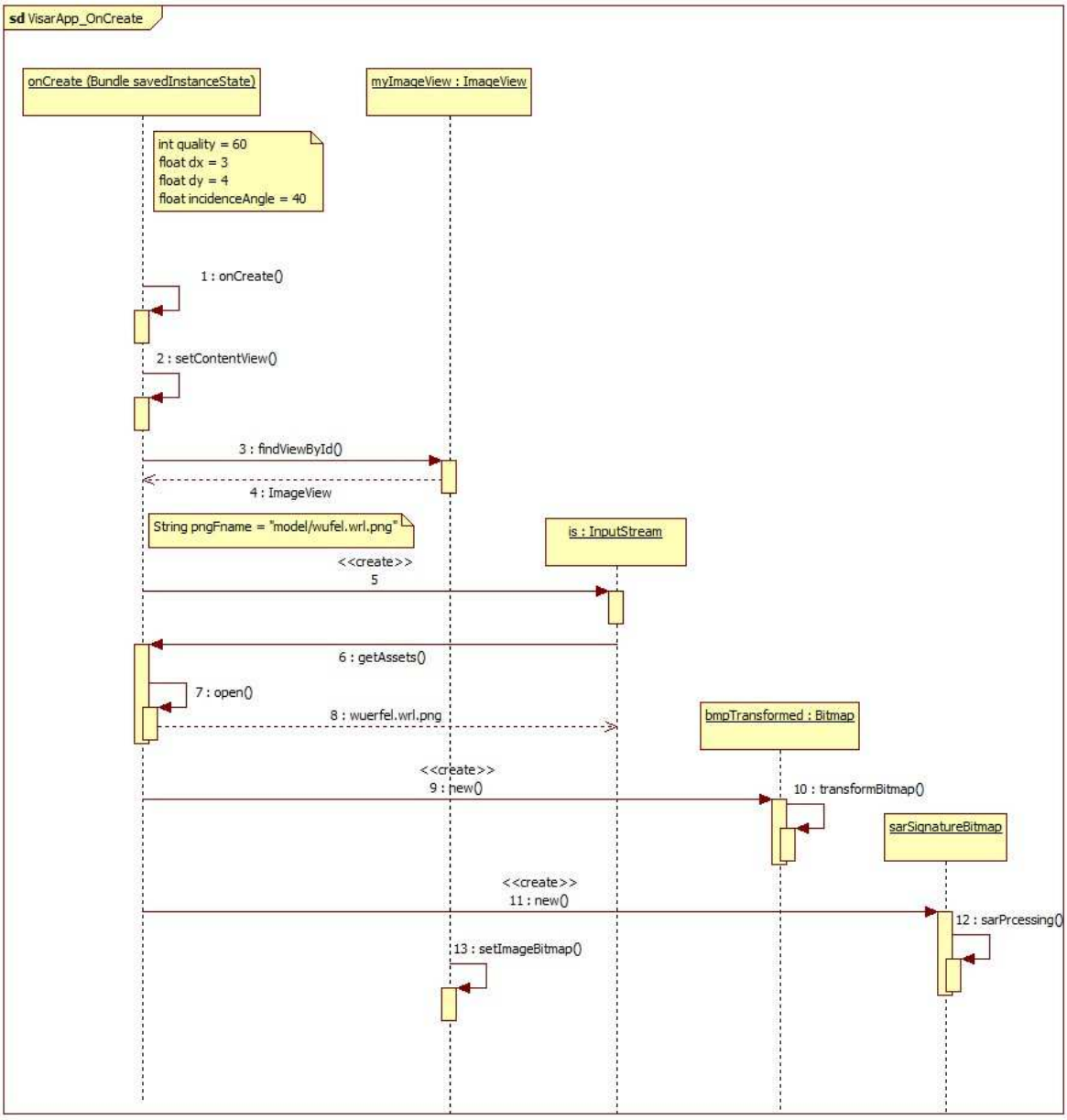


Abbildung 33: Auszug der OnCreate-Methode der Klasse VisarApp

Innerhalb der Methoden „transformBitmap“, wird die übergebene Bitmap-Grafik mit einem Canvas (der „Leinwand“) verknüpft, und einer Rotation sowie Translation unterworfen. Dies ist notwendig, wenn der Anwender den Würfel beispielsweise gedreht hat, und daher die PNG-Datei, aus der das neue Radarbild erzeugt wird, angepasst werden muss.

Das Resultat wird zurückgeliefert und im Folgenden durch die Methode „sarProcessing“ weiterverarbeitet.

Nach der Rückgabe erfolgt die Ausgabe auf dem Smartphone und der Speichervorgang auf der SD-Karte des Geräts.

6.4 Experimente / Ergebnisse

Um die Funktionalität des Prototypen zu testen, wird dieser auf dem Google Nexus S Smartphone getestet. Die App wird initialisiert, der Würfel befindet sich in der Ausgangsposition.

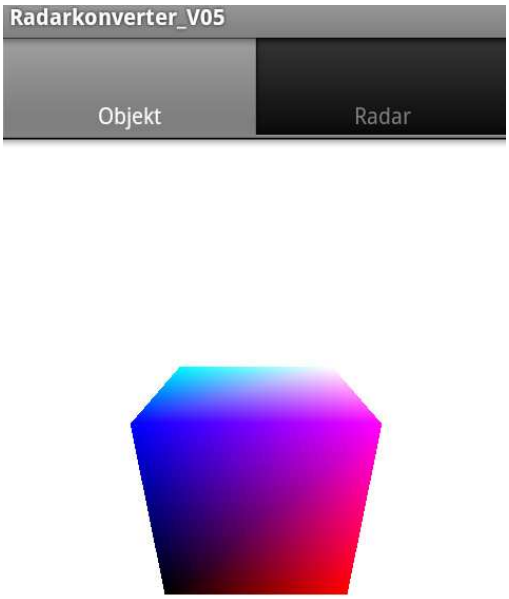


Abbildung 34: Startbildschirm der App mit Würfel

Im Folgenden wird der Würfel durch eine Veränderung der Lage des Smartphones gedreht. Durch das Auslesen des Näherungssensors kann dieser in jede beliebige Richtung gedreht und gewendet werden.

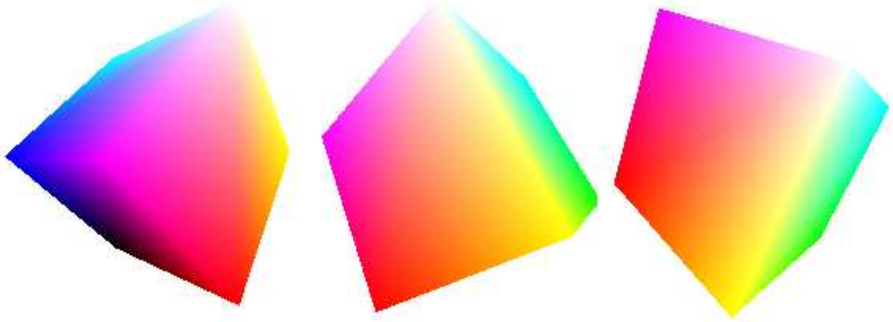


Abbildung 35: Verschiedene Rotationen des Würfels durch Veränderung der Smartphone-Position

Wie in „Abbildung 35: Verschiedene Rotationen des Würfels durch Veränderung der Smartphone-Position“ dargestellt, kann der Würfel innerhalb des 3-dimensionalen Raumes beliebig gedreht werden. Im nächsten Schritt wird aus einer dieser Positionen ein Radarbild generiert.

Als Ausgangspunkt dient folgender Würfel:

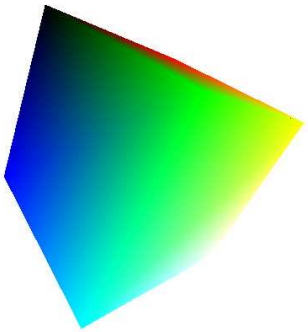


Abbildung 36: Würfel von Experiment 1

Über das Umschalten auf den ViSAR-Tab, wird nun das Radarbild generiert:

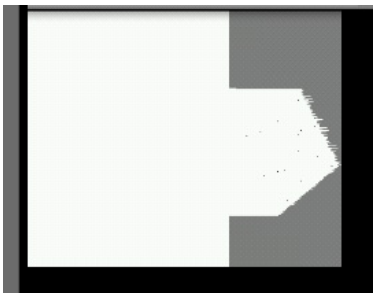


Abbildung 37: Das Radarbild als Ergebnis von Experiment 1

Die Manipulation des Würfels erfolgt zu Testzwecken über die Veränderung der Position des Smartphones oder über die Touch-Bedienung mittels Berührung des Würfels. In beiden Fällen kann der Würfel beliebig gedreht werden und die Generierung eines Radarbildes erfolgt problemlos. Ergänzend zur Ausgabe wird das Radarbild auf einer ggf. vorhandenen SD-Card gespeichert.

Im Folgenden wird der Würfel in eine andere Ausgangsposition gebracht und so ein andere Grundlage für eine zweite Generierung erreicht:

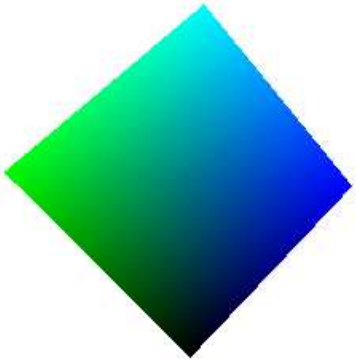


Abbildung 38: Würfel von Experiment 2

Die Generierung des Radarbildes bringt folgendes Ergebnis zustande:

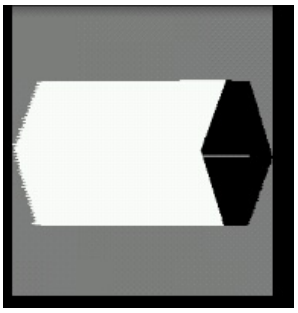


Abbildung 39: Das Radarbild als Ergebnis von Experiment 2

Abschließend wird der Würfel in eine ähnliche Lage wie in Experiment 1 gebracht, jedoch leicht abgewinkelt und über eine alternative Drehung. Hierdurch ändert sich auch deutlich der Winkel für die Generierung des Radarbildes.

Aus folgendem Ausgangswürfel:

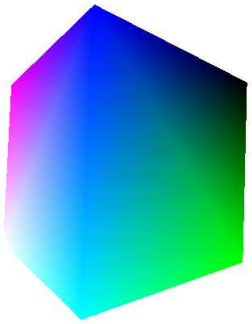


Abbildung 40: Würfel von Experiment 3

Wird folgendes Radarbild:

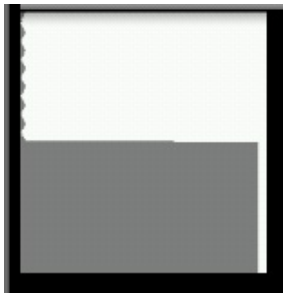


Abbildung 41: Das Radarbild als Ergebnis von Experiment 3

Es wird deutlich, dass auch geringe Veränderungen in der Position des Würfels durch Rotation, ein stark verändertes Radarbild generieren, da der Einfallswinkel ein deutlich anderer ist.

6.5 Diskussion

Das Experiment zeigt, dass die Erstellung eines Radarbildes wie gewollt funktioniert. Dabei entstehen, je nach Abhängigkeit des Einfallswinkels, leicht zu interpretierende Radarbilder (siehe „Abbildung 39: Das Radarbild als Ergebnis von Experiment 2“) oder kaum als solche Erkennbare Ergebnisse (siehe „Abbildung 41: Das Radarbild als Ergebnis von Experiment 3“).

Der 3-D-Würfel ist farblich unterschiedlich markiert (Blau, Grün, Violett), um die jeweiligen Seiten auch nach der Drehung voneinander zu unterscheiden.

Jedes der generierten Radarbilder wurde während des Experiments ebenfalls mit der Desktop-ViSAR-Version überprüft (siehe „Kapitel: 5.3ViSAR-Desktop-Standalone-Version“). Das Ergebnis stimmte in allen Fällen mit dem generierten Radarbild der Android-App überein. Es kann also davon ausgegangen werden, dass alle Radarbild korrekt generiert wurden.

Das Radarbild wird vor einem grauen, würfelförmigen Hintergrund ausgegeben.

6.5.1 Status des Prototypen

Der Prototyp stellt nahezu die vollständige Funktion zur Verfügung, die in der Konzeption angedacht war.

Durch eine Veränderung der Lage des Handys kann der Würfel gedreht werden, ebenso wie durch eine Berührung mit dem Finger. Ein Wechsel vom 3-D-Tab zur Radarbildanzeige generiert das Radarbild, was jedoch einige Sekunden in Anspruch nimmt. In dieser Zeit erfolgt kein Wechsel zum Radar-Tab. Erst wenn im Hintergrund die Generierung vollständig erfolgt ist, wird das Bild angezeigt. Es hat also für den Anwender den Eindruck, als ob die App für einige Sekunden eingefroren ist.

Ergänzend zur visuellen Ausgabe des fertigen Radarbildes, wird dieses auch auf einer potenziell vorhandenen SD-Karte gespeichert, um es später von dort weiterzuverwenden.

Aus Zeitgründen konnten jedoch zwei Elemente nicht mehr umgesetzt werden.

Ursprünglich angedacht war die Implementierung einer Zoom-Funktion durch das Bewegen von zwei Fingern auf dem Bildschirm des Smartphone. Zudem sind aktuell zwar sowohl die Rotationsmöglichkeit durch bewegen des Handys wie durch die Berührung mit dem Zeigefinger implementiert, jedoch können nicht beide Funktionen gleichzeitig eingesetzt werden. Innerhalb des Codes muss zwischen beiden Möglichkeiten umgeschaltet werden.

Somit ist immer nur eine Variante zur gleichen Zeit Verfügbar. Innerhalb des Codes sind beide Funktionen auf unterschiedliche Klassen aufgeteilt. Lediglich einer der Klasse ist jeweils eingebunden.

Eine weitere Einschränkung tritt auf, wenn der Anwender mehrere Radarbilder infolge erzeugen will. Wird, nachdem ein Radarbild erzeugt wurde, auf den Würfel zurückgeschaltet und dieser erneut manipuliert, erfolgt keine neue Generierung des Radarbildes. Die App muss also beendet und neu gestartet werden, damit ein neues Radarbild generiert werden kann.

6.5.2 Fazit und Ausblick

Ziel der Arbeit war es, in die Grundlagen von Android einzuführen, ein einfaches 3-D-Objekt zu entwickeln und eine App zur Radarkonvertierung zu erstellen. Dabei ging die Einarbeitung in die Grundlagen flüssig und problemlos vonstatten, ebenso die Konzipierung des Prototypen.

Einzig bei der Einarbeitung in den vorhandenen ViSAR-Code sowie die Überführung des Java3D-Codes in den Android-Code kam es zu Komplikationen. Hier war es stellenweise nicht einfach, die wichtigen Methoden des Ursprungsprogramm zu identifizieren sowie die Entsprechung der Java3D-Funktionen im Android-Code zu finden. Manche Funktionen mussten sogar komplett alternativ programmiert werden. Hierdurch kam es zu Verzögerungen im Zeitplan.

Im einem möglichen zukünftigen Schritt sollte der Prototyp um weitere 3-D-Modelle ergänzt werden. Neben dem Würfel können weitere Primitive und verschachtelte Elemente generiert werden. Zudem sollte die Touch- und Sensorsimulation fusioniert werden und das wechseln in den 3-D-Tab den ViSAR-Tab beenden. Die Zoom-Funktion über Multi-Touch-Gesten ist ebenfalls eine Funktion, die ergänzt werden sollte.

7 Anhang

Quellenverzeichnis

[And2010] *Verkaufszahlen: Android schlägt iOS und Symbian*
URL: http://www.tamspalm.de/2011/01/verkaufszahlen-android-schlaegt-ios-und-symbian
Aufgerufen am: 02.04.2011

[BESCHL] *Beschleunigungssensor*
URL: http://de.wikipedia.org/wiki/Beschleunigungssensor
Aufgerufen: 21.09.2011

[Bond2002] *A New Line Drawing Algorithm Based on Sample Rate Conversion*
Bond, C. 2002

[C&L2011] Click&Learn
Innovatives Vorzeigeprojekt
URL: http://www.clickandlearn.at
Aufgerufen am: 25.08.2011

[CampS] Campus Source
"Kurzbeschreibung der Systeme"
URL: http://www.campussource.de
Aufgerufen am 25.08.2011

[Comp2007] Competence-Site
„Fraunhofer ESK macht eCoach zum BIBCoach“
URL: http://www.competence-site.de/produktions-it/Fraunhofer-ESK-macht-eCoach-zum-BIBCoach
Aufgerufen am: 25.08.2011

[COSTE2009] *Cost-Efficient Development with Various OpenGL*
Vuorinen, J.
HELSINKI UNIVERSITY OF TECHNOLOGY, 2009

[CT2003] *Was ist ein Smartphone?*
URL: http://www-lehre.informatik.uni-osnabrueck.de/mc/material/smartphones/einleitung_02.html
Aufgerufen am: 02.06.2011

[DigHoeh] *Wie erstellt man Digitale Höhenmodelle?*
URL: http://satgeo.zum.de/satgeo/methoden/DHM_Web/Erstellung.htm
Aufgerufen am: 08.07.2011

[ESK2011] Fraunhofer ESK
“Mobile MENTOR – Lernen wo und wann man will”
URL: http://www.esk.fraunhofer.de/de/projekte/mobileMENTOR.html
Augerufen am: 25.08.2011

[IMF] IMF
"IFM - Trainingsunterlagen"
URL: http://www.ifm.com/ifmde/web/training-2.htm
Aufgerufen am: 21.09.2011

[PHROOD] *Scan-conversion of a line*
Wikipedia, 28.02.2007, Prood
http://de.wikipedia.org/wiki/Datei:Line_scan-conversion.svg
Abgerufen am: 29.08.2011

[PROA2010] *Pro Android 2*
Hashimi, S. Y.; Komatineni, S. & MacLean, D.
Anglin, S. *(Ed.)*
Apress, 2010

[Rob2010] *Semantik im Vektormodell*
Robineau, Peter
Diplomarbeit
Universität Wien, 2010

Abbildungsverzeichnis