



Department of Computer Science

Patterns and Pattern Languages for Mobile Augmented Reality

Donald Munro

Submitted in fulfilment of the requirements
for the Degree of Philosophiae Doctor in Computer Science
in the Faculty of Science at the Nelson Mandela University.

Promoter: Professor Andre Calitz
Co-Promoter: Dr. Dieter Vogts

April 2020

Declaration

I, Hugh Donald Munro (student number: s183007930), hereby declare that this thesis for the degree of Philosophiae Doctor is my own work and that it has not previously been submitted for assessment or completion of any postgraduate qualification to another University or for another qualification.



Hugh Donald Munro

Dedicated to my late mother.

Mother doesn't know where love has gone
She says it must be youth
That keeps us feeling strong
I see it in her face, that's turned to ice
And when she smiles she shows the lines of sacrifice
—Spandau Ballet, Through the Barricades

Abstract

Mixed Reality is a relatively new field in computer science which uses technology as a medium to provide modified or enhanced views of reality or to virtually generate a new reality. Augmented Reality is a branch of Mixed Reality which blends the real-world as viewed through a computer interface with virtual objects generated by a computer. The 21st century commodification of mobile devices with multi-core Central Processing Units, Graphics Processing Units, high definition displays and multiple sensors controlled by capable Operating Systems such as Android and iOS means that Mobile Augmented Reality applications have become increasingly feasible.

Mobile Augmented Reality is a multi-disciplinary field requiring a synthesis of many technologies such as computer graphics, computer vision, machine learning and mobile device programming while also requiring theoretical knowledge of diverse fields such as Linear Algebra, Projective and Differential Geometry, Probability and Optimisation. This multi-disciplinary nature has led to a fragmentation of knowledge into various specialisations, making it difficult to integrate different solution components into a coherent architecture.

Software design patterns provide a solution space of tried and tested best practices for a specified problem within a given context. The solution space is non-prescriptive and is described in terms of relationships between roles that can be assigned to software components. Architectural patterns are used to specify high level designs of complete systems, as opposed to domain or tactical level patterns that address specific lower level problem areas.

Pattern Languages comprise multiple software patterns combining in multiple possible sequences to form a language with the individual patterns forming the language vocabulary while the valid sequences through the patterns define the grammar. Pattern Languages provide flexible generalised solutions within a particular domain that can be customised to solve problems of differing characteristics and levels of

complexity within the domain. The specification of one or more Pattern Languages tailored to the Mobile Augmented Reality domain can therefore provide a generalised guide for the design and architecture of Mobile Augmented Reality applications from an architectural level down to the "nuts-and-bolts" implementation level.

While there is a large body of research into the technical specialisations pertaining to Mobile Augmented Reality, there is a dearth of up-to-date literature covering Mobile Augmented Reality design. This thesis fills this vacuum by:

1. Providing architectural patterns that provide the spine on which the design of Mobile Augmented Reality artefacts can be based;
2. Documenting existing patterns within the context of Mobile Augmented Reality;
3. Identifying new patterns specific to Mobile Augmented Reality; and
4. Combining the patterns into Pattern Languages for Detection & Tracking, Rendering & Interaction and Data Access for Mobile Augmented Reality.

The resulting Pattern Languages support design at multiple levels of complexity from an object-oriented framework down to specific one-off Augmented Reality applications. The practical contribution of this thesis is the specification of architectural patterns and Pattern Language that provide a unified design approach for both the overall architecture and the detailed design of Mobile Augmented Reality artefacts. The theoretical contribution is a design theory for Mobile Augmented Reality gleaned from the extraction of patterns and creation of a pattern language or languages.

Keywords: Augmented Reality, Design Patterns, Pattern Languages.

Acknowledgements

I would like to thank my promoters, Professor Andre Calitz and Dr. Dieter Vogts for their company on the long and winding road less travelled by, leading to the final chapter. To paraphrase Euclid “There is no ‘royal road’ to a PhD”, but their patience and support paved the way.

I would also like to thank the open source community in general and the Computer Vision and Augmented Reality communities in particular for providing inspiration, insights and solutions. In many ways the open source movement hearkens back to the era of Newton, Leibnitz and Gauss when the spread of knowledge was unhindered and Science was young and not yet captive to those who would subjugate all to profit.

Table of Contents

List of Figures	xii
List of Tables	xiv
Abbreviations	xv
1 Introduction	1
1.1 Background	2
1.2 Research Problem	4
1.3 Research Statement	4
1.4 Research Objectives	5
1.5 Research Questions	6
1.6 Ethics	6
1.7 Chapter Outline	7
2 Research Paradigm and Methodology	8
2.1 Introduction	9
2.2 Research Paradigms	10
2.2.1 Positivism/Post-positivism	10
2.2.2 Realism	10
2.2.3 Constructivism/Interpretivism/Subjectivism	11
2.2.4 Pragmatism/Mixed Method	11
2.2.5 Research Logic	11
2.2.6 Research Methodology	11
2.3 Research by Design	11
2.3.1 Design Science Research	13
2.3.2 Patterns and Pattern Languages in Design Science Research	17
2.4 Research Choices	20
2.4.1 DSR as a Paradigm	20
2.4.2 DSR as a Methodology	20
2.4.3 Research Process and Output	22

2.5	Summary	23
3	Patterns and Pattern Languages	24
3.1	Introduction	25
3.2	Software Reuse	25
3.2.1	Procedural Libraries	26
3.2.2	OO Libraries	27
3.2.3	Patterns	27
3.2.4	OO Frameworks	28
3.2.5	Product-line Architectures	28
3.3	Software Patterns	28
3.3.1	What's in a Name	29
3.3.2	What's the Context	29
3.3.3	What's the Problem	30
3.3.4	What's the Solution	31
3.3.5	Final Assembly	33
3.3.6	Pattern Misconceptions	34
3.4	Pattern Interactions	34
3.4.1	Pattern Aggregation	34
3.4.2	Pattern Complementation	35
3.4.3	Pattern Compounds	35
3.4.4	Pattern Sequences	36
3.5	Pattern Languages	37
3.5.1	Target	37
3.5.2	Name	37
3.5.3	Description	38
3.5.4	Solution	38
3.5.5	Applications	40
3.6	Summary	40
4	Augmented Reality	42
4.1	Introduction	43
4.2	History	44
4.3	Augmented Reality Hardware	45
4.4	Augmented Reality Software Development	47
4.4.1	Object Detection	47
4.4.2	Object Tracking	52
4.4.3	Mean-shift and Camshift	53
4.4.4	Median Flow	53
4.4.5	Discriminative Correlation Filters	54
4.4.6	Neural Network Based Tracking	55
4.4.7	Registration or Pose Determination	56
4.4.8	SLAM	61
4.4.9	Interaction	63
4.4.10	Presentation	65
4.4.11	Architecture	70
4.4.12	Prototyping	70

4.4.13 APIs and Frameworks	72
4.5 Summary	74
5 Architecture and Architectural Patterns for Mobile Augmented Reality	76
5.1 Introduction	77
5.2 Describing Software Architectures	77
5.2.1 Architectural Attributes	78
5.2.2 Architectural Styles	78
5.2.3 Patterns and Architectures	79
5.3 Architectural Styles for Mobile Augmented Reality	79
5.3.1 Related Work	79
5.3.2 Streams Based Architecture	80
5.4 A Task Graph Based Architecture for MAR	84
5.4.1 Implementation Technologies	84
5.5 Architectural Patterns for MAR	88
5.5.1 The STRUCTURAL TASK GRAPH Pattern	88
5.5.2 SHARED RESOURCE	90
5.6 Summary	91
6 Detection and Tracking	92
6.1 Introduction	93
6.2 Coordinate Systems	94
6.2.1 Global Coordinate Systems	94
6.2.2 Local Coordinate Systems	95
6.3 Patterns for Locational MAR	98
6.3.1 COMMON LOCATIONAL COORDINATES	98
6.3.2 SENSOR BASED ORIENTATION	100
6.3.3 SENSOR FUSION	101
6.3.4 SCRIPTED CONTENT	102
6.3.5 LOCATIONAL ARCHITECTURE	103
6.4 Patterns for Vision Based MAR	104
6.4.1 <i>Acquire Frame</i> (OS Idiom)	105
6.4.2 FEATURE EXTRACTION	108
6.4.3 FEATURE MATCHING	109
6.4.4 OBJECT TRACKING	113
6.4.5 RELATIVE POSE	116
6.4.6 ABSOLUTE POSE	118
6.4.7 POINT CLOUD POSE	121
6.4.8 BUNDLE ADJUSTMENT	121
6.4.9 KEYFRAME IDENTIFICATION	124
6.4.10 DIRECT 3D MAP	125
6.4.11 LOOP CLOSURE	126
6.4.12 DETECT/TRACK ARCHITECTURE	126
6.5 Pattern Language	128
6.5.1 Locational MAR	128
6.5.2 Vision Based MAR	130
6.6 Summary	135

7 Rendering and Interaction	137
7.1 Introduction	138
7.1.1 Rendering	138
7.1.2 Interaction	139
7.2 Rendering Hardware/Software for Mobile Platforms	140
7.2.1 Rendering Hardware	140
7.2.2 Rendering Software	140
7.3 3D Software Development	141
7.3.1 3D Rendering Concepts	142
7.3.2 Graphics Pipeline Stages	144
7.4 Interaction and Rendering Architectural Patterns	147
7.4.1 Interaction Architectural Patterns	147
7.4.2 Rendering Architectural Patterns	149
7.5 Rendering Patterns	151
7.5.1 The Level of Detail (LOD) Pattern	151
7.5.2 MULTI BUFFER	153
7.5.3 TIMEWARPING	154
7.5.4 PHOTOMETRIC REGISTRATION	154
7.5.5 IMAGE BASED LIGHTING (IBL)	155
7.5.6 PROJECTION SETUP	156
7.5.7 CALIBRATION ADJUSTMENT	158
7.5.8 BILLBOARD	158
7.6 Interaction Patterns	159
7.6.1 RAYCAST	159
7.6.2 VERTEX CODING	160
7.6.3 RETICLE	160
7.6.4 GUIDE	161
7.6.5 AFFORDANCES	161
7.6.6 ANCHOR	161
7.6.7 FREEZEFRAME	163
7.6.8 TANGIBLE TOOLS	163
7.6.9 MAGIC LENS	164
7.7 Pattern Language	164
7.8 Summary	165
8 Data Access	166
8.1 Introduction	167
8.2 Client/Server Distributed Systems	168
8.3 Pattern Language Design	169
8.3.1 Problem and Solution Scope	169
8.3.2 Forces	170
8.3.3 Architecture	171
8.4 Pattern Language Vocabulary	172
8.4.1 LAYERS	172
8.4.2 REQUESTOR	173
8.4.3 REQUEST HANDLER	175
8.4.4 MARSHALLER	175

8.4.5	FACADE	176
8.4.6	EVENT AGGREGATOR	177
8.4.7	Metadata Patterns	177
8.4.8	Concurrency Patterns	180
8.4.9	Caching	181
8.4.10	Rendering Patterns based on Data Access	182
8.5	Pattern Language Grammar	183
8.5.1	Bottom Up Design Sequence	183
8.5.2	REQUEST HANDLER Design Sequence	184
8.5.3	Top Down Design Sequence	184
8.5.4	EVENT AGGREGATOR Design Sequence	185
8.5.5	POI MARKER Design Sequence	185
8.6	Summary	185
9	Conclusions and Further Research	186
9.1	Introduction	187
9.2	Completed Objectives	188
9.2.1	RO ₁ : Literature Reviews	188
9.2.2	RO ₂ : Identifying architectural patterns	189
9.2.3	RO ₃ : Patterns and a pattern language for detection and tracking	190
9.2.4	RO ₄ : Patterns and a pattern language for rendering and interaction	190
9.2.5	RO ₅ : Patterns and a pattern language for data access	191
9.3	RO ₆ : Future artefact creation using the Pattern Language	191
9.3.1	Simple Locational Artefact	191
9.3.2	Combined Locational and Simple Vision Artefact	192
9.3.3	Feature Based MAR Artefact	192
9.3.4	SLAM MAR Artefact	192
9.3.5	MAR Object-Oriented Framework	193
9.4	Research Contribution	193
9.5	Further Research	194
9.6	Summary	195
Appendices		196
A	Mathematics for Computer Vision Overview	197
A.1	Coordinates and Cameras	197
A.2	Transforms and Pose	198
A.3	Epipolar Geometry	200
A.4	Optimisation	200
A.4.1	RANSAC	200
A.4.2	Levenberg-Marquardt	201
B	Gravity2Gravity Relative and Absolute Pose	202
B.0.1	Rotation between gravity vectors	202
B.0.2	Relative translation up to scale	203
B.0.3	Relative pose with full translation approximation	204
B.0.4	Absolute Pose	204

B.1	Implementation	205
B.1.1	Programming Details	205
B.2	Verification	206
B.2.1	Planar Relative Pose	206
B.2.2	3D Model (PnP) Pose	208
B.2.3	Data Set with Ground Truths	210
C	Pattern Index	211
C.1	Architectural Patterns	211
C.2	Detection Patterns	212
C.2.1	Locational MAR	212
C.2.2	Vision Based MAR	213
C.3	Rendering Patterns	214
C.4	Interaction Patterns	215
C.5	Data Access Patterns	216
References		217

List of Figures

1.1 Reality augmentation through the ages.	2
1.2 Caption for LOF	3
2.1 Visualisation of fields in terms of Analytic/Synthetic (horizontal) and Symbolic/Real (vertical) attributes [Owen, 1998].	12
2.2 DSR framework adapted from Hevner et al. [2004] and Hevner [2007].	15
2.3 Research onion with succeeding layers representing design choices (adapted from Saunders et al. [2007]).	21
4.1 Continuum between real and virtual worlds [Milgram and Kishino, 1994b].	43
4.2 Blob and corner detection.	48
4.3 CNN	55
4.4 The cosine law from trigonometry used in the P3P pose calculation.	59
5.1 Augmented Reality high level architecture [Reicher, 2004; MacWilliams et al., 2004].	80
5.2 Examples of pipeline architectures.	80
5.3 Video segmentation and tracking architecture using SAI [François, 2003].	83
5.4 A simple task graph based architecture for vision based detection and tracking.	85
5.5 TBB C++ source code for the task graph illustrated in Figure 5.4	86
5.6 An OpenVX graph.	87
6.1 Geographic Coordinate Systems (geodesic, ECEF, ENU) (Source: https://commons.wikimedia.org/wiki/File:ECEF_ENU_Longitude_Latitude_relationships.svg)	95
6.2 Pinhole camera model (Source: http://lfa.mobivap.uva.es/fradeg/phd/tracking/camera.html).	97
6.3 High level architectural design for a feature based MAR system utilising the STRUCTURAL TASK GRAPH pattern described in Section 5.5.1	104
6.4 Example where some detected features are mismatched	110
6.5 Factor graphs.	122
6.6 High level architectural design for a SLAM system utilising the STRUCTURAL TASK GRAPH pattern described in Section 5.5.1	127

LIST OF FIGURES

6.7 High level architectural design for a feature based MAR system utilising the STRUCTURAL TASK GRAPH pattern described in Section 5.5.1	129
6.8 The pattern language for detection and tracking for feature based MAR.	133
6.9 The pattern language for detection and tracking for SLAM based MAR (the path taken by dense/semi-dense SLAM is shown in grey)	135
7.1 Caption for Phong	143
7.2 Caption for texture mapping	144
7.3 Caption for graphics pipeline	145
7.4 Caption for scene graph	150
7.5 Capturing an IBL cube map (from [Akenine-Moller et al., 2018]).	156
8.1 Pattern Dependencies and Layered Architecture.	172
A.1 Pinhole camera model (Source: http://lfa.mobivap.uva.es/~fradelg/phd/tracking/camera.html)	198
A.2 Epipolar geometry for planar pose (Source: Arne Nordmann https://en.wikipedia.org/wiki/Epipolar_geometry)	200
B.1 Reference and reprojection images.	207
B.2 The effects of pixel noise.	208
B.3 Reprojection test for images with ground truth from PennCOSYVIO data set [Pfrommer et al., 2017]	210

List of Tables

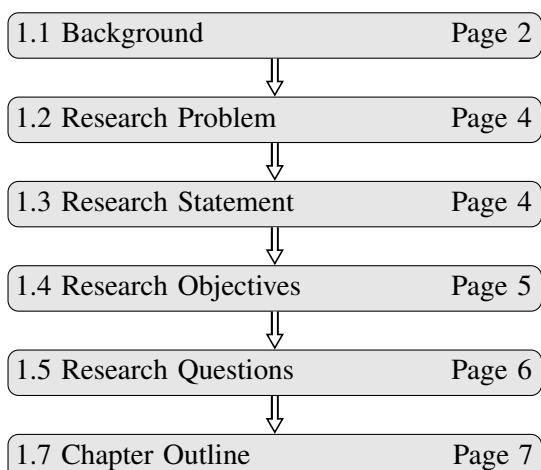
2.1 DSR outputs from March and Smith [1995], Purao [2002] and Iivari [2015]	17
2.2 DSR requirements versus Pattern properties summarised and adapted from Curley et al. [2013]	18
2.3 The research process of Nunamaker et al. [1990] mapped to the research in this thesis.	22
4.1 Advantages and disadvantages of low, medium and high fidelity MAR prototypes [De Sá and Churchill, 2012].	71
B.1 Reprojection mean error results for gravity vector planar pose with depth. The reprojections for the row in the table marked with an * are illustrated in Figure B.1b.	209
B.2 Reprojection mean error results for gravity vector PnP based 3D model pose.	209

Abbreviations

MVC	Model View Controller
ANN	Artificial Neural Network
AR	Augmented Reality
BSDF	Bidirectional Scattering Distribution Function
CG	Computer Generated
CNN	Convolutional Neural Network
CV	Computer Vision
DAG	Directed Acyclic Graph
DSR	Design Science Research
GPS	Global Positioning System
GPU	Graphics Processing Unit
IBL	Image Based Lighting
IMU	Image Based Lighting
IMU	Inertial Measurement Unit
KNN	K Nearest Neighbours
LOD	Level of Detail
MAR	Mobile Augmented Reality
MR	Mixed Reality
NDC	Normalised Device Coordinates
OO	Object-oriented
PL	Pattern Language
POI	Point of Interest
RANSAC	Random Sample Consensus
SIMD	Single Instruction Multiple Data
SLAM	Simultaneous Localisation and Mapping
TBB	Threading Building Blocks
TUI	Tangible User Interface
VR	Virtual Reality

1

Introduction



1.1 Background

Humans, like most other living organisms, interact with the world through their senses. Humans, however have a unique intellectual attribute of exhibiting a desire to embellish or enhance the reality they perceive. This desire was already evident early in human prehistory when hunters would paint hunting scenes often portraying prey with exaggerated traits such as size (Figure 1.1a).

As human societies evolved the requirement for recording more prosaic realities, for example debt, led to the development of symbolic representation and writing [Schmandt-Besserat, D., 2014]. Advances in the symbolic representation of reality in turn provided ever higher levels of abstraction for enhancing reality in many spheres of human endeavour, for example paintings in the visual arts and blueprints in engineering.

During the industrial age, the speed and flexibility of available technology tended to be a bottleneck limiting the creation of useful reality enhancing ideas. With the advent of the information age, computer based technologies such as computer graphics hardware and software have emerged which, by digital abstraction of information, allow for rapid and flexible access to varied representations of reality and the ability to mix reality with reality enhancing symbolism [Ohta and Tamura, 2014].

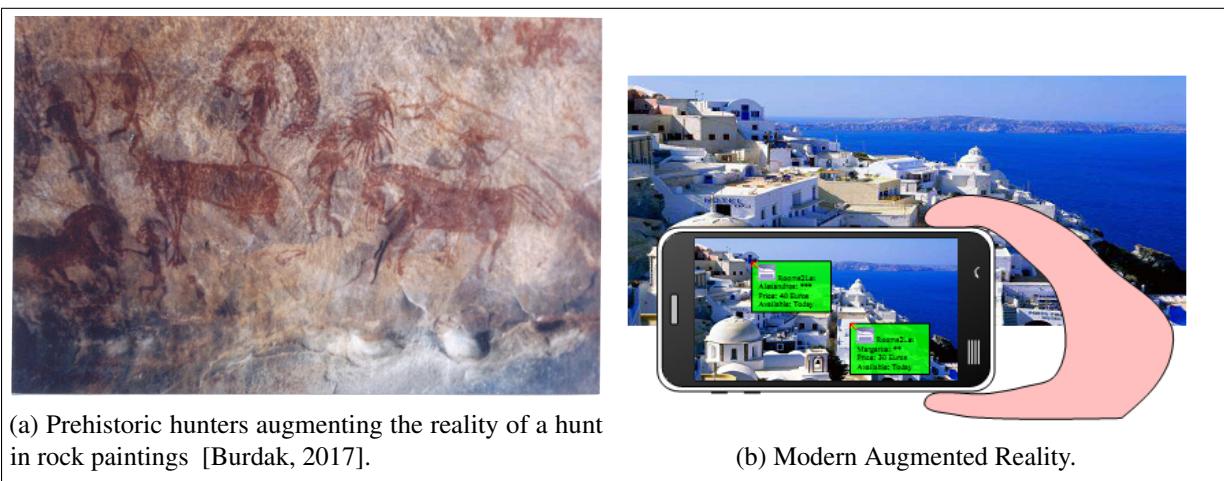


Figure 1.1: Reality augmentation through the ages.

The late 20th century saw the emergence of Mixed Reality (MR) as a field in Computer Science which strives to use technology as a medium to provide modified or enhanced views of reality (Figure 1.1b). Mixed Reality comprises a spectrum of technologies [Milgram and Kishino, 1994a] ranging from the technological depiction of the real-world to the completely virtual worlds of virtual reality (Figure 1.2). Augmented Reality (AR) is a subdomain of Mixed Reality that provides a modern perspective on

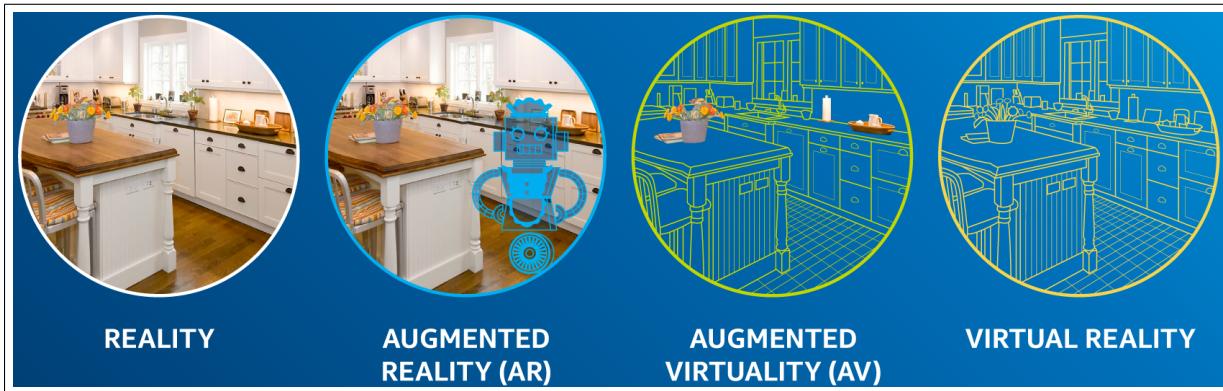


Figure 1.2: Mixed Reality Virtuality Continuum¹ [Milgram and Kishino, 1994a].

the age-old human desire to enhance our understanding of reality by overlaying it with multiple related information sources. As vision is the dominant sense for humans, visual augmentation is the predominant source, however other sensory inputs such as sound or haptic (touch) have also been used. AR in general and AR software development in particular will be discussed in greater detail in Chapter 4.

Historically AR was restricted to dedicated hardware, thus restricting its appeal to smaller audiences. In the last decade, however, the computing power of mobile devices has increased by orders of magnitude and Mobile Augmented Reality (MAR) implementations on these devices has become feasible. The expansion of AR into the mobile realm and the consequent increased demand for new and complex features has led to ever greater software complexity and a concomitant increase in the difficulty associated with the design and development of AR software.

The software design pattern concept was popularised by Gamma et al. [1995] and is based on Alexander's [1979] work on documenting best practices when designing real "brick and mortar" architectures. As the word "pattern" would suggest, software patterns document design problems which occur repeatedly in many different applications in terms of problem context, solution properties and constraints (known as forces) and best practices defining a solution space in which the designer may make different design decisions based on the unique problem being solved. Pattern Languages (PL) [Buschmann et al., 2007a] combine several individual software patterns in multiple possible sequences to form a language with the individual patterns forming the language vocabulary while the valid sequences through the patterns define the grammar. Patterns and pattern languages will be covered in greater detail in Chapter 3.

Patterns and Pattern Languages are non-prescriptive in the sense that they provide a solution space in terms of relationships between different roles defined by the pattern, and in the case of a PL, the

¹Perceived reality is on the left, while VR in which a completely computer generated virtual world is created is on the right.

relationships between the patterns that make up the language. Such solution spaces are in contrast to more prescriptive “blueprint” or “recipe” style design frameworks. Patterns and Pattern Languages can thus provide a key building block for a generalised and non-prescriptive approach to overcoming the difficulties in AR artefact design described in an earlier paragraph [Buschmann et al., 2007a]. Providing such a solution is the aim of this research and will be outlined in the next section describing the research problem.

1.2 Research Problem

The complexity associated with mobile AR design and implementation alluded to in Section 1.1 arises from MAR being a synthesis of many technologies including computer graphics, computer vision, machine learning and mobile device programming while also requiring multi-disciplinary knowledge of fields as diverse as Linear and Matrix Algebra, Projective and Differential Geometry, Probability and Optimisation. This multi-disciplinary nature of MAR has led to a fragmentation of knowledge into various specialisations, making it difficult to integrate different solution components into a coherent architecture that can integrate the components optimally. Academic papers and other information sources for MAR development tend to describe solutions to specialised problems occurring in MAR development which can make it difficult for non-specialists to understand the context in which the solution is to be applied. In addition, there frequently are a bewilderingly large number of different solutions documented for the similar or closely related problems. There is also little guidance for the overall architecture of MAR applications as most of the published solutions address specific technological problems.

The research problem can therefore be summarised as:

The design and implementation of Mobile Augmented Reality software artefacts² is difficult with minimal or no guidelines available on an overall architecture and fragmented low level guidance for individual components.

1.3 Research Statement

As related to the research problem defined in the previous section, the research statement is:

²Note that in this section and in subsequent sections the term *software artefact* will refer to interactive software applications, object-oriented frameworks [Fayad and Schmidt, 1997] and software libraries.

The design process for MAR can be encapsulated into one or more Pattern Languages which can provide detailed but non-prescriptive guidance for each step of the design of a MAR software artefact.

1.4 Research Objectives

As the aim of the research is to develop PLs to aid in the design of MAR software artefacts, the main objective encompasses this aim:

RO_M: *Specify one or more Pattern Languages providing a fully generalised, non-prescriptive design aid for MAR detailing and documenting the overall architecture and individual components required for various types and levels of complexity of MAR software artefacts.*

However in order to accomplish this objective, the main objective needs to be refined to identify or define an architecture for MAR, describe the architecture as patterns and identify other standalone patterns to be used in the PLs. Before commencing however, a literature review covering MAR and PLs should also be accomplished.

RO₁: *Conduct a literature review of Mobile Augmented Reality, Patterns and Pattern Languages.*

RO_m can then be further refined. First define an architecture suited for real-time processing of data streams from multiple sources combined with 3D graphical rendering and interaction:

RO₂: *Identify software architectures applicable to MAR and document them as architectural patterns.*

The next objective will be the identification of patterns for the primary components of MAR applications combined with the specification of pattern languages covering these patterns. These components comprise:

- Detection and tracking;
- Rendering and interaction; and
- Data source access.

RO₃: *Identify patterns and a pattern language applicable to MAR detection and tracking.*

RO₄: *Identify patterns and a pattern language applicable to MAR rendering and interaction.*

Many MAR applications need to retrieve data based on their current location. The nature of this data varies from annotations, images and other content about local features that must be augmented to visual features required by computer vision methods to detect local objects.

RO₅: *Identify patterns and a pattern language applicable to MAR data source access.*

As will be discussed in Chapter 3, PLs have a very wide scope. In the case of a PL for MAR, the scope could range from simple locational AR applications through complex computer vision based Simultaneous Localisation and Mapping (SLAM) (see Subsection 4.4.8 and Chapter 6) applications, all the way to a generic OO framework (Section 3.2.4) or product-line architecture (Section 3.2.5) covering all aspects of MAR development. In order to be able to exercise the PL in the real-world, the final research objective will then be to:

RQ₆: *Designate MAR software artefacts whose future design and development would cover a wide enough continuum in the solution space of the PLs to allow for both the evaluation of the PLs and probably also lead to improvements and extensions to the PLs.*

1.5 Research Questions

The main research question to be addressed is:

RQ_M: *How can the design of AR applications for mobile devices be abstracted and generalised into one or more Pattern Languages describing best practises for the design and development of AR artefacts with varied levels of size and complexity ?*

Several related questions on which the main question depends are:

RQ₁: *What software architecture or architectures can be identified, adapted or created and documented as architectural patterns for MAR?*

RQ₂: *Which general, reusable solutions for MAR tracking, detection, rendering and data source access can be identified and documented as patterns?*

RQ₃: *Which combination of architectural patterns identified in RQ₁ and individual patterns identified in RQ₂ can be combined into one or more Pattern Languages for MAR design?*

RQ₄: *What selection of MAR software artefacts could be designed using the Pattern Languages developed in RQ₄ that would be fully representative of the solution space provided by the PLs.*

1.6 Ethics

This research does not involve:

- Human research subjects;
- Animals or genetically modified organisms; or
- Sensitive official or archival documents.

therefore no ethical clearance was required.

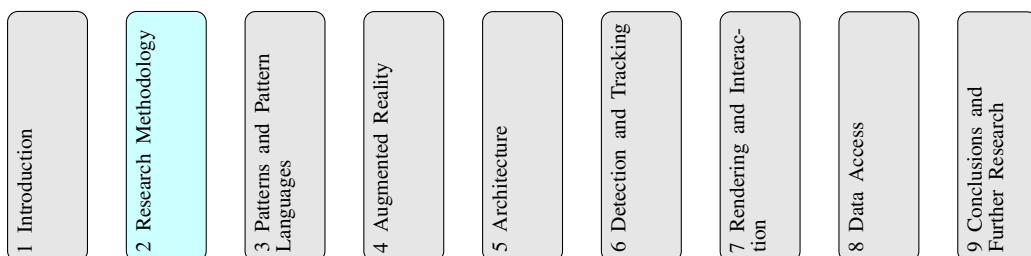
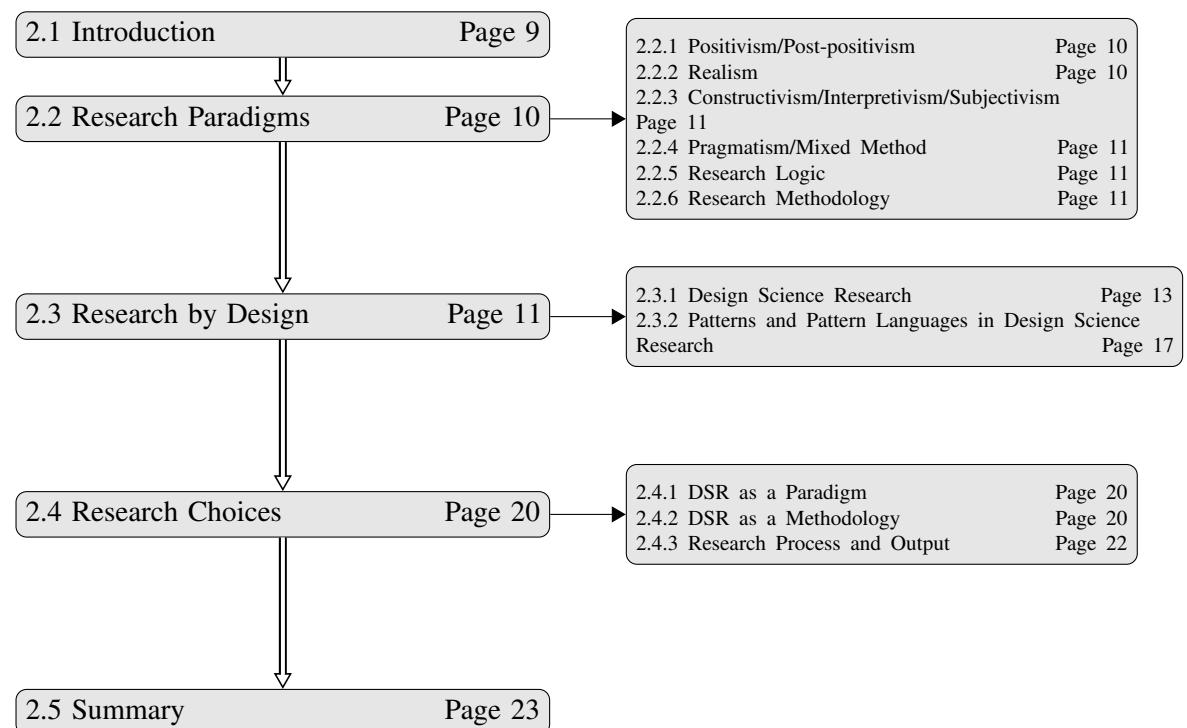
1.7 Chapter Outline

The document comprises nine chapters. Chapter two describes the research methodology, while chapters three and four provide a theoretical background for patterns, pattern languages and MAR. The rest of the dissertation describes the research in terms of patterns identified and pattern languages developed. The table below lists all nine chapters and, where applicable, the research questions they address.

Chapter	Description	Objective
1: Introduction	This chapter.	
2: Research Methodology	The research methodology.	
3: Patterns and Pattern Languages	A literature review of the theory underlying Patterns and Pattern Languages.	RO ₁
4: Mobile Augmented Reality	A literature review of the theory and practise of AR and specifically AR development on mobile devices.	RO ₁
5: Architecture	Identify and refine software architectures for MAR and develop a PL for MAR architecture.	RO ₂
6: Detection and Tracking	Identify patterns and define a PL for detection and tracking.	RO ₃
7: Rendering	Identify patterns and define a PL for rendering.	RO ₄
8: Data Sources	Identify patterns and define a PL for MAR data access.	RO ₅
9: Conclusions and Further Research	Evaluation of the research and discussion about future work.	RO ₆

2

Research Paradigm and Methodology



2.1 Introduction

The Oxford online dictionary defines research as being “The systematic investigation into and study of materials and sources in order to establish facts and reach new conclusions”. This somewhat insipid definition has the advantage of not offending any of the participants in the “science wars” of the 1990s, but does not go into the specifics of how the act of studying is accomplished or how to determine that the conclusions are conclusive. In the natural sciences, the scientific method has a long history of refinement by well known scientists and philosophers including Aristotle, Bacon, Descartes, Galilei, Newton, Hume, Kant and more recently Popper [Lewens, 2016]. Research in these fields combines theories expressed mathematically with experimentation to attempt to identify universal truths. In contrast, the social sciences have gravitated towards using a more qualitative approach which, in some cases, rejects the notion of physical truths in favour of only subjective truth. Attempts by several philosophers to apply this philosophy to the natural sciences, that is, to say that natural science theories are social constructs by scientists and should be given equal standing with other avenues of knowledge such as astrology or acupuncture, led to the above-mentioned “science wars” [Gauch, 2003].

The aforementioned research philosophies used in the natural and social sciences tend to be well defined, albeit diametrically opposed. In contrast, research in Computer Science (CS) and Information Systems (IS) tends to be multi-disciplinary often including facets from both the physical and social sciences as well as from the engineering sciences. Because of the lack of a historic precedent for a research philosophy in CS/IS research, there is a requirement to explicitly define the underlying research philosophy used. Additionally research in CS/IS, in common with engineering disciplines, differs from both the natural and social sciences in that its research output often consists of some form of artefact, or a description of optimal approaches for building an artefact.

As the research being conducted has the primary objective of designing a PL or PLs to aid in the design of MAR artefacts, it matches the description from the preceding paragraph of the research output being an artefact. The research is distinguished from many other CS/IS research projects in that the artefact is in fact a meta-artefact which, in turn, is used to create real artefacts.

This chapter will briefly describe the competing research philosophies and methodologies. It will then go into greater detail on research methodologies applicable to design in general and CS/IS design in particular. It will conclude by defining the relevant paradigms and methodologies used in this research.

2.2 Research Paradigms

A research paradigm is a set of common beliefs and agreements shared between scientists about how problems should be understood and addressed [Kuhn, 1970]. The underlying tenets of research paradigms vary between different disciplines. These differences are codified by the research paradigm or perspective utilised by the discipline, which defines the overall worldview of the researcher in terms of three components [Guba, 1990]:

- *Ontology* defines how the nature of reality is represented and what knowledge can be extracted from the defined reality;
- *Epistemology* defines the relationship between the researcher and the available knowledge as constrained by the ontology specified by the paradigm; and
- *Methodology* defines which strategies or methods can be used to gather knowledge given the ontology and epistemology selected by the paradigm.

Conceptually ontologies and epistemologies are tightly linked, indeed some authors refer to an ontology/epistemology combination as an epistemology. A number of these paradigms are described in the following subsections.

2.2.1 Positivism/Post-positivism

The theory of knowledge which historically forms the roots of Western Science is based on *positivism* which asserts that only information derived from logical and mathematical treatments or from verifiable observations obtained by experimentation can be considered as authoritative knowledge and that this is the only valid form of knowledge. Positivism is a combination of two earlier philosophies, namely rationalism or using reason to deduce theories with empiricism or taking measurements from nature or when performing experiments [Feichtinger et al., 2018].

Post-positivism is a more recent modification to positivism which places great emphasis on use of theories and the falsifiability of these theories. It also asserts that theories can only be falsified, not confirmed.

2.2.2 Realism

Scientific Realism's basic tenet is that scientific theories reflect a “real” reality [Saatsi, 2017]. To illustrate using an example from physics, to a realist elementary particles such as quarks have a real existence, while from a positivist/empirical viewpoint their existence is immaterial outside of being a part of a model of reality which agrees with observations. For the most part realism and positivism are similar in the ways

in which knowledge is gleaned using theories and experiments.

2.2.3 Constructivism/Interpretivism/Subjectivism

Constructivism/Interpretivism maintains that knowledge is subjective and is created by human societies, and there is no single “true” reality. Subjectivism claims that reality is what the observer perceives as reality. In these paradigms data is frequently interpreted qualitatively.

2.2.4 Pragmatism/Mixed Method

While positivism is quantitative in nature and constructivism/interpretivism/subjectivism mostly qualitative, pragmatism attempts to integrate quantitative and qualitative research and use whichever one is most likely to provide useful insights, without taking any philosophical positions about any universal truth.

2.2.5 Research Logic

Positivism and positivism derived paradigms use *deductive reasoning*, that is, a top down approach that starts with a theory, produces a hypothesis based on the theory and attempts to verify the veracity of the hypothesis in order to confirm the theory. *Inductive reasoning* is an alternate approach to the deductive method and is used most frequently by qualitative research ontologies. It is a bottom up approach that, given a set of observations, attempts to infer whether a given assertion is valid and therefore a valid theory can be inferred from the assertion. It is frequently combined with probabilistic inference, that is, the result is a probability of a particular assertion being valid. Pragmatic and mixed-method paradigms can use a combination of deductive and inductive logic dependent on which approach is likely to give the best results.

2.2.6 Research Methodology

The research methodology describes the strategy employed as a whole when executing the research. Numerous such methodologies exist and specific methodologies tend to be linked to particular discipline. The research pertaining to this thesis is CS/IS specific therefore methodologies suited to CS/IS will be investigated in the next section.

2.3 Research by Design

Research in the natural and social sciences provide their research output as new theories or to support, supplement or dispute an existing theory. In contrast, fields such as engineering and architecture are frequently concerned with creation of artificial constructs and so research in these fields must address the issue of creation of these constructs. Simon [1981] defined fields such as these as belonging to the

CHAPTER 2: RESEARCH PARADIGM AND METHODOLOGY

science of the artificial which encompasses the body of knowledge about artificial (man-made) objects as opposed to phenomena from the natural world.

Owen [1998] visualises how the sciences of the artificial differ from natural or sociological sciences based on two axes classifying research activities within the science (Figure 2.1). The vertical axis is the Symbolic/Real axis with sciences at the top being concerned primarily with the symbolic, for example pure mathematics while those at the bottom are concerned with the real-world. The horizontal axis is the Analytic/Synthetic with fields on the left concentrating their efforts towards exploration and discovery while those on the right are have a greater focus towards invention and manufacture.

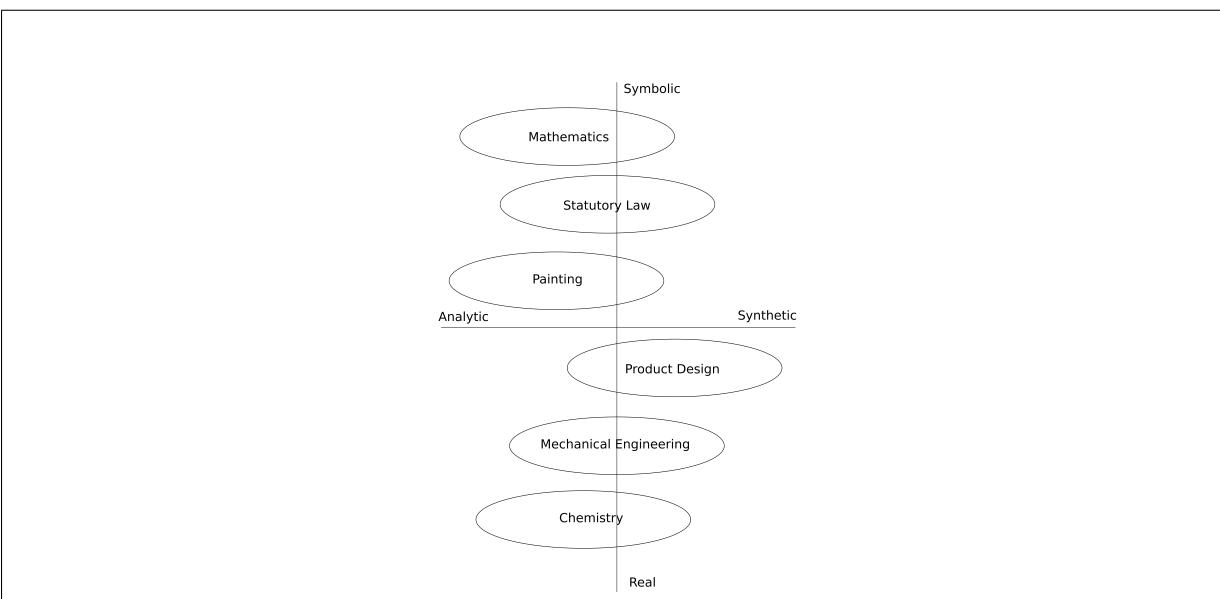


Figure 2.1: Visualisation of fields in terms of Analytic/Synthetic (horizontal) and Symbolic/Real (vertical) attributes [Owen, 1998].

Sciences of the artificial frequently interlink knowledge discovery or research with the creation of physical or intellectual artefacts which can be evaluated to improve understanding of the field and are also directly usable [Owen, 1998]. The pursuit of knowledge in these fields encompasses both research and design. Such fields usually have a larger synthetic component in Figure 2.1. If the artefact is a physical one such as a bridge or a building, then the field will appear near the bottom of the vertical axis of Figure 2.1, while fields that create abstract or intellectual artefacts such as software designs will appear near the top.

In the case of Computer Science considered as a science of the artificial, both the computer (hardware) itself and programs running on the computer (software) could be seen as human engineered artefacts to be investigated. The design of the software artefacts could also be seen as a kind of meta-artefact

worthy of investigation. These artefacts are frequently hierarchical systems comprising multiple sub-systems. Such systems frequently exhibit unpredictable *emergent behaviour* [Bar-Yam, 1997] due to the interactions between various components and sub-systems of the system. For such systems simulation can be used to infer theories about the behaviour of the system while prescriptive design theories (see Section 2.3.1.1) can be used to document best practises in creating the artefacts.

Historically Computer Science has a positivist, basis in the mathematics of Turing, Church and Von Neumann [Dyson, 2012]. In the modern era CS/IS research involving the theory of computation, discrete mathematical applications and Algorithmic analysis for example, are mathematical in nature. In the intervening decades since the birth of the computer, computer hardware, Computer Science and particularly human societies use of computers have evolved considerably. As a result, the majority of CS/IS research typically either involves the creation of software artefacts or enabling technology to assist the creation of such artefacts so CS/IS can for the most part be seen to belong to the science of the artificial, therefore research paradigms such as Design Science Research discussed in the next section are appropriate for CS/IS research.

2.3.1 Design Science Research

The requirement to incorporate research and design in Simon's [1981] sciences of the artificial led to the emergence of a new research approach known as Design Science Research¹. Whether DSR should be treated as a new research paradigm or a research methodology to be used in conjunction with existing paradigms is still controversial [Aljafari and Khazanchi, 2013]. Hevner et al. [2004] argue for DSR as a standalone paradigm, and Vaishnavi and Kuechler [2007] argue that DSRs metaphysics are unique in that the ontology and epistemology changes during different phases of the DSR cycle. Niehaves [2007] take a different view arguing that DSR in its present form has an implicit positivist epistemology and prefer a pragmatic approach to selecting traditional paradigms whilst using DSR as a methodology. McKay and Marshall [2007] also point to the use of different traditional paradigms within DSR, pointing out that existing applications of DSR can be divided into those using a positivist approach in treating a design artefact empirically, and those taking a more sociotechnical interpretivist approach.

In the following paragraphs a brief description of the proposed ontology and epistemology for DSR as a standalone paradigm will be outlined. The Methodology section which follows is common to either standpoint, although it is somewhat slanted to having positivist underpinnings as the proposed research for

¹The science in DSR distinguishes it from Design Research (DR) which concerns research into design rather than using design to do research.

which the methodology will be used in this document does not have an extensive sociological component. The actual choice of paradigm/methodology to be used is delayed until Section 2.4.

Ontology: The design of an artefact and its eventual realisation represent a “virtual reality” of sorts [Purao, 2013], as the design and eventually the artefact itself are an intellectual construct and form a reality of their own which can be manipulated by the researcher by altering the design or implementation. It is however implicitly understood by the researcher, that there is an underlying positivist “real” reality within which the “virtual” realities exist, for example the electrons in the CPU and memory of the computer which provide the continually shifting states representing the artefact are modelled by physics.

The incorporeal nature of the phenomenon being created/investigated (the phenomenon being the design and realisation of the artefact) provides a different view of reality compared to positivism. Firstly the ontology is complementary as the researcher can have differing views of reality for the design and for the artefact, as described in the previous paragraph. It is also evolutionary in that the understood reality varies during the research. For example, the artefact evolves from having an emergent reality, only existing as a intellectual conception in the mind of the researcher, to having a realist ontology while being evaluated as a existing phenomenon at conclusion of the DSR process.

Epistemology The epistemology of DSR can be described as knowing through making rather than observing or experimenting. For example, DSR experimentation involves altering the “reality” of the phenomenon, that is design or artefact, instead of performing experiments on a static phenomenon. [Purao, 2013].

The process of knowledge accumulation is iterative with feedback from the artefact construction and evaluation adding to the knowledge base with the objective of continually improving the artefact design with respect to the requirements specified by the evaluation criteria. The objective is to allow the researcher to improve his understanding of the research area while simultaneously improving the design of the artefact.

2.3.1.1 Methodology

The overall framework for DSR is illustrated in Figure 2.2, comprising three main boxes namely from left to right the *Environment*, *Research* and *Knowledge base* which are described below.

Environment The Environment represents the problem area under investigation. The problem area includes the problem as well as problem related entities, for example users of a proposed software artefact being designed as a solution for the problem. The problem itself may be classified as either [Wieringa,

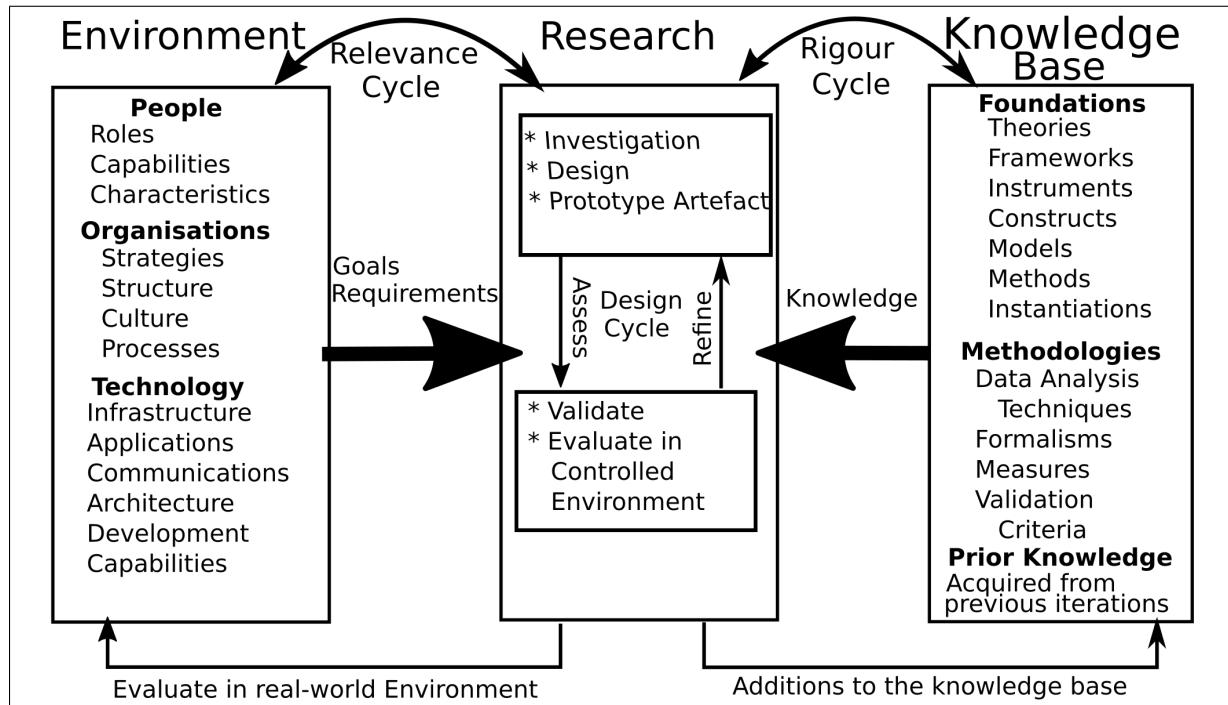


Figure 2.2: DSR framework adapted from Hevner et al. [2004] and Hevner [2007].

2009]:

- Design problems for which the solution is a design of some sort. In this case the use of ‘a design’ and not ‘the design’ implies that it is possible that many possible correct solutions exist, and the selected design frequently is dependent on the designer. The designer can also produce multiple different artefacts for comparative evaluation; or
- Knowledge problems that have a unique answer. In this case the artefact is a tool for the investigation of an empirical question, with the solution possibly being the correct one, or several solutions with varying probabilities of being correct or a partial solution.

Research The bulk of the research is carried out here where artefacts that address the problem are developed and evaluated. The *design cycle* is an iterative feedback cycle comprising several steps with knowledge flows in both directions resulting in iterative improvement. The steps are:

1. Problem investigation leading to design goals and requirements.
2. Design. The output of the design can optionally be used to construct a prototype solution. If the requirements from step 1 are not sufficient to create the design, return to step 1.
3. Validate the design or prototype. If the validation fails return to step 2.
4. Implement an artefact derived from the prototype.

5. Evaluate the implementation.

In the above validation (3) is seen part of the iterative experimental process which tests the design result (or prototype) within the problem context (or a simulation of the context) to verify that the goals are met. During validation the design is used to create and modify a design theory (discussed below) which models the interplay between the artefact and the problem context. Evaluation [Venable et al., 2012] can occur during the design cycle in a controlled environment and at the termination of a design cycle (although not necessarily the whole process). The latter involves evaluating a fully implemented artefact ‘in the wild’, that is, within a real-world context.

Knowledge Base The knowledge base comprises all external information sources concerning the problem area including prior research, related research, related disciplines and fields and results from the previous Design Science cycle.

Design Theories Design theories are the sciences of the artificial counterpart to scientific theories in the natural sciences. Baskerville and Pries-Heje [2010] differentiate between two forms of design theory:

- Design Practise Theory (DPT) which is prescriptive and describes in practical terms how to design an artefact. Walls et al. [1992, p. 37] provides an early definition for such a theory as “... a prescriptive theory based on theoretical underpinnings which says how a design process can be carried out in a way which is both effective and feasible”; and
- Explanatory Design Theory which relate generalised requirements to generalised components of a system. The generalisability means a wide range of possible designs can examined for different component combinations. The components are supplied by the design and relating specific component combinations to the requirements provides a means of evaluating the design.

Cycles In addition to the design cycle, Hevner [2007] describes two other cycles in the DSR framework. The *Relevance Cycle* provides a feedback loop between the Environmental context and the DSR context with initial design goals forming input to the DSR while proposed artefacts are output from the DSR to the Environment. The *Rigour Cycle* continually verifies the research contribution of the artefact by monitoring the Knowledge base.

2.3.1.2 Design Science Research Output

The output from the DSR process is important as it documents the research results and forms a theory of knowledge about the research area which in turn can be peer reviewed or utilised by other researchers when researching related areas. Unfortunately there are no set standards in regard to DSR output artefacts,

however the most widely accepted recommendations are constructs, models and methods as proposed by March and Smith [1995]. These, along with further enhancements from Purao [2002] (design theories as an additional output) and [Iivari, 2015] (types of the design theory artefacts) are enumerated in Table 2.1.

Table 2.1: DSR outputs from March and Smith [1995], Purao [2002] and Iivari [2015]

Output	Description
Constructs	The conceptual vocabulary of a domain
Models	A set of propositions or statements expressing relationships between constructs.
Methods	A set of steps used to perform a task.
Instantiations	The operationalisation of constructs, models, and methods.
Design Theories	Theories derived from meta-artefact construction or prescriptive knowledge extracted from concrete artefact construction [Iivari, 2015].

2.3.2 Patterns and Pattern Languages in Design Science Research

As a design pattern (see Chapter 3) is ‘a three-part rule, which expresses a relation between a certain context, a problem, and a solution’ [Alexander, 1979], it can be seen as a building block for Design Practise Theories (DPT) (see Section 2.3.1.1) as the solution provides a description of how to solve the problem within a given context. Walls et al. [1992] define a DPT as being both prescriptive and descriptive (as it ‘says how a design process can be carried out’). Patterns, however, are not rigidly prescriptive, instead they define a solution space in terms of roles and relationships and allow the designer to create a design using the pattern as a guide. Theories composed of patterns therefore provide greater generalisability than very specific prescriptive theories do which makes patterns attractive for DSR theories given that difficulties with generalisability is one of the issues encountered by designers using DSR [Baskerville and Pries-Heje, 2015].

Curley et al. [2013] suggest the correspondences between the problems addressed by DSR research and the solution providing guidelines provided by Design Patterns can provide a powerful approach to both formulate and document DSR theories (See Table 2.2 for a summarised version of the table of correspondences from Curley et al. [2013]). Also in 2013, Buckl et al. [2013] proposed a pattern based design research method comprising four main activities namely, observe and conceptualise, pattern-based theory building, solution design and application where a concrete artefact is created, and design and

CHAPTER 2: RESEARCH PARADIGM AND METHODOLOGY

artefact evaluation. Buckl et al. [2013] also provide an example design using this research method.

A single pattern provides solutions only to specific problems therefore it can only exist as a building block in a design theory and a low-level artefact in a DSR artefact collection. Moreover patterns rarely exist in isolation, instead patterns have relationships to other patterns (for example the famous Gang of Four book [Gamma et al., 1995] provides a map depicting the relationships between different patterns documented in the book). Types of pattern relationships include [Buschmann et al., 2007a]:

- Composition where a pattern combines with another to add functionality;
- Complement where one pattern complements another either as a competing alternative or in a part-whole relationship;
- Compound where the pattern is part of a larger collection of patterns, for example the well known Model-View-Controller (MVC) pattern is really a pattern compound; and
- Sequence where the pattern is part of a collection of patterns executed in a specific order.

In order to create a pattern based design theory, individual patterns need to be combined in some way. The resulting theory would be a meta-artefact which can be used by software practitioners (seen as users in the Environment in Figure 2.2) to construct specific concrete artefacts. The dichotomy between using DSR for creating meta-artefacts versus creating a solution to a specific problem by building a concrete artefact and extracting prescriptive knowledge from the design process is discussed by Iivari [2015].

Table 2.2: DSR requirements versus Pattern properties summarised and adapted from Curley et al. [2013]

DSR Requirement	Design Pattern Property
Unstable requirements and constraints based on ill-defined environmental contexts.	Encapsulation and Abstraction. Each pattern encapsulates a well-defined problem/context/forces/solution-space and makes clear in which context(s) they apply.
Complex interactions among subcomponents of problem and resulting subcomponents of solution.	Composability. Patterns can be combined with related patterns via relationships between roles provided by the patterns.
Flexibility to change design processes as well as design artifacts.	Relationships between roles defined by patterns can be varied to support structural, behavioral, domain, internal, language and platform variations [Buschmann et al., 2007a, ch. 2].

DSR Requirement	Design Pattern Property
Dependence upon human cognitive abilities (e.g., creativity) to produce effective solutions.	Generativity. Patterns provide solution spaces not specific solutions and require human designers to adapt the pattern to suit the design.
Dependence upon human social abilities (e.g., teamwork) to produce effective solutions.	Patterns provide a common vocabulary for teams of designers and practitioners.

One possibility for creating a pattern based design theory as a DSR meta-artefact would be to create a pattern collection [see Buschmann et al., 2007a, ch. 8], however such an approach would not provide a descriptive theory as there would be no guidance from the designer to the practitioner on how to combine the patterns to form concrete artefacts. For smaller meta-artefacts with a fixed sequence through the patterns, it might be possible to use a pattern sequence to create the meta-artefact. For most DSR problems requiring complex combinations of paths through the patterns a Pattern Language (PL) is required.

A PL combines multiple possible sequences through its patterns to form a language with the individual patterns defining the vocabulary while the valid sequences through the patterns map the available grammar [see Buschmann et al., 2007a, ch. 9, 10]. In the same way a pattern is generalisable through the fact that it provides a solution space instead of a specific solution, a PL is generalisable as it provides an even larger combinatorial solution space of all the patterns in its vocabulary and the different paths that can be taken through the PL. A PL therefore provides a generalisable descriptive design theory in which [see Buschmann et al., 2007a, p. 298]:

a specific solution is developed stepwise through many creative acts until it is complete and consistent in all its parts. Each individual act within the process of piecemeal growth differentiates existing space. A given design or situation in which a particular problem arises is thus gradually transformed into another design or situation in which the problem is resolved by an appropriate pattern of the applied language.

2.4 Research Choices

As the research involves the creation of a meta-artefact, that is an artefact guiding the construction of software artefacts, DSR will be used. As discussed in Section 2.3.1, the nature of DSR as either a paradigm or a methodology is still under debate. For the purposes of this research neutrality will be maintained regarding this dispute and briefly outline the design choices under both eventualities.

2.4.1 DSR as a Paradigm

With DSR as a paradigm, the ontology/epistemology is as described in subsection 2.3.1. The ontology posits the artefact as having a “virtual reality” within “real” reality which is constantly evolving during the design. This matches the use of both patterns and PLs which form an abstraction of reality represented by roles and the relationships between the roles. The patterns themselves are constantly evolving to accommodate new use cases [Buschmann et al., 2007a], and the PL also evolves as new patterns are added or the relationship between the patterns changes.

DSR epistemology involves iteratively extracting information from the artefact by repeatedly “altering its reality”, that is, making changes to the design, instead of performing experiments on natural phenomena [Purao, 2013]. This again corresponds well with the development of a PL using identified patterns, as a PL provides piecemeal growth and the PL itself can be built piece by piece as the solution space expands.

DSR research logic is not explicitly discussed in the literature, however a deductive approach is sometimes implicitly assumed. In the case of patterns though, an inductive approach is also required because the candidate patterns are identified from existing source code or design documentation. For this reason the research will use both inductive and deductive reasoning.

DSR methodology has already been fairly comprehensively described in Section 2.3.1.1, and so won’t be revisited again in this section.

2.4.2 DSR as a Methodology

Saunders et al. [2007] provide a summary of the various components of a research paradigm using the metaphor of a onion with succeeding layers representing the different choices available. A slightly amended version of this image is shown in Figure 2.3, with DSR added in light grey to both the research paradigm/philosophy layer and the methodology layer to illustrate both possibilities.

Using the DSR as methodology approach and starting at the outer layer of the metaphorical onion, a positivist paradigm would provide the best fit for the research, although perhaps not at quite the same

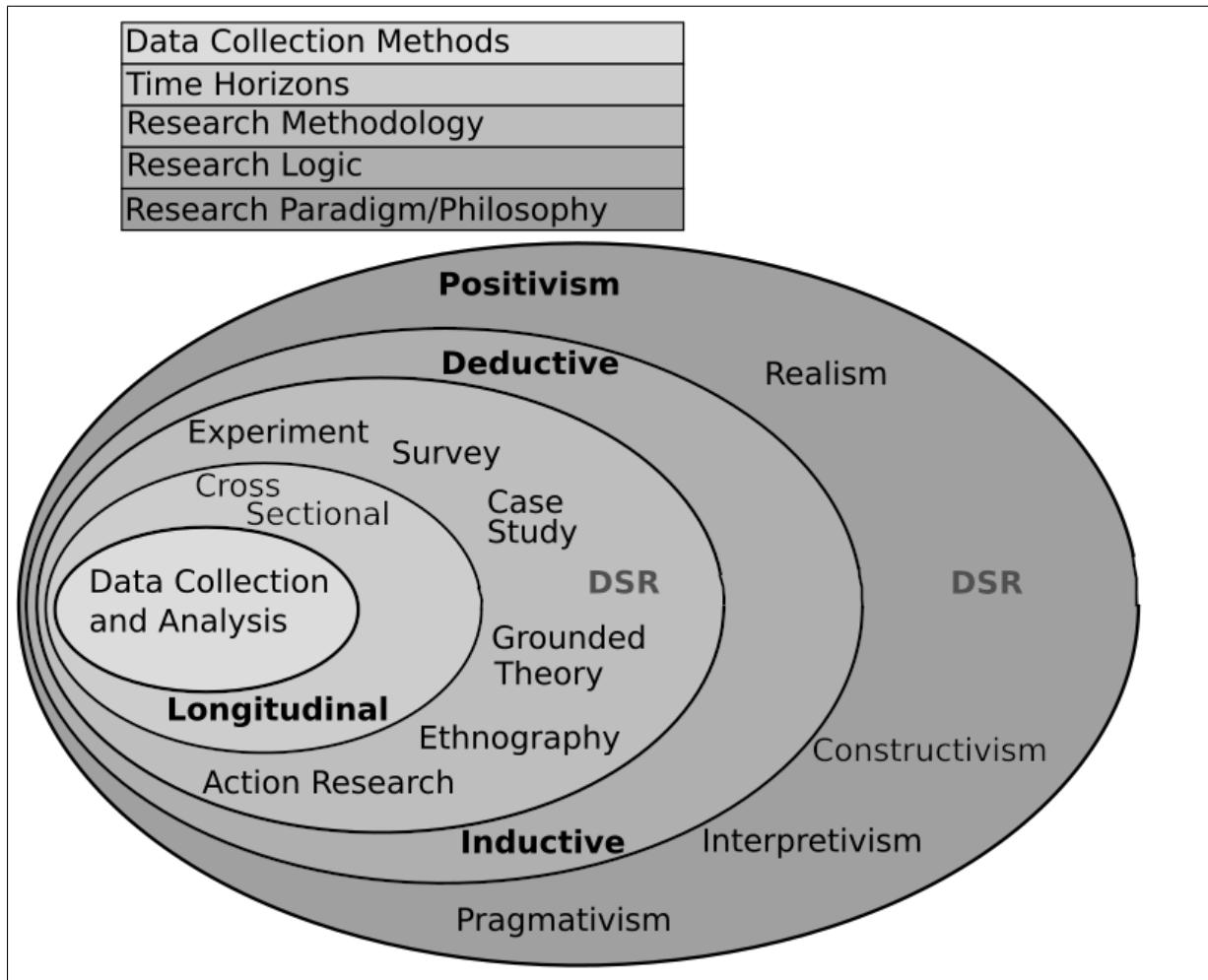


Figure 2.3: Research onion with succeeding layers representing design choices (adapted from Saunders et al. [2007]).

fine-grained level as that of the DSR specific ontology/epistemology. This would be due to the DSR concept of repeated experimentation on an artefact, combined with a theory of how the artefact should behave, or in the case of a PL, whether the result of using parts of the PL or individual patterns results in correct software proto-artefacts being created.

For the next layer (research logic), as mentioned in the previous section both deductive and inductive logic will be used. This differs from traditional positivism where only deductive logic is used.

As DSR is now considered to be at the research methodology level, it will now be used as such. See Section 2.3.1.1 for a description of the DSR methodology.

The next to last layer specifies the time extent of the research data, with cross-sectional reflecting a specific moment or short duration, while data taken over a longer term is longitudinal. Insofar as this

particular layer applies to this research, it would be longitudinal as it would refer to source code and design documents over some period of time from which patterns are gleaned.

The final layer would be the sources from which the patterns are extracted. For open source the data would be found on the Internet in various source code repositories. For closed source patterns would have to be inferred to the extent possible from design documents and API documentation. Patterns could also be extracted from academic papers where no implemented source code is available, in which case the data source would be academic archives and the Internet.

2.4.3 Research Process and Output

Iivari [2015] classifies two DSR strategies, the first targeted at creating a general solution to a general problem from which specific solutions can be generated, which will be referred to in this section as the meta-strategy. The second strategy is aimed at creating specific solution in the form of a single artefact based on specific client (the first strategy does not necessarily have any specific client) requests and then attempt to generalise a design theory from the solution. The second strategy will be referred to in this section as the is-strategy, as it is commonly, although certainly not exclusively utilised in Information Systems research. The meta-strategy output is conceptual while the is-strategy is specific, with a conceptual theory derived from the development of the artefact.

As the research in this thesis involves the design of a meta-artefact it corresponds to a meta-strategy. Iivari [2015] posit that the research process for meta-strategy research usually follows that of Nunamaker et al. [1990], which will be the process adopted here. The research process mapped to the research in this thesis comprises the stages listed in Table 2.3

Stage	Description	Objective	Chapter
Construct a Conceptual Framework	1. State research questions/objectives. 2. Understand the underlying theory (literature review).	RO ₁	1, 3, 4
Develop a System Architecture	Architectural pattern identification	RO ₂	5
Analyse & Design the System	Pattern identification and Pattern Language Development	RO ₃ , RO ₄ , RO ₅	6, 7, 8
Build the (Prototype) System	This stage is not applicable to a pattern language as it targets an entire solution space and not a single artefact		
Observe & Evaluate the System	Designate candidate artefacts for implementation using the pattern languages.	RO ₆	9

Table 2.3: The research process of Nunamaker et al. [1990] mapped to the research in this thesis.

As noted by Iivari [2015, p. 111] “it is not necessary or sometimes even possible to test and evaluate the constructed IT meta-artefact on the field when following DSR Strategy 1”, in the case of a pattern language there is no single artefact to evaluate as the pattern language generates artefacts from a

solution space. In this case a representative sampling of possible artefacts of differing complexities can be evaluated, although this does not guarantee full coverage.

The research output will be the architectural patterns, design patterns and pattern languages identified during the research. In DSR terms, the output will form a generic design theory for MAR software development.

2.5 Summary

This chapter provided a brief overview of some aspects of the philosophy of science in order to provide a background for the research choices made regarding the research being undertaken. In particular DSR was described in greater detail as it would appear to be the most appropriate research paradigm/methodology to use.

The research choices were then selected and elaborated on in Section 2.4 as:

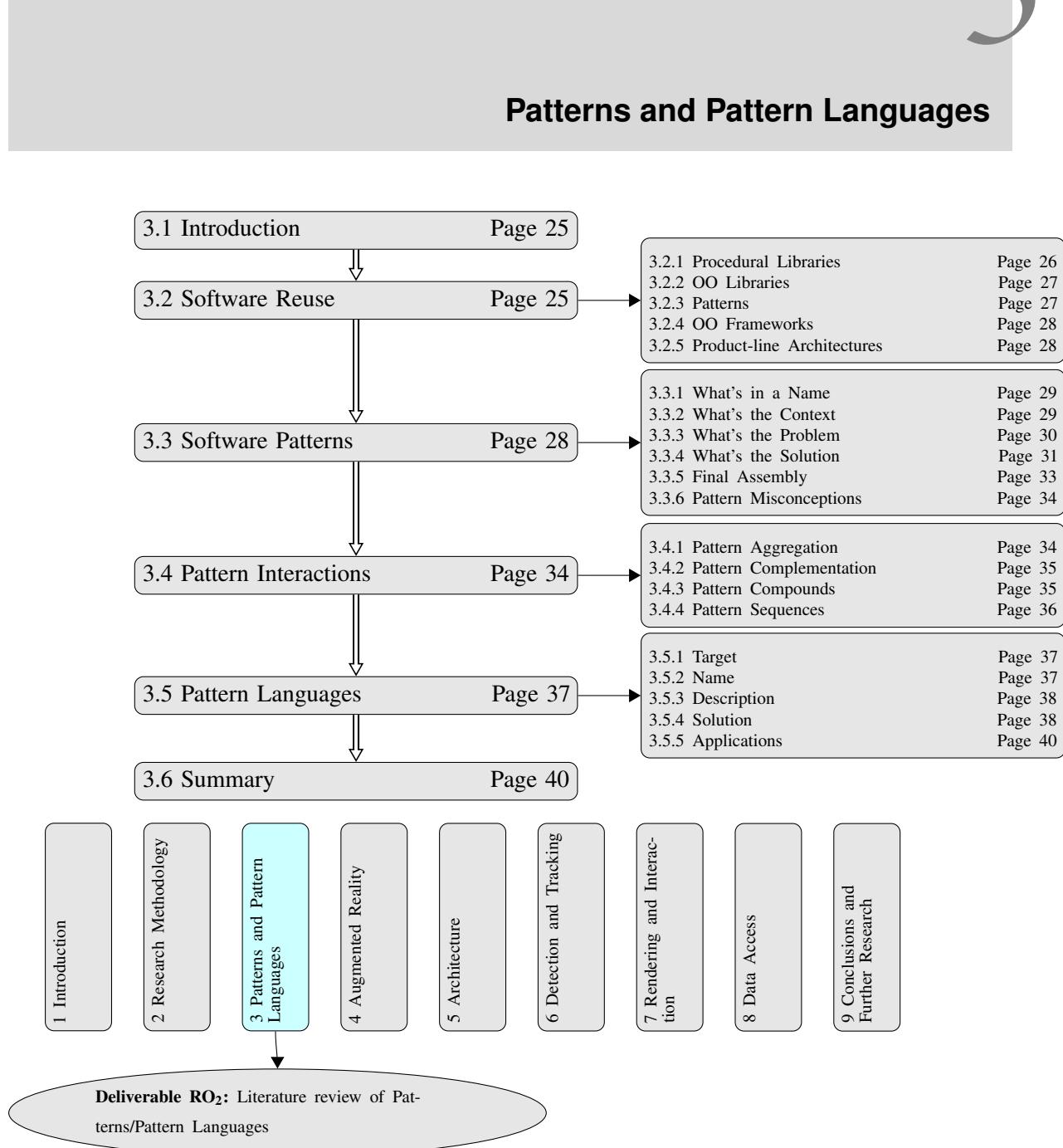
- Paradigm - DSR (as a paradigm) or positivist;
- Logic - Both deductive and inductive; and
- Methodology - DSR (as a methodology).

Section 2.4 also discussed pattern reaping from source code, APIs and academic papers as a research data collection and analysis element. Finally the research process was divided into research stages. The stages were also associated with corresponding research objectives for the thesis.

The following two chapters will provide a literature review for Patterns, Pattern Languages and Augmented Reality. After this the main research work identifying patterns and developing pattern languages will commence.

3

Patterns and Pattern Languages



3.1 Introduction

Most designers (software and otherwise) would agree that “reinventing the wheel” to solve a known problem is, at best, a poor use of resources, and at worst can result in a suboptimal solution that does not make use of the prior knowledge of, and experience invested in the solution by previous generations of designers. While reusing a design for a concrete artefact such as a wheel is usually straightforward, reuse of designs for abstract concepts can be more challenging as it depends on the design being documented in such a way that it can be reproduced. Software engineering in particular often encompasses complex abstractions which do not have physical realisations or analogues. In addition the different situations in which a software design must be reused may vary considerably, so the design documentation needs to be generalisable to different situations but still be specific enough to be useful.

The rest of this chapter will concentrate on a literature review of Software Patterns and Pattern Languages as specified by RO₁, which states:

Conduct a literature review of Mobile Augmented Reality, Patterns and Pattern Languages.

The applicable research process stage from Subsection 2.4.3 is:

Understand the underlying theory.

The output will be a review of research literature covering software patterns and pattern languages, and the theory covered will be utilised in Chapter 5 when describing architectural patterns and also in the Detection & Tracking (see Chapter 6), Rendering & Interaction (See Chapter 7) and Data Access (See Chapter 8) chapters when defining patterns and PLs.

The next section (Section 3.2) will briefly discuss the evolution of reuse in the software engineering sphere. The rest of the chapter will describe the pattern based approach for design reuse comprising:

- Patterns as standalone entities documenting singular problems the pattern solves and what the solution options are;
- Combinations of individual patterns and
- Formalised combinations of patterns into pattern languages.

3.2 Software Reuse

Software reuse can be defined as “... the use of engineering knowledge or artefacts from existing systems to build new ones” [Frakes and Isoda, 1994]. The history of reusability in Software Engineering dates back to the early 1960’s when, as computers became more commonplace, the size and complexity of the tasks computers were asked to address increased markedly. One of the early pioneers of computing,

Edsger Dijkstra, described this phenomenon as follows:

... the major cause (*of the software crisis*) is ... that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming had become an equally gigantic problem [Dijkstra, 1972].

The above quote dates back to 1972, when, by today's standards, computer hardware was still very limited. Since then Moore's law (the number of transistors in integrated circuits doubles approximately every two years) has further compounded the problem by multiple orders of magnitude. In recent years Moore's law has slowed but this has been offset by the development of parallelising technologies such as Graphical Processing Units (GPU), distributed "cloud" supercomputing using fast networking and the increased availability of fast primary and secondary memory.

There have been several different approaches to software reusability throughout the history of software engineering. The following subsections describe a few of the most prominent among these including procedural libraries, Object-oriented (OO) libraries, patterns, frameworks and product-line architectures.

3.2.1 Procedural Libraries

Much of the early history of reuse in software engineering centred on software libraries, collections of frequently used procedures or subroutines, which could be referenced by programs wishing to make use of the provided functionality. Initially such libraries had to be supplied in the form of source code (older languages such as COBOL and FORTRAN have language constructs for reading source libraries). Dependence on source proved to be inconvenient leading to the development of linkers and loaders which allowed routines to be provided in binary libraries which could be linked with the client code and the combined code could then be loaded into memory and executed using the loader. The invention of linking was probably one of the most significant contributions stemming from software libraries as the concept is now ubiquitous across operating systems and computer languages. The software library era produced several libraries which are still in use today, for example the linear algebra libraries BLAS and LINPACK/LAPACK which were written in the 1970s and 80s. Libraries are also popular in lower level areas such as embedded programming, while the system level APIs provided by operating systems can also be seen as a kind of library.

3.2.2 OO Libraries

The underlying concepts for OO programming first emerged in the early 1960's, but it was only in the 1970's when Simula 67 and Smalltalk became available that the OO paradigm started to enter the mainstream. Initially OO reuse was seen almost exclusively in terms of class inheritance and polymorphism, with reuse realised by inheriting from and extending functionality encapsulated in "base classes", possibly provided by external OO libraries. Subsequent experience gained in OO system development suggested that overuse of inheritance could lead to complicated chains of dependencies with small changes in these base classes breaking dependent inheriting classes. The "gang of four" design pattern book referenced in the next paragraph was one of the first to recommend the use of programming to interfaces and favouring composition over inheritance in OO design with inheritance being restricted to local (non-architectural) shallow hierarchies. Since the 1990's OO reuse has become predominant with the majority of popular programming languages such as Java, C++ and C# supporting it.

3.2.3 Patterns

Both procedural and OO libraries target source code level and not design level reuse which makes it difficult to generalise their use. Design Patterns, introduced by the aforementioned "gang of four" book [Gamma et al., 1995] revolutionised a new approach which sought to describe generalised software design¹ problems. The exact nature of how patterns are documented is described in the following section (Section 3.3), but the main premise is based on the architectural patterns in "bricks and mortar" architecture first introduced by Christopher Alexander.

Alexander's theory of patterns in architecture originated in the late 1960's [Alexander et al., 1968] and was documented in several subsequent publications in the 1970's [Alexander et al., 1975; Alexander et al., 1977; Alexander, 1979]. The theory involves documenting typical situations or circumstances encountered during the design process along with well known and tested solutions using a standard format for the documentation which simplifies finding a pattern for a given situation. After identifying a pattern an architect can follow links to one or more related patterns in the documentation. Software design patterns follow the same general strategy, albeit modified for use in a software design setting. In particular the documenting the situations where patterns apply and the links between patterns tends to be more complicated for software designs due to the complexity of the situations in which design variations occur and the combinatorial complexity of pattern linkage for software.

¹Patterns can also be identified in other areas, for example organisational patterns, but this document considers only software design patterns.

3.2.4 OO Frameworks

While patterns provide generalised solutions, OO techniques can be applied to utilise patterns in particular narrowly defined problem domains using a OO language. Frameworks are one such technique that provide a semi-complete application in a particular domain and allow the user to complete the application by filling in the necessary parts that distinguish the application. Such frameworks frequently reify patterns within the frameworks problem domain in order to provide both the architecture of the framework and to solve problems in the framework domain. Frameworks can be characterised as “white-box” frameworks that allow the user to customise the application using OO inheritance or “black-box” that allow customisation by composition and injection of user supplied objects.

3.2.5 Product-line Architectures

A Product-line architecture is in some ways similar to a framework, as it concentrates on a specific domain, in this case an identified set of software products. Commonalities in the feature sets of the products are identified and reusable software assets of all types (unlike a framework where only code is reused) are designed around the common requirements [van der Linden et al., 2007]. Like frameworks the assets specifically document points of variability where differing implementations might be required. These assets form the product line infrastructure. New applications are then built by configuring and where necessary implementing variation of the infrastructure assets. Product-line architectures have a narrower scope than pattern languages, and in some cases pattern languages are used in the architectural design of asset infrastructures, for example Buschmann et al. [2007a] describes the use of a PL in the design of a product-line for distributed systems implementing the Common Object Request Broker Architecture (CORBA).

3.3 Software Patterns

Alexander et al.’s [1977] original definition for patterns in architecture is:

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, In such a way that you can use this solution a million times over, without ever doing it the same way twice.

The commonly accepted definition for a software pattern is “... a general, reusable solution to a commonly occurring problem within a given context”. Both definitions emphasize the recurring nature of the problem, and the fact that a solution is documented in some way. The solution offered by the pattern should also

be as close to optimal as possible.

While both definitions in the previous paragraph are accurate, they do not tell the whole story. This section will attempt to fill in the gaps by describing several important pattern attributes such as pattern name, context, problem description, influencing forces and the solution. Perhaps just as importantly given the common misconceptions, the final subsection will attempt to specify what a software pattern is not.

3.3.1 What's in a Name

A pattern name is important because it enables the information carried by the pattern to be labelled and used as a means of design communication. Fowler et al. [2002] refer to this process as “chunking”. Choosing a name that is both descriptive but also easy to remember can be difficult as it involves a compromise between long descriptive but hard to remember names and shorter less descriptive names which don't summarise the pattern well, making it difficult for the pattern user to associate the pattern with a problem area. Some examples of pattern names that are easy to remember and summarise their task are **ADAPTER**, **ITERATOR** and **CHAIN OF RESPONSIBILITY**. An example of a cryptic pattern name is a UI pattern named **FACET** which is described as “a pattern for dynamic interfaces” but it would be difficult for designers unfamiliar with UI patterns to remember the connection unless it achieved a critical mass in popularity and became part of the design vocabulary.

Using specific formatting for pattern names helps readers to identify patterns from surrounding text. The later volumes in the POSA series [Buschmann et al., 2007b; Buschmann et al., 2007a], utilise **SMALL CAPS**, which is also the convention followed in this document.

3.3.2 What's the Context

The dictionary definition of context is “the circumstances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood”. The main concepts are an implied observer attempting to understand the idea/event given the circumstances. Mapping these concepts to pattern usage, the observer is the designer, the problem is the idea/event and the circumstances define when the pattern applies. An advantage of briefly specifying the context first is that the designer does not need to spend a lot of time reading an in depth problem description and/or solution to decide whether the pattern applies to the problem at hand.

As an example consider the **PROXY** pattern which Buschmann et al. [1996] describe as “... make the client of a component communicate with a representative rather than to the component itself”. Buschmann et al. identifies seven contexts for **PROXY**:

1. When accessing a sensitive object an access control layer is required;
2. When transparent access to a remote component in a distributed system is required;
3. When caching results from slow or expensive to access objects;
4. When access to shared concurrently accessed objects is required;
5. When reference counting for component instances are required;
6. When lazy or incremental loading of heavyweight or expensive components is required.
7. When network filtering is required (FIREWALL PROXY).

A possible problem with specifying pattern context is incompleteness, as it is possible that the pattern may be in use in other related contexts that the pattern author was unfamiliar with or future problems, for example new technologies, may utilise the pattern for a different context. A pattern can thus be viewed as evolving, with new contexts and other aspects such as solution options being added all the time. For this reason online pattern catalogues² are probably preferable to print books for pattern collections.

3.3.3 What's the Problem

After the context has set the stage, the problem description provides a more extensive overview of the problem. For patterns that have diverse contexts it may be necessary to give separate problem descriptions per context or alternately have the problem description have a variants subsection specialising a general problem description introduced in the earlier paragraphs.

Various criteria that shape or influence the problem are known as forces in the pattern design vernacular. Forces include desired properties, for example a solution should be adaptable and maintainable, as well as constraints, for example a solution should support concurrent asynchronous operation. These forces need to be described in the problem description, thereby allowing the user to recognise which forces apply to the design.

For example the general problem description for the PROXY pattern in Buschmann et al. [1996] reads:

²For example <https://hillside.net/patterns/patterns-catalog>.

It is often inappropriate to access a component directly. We do not want to hard-code its physical location into clients, and direct and unrestricted access to the component may be inefficient or even insecure. Additional control mechanisms are needed. A solution to such a design problem has to balance some or all of the following forces:

- *Accessing the component should be run-time-efficient, cost-effective, and safe for both the client and the component.*
- *Access to the component should be transparent and simple for the client. The client should particularly not have to change its calling behaviour and syntax from that used to call any other direct—access component.*
- *The client should be well aware of possible performance or financial penalties for accessing remote clients. Full transparency can obscure cost differences between services.*

Later contextual refinements are described, for example for item 2 in Subsection 3.3.2 above (transparent access to a remote component) is described as:

Clients of remote components should be shielded from network addresses and inter-process communication protocols.

3.3.4 What's the Solution

According to Alexander [1979] a solution should describe “both a process and a thing; both a description of a thing which is alive, and a description of the process which will generate that thing”. In a building architectural context the thing is the structure, while in software architecture it is a high level design generated by a process of design documented in the pattern solution. The solution must address the described problem as well as show how to resolve the forces catalogued in the problem description. It may also need to address or any compromises that may be required to be made to resolve conflicting forces. The solution may also describe the consequences, advantages and disadvantages of the solution or solution variation as well as list related patterns that may provide an alternative.

The full solution description may be preceded by a short summary thereby allowing the reader to get an overview of the solution. The description itself is normally given in terms of roles with different responsibilities and the relationship between these roles. It is important to note that roles can be implemented as classes in OO languages but there is not necessarily any one-to-one relationship between roles

and classes, and, if the pattern is implemented in an OO manner, then one class can play multiple roles. This is why using UML and other formal diagrams to illustrate patterns can be misleading to naive readers who assume the diagram is the pattern and not just one possible and simplified solution out of a myriad of solutions that can be generated. There are proposed extensions to UML to address pattern documentation. Pattern Instance Notation (PIN) [Smith, 2010] is one of the better such extensions.

An example design can also accompany the solution or be presented in a separate section. Examples are often the best way to materialise the pattern in the pattern users understanding, so choosing the right example is important. It may even be advantageous to include more than one example illustrating different aspects and variations of the pattern. The examples can be high level designs, pseudo code or actual code implementations.

A much summarised extract from the PROXY pattern solution in Buschmann et al. [1996] follows to illustrate the main points:

Let the client communicate with a representative rather than the component itself. This representative, called a proxy, offers the interface of the component but performs additional pre- and post-processing such as access-control checking or making read-only copies of the original.

... The client is responsible for a specific task. To do its job, it invokes the functionality of the original in an indirect way by accessing the proxy.

... the proxy offers the same interface as the original, and ensures correct access to the original. To achieve this the proxy maintains a reference to the original it represents. Usually there is a one-to-one relationship between the proxy and the original, though there are exceptions to this rule. ... The abstract original provides the interface implemented by the proxy and the original.

Later the remote proxy variation is described

... A Remote Proxy encapsulates and maintains the physical location of the original. It also implements the IPC (inter-process communication) routines that perform the actual communication with the original. For every original, one proxy is instantiated per address space in which the services of the original are needed. For complex IPC mechanisms, you can refine the proxy by shifting responsibility for communication with the original to a forwarder component ...

In the above the roles are the client, the original, the proxy and the public interface defined by the original.

3.3.5 Final Assembly

The elements that make up pattern documentation can be assembled in various ways. Several different documentation forms have been developed over the years, a few examples are briefly listed below.

- Alexandrian form originally used by Christopher Alexander. The format is mostly freeform text with no specific sections, but does have a bold “Therefore” preceding the solution. The preceding paragraphs describe the context and the problem, while the solution also discusses the forces. The description is designed to be used in a PL so it starts by listing any preceding patterns and ends with possible successor patterns.
- The Portland form is based on the Alexandrian one, although optimised for use in a hypertext environment. The solution paragraphs are structured in the form of the force being addressed “therefore” the solution with “stronger” (more important) forces dealt with first.
- GOF form as popularised by the “gang of four” book [Gamma et al., 1995]. Comprises sections for *name*, *intent* (brief problem description), *also known as* (other names), *motivation* (the pattern solution in an example), *applicability* (the context), *structure* (an OMT³ diagram), *participants* (the roles), *collaborations* (role relationships), *consequences*, *implementation*, *sample code* (usually for the example introduced in *motivation*), *known uses* and *related patterns*.
- The POSA form used by the “Pattern Oriented Software Architecture” (POSA) series of books such as Buschmann et al. [1996], is closely related to the GOF style. It starts with a highlighted short solution description (analogues to the GOF *intent*), followed by an illustrative *example* (GOF *motivation*). The *context*, *problem description* and *solution* follows with the solution described in terms of *structure* (roles) and *dynamics* (relationships). The *variants* section describes alternative solutions based on context for example the remote (distributed) PROXY previously encountered. GOF style *known uses*, *consequences* and *see also* (*related*) sections conclude the pattern description.
- The Coplien form has highlighted sections comprising *name*, *problem* often in the form of a question, *context*, *forces*, *solution*, *rationale* (extra information) and *resulting context*.

There are numerous other variations on these forms with the exact form dictated by the pattern authors stylistic preferences as well as constraints such as maximum allowed length in a pattern catalogue.

³OMT was a precursor to UML

3.3.6 Pattern Misconceptions

Possibly due to first encountering patterns through “soundbite” definitions, many designers misunderstand patterns to be prescriptive blueprints for design problems. As explained in Section 3.3.4, a pattern does not define a fixed solution, instead the solution is framed in terms of roles and relationships between the roles. A particular solution configuration is just one from a large solution space of possible solutions that depend on how and where the designer defines the roles and relationships within the rest of the design. Additionally, as will be seen in later sections, the pattern more often than not interacts with roles from other patterns in forming the solution.

Another misconception is that patterns are exclusively object-oriented . While it is true that many patterns are OO specific, many others are not. For example the PROXY pattern referenced earlier is used in distributed software systems such as CORBA, which can be used by procedural languages such as C with the proxy interfaces defined using a Interface Definition Language (IDL). The OBSERVER pattern (notify observers when an observed component changes state) is used in all sorts of non OO code such as operating systems and system software (the Linux kernel web site even has several articles on Linux kernel design patterns⁴).

3.4 Pattern Interactions

The preceding section discussed standalone patterns. In a real-world design scenario however, patterns frequently interact with other patterns. This section will describe various types of pattern interactions.

3.4.1 Pattern Aggregation

The *Related Patterns* section of the GOF pattern form in Subsection 3.3.5 frequently describes how the current pattern uses or is used by other patterns. An example is the the COMMAND pattern which uses the COMPOSITE and MEMENTO patterns. To illustrate the COMMAND/COMPOSITE link is explained. Firstly the pattern solutions are described [Gamma et al., 1995]:

COMMAND Encapsulate a request as an object, thereby letting you parameterise clients with different requests, queue or log requests, and support undoable operations.

COMPOSITE Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

The main role in COMMAND defines an interface with an *execute* method which realisations of the *ConcreteCommand* role implement, thus allowing requests to be treated as “black boxes”. The

⁴<https://lwn.net/Articles/336224/>, <https://lwn.net/Articles/336255/>

COMPOSITE pattern is used to implement higher level requests as sequences of normal requests. The example from Gamma et al. [1995] describes a text editor with menu options as *ConcreteCommand* roles and macros combining sequences of menu commands using **COMPOSITE**.

3.4.2 Pattern Complementation

In some cases multiple patterns solve the same or a very closely related problem and the designer has to choose which pattern is best within the context of the current design. For example both the **ITERATOR** pattern (“provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation” [Gamma et al., 1995]), and the **ENUMERATION METHOD** (“Support encapsulated iteration over a collection by placing responsibility for iteration into a method on the collection” [Beck, 1997]) provide a means to step through the items making up a collection. The way in which this is done differs markedly based on two differing philosophies. **ITERATOR** which has its roots in the C++/Java OO design school decouples the collection from the iterator by creating a iterator object solely responsible for iteration. **ENUMERATION METHOD** by contrast originated in the Smalltalk community and uses a functional approach to iteration by having the collection provide an iteration method which the caller supplies with either a **COMMAND** object or a callback function. The collection then invokes this **COMMAND** object or callback function for each element.

One possible reason for selecting one approach over the other might be the implementation language of the design artefact. When designing a project to be implemented in C++, Java or C# the designer might choose the **ITERATOR** approach, while a Smalltalk, Ruby or Haskell developer might prefer **ENUMERATION METHOD**. This is perhaps not as overriding a reason these days, as modern C++ ('11+) and more recent Java versions fully support functional constructs such as lambdas, so the choice may have more to do with the stylistic backgrounds of the designer and the implementers. Another reason might be the fit in the design itself. For example a black box OO framework naturally uses callback methods to allow framework users to implement functionality, so using **ENUMERATION METHOD** to call user supplied code would be a good fit.

3.4.3 Pattern Compounds

Pattern Compounds are documented combinations of patterns used to solve recurring problems. The classic example is Model-View-Controller (MVC) which, like many other OO technologies, originated in the Smalltalk environment. Gamma et al. [1995] define MVC in terms of the following constituent patterns:

OBSERVER	Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
COMPOSITE	Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
STRATEGY	Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.
DECORATOR	Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.
FACTORY METHOD	Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

and describe each patterns use within the compound as:

OBSERVER	Views are observers of the model.
COMPOSITE	Views and subviews form a composite hierarchy.
STRATEGY	Used to parameterise the Controller for handling input.
DECORATOR	Used to optionally embellish a View with specialised features.
FACTORY METHOD	Used to create Controllers.

Buschmann et al. [1996] also describe MVC as a pattern compound but extend the functionality slightly by adding a few more patterns.

The level of atomicity of standalone patterns can also be seen as somewhat arbitrary, as in some cases the internal structure of a standalone patterns appears very similar to parts of another standalone pattern. Smith [2012] introduced the concept of Elemental Design Patterns which defines the lowest level patterns from which most of the standalone patterns can themselves be composed.

3.4.4 Pattern Sequences

Pattern sequences are pattern aggregates or compounds where the patterns occur in an ordered sequence. The sequence can then be denoted by a list of patterns, for example < PATTERN 1, PATTERN 2 >. A minimal example would be the combination of COMMAND and COMPOSITE introduced in Subsection 3.4.1 where the sequence would be <COMMAND,COMPOSITE>.

In the case of the <COMMAND,COMPOSITE> (and generally for Pattern Aggregates) the order of the sequence is self-evident as COMPOSITE is used to extend COMMAND with the ability to group several commands transparently. To illustrate a more complex example a sequence for the design of a simple text editor as described in Gamma et al. [1995] is used. The sequence starts by defining the documents internal representation using COMPOSITE to represent composites of columns and rows containing glyphs (characters, images etc). The STRATEGY pattern is then used to represent different approaches to formatting the document. Next the UI is designed using the DECORATOR pattern to embellish the windows and COMMAND for the menuing system. The pattern sequence for the design is then <COMPOSITE, STRATEGY, DECORATOR, COMMAND> (note this is simplified for brevity

compared to the full example in Gamma et al. [1995]). Although the example in Gamma et al. doesn't do so, it is also possible to add elements of MVC to the UI by adding **OBSERVER** for the views to be notified when the model (the internal document representation) changes and **STRATEGY** to parameterise user input in which case the sequence would be <**COMPOSITE**, **STRATEGY**, **DECORATOR**, **OBSERVER**, **COMMAND**, **STRATEGY** (Controller)>.

Pattern Sequences are not as widely known as compounds, however their main utility is the part they play in describing Pattern Languages. This will be the topic of the next section.

3.5 Pattern Languages

Standalone patterns (Section 3.3) address single problem areas while the various types of pattern interactions introduced in Section 3.4 combine standalone patterns to solve larger problems. Pattern interactions, however, do not address architectural or system level design using patterns. In contrast Alexander's original conception of patterns was built around combining patterns in the design of larger structures such as designing rooms for a building and buildings for a town. In order to provide similar functionality for software requires some adaption from the approach used in building architecture due to increased combinatorial complexity of the components in a large software system. The approach is described in the following subsections.

3.5.1 Target

Pattern languages address larger scale design, however they are not as meant to act as a replacement for a software development lifecycle methods such as Waterfall or Agile. Instead PLs target specific domains such as distributed programming [Buschmann et al., 2007b; Voelter et al., 2004], Web site design [Graham, 2002] or even writing patterns [Meszaros and Doble, 1997]. The PL then provides a designer with guidance for developing different types of systems within the PL domain. These system types can vary considerably in complexity as the PL provides a solution space within the domain from which a multitude of different kinds of solutions can be chosen. The solution space itself is much bigger than the constituent patterns solution space as it represents the combination of solution spaces of all the patterns selected in the given design.

3.5.2 Name

As with patterns (Subsection 3.3.1) the PL name should be memorable and descriptive. Because PLs are used for domain design the PL name should usually provide a reference to the domain in its name.

3.5.3 Description

The PL description should describe its problem domain and how the PL covers the domain in terms of the design features and characteristics that generated designs will support and whether any constraints or limitations on the designs exist.

After describing the general approach of the PL to solving domain problems, the forces influencing the PL are enunciated. The forces for a PL differ from those of a standalone pattern in that different levels of forces exist. At the highest level the global forces are those dictating design architecture. The global forces and high level design in turn lead to clearly defined problem areas which have influencing forces too. Lastly within the problem areas individual patterns are applied where low-level pattern forces apply (the applicable pattern forces for a pattern when used in a PL are a subset of the usual ones). The overall PL description may choose to list the higher level architectural and problem area forces only and leave the PL specific pattern level forces to the individual pattern descriptions.

The description can also contain a summarised overview of the PL. The overview can be textual such as a short summary of the constituent patterns along with a list of predecessor and successor patterns. It may also be graphical depicting the topology of the PL with patterns as nodes and connections between patterns. For non trivial PLs the complexity may be to great to use a single diagram, instead the PL can be broken up into several such diagrams illustrating areas in the PL with relationships between the diagrams documented separately.

Attempting to use design diagrams, whether formal (such as UML) or informal, to illustrate PLs is usually not tenable as in most non-trivial PLs the complexity in the patterns and their interrelationships in the PL combined with the interrelationships of roles in the individual patterns is much too high.

3.5.4 Solution

Broadly speaking human languages consist of a vocabulary of words and a grammar with rules on how words can be combined. A PL can be expressed similarly by using patterns for the vocabulary. The grammar can then be defined by specifying valid pattern sequences (Subsection 3.4.4) using the patterns from the vocabulary. The sequence concept is extended to support branching and iteration. Branching allows selection of the next pattern in the sequence or another pattern sequence based on designer supplied criteria. Similarly iteration allows repeated application of a pattern or sequence.

How the grammar is specified depends on the author. Alexander's original PL listed all valid successor patterns as well as the valid preceding patterns at the start of each pattern's documentation. This

approach has also been followed by several software PLs, however for a large PL it can become difficult to visualise the connections. If this approach is followed then it can be supplemented by having a final section of the pattern documentation going into more detail on the selection of the next pattern in the sequence based on the different forces that apply to help the designer choose the best successor in the context of the given design. In particular, as the PL is providing a solution space some successor patterns may not apply for a particular design.

Another alternative is to use an external notation or design diagrams. Various approaches have been used such as Backus–Naur (BNF) form, railroad diagrams which provide a graphical alternative to BNF or the feature modelling graphical notation used by Henney [2006]. Such notations are particularly useful where the sequences have branches and iterations, especially if a branch can lead to a different sequence. The external forms can also be used in addition to the successor listing method of the preceding paragraph to allow a designer to get an overview using the external notation.

A design has to start somewhere, therefore PLs have the concept of an initial or entry-point pattern which forms the starting point from which pattern sequences emanate. It is also possible for a complex PL to have more than one such initial pattern.

The context for every pattern in the PL is defined by how it is used within the PL, as compared to standalone patterns where the context is generalised to all known uses of that pattern. If space permits then the context description for the pattern in the PL can give a brief generic overview of a specific pattern context before going into greater detail on the context within which it is used within the PL. The narrowed down context description should provide a designer with enough information to decide whether the pattern is appropriate for a given design, particularly when there is a choice of different patterns in the sequence.

The process of designing using a PL is evolutionary in that at each step a pattern from a sequence is added to the design, therefore each step expands the design starting from the entry-point pattern. For PLs that support different architectures within the solution space, the entry-point pattern may be an architectural one which then guides future pattern choices based on the chosen architecture. Regardless of whether the entry-point pattern is architectural, the process of pattern by pattern design means designers iteratively make decisions on the desired design, with the design decisions prioritised by the pattern sequence. At each point the design is logically whole and ready for further expansion, but if a design mistake is detected then the designer can backtrack to the last correct point and restart. This approach resembles that adopted by the Agile methodology, and is in contrast to models such as Waterfall where

the design of the entire system is seen as a stage.

While each pattern is being added, it is important that the designer remember the importance of roles in patterns as described in Subsection 3.3.4. This means that adding a pattern does not necessarily mean all the pattern roles should be created as new OO classes, as in many cases the roles can be integrated into existing classes from previous steps. Buschmann et al. [2007a, §11.5] describes some best practises for integrating pattern roles into existing designs when realising a design using a PL.

The PL documentation should also provide one or more examples of the application of the PL in creating a design in order to clarify the design process. The example can be integrated into the pattern documentation as a running example or presented in a separate section. Several examples at different levels of complexity provide better feature coverage to aid the designers comprehension, for example an introductory example of a simple artefact as well as a complex design such as a OO framework. Larger PLs may also partition examples into different problem areas covered by the PL.

3.5.5 Applications

The simplest application is the direct creation of software artefacts in the PL domain. The PL allows for the creation of a large spectrum of different artefacts of different size and complexity covering parts or the whole of the domain.

PLs can also be used to create meta-artefacts that can, in turn be used to create artefacts. Object-oriented frameworks (Subsection 3.2.4) are one example of such a meta-artefact, and provide a good fit for the application of PL [Braga and Masiero, 2002] as they also target particular application domains. A PL can be used to generate multiple frameworks with differing architecture and characteristics from the solution space provided by the PL. For example a black-box or a white-box framework or a framework with a simplified narrowed domain. PLs can also help document a framework, and in the case of frameworks that were not created using a PL, a simplified PL can be reverse-engineered from the framework in order to document the framework [Kirk, 2005].

Other meta-artefacts that can be generated include Reference Architectures [Jazayeri et al., 2000; Guerra and Nakagawa, 2015] and Product-line Architectures (Subsection 3.2.5).

3.6 Summary

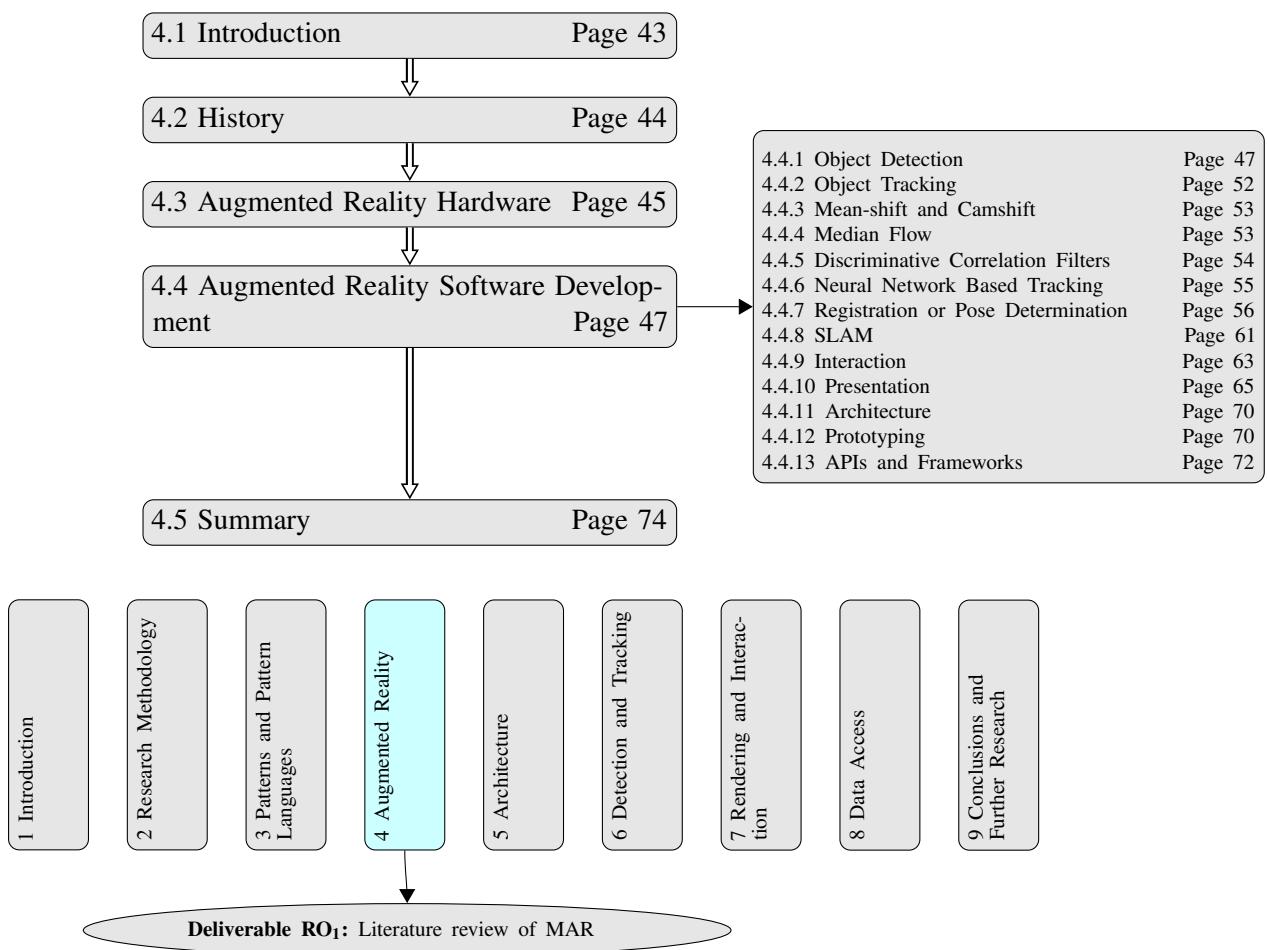
Software reuse becomes increasingly important as software complexity and size increases. While many software reuse technologies of the past concentrated on code level reuse, design patterns allow design best practises to be reused in a non-prescriptive way by documenting the design concepts for individual

problems commonly encountered in software development in terms of roles and relationships between roles.

Pattern Languages extend the idea of design reuse into problem domains. They combine individual patterns into more complex structures capable of solving large scale design problems. The patterns are combined using sequences which support branching and iteration, with the resulting PL providing a very large solution space from which designs of varying complexity can be created.

4

Augmented Reality



4.1 Introduction

In the 2002 Science Fiction movie, *Minority Report*, the hero is seen walking through a shopping arcade with personalised advertisement messages overlaying advertisements in the mall¹. In this scene the reality perceived by the hero is modified, presumably through the contact lenses he is wearing, to provide personalised augmentation of the advertisements. Presently, technology has not yet caught up with some of the more intrusive aspects of reality augmentation depicted in the movie, however the underlying precepts of providing an augmentation of perceived reality through some form of technology are adopted in the field of AR.

Ten years before, in 1992 the movie, *The Lawnmower Man*, explored a related field, Virtual Reality (VR), which creates a completely computer generated artificial world with which a human can interact. AR and VR are related fields with Milgram and Kishino [1994b] defining a continuum between the real-world on one end and virtual reality on the other (See Figure 4.1). In between these two poles various forms of mixed reality are possible with Augmented Reality (AR) being closest to “real” reality.

While VR generates a new synthetic world, AR enhances the existing world by superimposing computer generated objects or information over the real scene in order to provide a composite 3D view. In order to accomplish the superimposition an AR application needs to identify real objects based on the location of the user in order to be able to overlay pertinent information or objects. AR applications are by definition interactive, if only because they need to react to changes to the users location, but most AR applications will also provide other forms of interaction allowing the user to interact with the generated objects or even real-world objects.

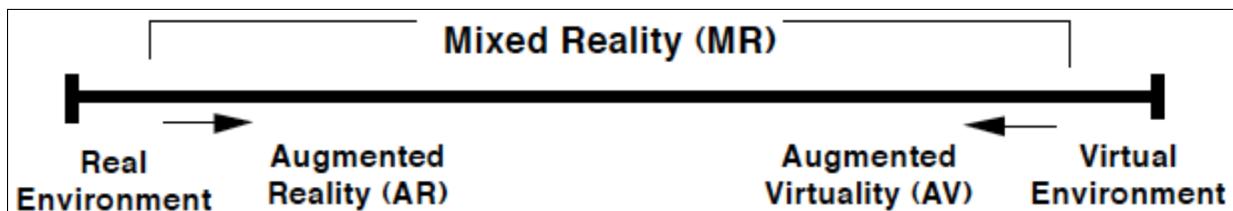


Figure 4.1: Continuum between real and virtual worlds [Milgram and Kishino, 1994b].

¹https://www.youtube.com/watch?v=7bXJ_obaiYQ

Azuma [1997] defines Augmented Reality (AR) as:

An environment that includes both virtual reality and real-world elements. For instance, an AR user might wear translucent goggles; through these, he could see the real-world, as well as computer-generated images projected on top of that world.

Craig [2013] provides a more recent definition which includes the attributes of interactivity and object recognition/registration:

A medium in which digital information is overlaid on the physical world that is in both spatial and temporal registration with the physical world and that is interactive in real time.

Milgram and Kishino's [1994b] taxonomy is technical, Hugues et al. [2011] provide a functional taxonomy of AR. Their taxonomy identifies two main branches of AR namely Augmented Perception for assisted decision making and Artificial Environment which augments with objects from a different time or the physically impossible for example virtual objects not subject to the laws of physics. These two branches are further subdivided into sub-branches in a taxonomical tree.

The rest of this chapter will concentrate on a literature review of Mobile Augmented Reality (MAR) as specified by RO₁, which states:

Conduct a literature review of Mobile Augmented Reality, Patterns and Pattern Languages.

The applicable research process stage from Subsection 2.4.3 is:

Understand the underlying theory.

The output will be a review of research literature covering MAR, which will be utilised in later chapters on patterns and PLs for Detection & Rendering (see Chapter 6) and Rendering & Interaction (see Chapter 7).

The following section will briefly discuss the history of AR which is followed by a taxonomy of AR hardware classified by display position and visual presentation method. The chapter then focusses on the primary objective which is an overview of the architecture, design, algorithms and techniques involved in MAR software development.

4.2 History

The first steps towards computer based Virtual and Augmented Reality were taken in 1966 at Harvard University when Ivan Sutherland and Bob Sproull constructed the first head mounted display (HMD)

capable of generating simple indoor scenes using wireframe graphics [Billinghurst et al., 2015]. The HMD was designed to allow its wearer to vary the view depending on where he was looking. The HMD however, was not really head mounted as it was too heavy and had to be suspended from above. It was not until 1989 that the term Virtual Reality was applied to computer synthesised reality by Jaron Lanier [Sherman and Craig, 2002] who went on to found a company that sold VR goggles, a much lighter version of Sutherland's original HMD.

The term Augmented Reality is thought to have originated at Boeing when Tom Caudell coined the phrase while developing software that could overlay a virtual depiction of cables onto a real scene to aid airplane construction workers [Caudell and Mizell, 1992]. The field was formally launched when the Communications of the ACM published a special issue on computer augmented environments with Feiner et al.'s [1993] paper being the first academic article in the new field. In 1997, Azuma's [1997] survey on Augmented Reality provided the first formal definition of AR. The first steps towards the acceptance of AR outside the academic arena came when Kato and Billinghurst [1999] released an open source library named ARToolkit which is still in use today.

In 1997, Feiner et al. [1997] developed the first mobile AR system which utilised a backpack based computer, GPS and networking over a radio link (this predated the consumer wifi era) to provide a 3D tour guide with information on buildings or artefacts superimposed on the transparent display goggles. Smartphones and PDAs/tablets eventually evolved to the point where they had sufficient capabilities to support AR with Mohring et al. [2004] implementing the first mobile AR application using a mobile device camera for video input. The implementation of mobile AR using the device camera became standard with more recent mobile AR applications also making use of the increasing number of sensors provided by modern mobile devices.

4.3 Augmented Reality Hardware

In the early years, AR was exclusively developed using custom hardware as epitomised by Sutherland's HMD [Peddie, 2017]. In the modern era the use of dedicated custom hardware is still widespread, however the rise of AR on mobile devices has led to increased development of AR running on commodity mobile devices, which may also lead to the eventual commoditisation of AR itself. A niche market for consumer HMDs has also emerged, initially when Jaron Lanier formed a company called VPL that sold VR goggles, followed by Oculus which released the Oculus Rift², a VR headset designed for 3D gaming

²<http://www.oculusvr.com/>

which was released to developers in 2013. The Oculus Rift can also be modified to support AR using an attached camera³. Microsoft's Hololens⁴ is a mixed reality headset supporting both VR and AR which was released in 2016.

Google's Project Tango combined a software AR framework with a hardware specification for Android mobile devices that specified 3D camera support that could return depth information in 3D point clouds from a separate depth camera. Unfortunately, the project was discontinued in 2017 in favour of the monocular software only ARCore project (there is a limited hardware specification for ARCore for providing camera calibration details but 3D cameras are not supported).

AR hardware may be classified by the visual presentation method and the display position. The visual presentation of AR has been implemented using three methods:

- Video feed displays where video input from a video camera or cameras is combined with overlaid virtual content as is the case with mobile AR using a mobile device camera for a video feed;
- Transparent displays where virtual objects are projected onto the display over real-world objects seen through the display. This approach can be used for HMDs, eyeglasses, contact lenses, transparent hand-held devices and car windscreens; and
- Spatial Augmented Reality where the AR content is projected directly onto real-world objects [Bimber and Raskar, 2005].

AR display positions currently include:

- Hand-held such as mobile smartphones and tablets as well as hand-held transparent displays;
- Head mounted as in the original HMD. HMDs can be further classified into those using a video feed and those using a transparent display;
- Eyeglasses such as Google Glass;
- AR contact lenses have been used by the military and consumer versions are under development; and
- Conventional or see-through displays laid out in a predefined environment to surround the user with a augmented video feed of the real scene.

³For example <https://www.stereolabs.com/zed-mini/>

⁴<https://www.microsoft.com/en-us/hololens>

4.4 Augmented Reality Software Development

This section describes the primary problem areas encountered during AR software development. The coordination of these problem areas into a high level architecture of a typical MAR application is then described. Testing and prototyping AR applications are also discussed followed by a brief review of some available and free to use AR frameworks.

In terms of Section 4.3 on AR hardware, MAR uses a hand-held display position and a video feed display method. As this document will be concerned exclusively with MAR the following sections will concentrate on techniques used by MAR, that is techniques for video feed display method and hand-held display position will be emphasised although other techniques may also be described where they contribute to understanding of MAR.

4.4.1 Object Detection

In order to be able to augment real-world objects, the objects need to be detected in the incoming image frame. The augmentation requires not only detection, but also the object position in the video frame, for example, a bounding box. In order to do this some form of database containing data that can be used to recognise the object must be available, with the kind and quantity of the data varying depending on the detection method. For Simultaneous Localisation and Mapping (SLAM) (see Section 4.4.8) AR applications a database of features is built while the user moves.

Currently the primary method of object detection in MAR is CV based detection using features extracted from an image or images of the object. In the last decade, Neural Networks have emerged as an alternative approach, however in a mobile device setting they do have some obstacles to their adoption, as will be discussed later.

4.4.1.1 Vision Based Detection

CV based detection works by recognising features in an image and what exactly the features are, depends on the CV method used. Commonly used methods range from the use of special markers that have to be manually added in the real-world to using features naturally occurring in images of the object. The latter approach can also be further separated into 2D planar matching where co-planar features in a query (video frame) image are matched to corresponding co-planar features in a training image of the object, 2D non-coplanar matching and 3D matching where features in the query image are matched to 3D model or point cloud data in the AR database.

Fiducial Markers

The earliest approaches placed special artificial markers, known as fiducial markers, on objects allowing the objects to be identified by association [Kato and Billinghurst, 1999]. In this case, the database need only contain associations between markers and objects. The process of fiducial marker recognition begins by thresholding the input image to remove extraneous information (the markers themselves are black and white making thresholding easier). Contours are then extracted from the image and matched against the marker. If a marker is identified, then it can be matched against a specific marker by quadrilateral matching.

2D Features (interest points)

Fiducial markers are dependent on the ability to place markers in the real-world before starting the AR application. In contrast, interest point detection detects naturally occurring features in the image and extracts relatively unique descriptors for the feature that can then be compared against features extracted from another image [Szeliski, 2010]. Feature extraction and descriptor creation can be seen as separate processes, although some detection algorithms provide a descriptor as well.

When matching features, two difficulties arise. Firstly one of the images may be rotated relative to the other, therefore it is desirable that the descriptors be rotationally invariant [Szeliski, 2010]. Secondly the features may be at different scales, for example if one image is zoomed in or out versus the other. In this case scale invariance is important [Szeliski, 2010]. As both issues can and often do occur together, descriptors that are both scale and rotation invariant will be the most robust. In reality even scale and rotation invariant descriptors are only invariant up to a point, so a feature database can improve the chance of detection by storing multiple descriptors imaged at different scales and angles.

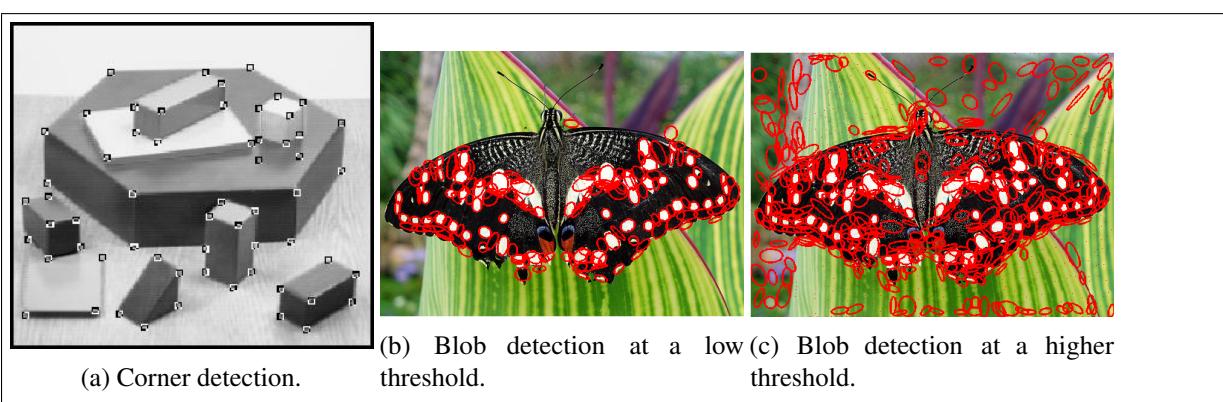


Figure 4.2: Blob and corner detection.

There are two main approaches for detecting interest point, namely corner detection and blob

detection [Kaehler and Bradski, 2016]. Corner detectors detect intersecting edges as natural features (see Figure 4.2a). Blob detectors consider natural features as blobs or regions that have distinctive texture features, particularly texture gradient (see Figures 4.2b and 4.2c).

Descriptors also occur in two main groups. The first type is based on encoding the texture gradient around a detected keypoint as a histogram. This is frequently combined with ascertaining the dominant direction of gradient difference and then adjusting the gradient histogram to match this dominant direction (thereby providing some degree of rotation invariance). The histograms can then be normalised and compared. The second descriptor type is known as Local Binary Patterns (LBP). The premise here is, instead of creating a histogram, a binary string describing the texture is created which can then be compared using Hamming distance. The pattern is created by dividing the region into cells and then comparing pixels in cells to their neighbours and outputting ones or zeroes depending on whether the intensity is higher or lower than the neighbour.

Two of the earliest and best known detectors are SIFT and SURF. The SIFT algorithm [Lowe, 1999] is a blob detector which creates a pyramid of scaled images and then uses a difference of Gaussians between levels to approximate the Laplacian (2nd derivative) of the Gaussian which in turn is used to find extrema representing keypoints. The SIFT descriptor is created by finding the keypoint as the best match in the scale pyramid and then encoding a gradient based descriptor for it. The SURF algorithm [Bay et al., 2008] provides improved performance by approximating a Gaussian second derivative mask to an image at many scales. Both the above algorithms are patented in the US, although they can be used in open source and academic settings.

The FAST detector [Rosten and Drummond, 2006] was one of the earlier corner based detectors⁵. FAST uses the difference between pixel intensities in a circular stencil around a pixel for detecting corners. FAST is, as the name would suggest, computationally very efficient, however it does have some disadvantages in that it is not scale or rotation invariant and does not handle image noise very well. FAST has since been improved resulting in FAST-ER [Rosten et al., 2010] while Mair et al. [2010] optimised the decision tree used in FAST and FAST-ER resulting in the AGAST algorithm. ORB [Rublee et al., 2011] provides a rotation (but not scale) invariant version of FAST combined with the generation of a BRIEF descriptor (later versions of BRIEF as implemented in OpenCV are rotation invariant too). BRISK [Leutenegger et al., 2011] extends AGAST to include scale and rotational invariance and, like ORB, provides descriptor generation.

⁵The Harris/Shi-Tomasi algorithms predate FAST, however they are too slow for real-time work, at least at the scale of a full image, as they depend on deriving a matrix and extracting Eigenvalues.

KAZE [Alcantarilla et al., 2012] uses non-linear blurring instead of Gaussian blurring when creating the scale space by using a diffusion filter simulating the diffusion of image intensity by approximating a partial differential equation borrowed from physics. This process better preserves image detail during scaling while still removing noise. KAZE uses a normalised determinant of the Hessian rather than the Laplacian for keypoint detection and is rotation and scale invariant. AKAZE [Alcantarilla et al., 2013] is an accelerated KAZE which improves the diffusion filter performance to provide better performance and uses Scharr kernels for image differentiation to improve rotational invariance.

2D Interest Point Detection Mechanics

It may be discerned from the sheer number of different detectors and descriptors (far more than are mentioned in the previous paragraphs) that interest point detectors and descriptors have been a very active research area. Despite this, the ill-posed nature of the underlying problem means that bulletproof object detection remains elusive [Tareen and Saleem, 2018]. Descriptor matching is far from 100% accurate and mismatches occur frequently. Another common issue, particularly with self similar symmetrical images is incorrect matches between similar looking features or multiple matches for one keypoint to several keypoints. For this reason, depending on a particular feature being matched is not a good strategy, and object detection should best be viewed as probabilistic over a large number of potential matches some of which may match correctly rather than deterministic with all possible matches matching.

After feature detection the next step in detecting an object is to match image descriptors with stored descriptors of the object from the database. The matching commonly involves K Nearest Neighbours (KNN) algorithms using either an exact brute force approach or a fast approximate search using a spatial index such as FLANN [Muja and Lowe, 2014]. Best matches are found by using the distance ratio of the second nearest neighbour to first with ratios above a given threshold considered good. Next, if the points are coplanar a homography (a linear relation between the train and query points on a plane) is computed using an optimisation technique called Random Sample Consensus (RANSAC) [Fischler and Bolles, 1981] (see also Appendix A.4.1) to eliminate candidate matches that do not conform to the optimal homography, or for non-coplanar points a similarity or affine transform can be computed instead. This process usually results in a much reduced number of more reliable matches.

The use of a homography means that, theoretically at least, non-coplanar points should be eliminated. The matching process can therefore be optimised if at least the training images have only co-planar keypoints stored in the database. While it is theoretically possible to automatically detect planes in images, intelligently selecting co-planar points connected with an object of interest in a training image

is still best done manually. Conversely, where non-coplanar pose calculation techniques are used (see Subsection 4.4.7) co-planar points should be removed as they adversely effect relative pose calculations.

3D Matching

The procedure for using 3D data to recognise objects starts in a similar fashion as for 2D matches, by detecting natural features in the image using any of detector and descriptors previously described. The detected points need no longer represent co-planar points in the real-world as they are matched to 3D points from a 3D model or point cloud in the AR database. The object detection itself is linked to pose determination, discussed in Section 4.4.7 as various approaches use pose space clustering [Olson, 1997] or RANSAC optimisation of the potential poses, with the absence of a solution or a bad solution indicating the object is not found.

4.4.1.2 Neural Network Detection

In recent years Artificial Neural Networks (ANN), particularly Convolutional Neural Networks (CNN), have eclipsed other techniques of object classification in CV. Object classification differs subtly from object detection in that object detection involves finding a specific instance of an object in an image (and where it is located in the image) while classification involves identifying whether an image contains a general class of objects, for example a bicycle (of any type) or a dog (of any breed).

Describing how ANNs and CNNs work is beyond the scope of this text, see Reagen et al. [2017] for a high level overview or Goodfellow et al. [2016] for an in depth treatment. A very brief summary is that ANNs approximate some continuous multivariate function $f(\vec{x}; \vec{\theta})$ where \vec{x} is input data and $\vec{\theta}$ are parameters. The approximation is done using a graph of nodes⁶ with graph nodes arranged into layers. The connections between nodes provide data inputs from the previous layer along with an adjustable weighting factor per input, with the first layer of input being the training data. The graph is usually a Directed Acyclic Graph (DAG)⁷ with each node being connected to every node in the following layer for normal ANNs, but for CNNs some layers are not fully connected while others are, making them more suited to image processing as the image can be processed in chunks with the result combined and then split again. The weights are adjusted by gradient descent optimisation using a process called backpropagation based on a numerical approximation of the multivariable chain rule from calculus known as algorithmic (also automatic) differentiation. The universal approximation theorem [Hornik et al., 1989] applied to this technique states that ANNs, given enough nodes in the network, can approximate any continuous

⁶Known as neurons as the original idea was to simulate the biological brain, although for more recent ANNs the resemblance is only in passing.

⁷A type of ANN called a recurrent ANN can have cycles.

function to any required degree of accuracy, however exactly how to achieve this approximation is not revealed so the “devil is in the details” for ANN practitioners.

The use of ANNs has been extended to object detection, for example Redmon et al. [2016] uses a CNN with 26 layers with excellent results on a desktop computer with a NVidia Titan X Graphics Processing Unit (GPU). Altwairjry et al. [2016] use an alternative approach emulating the “old fashioned” CV approach by training from a set of detected keypoints in order to distinguish good keypoints from bad and then extract descriptors for good keypoints.

Using ANNs and CNNs on mobile devices which are, despite tremendous hardware advances in the last decade, still computationally challenged compared to desktop computers, presents several problems. For example, ANNs are very dependent on GPUs to achieve usable performance, particularly in a real-time setting, and mobile device GPUs are still a long way behind their desktop brethren in terms of number of cores and GPU shared memory. This situation is not likely to change much in the foreseeable future due to:

1. The rate of miniaturisation (Moore’s law) is approaching the asymptote, and the cost of miniaturisation is increasing.
2. The amount of heat dissipated by GPUs is extreme, for example a typical NVidia 1080 based GPU has two fans and a large heat sink and frequently weighs more than the desktop motherboard it is plugged into, which makes it hard to imagine a similar powered GPU in a “mobile” device.
3. The more powerful the GPU is, the more power is required to run the GPU, which, for mobile devices means rapid battery drainage.
4. The amount of GPU and CPU (for transfer to GPU) memory required by the very deep CNNs used for image processing, given that mobile devices trail desktops both in quantity and speed of memory.

For these reasons, the traditional CV based approaches to object detection will remain relevant in MAR for some time, although CNNs may eventually overtake them, particularly if ANN specific hardware replacing GPUs becomes available in the future.

4.4.2 Object Tracking

Once an object has been detected, augmentation in subsequent video frames requires that the objects position be updated as the object, camera or both move. This process is known as tracking. Several approaches to tracking have been developed, the main ones of which will be described in the following subsections.

4.4.2.1 Optical Flow

Optical flow is one of the earliest approaches to tracking and is based on an assumption of colour (or brightness) constancy of pixels representing the same 3D point between video frames. Expressing this assumption mathematically:

$$I(x(t), y(t), t) = I(x + \frac{dx}{dt} \partial t, y + \frac{dy}{dt} \partial t, t + \partial t)$$

and applying the chain rule:

$$\frac{dI}{dt} = \frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

The required velocities correspond to $\frac{dx}{dt}$ and $\frac{dy}{dt}$ in the x and y directions, however it is not possible to solve for two unknowns with only one equation. Making a further assumption that the tracked pixels belong to a single object so they move uniformly, allows a overconstrained system of equations to be derived from multiple pixels/keypoints.

The above has also assumed that the time increments are small enough so that the tracked points belonging to an object do not exhibit large jumps between video frames. This is not always the case so a further optimisation is to create a scaled image pyramid to try and detect the flow at different scales.

Tracking every pixel in this way is known as *dense optical flow* and is generally too computationally expensive to be done on mobile devices. The alternative is to track some form of keypoint, preferably corner based, which is known as *sparse optical flow*. The sparse optical flow method is adapted from a dense optical flow method described by Lucas and Kanade [1981] with the scale pyramid detection supplement by Bouguet [2000].

4.4.3 Mean-shift and Camshift

Mean-shift is a general statistical algorithm for finding the mode (the value that appears most often) of a dataset. In an image processing context it can be used to find the centre of mass of a rectangle of pixels in terms of colour or texture gradient [Comaniciu and Meer, 1999]. Mean-shift can therefore be used for tracking by finding modes within the following frames. Camshift is an image processing specific version of Mean-shift that automatically adjusts the size of the detection rectangle.

4.4.4 Median Flow

The Median Flow tracker [Kalal et al., 2010] introduces an error measure based on tracking pixels or keypoints in both temporal directions and comparing the resultant forward/backward trajectories to obtain the error. Both tracks start in the initial detection frame and for each successive frame the forward track and backward track is appended to. The tracks are used to calculate forward and backward trajectories

used for comparison using some metric, for example Euclidean distance between the initial and end points. The tracker itself applies this error measure to points returned from each frame of a sparse optical flow track.

4.4.5 Discriminative Correlation Filters

Correlation and convolution filters are sliding window kernels that can be applied to an image, for example for a 1D discrete correlation/convolution filter:

$$\text{Correlation: } F \circ I(x) = \sum_{i=-N}^N F(i)I(x+i)$$

$$\text{Convolution: } F * I(x) = \sum_{i=-N}^N F(i)I(x-i)$$

where F is the filter, I is the image and $\text{length}(F) = 2 * N + 1$.

Simple matching by correlation can be accomplished by using the filter as a template and minimising the difference of squares $\sum_{i=-N}^N (F(i) - I(x+i))^2$ or using normalised correlation

$$\frac{\sum_{i=-N}^N (F(i)I(x+i))}{\sqrt{\sum_{i=-N}^N (I(x+i))^2} \sqrt{\sum_{i=-N}^N (F(i))^2}}$$

which usually provides more accurate results. A naive approach would then be to crop the trackable bounding box from the image and use it as a matching filter using either of the techniques above. Such an approach works tolerably if the target scale, rotation and lighting stays exactly the same whilst tracking, but this is unlikely in real life.

The Minimum Output Sum of Squared Error (MOSSE) tracker [Bolme et al., 2010] uses the initial target frame to create a set of training and target images where the target images are translated using a shifted Gaussian representation of the sample center. These images are used for initial training in the frequency domain (via a Fast Fourier Transform). A correlation filter is then created by minimising a sum of squared errors between the filter applied to the image and the target image.

The Kernelised Correlation Filter (KCF) [Henriques et al., 2015] tracker combines cyclic shift permutation matrices into circulant matrices [Edelman, 2017] in order to represent all possible horizontal and vertical shifts of a sample and can then utilise a property of such matrices that they are always diagonal in the frequency domain (after a Fourier transform). The circulant matrix is then used in the frequency domain to simplify Ridge linear regression [Draper and Smith, 2014] for minimising the squared error of the samples, before converting back to the spatial domain. The conversion results in a correlation filter, but doing the calculations in the frequency domain reduced the complexity from $O(n^3)$ to $O(n \log n)$. Henriques et al. [2015] also extend this algorithm to non-linear regression by using the SVM ‘Kernel

Trick” from Machine Learning.

The Discriminative Correlation Filter with Channel and Spatial Reliability (CSR-DCF also known as CSRT in OpenCV) [Lukezic et al., 2017] starts by constructing a spatial reliability map with binary entries indicating which pixels may belong to the target by using a Bayesian probabilistic approach. A correlation filter is then learned by minimising a convex matrix function combining the spatial reliability map with a candidate correlation filter. Separate filters are learned for all channels in the image with a channel learning reliability score calculated per filter. At each update step the spatial map and filter are updated along with foreground and background histograms. The filter channel reliability score is also updated and combined with a per channel reliability score to reflect a combined reliability.

4.4.6 Neural Network Based Tracking

Most ANN approaches to tracking combine detection and tracking. They also take entire video images as input as opposed to regions. As an example of the detect to track approach Feichtenhofer et al. [2017] extend the Region of Interest (ROI) detector of Dai et al. [2016] to correlate results from two CNNs taking successive frames as input (Figure 4.3).

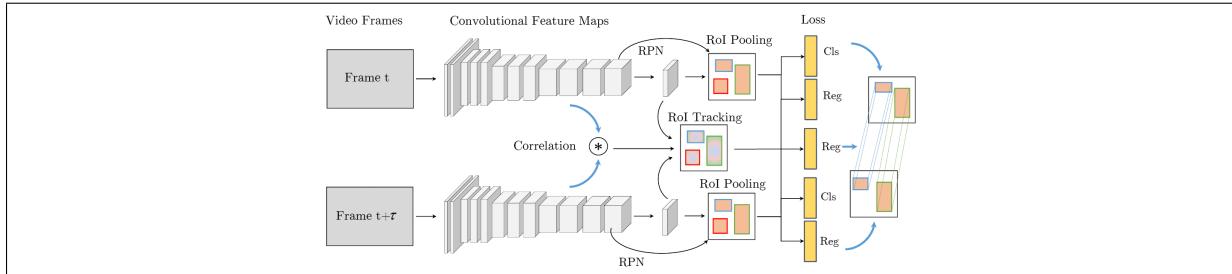


Figure 4.3: CNN for a ROI based detector/tracker [Feichtenhofer et al., 2017].

Bertinetto et al. [2017] provide a CNN based tracker which learns correlation filters (as described in the previous subsection). As CNN are convolutional by name and nature, it is relatively easy to modify one to implement convolutions alter ego, correlation. The NN consists of two learning tasks one to learn a correlation filter and one that does the tracking using the filter. The errors from using the learned filter on the tracking task are backpropagated to the filter learning task. The paper also provides a closed form expression for the derivative of the correlation filter which is used by the backpropagation.

As far as can be ascertained at the time of writing, no tracking CNN based solution has been fully implemented on a mobile device.

4.4.7 Registration or Pose Determination

MAR systems overlay computer generated (CG) visual models and information over the camera display of the real-world. In order to do this, there must be a common coordinate system for the CG rendering and the camera display. The camera output however, depends on the position of the device in the real-world, for example the user may rotate the device. The position and orientation of the camera is known as the pose and the process of determining the pose is also known as registration.

Pose can be determined using hardware sensors or by applying CV techniques to a camera frame or sequence of frames. Unfortunately sensor methods rarely provide the accuracy necessary for seamless virtual/real integration, therefore hybrid methods combining the two approaches are popular.

4.4.7.1 Sensor Based Pose

Mobile devices usually have several motion sensors, although the quantity and quality of the sensors tend to depend on the cost of the device. The sensors are usually based on micro-electromechanical (MEMS) technology [Kempe, 2011], combining the necessary degree of miniaturisation with reasonable levels of accuracy. The following sensors are commonly available on modern mobile devices:

- **Gyroscope:** The gyroscope measures rotational velocity by integrating readings over time. Most devices combine three gyroscopes into one MEMS circuit in order to provide rotation for three axes. Due to the measurement process being additive, gyroscopes suffer from accumulated drift over time. Gyroscopes can only provide relative measurements and therefore they are normally combined with other sensors.
- **Accelerometer:** Measures acceleration, usually on three separate axes. In addition to the raw accelerometer, mobile device operating systems normally provide two software sensors, the *gravity sensor* which uses a Butterworth filter [Cohen Tenoudji, 2016] to isolate the gravity component for each axis and the *linear accelerometer* which returns acceleration without any gravity.
- **Magnetometer:** Attempts to measure surrounding magnetic fields and use this information to provide a bearing to magnetic north. Magnetometers are usually the weakest link amongst mobile device sensors as unrelated local magnetic fields easily cause large errors. Unfortunately the magnetometer is the only motion sensor that provides global orientation data, as opposed to local data (relative to the device) which means that attempts to provide global orientation by combining the magnetometer with other sensors are usually unreliable.
- **Combination Motion Sensors:** Mobile OS API also provide several fused software sensors that

combine input from multiple hardware sensors. For example Android has the *Rotation Vector* sensor that fuses the gyroscope and accelerometer using a Kalman filter to provide the device orientation in axis-angle format, while the *Significant Motion* sensor detects significant translational motion.

- **Position Sensors:** Global Positioning System (GPS) sensors provide global absolute position in terms of latitude and longitude. Most modern mobile devices fuse GPS, WiFi and cellular tower information in order to provide better accuracy, and also to provide a service indoors where a GPS signal may not be available. Coordinates in latitude and longitude may also be converted to global Cartesian coordinates in the Earth-Centred Earth-Fixed (ECEF) coordinate system or to local East, North, Up (ENU) Cartesian coordinates which are usually more convenient in a MAR application setting.
- **Indoor Position Sensors:** Bluetooth beacons and the IEEE 802.11mc WiFi Round Trip Time (RTT) standard provide positions in a local (indoor) coordinate system based on triangulation.

4.4.7.2 Vision Based Pose

CV based methods can be categorised into 2D relative pose and 3D absolute pose methods. Relative pose estimates the relative pose change between two images while 3D pose estimates the pose between a (2D) image and known points on a 3D model or pointcloud.

Both of these methods require calibration knowledge about the camera providing the images. The calibration information is required to relate a 3D position in the real-world as viewed by the camera with its 2D camera projection. Most pose algorithms assume the intrinsic and extrinsic parameters from the pinhole camera model [Hartley and Zisserman, 2004] which is described in Appendix A.1 (and in much greater detail in Hartley and Zisserman [2004]).

The intrinsic parameters have to be obtained in order to do the pose calculations. The mathematics behind this process is described in Hartley and Zisserman [2004] and are implemented by CV libraries such as OpenCV or MATLABs CV toolbox. Modern devices that support OS specific AR APIs such as ARCore (Android) or ARKit (iOS) can usually access these parameters directly via an API call as the hardware standard usually specifies that this information be available. Techniques do also exist to allow the camera to be calibrated automatically, possibly at application startup, by the user [Hemayed, 2003; Tan et al., 2017], although ensuring the quality can be problematic. It should be noted that it is assumed the focal length does not vary, that is any auto focus is disabled by the AR application.

Relative (2D) Pose

Relative pose uses keypoint correspondences from training and query image obtained as described in

Section 4.4.1.1 to calculate the extrinsic parameters (translation and rotation) of the query image relative to the training image. In AR the training image keypoints may be pre-detected keypoints stored in an AR database. Many relative pose algorithms depend on a relationship that exists between two images of the same scene viewed from different positions known as Epipolar Geometry (see Appendix A.3 and in much greater detail in Hartley and Zisserman [2004]).

Relative pose methods predate CV as many of the techniques were developed for Photogrammetry in the early and mid-twentieth century long before computers emerged. The five-point relative pose problem is one such technique that was first addressed by Kruppa [1913] in 1913, who showed that the minimum number of points required to find the relative pose without extra information is five points with a maximum number of possible solutions of eleven. The method of Nistér [2003] is a more recent computationally tractable version intended for use with RANSAC using five points as the minimum sample. The method solves a tenth degree polynomial to obtain the essential matrix \mathbf{E} , which is then decomposed using Singular Value Decomposition (SVD) to obtain up to four possible solutions. The correct solution must then be ascertained by using the cheirality constraint (the points should be in front of the camera). Stewénius et al. [2006] provide a more stable solution based on solving a slightly different set of polynomials using Gröbner bases from Algebraic geometry. Batra et al. [2007] reduce the required number of polynomials to nine quadratic equations which are solved using constrained optimisation to provide a more noise resistant solution.

Absolute (3D) Pose

Three dimensional pose algorithms relate the transformation of objects in terms of rigid transforms (transforms which preserve distance between points). Such transformations can be described in terms of rotation and translation. Translation can be described as a vector \vec{t} with each component specifying the distance a point is moved in the coordinate system axis represented by the component. Rotation can be described by an orthogonal $n \times n$ rotation matrix \mathbf{R} with $\mathbf{R} = \mathbf{R}^T$ and the determinant $|\mathbf{R}| = \pm 1$ (in the case of -1 it is an improper rotation such as a reflection). The combined rigid transform of a 3D point \vec{X} in world coordinates to camera coordinates by a rotation \mathbf{R} and translation \vec{t} is given by:

$$\vec{x} = (\mathbf{R}\vec{X} + \vec{t}) \quad (4.1)$$

The projection of the point \vec{X} to image coordinates is given by:

$$\lambda\vec{u} = \lambda(u_x, u_y, 1) = \mathbf{K}(\mathbf{R}\vec{X} + \vec{t}) \quad (4.2)$$

where \vec{u} is expressed in 3D homogeneous coordinates with the third component being 1, λ is the homogeneous scaling constant and \mathbf{K} is the camera calibration matrix. This can also be expressed in terms of a single 4×4 matrix multiplication $\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 0 & 1 \end{bmatrix} \vec{X}_h$ with \vec{t} forming the first three elements of the fourth column of the matrix and \vec{X}_h being a four component homogeneous representation of \vec{X} .

The pose algorithm uses matches specified by 2D points (pixels) in the image, possibly detected by the natural feature methods from Subsection 4.4.1.1, matched to 3D coordinates (in a unit of length such as metres) on a 3D model or in a pointcloud. The actual correspondences needs to be specified in the AR database.

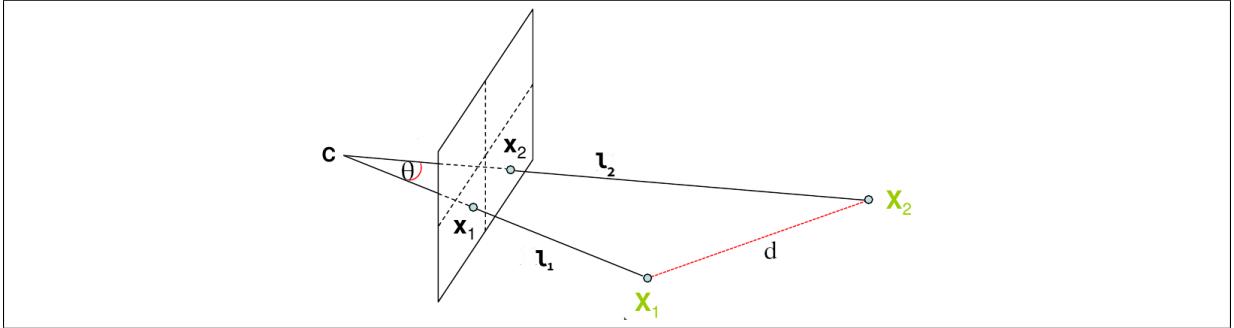


Figure 4.4: The cosine law from trigonometry used in the P3P pose calculation.

Solving the pose from matched points on a 3D model is known as the PnP problem, and as the minimal number of points required for a solution is three the three point solution is known as P3P. Most P3P solutions are based on using the cosine law for triangles (Figure 4.4) where $d = l_1^2 + l_2^2 - 2l_1l_2 \cos \theta$ where d is known and θ can be found using the dot product of the image rays of x_1 and x_2 , that is $(K^{-1}x_1)^T(K^{-1}x_2) = |K^{-1}x_1||K^{-1}x_2| \cos \theta$. The three points then provide three quadratic equations that can be used to solve the rest of the unknowns. Gao et al. [2003] provide a complete analytical solution for the P3P system of equations, classify the solution space by number of possible solutions and provide a numerical algorithm. Kneip et al. [2011] optimise the solution space by using intermediate camera and world frames defined by the image rays. Using these coordinate systems they formulate a single quartic equation, the solutions of which can be backsubstituted into the original coordinate system to provide the poses (in all cases where multiple solutions occur they are disambiguated using a fourth point).

A problem with the P3P solutions is that they are not noise resistant. Combining P3P use with RANSAC does provide a partial panacea, but solutions able to scale up to use available points, can often offer better results. These PnP solutions can be classified as iterative or non-iterative, with iterative

solutions being generally more accurate but much slower.

Iterative solutions use the current pose to reproject the 3D points to 2D image points and attempt to minimize the difference between the reprojected point and the actual point⁸. The most common optimisation method used is Levenberg-Marquardt (see Appendix A.4.2 or Treiber [2013]), which can be applied to over-determined multivariate least squares problems. Apart from the runtime costs associated with iteration, another possible disadvantage is that (non-convex) optimisation cannot guarantee to find a global critical point. Many iterative solutions in libraries such as OpenCV [Kaehler and Bradski, 2016], use another fast P3P or P5P method to find an initial solution and then iterate starting at the found solution. On the whole iterative solutions do give the most accurate results, albeit at a cost of degraded performance.

Several non-iterative solutions have been proposed. EPnP, the method of Lepetit et al. [2009], is $O(n)$ (n is the number of points) and is based on transforming the n points to homogeneous barycentric coordinates expressed in terms of four control points. Solving for the control points results in a 12×12 matrix, the Eigenvectors of which provide a constant number of simultaneous quadratic equations. Penate-Sanchez et al. [2013] extended this method for the uncalibrated case by adding support for finding the focal length.

Zheng et al. [2013] parametrise the rotation using *non-unit* quaternions and use Gröbner bases to generate a unconstrained system of polynomials. UPnP (Ultimate PnP) from Kneip et al. [2014] uses unit quaternions for rotations and Gröbner bases to provide a guaranteed global optimum solution, while also supporting generalised (non-central) cameras.

The preceding algorithms all assumed noise-free point correspondences, possibly depending on RANSAC optimisation to deal with noise. Alternative methods that attempt to deal with noise internally without iteration have emerged recently. For example Ferraz et al. [2014] use the barycentric coordinate system of EPnP reformulated to implement maximum likelihood minimization on correspondences to remove outliers. Urban et al. [2016] also use maximum likelihood based on correspondence covariance matrices for monitoring pose error in order to eliminate outliers.

4.4.7.3 Hybrid Sensor/Vision Pose

The preceding algorithms documented in Subsection 4.4.7.2 used only information available from incoming video frames. As previously noted in Subsection 4.4.7.1, mobile devices have access to other sensory information available from built-in sensors. Unfortunately, the accuracy of the sensor data from some sensors are questionable. In particular the magnetometer is error prone in the presence of magnetic fields

⁸ Alternately the deviation between ray directions from object points can be minimised

generated by nearby objects and the gyroscope is prone to drift and can only be used to obtain readings relative to a start position as its output is the current rotational velocity. The gravity sensor, which is a software sensor that fuses readings from the accelerometer and gyroscope, is more reliable⁹. The earliest relative pose algorithm utilising the gravity vector dating back to the advent of Inertial Measurement Unit (IMU) sensors in mobile devices, is that of Fraundorfer et al. [2010] that requiring five points. Sweeney et al. [2014] provide a more recent approach requiring only three points. The use of gravity in absolute 3D pose also dates to 2010 when Kukelova et al. [2011] formulated a two point solution, with Sweeney et al. [2015] providing a more efficient and noise resistant solution, also only requiring two points. For more detail on these algorithms see Sections 6.4.5 and 6.4.6.

While sensors other than the gravity sensor are not necessarily accurate enough for standalone pose calculation, there are circumstances where they can be fused to provide alternative or complementary pose when vision based pose is lost or compromised. For example, if there are no recognisable features in view then, even though precise pose is probably not necessary as nothing is being augmented, it may still be necessary to have access to an approximate pose for other functions such as displaying the UI. In such cases using fused pose from multiple IMU sensors provides a usable alternative option. Another example where fusion is useful, is when using SLAM vision based methods which may not always be reliable or have data while the area is being mapped, so combining the vision and IMU inputs through a probabilistic filter provides an alternative. This situation is discussed in the next Subsection (4.4.8).

4.4.8 SLAM

Simultaneous Localisation and Mapping (SLAM) is a combination of different techniques used to build a 3D map of the surrounding environment and recover device pose as the agent¹⁰ moves. The 3D map is used to both continually update the pose and to subsequently localise the agent in a world coordinate system when previously mapped features are re-encountered (known as loop closure). Approaches using vision only are referred to as vSLAM, others fuse visual and IMU input while some robotic implementations have the luxury of being able to use expensive laser range finders and other specialised hardware. Early SLAM systems were filter based, using extended Kalman or particle filters, but this approach proved to be too tightly coupled to be parallelisable, resulting in most modern MAR SLAM systems replacing a filter based approach with the optimisation based approach described in the following paragraphs.

The process of building the 3D map of the surroundings based on processing multiple images from

⁹The gravity fusion filter compensates for the gyroscope drift by utilising hardware specific calibration parameters supplied by the sensor manufacturer.

¹⁰User for AR or robot for robotics, as SLAM originated in the field of robotics.

different viewpoints is known as Structure from Motion (SfM) [Mohr et al., 1995; Beardsley et al., 1996]. As the process relies on repeated determinations of camera pose from triangulated points, error accumulation eventually leads to invalid or non-existent mapping. To counter this Bundle Adjustment (BA) [Triggs et al., 2000] is a non-linear optimisation applied to the points and frame poses in order to apply global refinements. The original versions of BA were applied to all frames which proved to be computationally intractable. Klein and Murray [2007] introduced the concept of keyframes based on a similar concept from video compression, which asserts that the difference between subsequent frames in a video sequence are negligible and can be ignored for some given period or until the change becomes measurable. Keyframe SLAM designates a given frame as the keyframe which is used for BA and other 3D map processing, with the keyframe changing based on some time and/or image difference criteria. Most modern SLAM systems maintain and perform BA on a data structure known as the pose graph, representing a probabilistic graphical model of all keyframe poses and 3D points in the lifetime of the application.

The initial state in the SLAM process is initialisation where a global coordinate system and an initial keyframe are chosen. Following initialisation several threads are started to perform the main functions, with the usage of such a concurrent architecture pioneered by Klein and Murray [2007]. In the tracking thread features from the last keyframe are projected into the current frame using a velocity model with the result used to perform optimisation on matches to obtain a pose which is added to the pose graph. Keyframes are switched when a large enough change from the previous keyframe is detected or a given time period has elapsed.

Mapping performs a local BA which occurs every time the keyframe changes. The BA is performed on all preceding keyframes that share features with the current keyframe. Loop Closing compares the keyframe with past keyframes to check for loop closure, and if detected performs a full global BA to optimise the pose graph to reflect the loop (although hardware constrained SLAM systems may not have loop closure in which case they are known as an “infinite corridor” system). The mapping thread described above assumes a monocular camera, systems that have 3D cameras capable of providing pointclouds such as Google Tango specified devices or Kinect can use the pointclouds to create the map instead thereby reducing computational complexity.

Two different approaches to 3D mapping and pose determination in SLAM have evolved as these systems became more sophisticated. Sparse systems use 2D feature matching (as described in Subsection 4.4.1.1), so they use very small parts of the entire image hence the name sparse. While sparse SLAM

is computationally efficient, it can encounter problems with identifying features in bland textureless environments. Dense/semi-dense SLAM is direct pixel based using either the entire image or large parts of the image (for example regions of high gradient). The fully dense approach is computationally unfeasible on a mobile devices, but semi-dense methods have been successfully implemented [Engel et al., 2014]. The semi-dense approach is however, still computationally expensive on devices with modern high resolution cameras where large numbers of pixels must be processed even in the semi-dense case. Dense and semi-dense methods also suffer from accuracy issues related to mobile device cameras using rolling shutters [Yang et al., 2018]. Hybrid approaches using both sparse and semi-dense approaches have also been implemented [Engel et al., 2018].

SLAM systems combining visual and IMU data also follow the processes described above but include input from the IMU. As with visual only SLAM earlier systems used filters to combine IMU and visual data, but many modern systems add the IMU data to the pose graph for BA processing [Forster et al., 2015; Stumberg et al., 2018].

4.4.9 Interaction

Many AR applications do not directly interact with augmented or real objects, but instead just supplement real-world objects with information. This class of AR applications are classified as “Reality with Augmented Perception” in Hugues et al.’s [2011] AR taxonomy, and form an important subset of real-world AR applications with medical, engineering and tourist applications being good examples. There is also a continuum of increasing interaction with many such applications allowing limited interaction with augmented information displays, such as increasing or decreasing the level of detail or displaying visualisations or graphs.

For those AR applications that do support interaction, such interaction implies an action, an actor performing the action and an object on which the action is performed. For AR the actor is the user of the AR application, the object is a virtual or possibly a real object in the AR world and the action itself is dependent on the application. The fact that the application controls the action means that it is possible to perform actions that would not be possible in the real-world, at least with virtual objects. For example, a user swiping a virtual object in a MAR application to make it disappear.

In order to interact with the AR world the position of the user in the world needs to be specified. Poupyrev et al. [1998] classify AR interfaces as being either exocentric meaning the interaction occurs from outside the virtual world or egocentric where the interaction occurs inside the world. Most augmented reality interfaces are egocentric as they place the user in the combined real and virtual world. However, it

is possible to combine the two approaches, for example, to allow searching or navigation with a exocentric mode and normal interaction and navigation in an egocentric one.

Selection and manipulation of objects is one of the most fundamental AR interactions. Traditional egocentric interfaces have used either a pointer metaphor, where selection occurs by casting a ray into the virtual world to select objects that intersect the ray or a virtual hand metaphor, where a user uses a virtual hand to grasp objects. When using the virtual hand, the virtual arm attached to the hand can extend in length as either a linear function of its extension or in the case of the Go-GO technique [Poupyrev et al., 1996] initially linearly and then becoming exponential. The ray casting approach has been adapted by Feiner and Olwal [2003] to allow for curved rays in order to point “around” occluding objects.

Augmented reality deals with both real and virtual objects, so user interfaces that can incorporate real objects into interaction with the user are desirable. Although it was conceived separately from AR, the Tangible User Interface (TUI) [Ishii and Ullmer, 1997] provides such functionality. A TUI in an AR system uses real objects in the augmented world which the AR software tracks and allows the user to use these objects as object manipulators such as handles or containers. Billinghurst et al. [2001] show how TUIs can be combined with AR to form what they call Tangible Augmented Reality, the main characteristics of which are:

1. Each virtual object is registered to a physical object; and
2. The user interacts with virtual objects by manipulating the corresponding physical object.

In order to select objects a means of identifying objects being pointed at needs to exist. Virtual objects are created by the application so they can be identified based on some form of internal knowledge or encoding of the object. Manipulating un-augmented (that is, the real object has not already been detected and augmented) real-world objects are more problematic and dependent on the object detection. Detecting such objects would be dependent on the object detection and tracking components updating their location in a shared data structure.

4.4.9.1 Mobile Device Specific Interaction

Mobile devices are hand held and can detect changes in orientation as well as movement and have a touch sensitive display so they can recognise touch based gestures. In addition, they are hand held which needs to be borne in mind when designing the interaction technique as the user may only have one hand free during the interaction and extra input devices for pointing cannot be used. Gervautz and Schmalstieg [2012] classifies current AR interaction types on hand held devices to be one of:

- Embodied interaction where a combination of the device movement, orientation and touch screen

interactions are used to interact with AR objects; and

- Tangible interaction (TUI) as previously described.

During embodied interaction one of the consequences of one handed interaction is that it is difficult for users to hold the device steady while interacting with the scenery. To counteract this, Lee et al. [2009] describe a technique called *freeze-set-go* which allows the user to freeze the scene, make a selection and then continue (this technique was adapted by Chowdhury et al. [2013] to allow freezing of a single virtual object instead of the entire scene). Mossel et al. [2013a] describe using one-handed 2D touch combined with device orientation to provide 6DOF manipulation and mapping the device orientation onto the selected object to provide rigid-body transformations on the object.

In order to make optimal use of limited screen space on mobile devices, Mulloni et al. [2010] propose using a zoomable interface where the user can zoom out to an exocentric view, select an object and then zoom back to egocentric view by manipulating an axis of the mobile device. Mossel et al. [2013b] describe a selection technique they call *DrillSample* which provides a two-step selection mechanism where if more than one object is selectable where the user is pointing then a zoomed 2D display of all selectable items are presented to the user so the selection can be refined. Such a technique is useful where, due to limited screen space, selectable objects are densely clustered in the display¹¹.

Hurst and Van Wezel [2011] contrast touch based embodied selection and manipulation techniques with a TUI equivalent that uses the user's finger held out in the real-world space in front of the camera and tracked by computer vision code to select and manipulate objects. Their conclusion is that most users prefer, and are more accurate, using the touch based interface, however, some tasks such as translation of the selected object may be easier using the TUI approach. Further uses of hand recognition in AR interfaces are described by Billinghurst et al. [2014].

4.4.10 Presentation

The presentation component is responsible for compositing the real-world feed, 3D virtual objects and 2D informational objects that combine to form the AR world experienced by the user. From an AR perspective, the most important attribute is the correct registration of real objects that are to be augmented as any errors will lead to misplaced or inconsistently placed augmentations. The correct registration is dependent on having reliable pose estimation (See Subsection 4.4.7.2). Pose determination is itself dependent on correct object detection and a correctly calibrated camera.

¹¹Mobile browsers such as Firefox or Chromium use a similar technique in 2D when clicking on hyperlinks that are close together

Apart from the accuracy of the augmentation, depth perception also influences the fidelity of 3D AR portrayal. Factors relating to the reasonably accurate rendition of depth onto a 2D medium are discussed next.

4.4.10.1 Depth Perception

Humans and most other animals having eyes, have evolved stereo vision, that is, the processing input from two eyes in the visual cortex which enables depth perception. Current cognitive science theories suggest visual processing is part of an extremely complex system which builds a 3D model of the world from multiple sources [Jerath et al., 2015]. An example of the complexity of the system is that people who have lost the use of one eye can still see in 3D from which it can be inferred that the visual system combines stereo view with several other depth cues in order to build a 3D view of the world.

Depth cues include amongst others:

- Perspective in which parallel lines converge to a point at infinity and distant objects appear smaller. Computer graphic algorithms for 3D rendering apply this effect through a perspective projection transformation;
- Shading which is manifested by how light is reflected from surfaces;
- Shadows are related to shading in that they are an artefact of light shining onto an object;
- Occlusion occurs when one object is in front of another and appears to overlay it; and
- Motion parallax occurs as a result of nearby objects appearing to travel further and faster than faraway objects when the observer is moving.

These depth cues will be elaborated on in the following paragraphs.

Perspective: Scenes viewed in perspective have objects nearer to the viewer appearing larger than those further away and parallel lines appear to converging at infinity. As the real-world scene is captured in perspective and the virtual objects can be rendered using a perspective projection [Akenine-Moller et al., 2018], a frame from an AR scene is viewed in perspective. As with calculating pose, it is also important for correct perspective that the camera calibration is correct as the perspective transform matrix used for rendering is built using camera calibration parameters.

Occlusion: The default rendering process of video feed hardware involves rendering the geometry of the virtual objects into a framebuffer containing the live video feed which results in these virtual objects overlaying, or to use computer graphics terminology occluding, the real-world objects and scenery regardless of the depth of the virtual objects in the AR scene (although occlusion between virtual objects can be handled by the graphics systems depth buffer).

Resolving occlusion boils down to comparing the depth (z axis) in the AR world of the virtual objects and the real objects, however this is dependent on information about the depth of the real-world objects being available which, for a monocular camera is not by the case. Three dimensional cameras that support returning point clouds¹² can provide depth information, albeit limited by both a maximum depth detection limit¹³ and increasing measurement inaccuracy as point depth increases. As a result generalised occlusion detection is still an open research question. One approach is to create a 3D mesh of the surroundings and using it to build an occlusion map, although doing this in real time on a mobile device is problematic. Such approaches can use single or multiple point clouds or the sparse 3D map built by a SLAM pipeline, however achieving the necessary accuracy and performance is difficult in general settings.

An alternative technique relies on 3D models of real-world objects being available. The real-world objects can then be fully or partially represented by “phantom” 3D meshes without any colour or texture information [Breen et al., 1996]. These “phantom” models are then rendered to their real-world equivalents position, but with the colour buffer disabled so only the depth buffer for the pixel fragments of the phantom object is updated (the framebuffer still contains the colours from the original real-world objects image). When the virtual objects are rendered depth buffer comparisons will then provide occlusions where necessary. DiVerdi and Höllerer [2006] optimised this approach by using a GPU shader to track the silhouette edges of the image compared with the models and the corrected edges are then alpha blended to achieve an occlusion status.

Shading and Lighting Simulating real-world lighting has been a complex component of the 3D graphics field and several lighting models of varying complexity have been designed to attempt to simulate lighting encountered in the real-world. Most of these models view the interaction of light with visible objects as consisting of four separate types:

- Emitted light is shown as originating from a light source but has no further effect on any other objects;
- Ambient light affects all objects in a scene equally, but has no direction and is not perceived as reflecting from objects;
- Directional light which has a definite source as a point emitting light in a 360°sphere; and
- A distant directional light such as the sun with parallel rays.

¹²Or multiple images from a monocular cameras.

¹³Expensive Time-of-Flight cameras [Li, 2014], which are not typically available in mobile devices do have a greater range.

The way the light reflects from objects defines the visual appearance of the objects. In diffuse reflection light rays are scattered equally in all directions and are dependent only on the angle of incidence of the light ray. Specular reflection is dependent on the position of the viewer and the angle of incidence of the light ray and is manifested by highlights on the surface of shiny objects.

More complex lighting models also exist, for example, that of Schlick [1993] that models more advanced concepts from the physics such as energy conservation, Helmholtz reciprocity, Microfacet theory and the Fresnel equation. An alternative approach which is frequently used by movie special effects is ray tracing which traces the path of individual rays to create the image. Ray tracing is typically not seen as a real-time technique, as it is extremely computationally expensive, although NVidia's recent introduction of RTX cards with GPU hardware support for ray tracing may change this. While advanced lighting models contribute to a greater sense of realism, the human visual systems ability to unconsciously compensate means that even the standard lighting models provide an adequate experience [see Thompson et al., 2011, ch. 12] in AR when combined with several AR specific techniques.

Mobile augmented reality displays the real-world as captured by a camera which presents a potential inconsistency when trying to render virtual objects as the light sources positions in the lighting model may differ from reality. Several approaches have been used to address the lighting issue. Agusanto et al. [2003] use an image based approach that captures lighting from the scene with radiance sensing hardware and then uses the captured image as an environmental illumination map¹⁴. Pessoa et al. [2010] present a more advanced and up to date environmental map solution that uses GPU shaders to generate per object environment maps, which results in better visual effects including shadows. Both solutions do require a pre-calculated environment map image to exist. Calian et al. [2013] describe the use of a hardware shading probe which captures shading information (as opposed to radiance which requires potentially unreliable calibration) and stores it in precomputed form for rendering. The probe can be adapted for indoor or outdoor conditions.

Detecting light sources in images is surprisingly difficult and is an open research area. Many approaches depend on some type of apparatus such as a Lambertian sphere in the image. Approximations such as those in Google's ARCore (see Subsection 4.4.13.1) appear to estimate light intensity distribution from pixels in the image to give a rough approximation, which, when combined with the human visual systems self correction ability, gives reasonable results when used in a GPU shader.

An alternative to the image based approach is a non-photorealistic approach that creates stylised

¹⁴A 3D computer graphics method that looks up lighting RGB values from a image or table instead of using a lighting model calculation

representations of both real-world and virtual objects so they become indistinguishable and allows embellishment of the stylised objects with additional information that may be useful to the user. Chen et al. [2012] presents a framework for such a non-photorealistic method.

Shadows Shadows are important visual cues to help perceive depth. Depth perception in an AR world could be improved if virtual objects were to have shadows resembling those of the real world objects. Haller et al. [2003] uses shadow volumes to generate shadows, however in order to generate real-world object shadows, the real-world object must have a corresponding “phantom” model registered for it as discussed in the section on occlusion. Madsen and Laursen [2007] presents a image based solution using both shadow maps and environment maps which also allows varying output quality so different levels of hardware can be supported and uses a GPU to do much of the processing, however this technique is best suited to indoor AR as it depends on the environment map being captured on a polished sphere which is impractical for outdoor scenes. Madsen et al. [2006] estimates outdoor lighting in terms of the position of the sun as the primary light source using diffuse surfaces in the scene.

On modern mobile devices with location sensors it is possible to calculate the position of the sun given the location and time of day¹⁵. This information can be used to implement simple lighting and shadows for virtual objects. Barreira et al. [2013] extends this idea for devices which have an ambient light sensor to detect the scene illuminance.

4.4.10.2 Annotations

Many AR systems such as the previously mentioned augmented browser applications augment real objects with annotations conveying information about the object which need to be rendered. Wither et al. [2009] classify various types of annotations and how and where they should be rendered while Rosten et al. [2005] describe rendering techniques that attempt to ensure annotations are rendered in such a way that the annotations are fully visible but do not occlude important features in the scene.

4.4.10.3 Scene Graphs

Scene graphs abstract 3D rendering by placing renderable objects in a graph or tree. For example, rendering the solar system with the sun, planets orbiting the sun and moons orbiting the planets could be done by having the sun as a root node, attaching the planets to the sun and the moons to the planets and so on. Real scene graphs are more complicated as nodes can have different types such as geometry nodes for renderables, billboard nodes for text and transform nodes to perform a transformation such as translation

¹⁵<http://pveducation.org/pvcdrum/properties-of-sunlight/suns-position>

or rotation to all its child nodes. Making changes to such a data structure, for example moving a moon between planets or attaching the root node to a new root node representing a galaxy, is much easier than direct 3D API programming in APIs such as OpenGL, Vulkan or DirectX would be.

Using scene graphs improve performance by optimising the culling of non visible objects, correct ordering of transparent object rendering and texture reuse. They also enhance programming efficiency by acting as a framework encapsulating lower level API detail and portability between different APIs. For AR systems, adding or removing dynamic AR content is much simpler using a scene graph. Both Google's ARCode and Apple's ARKit also have scenegraph extensions (Sceneform and SceneKit respectively).

4.4.11 Architecture

The architectural attributes of a MAR artefact varies depending on the complexity of the artefact. In general some common requirements are:

- Real-time support for video and sensor data;
- Concurrency and multithreading support for tracking and detection;
- User interaction abstraction; and
- 3D rendering with optional support for scenegraphs.

Optional attributes include distributed object/network support and platform independence. Chapter 5 describes MAR architecture in greater depth.

4.4.12 Prototyping

The nature of MAR applications does not easily lend itself to laboratory prototyping. The rapidly evolving nature of the field also means that most research emphasis is dedicated to implementation specific details such as tracking algorithms and as a result there is very little in the MAR literature on prototyping.

De Sá and Churchill [2012] experimented with MAR prototyping by creating low, medium and high fidelity prototypes for a proposed MAR application. The low fidelity prototype was simply a hollowed out smartphone with a rectangular see through hole covered by a glass screen with interaction tool icons glued to the glass where the screen would usually be. The program was “implemented” by a developer who held up paper cutout icons at appropriate times while prototyping was under way (the so called “Wizard of Oz” prototyping method, although in this case the user was aware of the developers presence). The mixed fidelity implementation was implemented using short videos which were edited to contain icons at appropriate locations. This approach provided a high degree of visual fidelity but less interactivity. The high fidelity prototype was a prototype MAR implementation in which selected features

were implemented while other features were simulated.

The experiment involved asking different users to use each of the prototypes and following the use fill in a questionnaire. The low fidelity approach was the lowest rated with many users not understanding the underlying concept. The mixed fidelity version proved to have the best cost benefit as it out performed the high fidelity version on some questions and was relatively close to the high fidelity version for the rest. The high fidelity version was rated best for providing a realistic experience although users tended to have higher expectations of it as it bore a closer resemblance to a fully working application. De Sá and Churchill [2012] also evaluated the three prototypes in terms of probing or stimulating user creativity, concept validation, feature validation, usability and user experience (see Table 4.1).

	Probing	Concept Validation	Feature Validation	Usability Testing	User Experience
Low-Fidelity	Good for probing as it allows to easily add features.	Not very good to demonstrate the concept or interaction flow.	Difficult to demonstrate features but easy to include new ones in-situ.	Not great to test usability issues, but enough to validate icon size and readability.	Not adequate to evaluate more complex dimensions such as excitement, aesthetics, etc.
Mixed-Fidelity	Not great for probing as it is non-interactive and inflexible.	Very good for concept validation as it shows features and interaction flow.	Good for simple features. Difficult to demonstrate more complex features as it is non-interactive.	Not ideal for usability testing but still allows for the detection of some issues.	Good to assess some aspects of user experience(e.g., aesthetics, flow).
High-Fidelity	Supports probing to some extent but does not allow for in-situ add-ons.	Good for concept validation mainly because it is interactive.	Good for feature validation as it allows users to explore them in detail.	Very good for usability testing as it supports interaction and functionality.	Good for experience evaluation, if care is taken to polish the interactivity.

Table 4.1: Advantages and disadvantages of low, medium and high fidelity MAR prototypes [De Sá and Churchill, 2012].

Building a high fidelity prototype could be simplified and quickened if prototyping tools were available. Mullen [2011] demonstrates using the Processing language to create AR prototypes. Processing is a simplified programming language designed to be easy to learn for non-programmers or part time programmers. To quote <http://processing.org>:

Initially created to serve as a software sketchbook and to teach computer programming fundamentals within a visual context, Processing evolved into a development tool for professionals. Today, there are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning, prototyping, and production.

Processing has been used extensively both for prototyping and creating Information Visualisation

software. Because it supports 3D and AR programming through plugin libraries it is also well suited for creating AR prototypes.

Wither et al. [2011] developed Indirect Augmented Reality, an AR solution which replaces the live video feed background with a previously captured panoramic image¹⁶. The primary reason was to provide an alternative tracking method for outdoor AR which would ameliorate problems related to incorrect registration of real-world objects. However such an approach could also be useful for prototyping AR. Berning et al. [2013] proposes such a solution for AR prototyping using an inexpensive lens attachment for mobile devices to record the panoramic video which can then be augmented by a video editor. A prototyping user then uses a prototype application to interact with the panoramic video of the real-world using the mobile device sensors to track orientation.

Munro et al. [2017] developed a MAR recorder application for Android that allows the recording and off-site playback of video and sensor data. A separate class library for Android was also developed which provides mock implementations [Beck, 2002] of the Android *Camera* class and the more recent Camera2 API *CameraDevice* class. This allows similar camera setup and previewing code to be used for real and emulated cameras in order to minimise the number of changes required to switch from emulation to real code, with the only change required to run the emulation being a method to set the playback files location.

4.4.13 APIs and Frameworks

The recent advent of the “Big-2” of mobile devices into the AR SDK milieu (ARCore for Android and ARKit for Apple) has started the move towards the commodification of MAR. In addition a large number of proprietary and open source third party AR frameworks have been developed over the last few years. A reasonably comprehensive comparison chart can be found at <http://socialcompare.com/en/comparison/augmented-reality-sdks>. This section will give a brief overview of some of the available options.

4.4.13.1 ARCore and ARKit

Both ARCore from Google and ARKit from Apple are SLAM based (Subsection 4.4.8) APIs using filters to fuse vision and sensor inputs. They also allow 2D planar object recognition and plane detection (implemented using the 3D SLAM map). Both allow 3D models to be added either on a previously detected plane or at a developer specified anchor position in 3D space and allow user interaction via

¹⁶The idea of placing a user in the centre of a panoramic scene dates back to Apple’s QuickTime VR format although unlike AR, Quicktime did not support 3D interaction.

touch screen interaction by ray intersection. Unfortunately, unlike Microsoft's HoloLens API, neither API allows direct access to the SLAM map although they do allow access to the raw video frames (unlike some other proprietary AR SDKs).

ARCore programming can be done in Java, C using the Android NDK or C++ using either Unity or Unreal Engine for Android. The Unity and Unreal toolkits have also been ported to iOS so ARCore can also be used on an Apple device. ARCore also has a proprietary scene graph called Sceneform which can be used by Java and C NDK applications (Unity and Unreal have their own scene graphs). Alternately rendering can be done directly using OpenGL.

ARKit is primarily programmed using Apple's Open Source Swift language although it is also possible to use Objective C. As with ARCore, ARKit has a proprietary scene graph called Scenekit while direct rendering can be done using either OpenGL or Apple's new Metal 3D API. Both Unity and Unreal Engine also have third party support for ARKit.

4.4.13.2 Vuforia Engine

Vuforia is a cross platform proprietary AR SDK which allows free downloads for developers and limited free use. Unlike ARCore and ARKit, Vuforia does not use SLAM or use of sensors, instead it provides marker and markerless vision tracking, although for markerless tracking it relies on a cloud based service for AR database storage. Vuforia can be used to create AR applications for Android and iOS either natively or using Unity. It is implemented in C++, however there is a Java wrapper layer for Android. On Android it is therefore possible to create AR applications using Java and the Android SDK or C++ and the Android NDK or a combination of the SDK and NDK. There is currently no corresponding Objective C layer for iOS so Vuforia AR application for iOS need to be in C++¹⁷. The API also supports the equivalent of ARCore/ARKit anchors with the ability to place 3D objects at anchored positions.

4.4.13.3 ARToolkit

ARToolkit is a open source multi-platform AR library that includes marker (with a specialised fiducial marker system) or markerless tracking, camera calibration and stereo camera support. It is programmable in C++ with platform/user defined rendering as it functions as a library and does not do rendering or AR database management. It also has Unity and Unreal plugins and a version of ARToolkit that combines it with OpenSceneGraph named OSGART has also been created.

¹⁷It is possible to use Objective C++ which allows linking C++ code with Objective C.

4.5 Summary

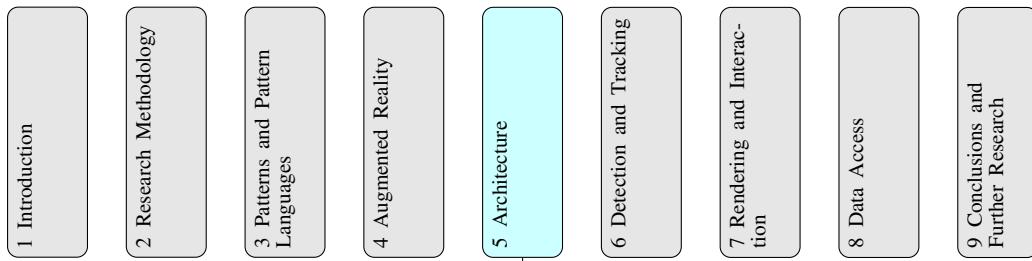
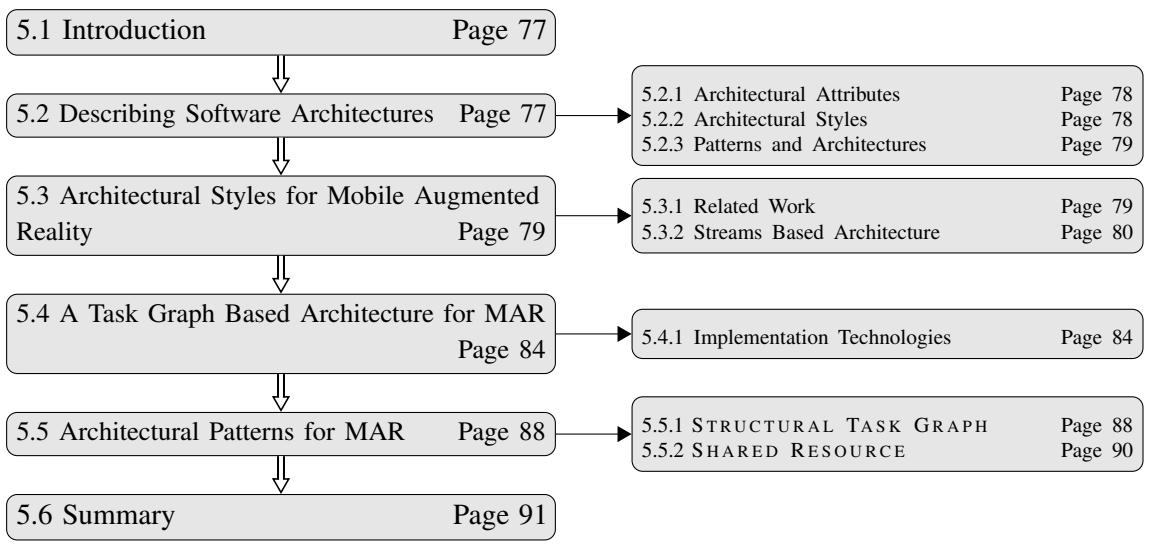
AR as a field in computer science is relatively new, dating back to the 1990's. More recently the availability of cheap mobile and wearable devices has led to the increased commodification of AR for mobile devices or MAR. MAR software development encompasses several different areas:

- Object detection is the process of detecting and identifying objects in the incoming video frames. For mobile devices CV techniques such as corner or blob based feature detectors are most commonly used, although in the future neural network based solutions may become more mainstream depending on hardware advances;
- Once objects have been detected, the tracking component updates the object movements. There are numerous tracking methods including optical flow, mean/camshift, median flow and correlation filter approaches;
- Pose determination uses detected features to calculate the transformation between the world coordinate system and the device and camera coordinate systems. Pose determination methods can either use two sets of 2D image points or matches between 2D image points and 3D model points. It is also possible to combine the use of image based features and mobile device sensors or use sensors exclusively to obtain pose, although the latter approach lacks accuracy compared to image based or combination methods;
- SLAM based AR builds an internal map as the user moves in the 3D world. It utilises feature detection to recognise previously encountered features instead off finding features from an AR database;
- Interaction in MAR can be either egocentric (occurs inside the augmented world) or exocentric (originates outside the augmented world), with most MAR interaction being egocentric. Interaction can involve real or virtual objects and TUI interfaces provide a method for such direct interactions with objects, while mobile device sensors can also be employed to provide more interaction possibilities;
- Three dimensional graphics are employed for AR presentation, therefore techniques such as perspective, shading, shadows and lighting derived from the field of 3D graphics are important. Scene graphs may also be used to improve both rendering and programming efficiency;
- Prototyping MAR applications are difficult but various techniques described in Subsection 4.4.12 may provide solutions; and
- Several MAR APIs are available, some of which are open source or free to use. Both Apple and

Google have recently commenced providing AR APIs with their mobile OSs.

5

Architecture and Architectural Patterns for Mobile Augmented Reality



Deliverable RO₂: Identify software architectures applicable to MAR and document them as architectural patterns.

5.1 Introduction

MAR is a new field and during its evolution over the past decade most design and development effort has been expanded into research and implementation of specific highly technical details such as those described in Chapter 4, with the rest of the application usually added as an afterthought. As MAR becomes a mainstream technology and the size, complexity and longevity of MAR artefacts increase, the importance of having an underlying design to guide the overall structure of such artefacts will increase.

IEEE standard 1471–2000 [Maier et al., 2001] provides a definition for software architecture as “*The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution*”. For MAR the complexity of the overall system is multiplied by the complexity of the individual components such as low level camera and sensor access, object detection and tracking, sensor filtering, 3D rendering using scene graphs and 3D interaction, all of which need to be combined efficiently.

The aforementioned IEEE definition is generic and applicable outside of software architecture, but does not fully reflect the link between design and architecture. Grady Booch, one of the leading figures in OO design in the past two decades supplies an alternative which highlights this linkage: “*Architecture is design but not all design is architecture. Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change*” [see Booch, 2008, page 1]. This definition also highlights the effect of architecture on the longevity of a designed artefact in the sense that the quality of the architecture is reflected in the ease with which the artefact can be maintained or enhanced along with the concomitant costs. The cost of change where architecture is non-existent, the so-called “Big ball of mud”, is described by Foote and Yoder [1999].

The rest of this chapter will concentrate on finding a solution for RO₂, which states:

Identify software architectures applicable to MAR and document them as architectural patterns.

The applicable research process stage from Subsection 2.4.3 is:

Develop a System Architecture.

The output will be an architecture for MAR defined by the identified architectural patterns.

5.2 Describing Software Architectures

Software architects need to be able to identify suitable architectures which are both “good” as well as suited to the design. This section describes several approaches to describing software architectures in order to make this task easier.

5.2.1 Architectural Attributes

In order to extend the definitions of software architecture, it may be instructive to attempt to highlight desirable attributes for a software architecture. These attributes, like patterns, are derived from best practises that have evolved over several decades of designing large software systems.

Modularity entails the decomposition of a system into multiple separate functional units or modules which may in turn contain components. Components or constituents of a module should be *cohesive*, that is the constituents should “belong together” with respect to the underlying architecture. Highly cohesive modules provide *separation of concerns* by grouping related responsibilities. Modules should also only depend on other modules where completely necessary, a property known as low *coupling*.

Modularity isolates the effect of changes thereby reducing the cost of change and also makes testing changes much easier. Modules with well designed interfaces also make it possible to provide “pluggability” or the ability to swap improved or updated module implementations. Modularity also reduces complexity by allowing subsequent design effort to concentrate on modules and interfaces between modules.

Encapsulation packages related modules or components behind interfaces which provide an *abstraction* for the functionality provided by the encapsulation. The abstractions frequently provide a simplified high level facade by combining lower level functions, sometimes by adding new modules which provide added layers of abstraction, although, as pointed out by Hopkins and Jenkins [2008], over-abstraction can also lead to over-simplification.

While a large part of architecture involves reduction to constituent parts, the essence of the system being designed lies in the interactions between the parts which allow the system to be greater than the sum of the parts.

5.2.2 Architectural Styles

Different types of systems entail different architectural approaches. For example, a batch based mainframe system will require a different approach from that of an interactive massively multiplayer online game. Differing philosophies from designers also lead to different architectural approaches, for example a monolithic as compared to a microservices architecture. Different approaches are known as architectural styles and numerous different styles have been classified [Sharma et al., 2015]. Some examples include monolithic, layered, client-server, pipes and filters and blackboard amongst many others.

In many cases, particularly for complex systems however, the design may necessitate a combination of styles, perhaps with a main underlying style combined with other styles used by a module or combina-

tion of modules within the system. These combination styled systems are known as *heterogeneous styled architectures* [Abd-Allah, 1996].

5.2.3 Patterns and Architectures

Patterns provide an ideal approach to documenting architectural styles, and most of the styles from Subsection 5.2.2 are expressed as patterns. These high level patterns are known as architectural patterns to distinguish them from design patterns used to solve lower level design problems.

Architectural patterns can also be used in a PL. For example when used as an initial pattern it specifies how the rest of the design unfolds, particularly in the case where multiple different architectural styles are supported. The use of architectural patterns in PLs documents architectural design decisions, where different decisions lead to different paths through the PL. As these decisions guide the system design process, documenting them provides an important record which may be used by future developers when enhancing or maintaining the system.

5.3 Architectural Styles for Mobile Augmented Reality

The rest of the chapter will focus on architecture for MAR. Finding the architectural style that best fits MAR will be addressed first. Various patterns enabling the architectural style will then be investigated. Finally the integration of these patterns into a PL will be discussed.

5.3.1 Related Work

Earlier work on AR architecture [Reicher, 2004] use the Model View Controller (MVC) architectural pattern to define the architecture. Reicher [2004] describe a generic high level architecture for the application and AR components comprising several sub-systems (Figure 5.1):

- **Application:** Encapsulates a high level application that utilises the AR components combined with application specific features.
- **Interaction:** Handles direct interaction with the augmented world, for example using the touch-screen, but not sensor based interaction such as physical movement;
- **Presentation:** Corresponds to the view part of the MVC pattern and displays the augmented output. It utilises shared data such as the pose calculated by the tracker, accessed via the context sub-system;
- **Tracking:** Handles object detection, tracking and pose using video feed and sensor data. The pose is held as shared data in the context sub-system and is updated by the tracking sub-system;
- **Context:** Stores the current context and provides contextual information to other sub-systems. It

makes use of the **REPOSITORY** pattern to share the pose and other information between subsystems; and

- **World Model:** Provides spatial information about the AR world coordinate system.

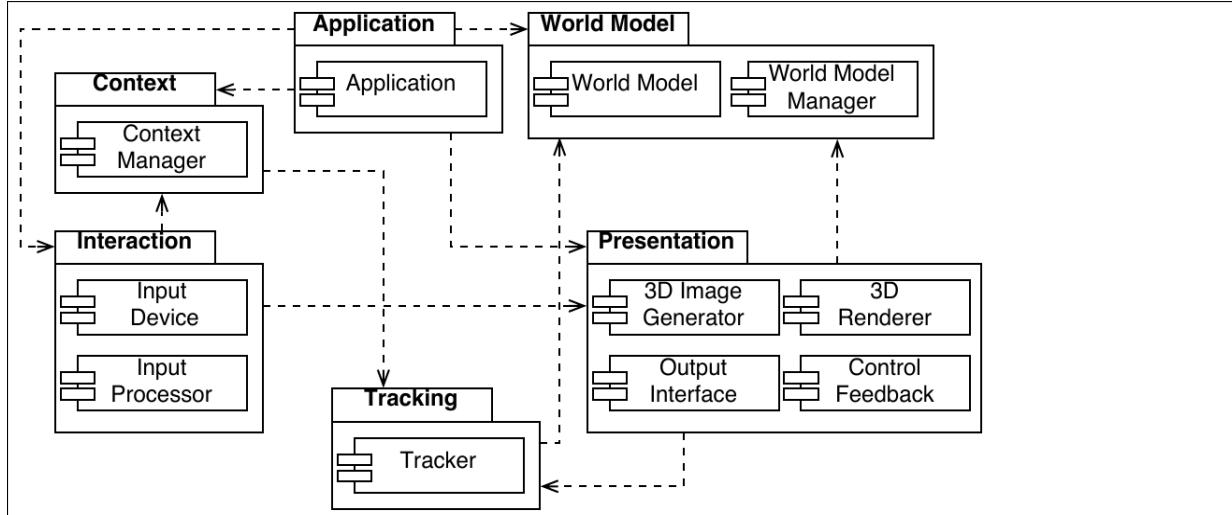


Figure 5.1: Augmented Reality high level architecture [Reicher, 2004; MacWilliams et al., 2004].

5.3.2 Streams Based Architecture

When deciding on a primary architectural style, the requirements and constraints of the design drive the choice. For MAR real-time processing of streams of video and sensor data is both a requirement and a constraint, therefore choosing a style that targets real-time data streaming is appropriate.

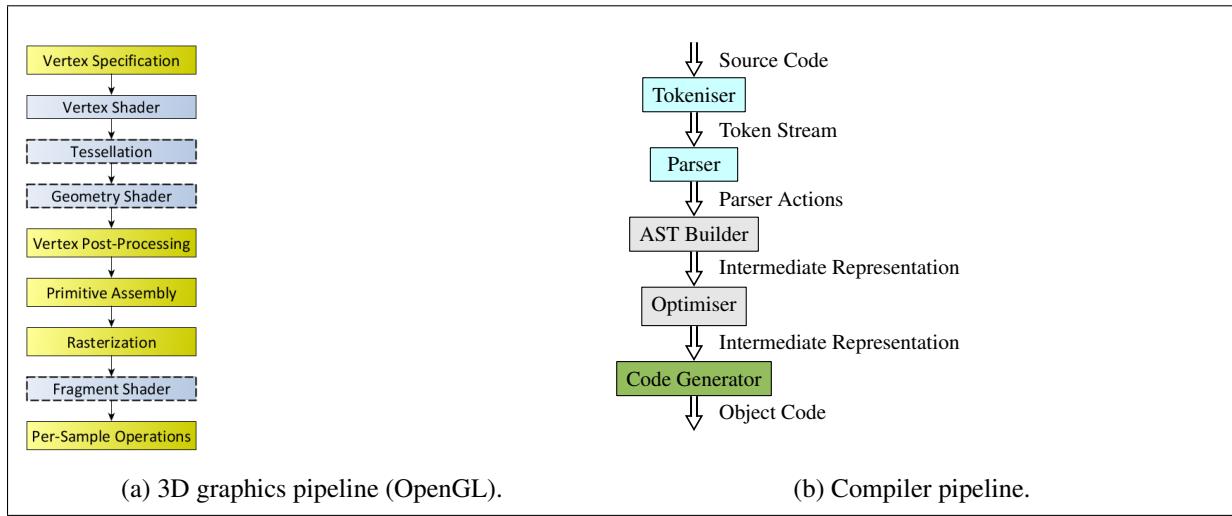


Figure 5.2: Examples of pipeline architectures.

A commonly accepted approach to dealing with data streams is the use of the **PIPES AND FILES** pattern.

TERS architectural pattern [Buschmann et al., 1996] which provides a structure for systems that process streams of data. Each processing step is encapsulated in a *filter* (also known as a stage) which can execute concurrently with other filters when data is available (using a single thread per filter). Data originates from a *source* and is passed through *pipes* between adjacent filters until the final *sink* node is reached. The pipes are usually implemented as First In First Out (FIFO) queues that allow synchronisation by blocking when there is no data available or when the pipe is full. Filters frequently provide incremental output to their output pipe while they are processing so the next filter does not have to wait until the current filter completes work. Pipelines can be composed, for example, a stage in the main pipeline can in turn be another pipeline. Examples of pipeline architectures include the 3D graphics rendering pipeline implemented by 3D APIs such as OpenGL, Vulkan and DirectX which transforms 3D vertices in a world coordinate frame to 2D projections on a computer screen (see Figure 5.2a) and the compiler parsing pipeline which transforms source code text into object code (see Figure 5.2b).

The PARALLEL PIPELINE pattern [McCool et al., 2012] extends PIPES AND FILTERS to support fully parallel filters, that is filters that can process more than one data item from the stream at a time. This significantly complicates the pattern as subsequent filters could potentially receive data items out of order. The pattern allows for three kinds of filters namely serial-in-order which resynchronises incoming data, serial-out-of-order which allows incoming data in any order and the aforementioned fully parallel filters.

The PIPES AND FILTERS/PARALLEL PIPELINE patterns promote modularity as there are no dependencies between filters, each filter is responsible only for processing data from its input pipe. Filters are also interchangeable so a new filter which is able to process the same data can be plugged in, conceivably even at runtime.

Pipelined architectures are ideal for traditional dataflow applications such as 3D graphics or compilers (see Figure 5.2), but they do have some drawbacks when processing real-time or interactive streams:

- As all data is physically copied byte by byte between filters, large data objects such as video frames can impose large overheads;
- More advanced architectures may require multiple pipelines with dependencies between the output of each pipeline and subsequent components or pipelines that use the outputs. In particular there is no direct support in the architecture for synchronising on the output of a pipeline;
- When two stages can act on the same data independently then there is unnecessary latency while the first stage processes. For example if P1, P2 and P3 are stages with P2 and P3 dependent on P1

but not on each other:

$$[P1] \rightarrow [P2] \rightarrow [P3] \rightarrow [P4]$$

then P3 has to wait until either P2 completes, or if P2 supports incremental output then at least until after P2 has started output. If P4 is only dependent on P2 then it too has to wait for P3; and

- There is no feedback support, that is the result of processing one filter cannot affect another filter. An example of where this kind of feedback is important is user interaction, where a user selection may change the the processing requirements for a filter.

5.3.2.1 The SAI Architecture

François [2003] proposed SAI (Software Architecture for Immersipresence) which adapts PIPES AND FILTERS for use in a real-time setting. SAI replaces pipes and filters with cells, sources and pulses. Cells correspond to filters in that they are components that process incoming data and forward the result to the next cell. Data is considered to be either volatile for example, a video frame or persistent, for example user settings. Sources hold persistent data (parameters and variables) for a cell or cells, that is, sources are potentially shared between several cells, but a cell can only have one source.

Pulses represent a time ordering with pulses being generated at regular intervals or on appearance of a recurring volatile data item such as a video frame. The pulse holds this volatile data as well as a timestamp, which is then propagated down the pipeline of cells ordered by timestamp using message passing as opposed to FIFO queues. A pulse can hold multiple data items identified by a unique key and cells can add further data items to the pulse.

Cells receive the pulse and can extract the volatile data from the pulse by key, process it and then update the pulse data or add new data to the pulse before forwarding the pulse to the next cell. Cells can also potentially execute more than one pulse at the same time, for example when a new pulse arrives before the last one is fully processed. The pulse itself is passed as a pointer (in process) or token (out of process) to a pulse structure in shared memory, thereby avoiding repeated copying of data. The pulses described so far are used to transfer volatile data, and are known as passive pulses. Cells also access persistent data in their sources using local pulses known as active pulses to query the source using a known key for a given data item.

Figure 5.3 is an example of a SAI architecture. Squares represent cells while circles represent sources. Passive pulses travel along single line connectors while active pulses allowing cells to access persistent data in sources are represented by double lines. Text with a + in front denote a cell addition to the pulse. There is a feedback loop in the second segmentation cell which splits the pulse in two with the

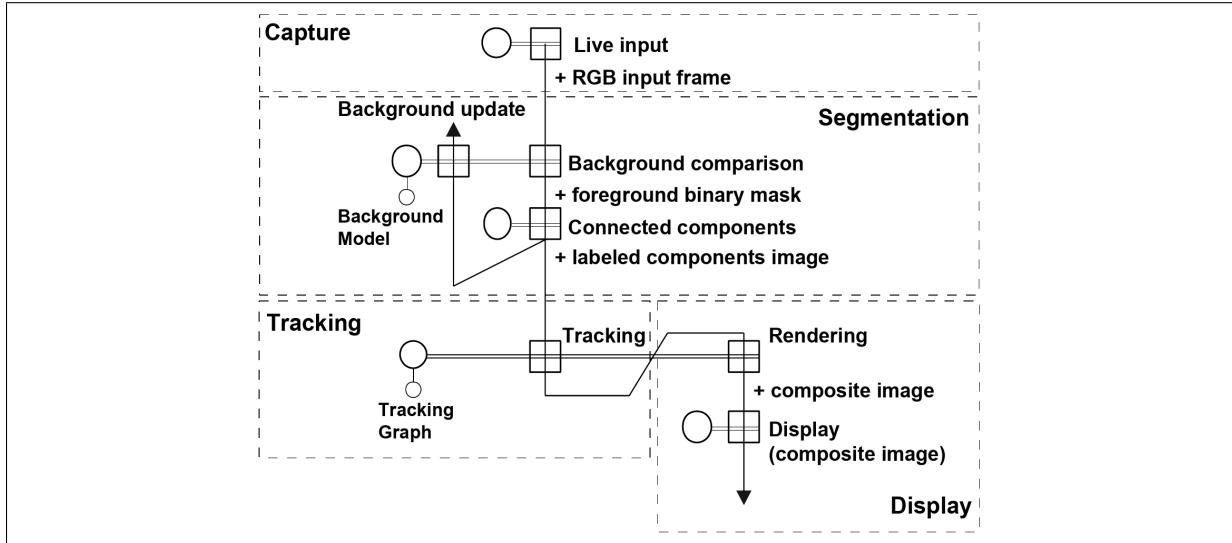


Figure 5.3: Video segmentation and tracking architecture using SAI [François, 2003].

feedback part of the pulse updating a statistical model used during segmentation.

5.3.2.2 Task Graph Based Architecture

Pipeline architectures are best suited to single stream applications, as multiple streams need to be represented as multiple pipelines, which need to be synchronised on each pipelines final sink. In some cases new pipelines need to be created to continue processing the result of the first stage pipelines. Many real-time multimedia style applications, including MAR, fall into the multi-stream category. In these cases, a more natural approach is to structure the design into a Directed Acyclic Graph (DAG) with the nodes of the graph corresponding to filters and the edges to pipes.

The **TASK GRAPH** pattern [Miller, 2010] documents modelling multiple atomic tasks with dependencies between the tasks as a DAG, with nodes representing tasks and edges representing dependencies and predecessor results. The graph can be defined either at compile time or, as a variation, at runtime and is documented as a computational pattern as opposed to an architectural one. Keutzer and Mattson [2011], who use Miller's [2010] **TASK GRAPH** pattern in their PL do mention a yet-to-be defined (as at November 2018) architectural version they refer to as the **ARBITRARY STATIC TASK GRAPH** pattern. As their description of the pattern is missing, a **STRUCTURAL TASK GRAPH** pattern for architectural use in a MAR PL setting will be documented in Section 5.5.

5.4 A Task Graph Based Architecture for MAR

The STATIC TASK GRAPH pattern provides an architectural frame onto which stream processing components can be attached. In a MAR setting however, it may be desirable to allow components to access shared data. Some examples include:

- As alluded to earlier in the chapter, video frames are far to big to be copied between components in the graph. Instead a frame id representing the frame is communicated between the nodes with the video frame held in shared memory;
- CV based algorithms may require access to a previous frame or frames. Storing frames in a size restricted hash table accessed by a frame id would allow components access to previous frames;
- The pose calculation task can update a shared pose accessible to any other components that may require it; and
- The sensor handler task could maintain a queue of recent sensor readings which could be accessed by sensor fusion and pose calculation tasks.

The SHARED RESOURCE pattern [Ortega-Arjona and Roberts, 1998] is a pattern that provides a synchronised shared repository that is concurrently usable by multiple clients. In the examples above, the SHARED RESOURCE could provide an interface to access video frames by id, the most recent pose and sensor readings accessed by timestamp.

A heterogeneous architectural style combining STRUCTURAL TASK GRAPH and SHARED RESOURCE can then be used as a basis architecture for MAR and other multimedia designs that require processing data streams such as video frames and sensor readings. While this combination forms the basis architecture, it does not preclude other MAR modules from having module specific architectural styles, for example, the network data and web access module may use the BROKER pattern when interacting with database and web services as will be seen in Chapter 8. See Figure 5.4 for a simple example of an architecture for an application performing vision based detection and tracking, while Subsection 5.4.1.1 describes a possible implementation.

5.4.1 Implementation Technologies

While patterns describe best practises for solving problems, the solutions need to be implemented using an implementation technology. As a designer needs to take into consideration the high level implementation details at least as far as they apply to chosen architectural patterns a description of some of the more mature or widely used technologies is in order [Van Heesch et al., 2011].

Several of the technologies described in this subsection are concurrency supporting libraries that use the concept of a logical task instead of a thread. Historically, parallel programming was expressed using low level software threads that were mapped onto hardware threads by either the OS or a threading package such as pthreads, using preemptive time slicing. An alternative high level approach is to define potentially parallel units of execution as tasks and leave it to a task scheduler to decide on how to optimally allocate tasks to hardware threads based on the current status of all available processors. The tasks themselves can then be implemented as lightweight non-preemptive threads or fibers, thereby avoiding the expense of preempting hardware threads. However, it is entirely possible to implement a task graph using threads, however it is both more complex and, more often than not, sub-optimal.

5.4.1.1 Threading Building Blocks

Threading Building Blocks (TBB) [Voss et al., 2019] is an open source C++ task based template library for parallel programming using multicore CPUs. TBB has also been ported to mobile OSs such as Android and iOS. TBB directly supports task graphs through its flow graph template class. Several predefined node types are supported including amongst others *function_node*, *source_node*, *buffer_node*, *queue_node*, *sequencer_node*, *join_node* and *limiter_node*¹. New node types can also be created by composition and inheritance of primitive node types. Edges between nodes can be specified with a concurrency limit which designates the the number of tasks the node can run in parallel, although if more than one is allowed then a *sequencer_node* may need to be used as the successor to ensure the output is ordered correctly if ordering is required.

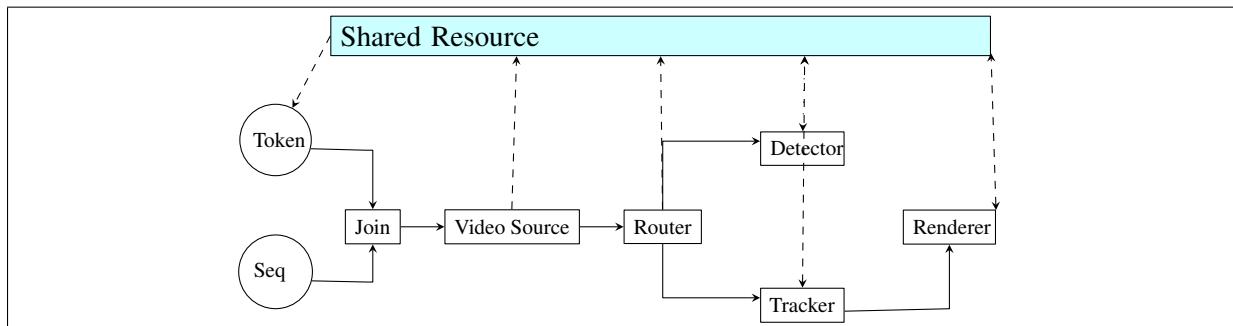


Figure 5.4: A simple task graph based architecture for vision based detection and tracking.

The code to create the simple task graph illustrated in Figure 5.4 using TBB is listed in Figure 5.5.

The code is relatively self documenting consisting of node declarations and defining edges between nodes, although a few clarifications may be necessary. The join node waits for both a sequence number

¹See <https://software.intel.com/en-us/node/517350> for a more complete listing.

```

using dag = tbb::flow;
dag::source_node<size_t> seq_node(graph, source);
TokenHandler tokens(shared_resource);
dag::source_node<size_t> token_node(graph, tokens);
dag::join_node<std::tuple<size_t, size_t>, dag::reserving> join_node(graph);
Camera acquisition(camera_no, shared_resource);
dag::function_node<std::tuple<size_t, size_t>, size_t, dag::queueing> video_node(graph, 1,
                                         acquisition);
dag::multifunction_node<size_t, dag::tuple<size_t, size_t>> router_node(graph, 1,
                                         router);
LKTracker tracker(shared_resource);
dag::function_node<size_t, size_t, dag::queueing> tracker_node(graph, 1, tracker);
SiftDetector detector(shared_resource);
dag::function_node<size_t, size_t, dag::queueing> detector_node(graph, 1, detector);
OpenGLRenderer renderer(shared_resource);
dag::function_node<size_t, size_t, dag::queueing> renderer_node(graph, 2, renderer);

dag::make_edge(seq_node, dag::input_port<0>(join_node));
dag::make_edge(token_node, dag::input_port<1>(join_node));
dag::make_edge(join_node, video_node);
dag::make_edge(video_node, router_node);
dag::make_edge(dag::output_port<0>(router_node), detector_node);
dag::make_edge(dag::output_port<1>(router_node), tracker_node);
dag::make_edge(detector_node, renderer_node);
dag::make_edge(tracker_node, renderer_node);

```

Figure 5.5: TBB C++ source code for the task graph illustrated in Figure 5.4

and a token from the sequence and token sources before continuing to the video node. The token source attempts to atomically decrement the token count using a shared resource method which blocks if the token count is zero, thereby acting as a frame rate limiter. The video node grabs a frame from the camera, updates the frame map in the repository and then passes the token and id pair to the router. The router then passes the id/token pair to the tracker and, if it is not currently busy, the detector. The detector and tracker update detection and tracking results in the repository. The tracker passes the id/token to the renderer, which for this simple example, just draws a bounding box around objects using the the sequence number to get the frame and tracking results from the shared repository. After rendering the renderer increments the token count in the repository. The tracker updates the detector data in the shared resource when an object track is lost, while the detector adds newly detected objects to the tracker data. Detection is an expensive operation and may exceed the frame processing time.

5.4.1.2 FastFlow

FastFlow is “a C++ parallel programming framework advocating high-level, pattern-based parallel programming. It chiefly supports streaming and data parallelism, targeting heterogenous platforms composed of clusters of shared-memory platforms” [Aldinucci et al., 2011]. FastFlow and TBB share many features in common, with TBB possibly supporting more features while FastFlow provide benchmarks with

slightly better performance. In FastFlow task graphs can be created using the a Macro Data Flow model (MDF) or by composing pipelines and task farms.

5.4.1.3 OpenVX

OpenVX is a standard for providing acceleration of CV applications [Khronos, 2015] which originally targeted embedded hardware but has subsequently been ported to general purpose OSs. As OpenVX is a standard, specific vendors supply concrete implementations optimised for their hardware.

OpenVX defines a set of CV functions, for example convert to greyscale or detect corners in an image, that may be used as building blocks in an OpenVX CV application. These CV functions may be implemented using vendor specific hardware optimisations such as GPUs or FPGAs. The functions take specific parameters specified as OpenVX data objects. The functions are then implicitly combined as nodes in a graph with the edges inferred from the data dependencies between the functions.

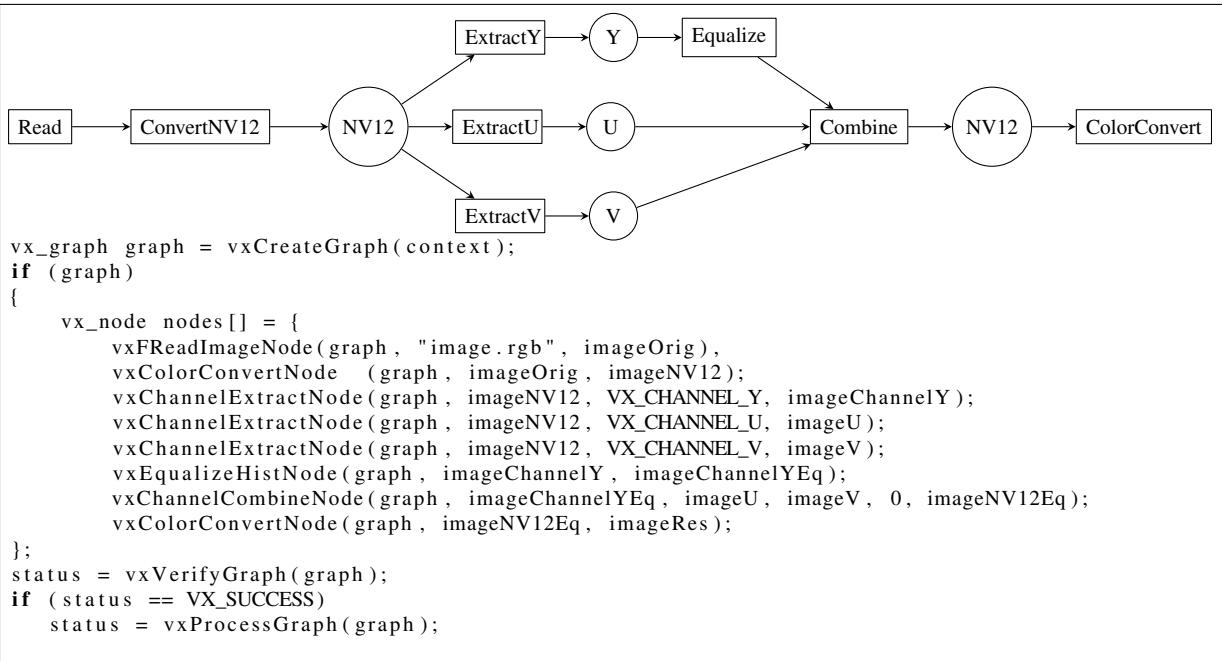


Figure 5.6: An OpenVX graph.

An example of a graph diagram and corresponding code for a small OpenVX fragment is shown in Figure 5.6. In the code *imageNV12*, is the output of *vxColorConvertNode* and the input to three instances of *vxChannelExtractNode* so edges are inferred to exist from *vxColorConvertNode* to the three *vxChannelExtractNodes*.

A disadvantage of OpenVX compared to the general purpose threading libraries discussed in this section is an OpenVX graph cannot be pipelined, that is the *vxProcessGraph* in the code above blocks

until the graph has completed execution before returning. Internally the graph may well execute nodes in parallel, particularly as many implementations are for parallel systems such as GPUs, but the pipelining of data streams is not allowed (it is possible to execute an asynchronous version of *vxProcessGraph* and execute another graph simultaneously, but not the same graph). This means that it would probably be better to use OpenVX graphs as nodes within an architectural task graphs created using a technology that supports pipelining rather than as an architectural task graph itself.

5.5 Architectural Patterns for MAR

This section will document the main MAR architectural patterns that have been identified. As most MAR architectures will probably be hybrid with related architectural patterns being used in sub-systems, the definition of these related architectural patterns will be deferred until the sub-system itself is described. All patterns contexts are relative to their use with the overall PL as opposed to their general purpose use.

5.5.1 The STRUCTURAL TASK GRAPH Pattern

Context: Defining a base architecture for MAR where the design is required to process multiple data streams in real-time.

Problem: The chief requirements for MAR (see Section 4.1) are real-time interactivity and augmentation that is in both spatial and temporal registration with the physical world. The realisation of these requirements are dependent on rapid and efficient processing of streams of data such as video frames and sensor readings. The streams are usually periodic and high frequency, and, as in the case of video frames, may also comprise large data items comprising many bytes. The streams are usually processed by different components which extract information from the streams or transform the data from the stream. Some components depend on the output of other components, either as information extracted from the stream or changes made to the data.

The probable existence of multiple data streams combined with the possibility that the output from some MAR components may be split into inputs for several descendent components and later recombined means a simple pipeline is not practicable.

Solution: Structure the design components responsible for handling and transforming data streams into tasks and place them in a Directed Acyclic Graph where the nodes are tasks and the edges are data stream dependencies along which the data stream can flow. The component lifetimes are assumed to match that of the artefact being designed, so the graph exists for the lifetime of the artefact and defines its architecture.

As mentioned in the problem description, it is possible that the data packets can be large, for example

video frames. In such a case it is undesirable and inefficient to repeatedly copy the data between nodes. A better solution is to combine this pattern with the SHARED RESOURCE pattern (See Subsection 5.5.2) and pass a key or token representing the data between the nodes while gaining access to the actual data through a shared resource object or method.

The implementing technology should support task parallelism, and the components should be designed so as to be able to support processing the next item in the stream as soon as the current one is complete so as to be able to maximise throughput. Prefer implementing technologies that use task based scheduling (see Subsection 5.4.1) compared to those that require low level threading, but if low level threading is used then care must be taken in assigning logical threads in components to physical hardware threads.

It is also possible to combine other forms of parallelism into the graph. For example, a task node may utilise a scan or map-reduce pattern or execute an OpenVX graph within the component. To fully utilise available parallelism the component could use a GPU instead of a CPU core, that is making use of heterogeneous computing.

Nodes and edges are the structural elements with nodes representing tasks and edges dependencies with data flow. While the functional task is the primary node in the DAG, various other node types exist in various implementations, or can be implemented using lower level primitives. These nodes provide a means to synchronise and buffer graph flow. Some examples include:

- Data generation and processing nodes such as source nodes (for example a node that obtains a camera image), functional task nodes, multifunctional task nodes that can output results to multiple descendent nodes selectively and broadcast nodes that copy their input to multiple descendants;
- Buffering nodes including standard buffer, ring buffer, queue, priority queue and sequencer nodes which resynchronise output based on an incoming sequence number; and
- Synchronisation nodes such as a join node that receives input from multiple nodes but only transmits to its successor when all input nodes have data available, and the limiter node that stops outputting data when a given count is reached until the counter is reset.

The critical path through the DAG which is determined by the time taken by tasks in paths in the graph determines the maximal throughput. It is possible to work around the limitations of slower components by allowing them to “skip turns”, that is not process for every periodic iteration such as video frames. For example, if it is known that a SIFT detector is slower than other components, then the detector node can be set to have a parallelism of two and have one thread doing the detection while the second one

simply passes the received token on to the next task until the main thread completes its current detection. The obvious disadvantage is that the detection may be several frames behind the other components, but this is frequently acceptable given that inter-frame changes are normally quite minimal.

5.5.2 SHARED RESOURCE

Context:

- When defining a **STRUCTURAL TASK GRAPH** the size of the data that will have to be passed between task nodes is too large to be efficiently copied.
- When results computed from data streams need to be shared amongst all nodes.

Problem:

- The default means of communicating data between nodes in a **STRUCTURAL TASK GRAPH** is by message passing between nodes, however multimedia data such as images tend to consume large amounts of memory space and are slow and inefficient to copy when implementing task graph message passing.
- When the results computed using the streams by one task should be available to other nodes in the graph, regardless of their position in the graph relative to the node that computed the result.

Solution: Use the shared resource pattern [Ortega-Arjona, 2003] to provide shared data access to components acting as nodes in a **STRUCTURAL TASK GRAPH**. Data consistency is enforced by the shared resource component, while ensuring correct sequencing is left either to the nodes or the design (for example if a task updates a frame and there is a possibility of out-of-sequence access causing inconsistencies then the designer could specify the updated frame should be stored separately from the original).

When using **SHARED RESOURCE** in conjunction with **STRUCTURAL TASK GRAPH**, nodes from the task graph play the role of the sharer components while the shared resource object or API plays the role of the SharedResource in the pattern as described by Ortega-Arjona [2003]. In order to adhere to the principle of programming to an interface, the shared resource component should export a standardised **EXTERNAL INTERFACE** [see Buschmann et al., 2007b, p. 281] which would support pluggability and maintainability and minimise dependencies.

The locking that may be necessary to ensure data consistency will impact performance and could even be the source of bugs. In some cases lock-free algorithms for some data structures such as queues may lead to improved performance although performance gains for lock-free structures are not guaranteed and

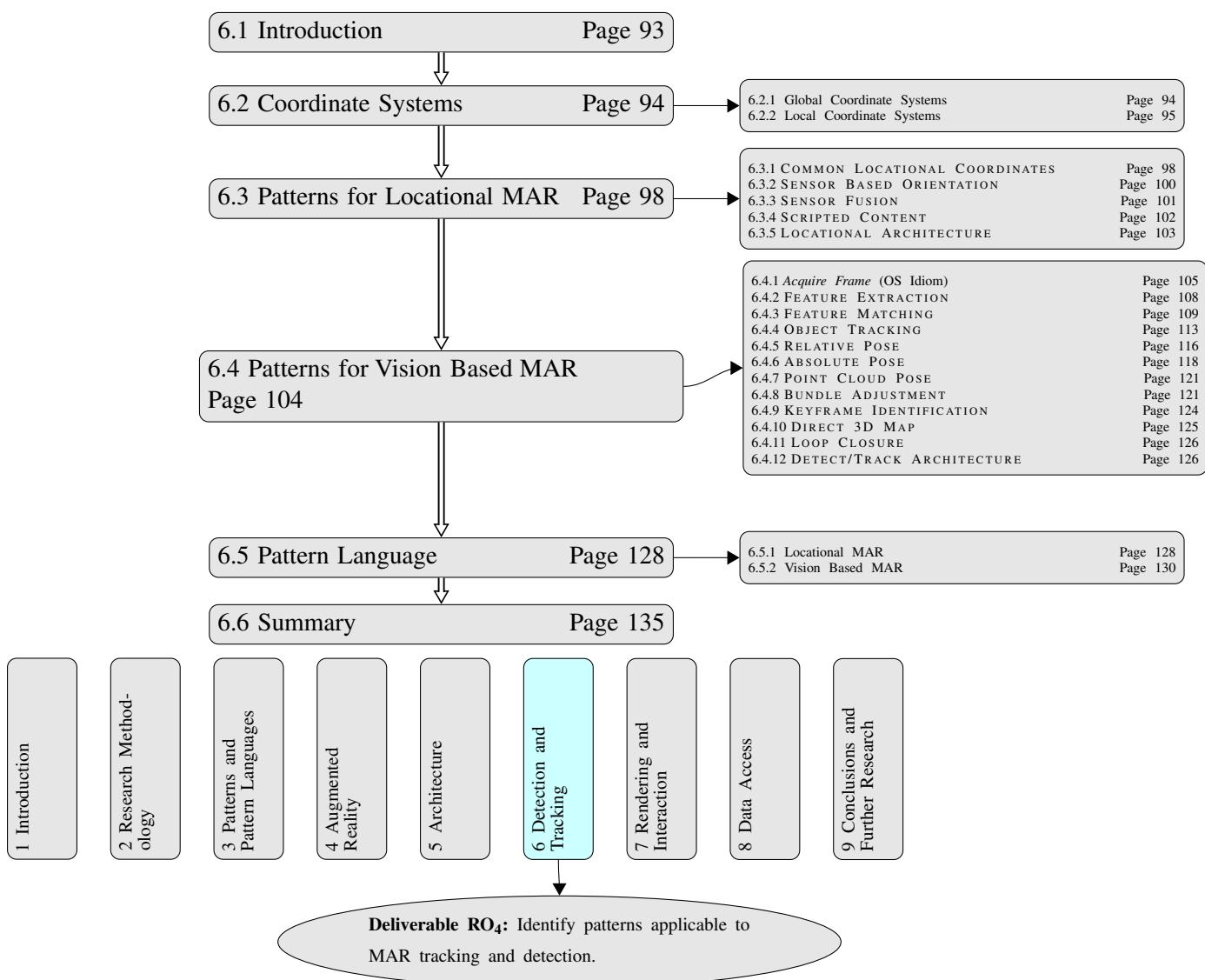
require real-life benchmarking.

5.6 Summary

The chapter commences by highlighting the importance of software architecture for complex systems. Architectures were described in terms of attributes such as modularity and encapsulation. The concept of architectural styles such as *monolithic, layered, client-server or pipes and filters* was introduced, along with the use of a specialised kind of pattern known as an architectural pattern which can be used to document these styles. The rest of the chapter concentrated on architecture for MAR, in particular the primary architectural patterns embodied by **STRUCTURAL TASK GRAPH** and **SHARED RESOURCE**.

6

Detection and Tracking



6.1 Introduction

Detection and tracking is the engine room of an AR system as, in order to perform any augmentation, the system needs to be able to locate what is to be augmented in the currently perceived view or location. For vision-based systems, CV techniques can be used to detect objects that should be augmented, while location based AR can use the location and orientation sensor to decide what augmentation to display based on the current location and the direction of view obtained from the orientation sensors. The majority of this chapter will be concerned with vision-based systems as these systems have far greater complexity, however patterns aiding localisation and orientation for locational MAR (or combined approaches) will also be described.

Computer Vision systems themselves can also be classified into SLAM-based systems (see Subsection 4.4.8), where previously unknown but re-recognisable features are detected and added to a collection that helps to build a map of the surroundings as the user moves, and feature based AR where objects with pre-stored features are recognised for augmentation. In this context features are recognisable collections of pixels, normally corners or textured blobs, while objects are higher level real-world entities comprising many individual features. Some SLAM systems also support a hybrid mode where they can also recognise stored objects, and can, in addition, also detect other augmentable non-stored objects occurring randomly in the scene such as planes.

CV based detection and tracking MAR subsystems perform the following functions:

- Initial detection of predefined objects or, in the case of a SLAM system, re-detection of previously detected features and detection of new features with properties that make them uniquely re-recognisable later. Non-SLAM systems perform object detection in that they recognise higher level objects comprised of multiple features by comparing the detected features against stored features;
- Tracking previously detected objects or features. Objects are tracked in order to correctly augment them as the camera or the object moves. Features are tracked by SLAM systems in order to use feature positions for triangulation between multiple frames; and
- Pose calculation (also known as registration) using detected features, sometimes combined with mobile device orientation sensors using filters such as a Butterworth [Cohen Tenoudji, 2016] or Kalman [Chui and Chen, 2017] filter.

In addition the cameras internal parameters (for example the focal length) are necessary for correct detection and tracking, therefore camera calibration could also be seen as part of the detection and tracking

subsystem. This could be built into the system workflow by providing a calibration mode which is enabled only if the calibration details are not available in order to allow an end user to calibrate the camera before starting the application proper¹. With the advent of vendor supplied MAR APIs such as ARCore and ARKit, it has also become common for hardware suppliers to specify their camera parameters in the device firmware which can then be accessed using the API, regardless of whether the API is utilised in the rest of the application.

The rest of this chapter will concentrate on finding a solution for RO₃, which states:

Identify patterns and a pattern language applicable to MAR detection and tracking.

The applicable research process stage from Subsection 2.4.3 is:

Analyse & Design the System.

The output will be:

1. A set of patterns applicable to MAR detection and tracking; and
2. A pattern language built on these patterns.

6.2 Coordinate Systems

In order to perform detection and tracking and utilise the results, common coordinates systems need to be defined. These coordinate systems may be classified as global or local, with global coordinates being defined in terms of common geographic coordinates such as latitude/longitude, while local coordinates exist in device space, for example a room or building. Global coordinates are determined by location sensors such as the GPS sensor, while local coordinates can be determined from sensors or defined arbitrarily by the vision based software component. Both types of coordinate systems can be used by a MAR application, for example using global coordinates to obtain information on location-keyed artefacts (as described in Chapter 8), while simultaneously using local coordinates for vision based tasks such as detection or registration.

6.2.1 Global Coordinate Systems

The most common global coordinate system is the geodetic (WGS-84) which uses latitude, longitude and altitude to identify any point on earth [Van Sickle, 2017]. All of the subsequently described geographical systems are calculated from WGS coordinates using known formulas, as GPS sensors by default only return WGS coordinates [Kaplan and Hegarty, 2017].

In many cases it can be more convenient to use a Cartesian coordinate system when performing

¹Methods for online calibration during detection and tracking also exist [Pflugfelder and Bischof, 2005; Zhang et al., 2016]

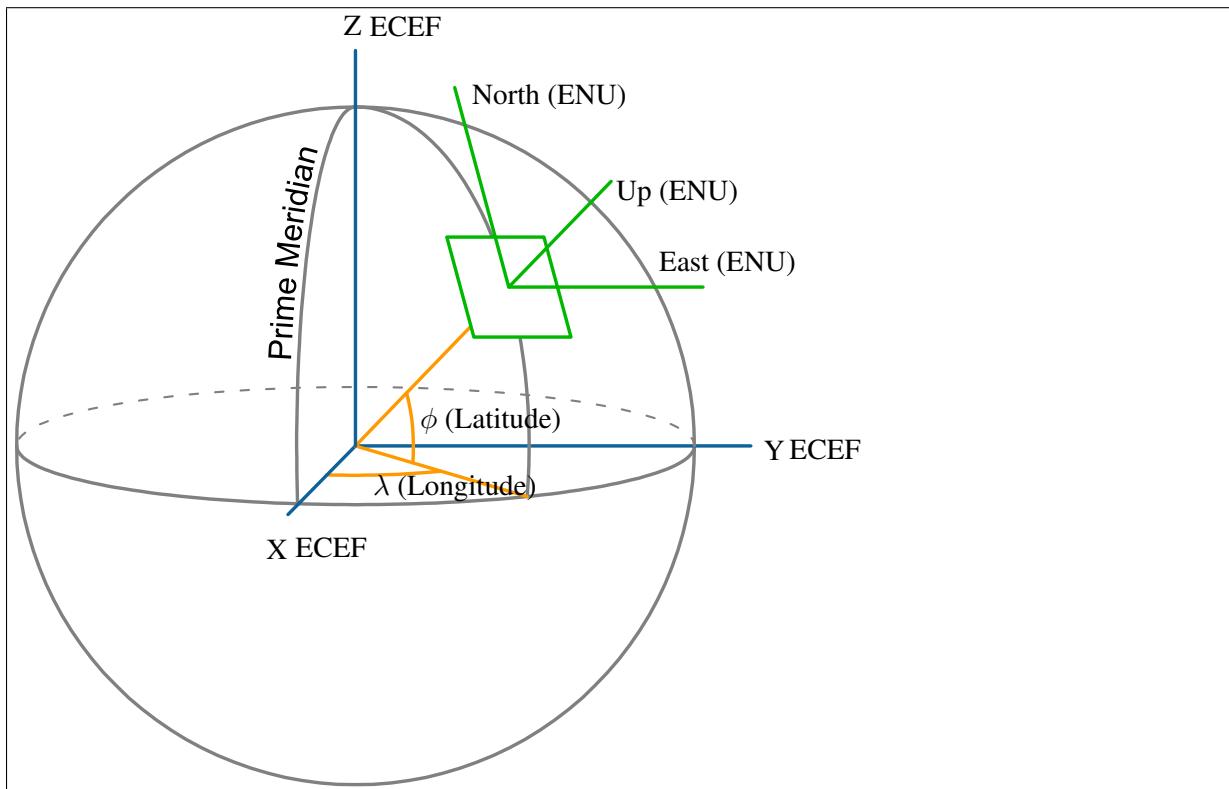


Figure 6.1: Geographic Coordinate Systems (geodesic, ECEF, ENU) (Source: https://commons.wikimedia.org/wiki/File:ECEF_ENU_Longitude_Latitude_relationships.svg)

calculations, particularly in cases where distances between entities are small which is usually the case for MAR. The Earth-Centred, Earth-Fixed (ECEF) system (Figure 6.1) provides a right handed Cartesian system with the origin at the centre of the Earth, the Z axis positive towards true north, the X axis positive towards the intersection of 0 latitude and 0 longitude and the Y axis perpendicular to the X and Z axes [Kaplan and Hegarty, 2017].

6.2.2 Local Coordinate Systems

Local coordinate systems for MAR can be sensor based or vision based. Sensor based systems in turn may be geographical, that is ultimately derived from GPS coordinates transformed to a local system, or some other positional sensor such as Bluetooth beacons or WiFi, which, like the vision coordinates, tend to be arbitrarily defined by the application.

Sensor Based Coordinate Systems Geographic local coordinate systems are defined by the tangent plane to the Earth at the local point defined as the origin. The *East, North, Up* (ENU) system has X pointing east, Y pointing north and Z pointing up (Figure 6.1), while the *North, East, Down* (NED) has X pointing north, Y pointing east and Z down [Kaplan and Hegarty, 2017]. Conversion from geodetic

to ENU/NED depends on first converting the geodetic latitude and longitude to ECEF and thence to ENU/NED.

Non-geographic indoor coordinate systems are obtainable using several different approaches [Davidson and Piché, 2017], of which WiFi signal localisation such as the IEEE 802.11mc WiFi Round Trip Time (RTT) [Statler, 2016] standard or Bluetooth beacons [Statler, 2016] are commonly used. The specification of these coordinate systems depends on the location of the hardware supplying the localisation signals, with room, floor or building level coordinate systems being typical examples [Wu et al., 2018].

In addition to the above-mentioned locational coordinate systems, other sensors such as orientation sensors have their own local device coordinate system. Fortunately, the leading mobile device OSs use the same convention of having the Z axis positive outwards from the device screen towards the user, the Y axis positive towards the top of the device and the X axis positive towards the right. Top and right is defined with respect to what is considered the “natural” orientation of the device, for example most smartphones natural orientation would be portrait, while tablets may (or not) default to landscape (the orientation can be queried from the OS API).

Vision Based Coordinate Systems Local coordinate systems for vision based systems are used for detection, tracking and pose registration. These coordinate systems are defined by the choice of camera model, and consist of, at minimum, a world coordinate system and a camera coordinate system. The world coordinate system is still a local one, but defines coordinates in the users world for example the room the user is in while camera coordinates refer to a virtual coordinate system defined relative to the camera.

The pinhole camera model [Faugeras, 1993] is the most widely used model (Figure 6.2). The main abstraction of this model is that the lens through which light enters the camera is replaced by an infinitely small aperture. The pinhole model defines three coordinate systems:

1. The 3D world coordinate system represents coordinates of real-world objects in an arbitrary user defined coordinate system.
2. The camera coordinate system is an intermediate 3D system defined with the origin at the cameras' optical centre.
3. The 2D image coordinate system where the camera coordinates are projected onto.

When using this model, a calibrated camera is assumed with known camera specific parameters. These are known as intrinsic parameters and should include at least the focal length and the camera centre of projection offsets. Some calibration techniques also calculate an axis skew parameter, although this parameter is less relevant with modern digital cameras.

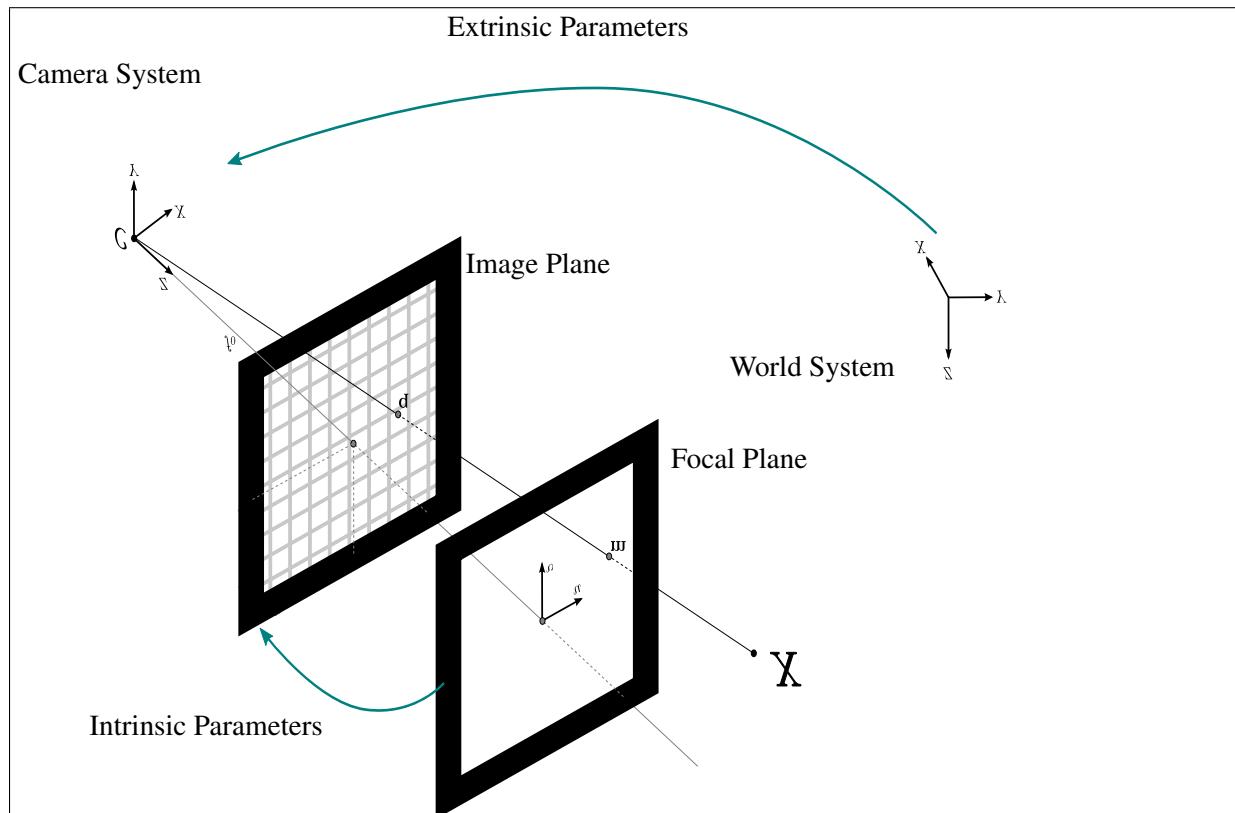


Figure 6.2: Pinhole camera model (Source: <http://lfa.mobivap.uva.es/fradelg/phd/tracking/camera.html>).

In order to transform coordinates in the world system to camera coordinates, parameters specifying an affine transformation consisting of the translation and rotation between the world and camera systems is required. In order to represent rotation and translation in a single matrix, homogeneous coordinates are used. Homogeneous coordinates represent n-dimensional points with multiple points in $n + 1$ dimensions. Coordinates $1 \dots n - 1$ are then multiples of the n th coordinate, for example (mx, my, m) would be a homogeneous representation of (x, y) as would (nx, ny, n) and $(x, y, 1)$ or any scalar multiple of the vector $(x, y, 1)$. Homogeneous coordinates provide a convenient tool when dealing with 3D to 2D projections and are an important part of Projective Geometry [Goldman, 2009].

Both intrinsic and extrinsic parameters are packaged into matrices. The intrinsics matrix is of size 3×4 and is given by $\mathbf{K} = \begin{pmatrix} f_x & \frac{s}{f_x} & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$ where f_x and f_y are the focal lengths in the x and y direction, c_x and c_y are the centre of projection offsets and s is the skew if available or zero if not. The extrinsics matrix is a 4×4 matrix $\mathbf{E} = \begin{pmatrix} \mathbf{R} & \vec{T} \\ \vec{0}^T & 1 \end{pmatrix}$ where \mathbf{R} is a 3×3 rotation matrix and \vec{T} is a

translation vector with three components.

The intrinsic and extrinsic matrices are then multiplied to form the projection matrix $\mathbf{P} = \mathbf{KE}$, which can then be used to transform world coordinates to image coordinate (As will be seen in Chapter 7, a further transformation is required for use with 3D APIs such as OpenGL or Vulkan.)

While the pinhole model is the most popular camera model, various other models exist. A variation of the pinhole model allows for multi-camera systems by supporting multiple camera coordinate systems as viewpoints on a single world coordinate system as implemented in OpenGV [Kneip and Furgale, 2014]. Specialised camera models also exist for more exotic types of camera lenses with wide but distorted field of view such as the Fisheye lens. Such cameras can be modelled using the approach of Devernay and Faugeras [2001] or OpenCVs alternate fisheye model [OpenCV, 2017].

The pinhole model itself can be modified to model the use of a lens instead of an aperture. The *paraxial refraction model* [Katz, 2002] is one such model that extends the pinhole model with parameters, such as the radius of curvature and index of refraction of the camera lens. Apart from its increased complexity, the practical use of this model is also dependent on hardware vendors providing these parameters which limits its application (although it may be used in a computer graphics ray tracing setting where such parameters can be arbitrarily set).

6.3 Patterns for Locational MAR

This section will describe patterns for use in a locational MAR artefact, although many of the patterns could also be of use in hybrid vision/locational artefacts. In addition to these patterns, two other locational MAR patterns are documented in Section 8.4.10, as their description is bound to the data access patterns described in Chapter 8. The PhD thesis of Michel [2017] also provides a comprehensive resource for locational MAR.

6.3.1 COMMON LOCATIONAL COORDINATES

Context: Provide a common geographically derived local coordinate system in which Points of Interest (POIs), annotations and other augmentable items can be placed.

Problem: While it is possible to use only latitude and longitude for coordinates, it is difficult to reconcile these coordinates with the concept of a 3D world viewed through a camera. Calculations involving direction of view of the user and resulting visibility tests and short range distance calculations for geographic objects is much easier and more intuitive using a familiar Cartesian system.

Solution: Convert the geodetic coordinates to ENU coordinates (see Section 6.2.2). The origin of the ENU system is then defined by the users current position in latitude and longitude as reported by the GPS sensor. Note that the relevant distance and directional calculations have to be re-calculated as the user moves, which also allows for the size of rendered objects to vary depending on the distance to the user/origin. Having the origin coincide with the users position also makes it easier to process data from other sensors which use a device local coordinate system with the device at the origin.

In order to calculate the ENU coordinates, the latitude and longitude must first be converted to ECEF using the following formulas:

$$\begin{aligned} e^2 &= (a^2 - b^2) / a^2 & X &= (v + h) \cos \phi \cos \lambda \\ v &= \frac{a}{\sqrt{(1 - e^2 \sin^2 \phi)}} & Y &= (v + h) \cos \phi \sin \lambda \\ & & Z &= (v(1 - e^2) + h) \sin \phi \end{aligned}$$

where ϕ is the latitude, λ is the longitude, h is the altitude, $a = 6378137m$ is the Earth's semi-major (Equatorial) axis, $b = 6356752.3$ is the Earth's semi-minor axis, $e = 0.081819190842622$ is the earth's eccentricity and X, Y, Z are the ECEF coordinates. Note that v is undefined at the pole (latitude $\frac{\pi}{2} = 90^\circ$).

Given the ECEF coordinates for the users position (X_O, Y_O, Z_O) , the ECEF coordinates for a augmentable object (X, Y, Z) and the geodetic coordinates of users position (ϕ_O, λ_O) the ENU coordinates (U, V, W) are calculated by the following matrix multiplication:

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} -\sin \lambda_O & \cos \lambda_O & 0 \\ -\sin \phi_O \cos \lambda_O & -\sin \phi_O \sin \lambda_O & \cos \phi_O \\ \cos \phi_O \cos \lambda_O & \cos \phi_O \sin \lambda_O & \sin \phi_O \end{bmatrix} \cdot \begin{bmatrix} X - X_O \\ Y - Y_O \\ Z - Z_O \end{bmatrix}$$

The users position is given by $(0, 0, 0)$ in ENU coordinates.

The python pyproj library² provides a useful prototyping and testing tool for locational coordinate transformations. For example to convert the location of the Eiffel tower to ECEF:

```
ecef = pyproj.Proj(proj='geocent', ellps='WGS84', datum='WGS84')
lla = pyproj.Proj(proj='latlong', ellps='WGS84', datum='WGS84')
pyproj.transform(lla, ecef, 2.2945, 48.8584, 300)
(4201133.036813931, 168331.00430233107, 4780438.968075508)
```

MATLAB also provides locational coordinate transformation functions in the Aerospace toolbox [The MathWorks Inc (Aerospace Toolbox), 2019].

Rendering of augmentable items is performed using the ENU coordinate space as the renderer world coordinate system (see Chapter 7). Items to be rendered can be divided into augmentations such

²<https://github.com/pyproj4/pyproj> or pip install pyproj

as annotation text and POI indicators on the one hand and 3D models such as Pokémon on the other. The first type should probably have constant size and therefore use an orthographic projection while 3D models can use a perspective projection so as to appear smaller when further away. In some cases it may also be desirable to render POI indicators using a perspective projection to vary size based on distance, however such items may end up being scaled to invisibility even when relatively close depending on the projection matrix in use. As an alternative, the Level of Detail (LOD) pattern (Section 7.5.1) could be utilised in conjunction with an orthographic projection to ensure POIs remain visible.

The perspective projection matrix defined in Equation 7.1 can be modified to use the device cameras field of view (FOV):

$$P = \begin{bmatrix} \frac{1}{a \tan \frac{f}{2}} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan \frac{f}{2}} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where $a = \frac{\text{width}}{\text{height}}$ is the aspect ratio, f is the vertical field of view, and n, f are the near and far Z coordinates.

The device OS should support obtaining the FOV or information from which it may be calculated. For Android for example the Camera2 API supports the *CameraCharacteristics* structure containing fields such as *SENSOR_INFO_PHYSICAL_SIZE* which can be used to calculate the FOV³.

Michel [2017] describes a variation for defining MAR coordinate systems which uses ECEF coordinates instead of ENU as the primary coordinate system. This approach obviates the need to recalculate POI coordinates when the user position changes, but complicates sensor calculations as the sensor coordinate system is centred on the device.

6.3.2 SENSOR BASED ORIENTATION

Context: Using device sensors for orientation of the MAR scene.

Problem: Assuming the MAR scene is represented using a coordinate system, as specified in Subsection 6.3.1 with the user/device at the origin then a means of determining the orientation of the device relative to the selected coordinate system is required.

³See <https://stackoverflow.com/questions/39965408/what-is-the-android-camera2-api-equivalent-of-camera-parameters-gethorizontalvie> for a comprehensive discussion on this topic.

Solution: Make use of the devices IMU sensors such as the accelerometer to obtain the vertical axis from the acceleration due to gravity and the magnetometer to obtain the direction to magnetic north to derive a rotation matrix or quaternion that can rotate between the device coordinates and a modified ENU coordinate system which uses geomagnetic north instead of true north. The necessary angles required to rotate the one coordinate system into the other can then be extracted from this matrix if required, often in the form of Euler angles. The inclination angle between magnetic north and true north can also be obtained at a given latitude and longitude and incorporated into a further rotation matrix. These rotations can then be applied as a model transformation when rendering items.

Fortunately the mobile device OS do supply APIs that provide most of the required functionality “out of the box”. The IMU data is obtained by an OS supplied OBSERVER [Gamma et al., 1995, p. 293], with an event callback in which the IMU data is part of the event along with a timestamp. The IMU data access is already conveniently parallelised as the OS callbacks take place in the context of an OS thread, although this means that the callback should execute as expeditiously as possible. Most mobile device OSs also supply APIs for extracting pose from the raw IMU data, for example in Android the *SensorManager.getRotationMatrix* call obtains rotation and inclination matrices given an acceleration and magnetic reading, the *SensorManager.getOrientation* API extracts the orientation from the rotation matrix as Euler angles (if required) and *SensorManager.getInclination* calculates the inclination. As an alternative to directly extracting the pose from the raw IMU data see the next pattern (SENSOR FUSION).

6.3.3 SENSOR FUSION

Context: Improving the accuracy of sensor based orientation derivation.

Problem: The accuracy of the orientation determination described in Subsection 6.3.2 is exclusively dependent on readings from the magnetometer and accelerometer. Unfortunately both types of sensors are prone to errors, particularly the magnetometer which is very sensitive to local magnetic fields in the devices vicinity. Most accelerometers provide fairly consistent reading for the direction of gravity, but do suffer from some noise in the readings which need to be smoothed. All of the above means the orientation reported by using each sensor in isolation will suffer from lack of accuracy and consistency.

Solution: Combine the results of several sensors by using a filter which allows readings from different sensors to compensate for and correct errors and smooth the resulting output. Mobile device sensor fusion for locational AR combines the magnetometer and accelerometer with a gyroscope and possibly GPS.

The filters are either stochastic, for example variants of the Kalman filter, or complementary which exploit known sensor combinations that provide readings that can supplement each other. A full description of sensor fusion is beyond the scope of this thesis, see Kok et al. [2018] for a comprehensive tutorial and overview of the use of sensors for orientation as well as sensor fusion.

Some examples of sensor fusion implementations include those of Renaudin and Combettes [2014], Madgwick et al. [2011] and Fourati et al. [2011]. See Michel [2017] for a comprehensive comparison and another implementation.

Mobile device OSs also supply sensor fusion in the form of “virtual” sensors which internally fuse several hardware sensors and then provide the filtered orientation via the OS callbacks similarly to hardware sensors. The Android rotation vector is an example of such a “virtual” sensor which fuses the gyroscope, magnetometer and accelerometer. The AOSP reference implementation of the rotation vector is an Extended Kalman filter (see <https://android.googlesource.com/platform/frameworks/native/+/refs/heads/master/services/sensorservice/Fusion.cpp>). It does not do any correction for magnetic anomalies however, which many of the previously mentioned fusion methods do.

It is also possible to fuse input from the camera in a hybrid visual-inertial tracking system. Drift and errors in the sensors can then be compensated for by using information from a camera frame where the timestamps on the frame and the sensor data match⁴, while the sensors can be used when camera data cannot be computed, such as when there are no recognised features in the frame. The development of such systems date back to the early history of AR [Suya et al., 1999], but the advent of SLAM system in the past decade has led to increased sophistication in the implementation [Mourikis and Roumeliotis, 2007; Leutenegger et al., 2015; Forster et al., 2017].

6.3.4 SCRIPTED CONTENT

Context: Providing MAR content or markers at pre-defined, pre-scripted locations.

Problem: Locational MAR authors⁵ require a means of describing the AR world in terms of placement of various types of content such as POIs, 3D models, annotations or multimedia content such as video or audio clips.

Solution: Utilise a textual description of the AR world, in a manner similar to HTML describing a web page, or, perhaps more appositely, the Keyhole Markup Language (KML) [Google Developers, 2017]

⁴Or approximately match as obtaining precise times for when the camera frames was captured as opposed to when the frame data was transferred to the OS from the camera hardware is difficult.

⁵Where the author is someone that can define and populate the AR world within the application with various types of content as opposed to the developer who creates the application.

used by Google Maps and other GIS providers to display geographic data in web pages or mobile device applications.

The concept of a file based definition for placing content in a 3D world dates back to 1994 when the initial version of the Virtual Reality Modelling Language (VRML) was specified. Various AR specific standards have been proposed including the VRML derived Augmented Reality Application Format (ARAF) [Lavric, 2017] and the Augmented Reality Markup Language (ARML) version 2.0 [Lechner, 2013]. This pattern may also be combined with the patterns from Chapter 8 to retrieve locational data specified in a format file.

6.3.5 LOCATIONAL ARCHITECTURE

Context: Architecture for locational MAR artefacts that can be used within the overall architectural framework defined in Chapter 5.

Problem: In order to design a MAR artefact a reference architecture is required.

Solution: Define a concurrent architecture based on the approach advocated in the **STRUCTURAL TASK GRAPH** pattern (Section 5.5.1), with the proviso that designers may choose a simpler architecture for small locational MAR artefacts that may not require a high level of concurrency due to their comparative simplicity requiring less computational efficiency.

Locational MAR is considerably less computationally intensive when compared to vision based MAR, therefore the requirement for large scale concurrency is not as high. The main drivers leading to the use of the **STRUCTURAL TASK GRAPH** are still present however, these being handling streams of large video frames and streams of high frequency but smaller size IMU data. Because of this a reference architecture utilising **STRUCTURAL TASK GRAPH** is described here.

Firstly implement a shared resource that can be concurrently accessed by tasks as described in **SHARED RESOURCE**. The IMU data can be obtained as described in **SENSOR BASED ORIENTATION**. As mentioned in **SENSOR BASED ORIENTATION** the OS supplied IMU callback should execute as expeditiously as possible as it is executed within the context of an OS thread, therefore using the **SHARED RESOURCE** pattern to hold an event queue where the sensor data is enqueued so that other components such as a **SENSOR FUSION** or non-fused pose calculation can use this to access sensor events is indicated.

The incoming video frames can be converted to RGB as described in *Acquire Frame* (Subsection 6.4.1 Page 105) and stored in a limited size frame queue in the **SHARED RESOURCE**. The renderer

can then run as a separate task awaiting frame availability.

In order to match pose to frame the **SHARED RESOURCE** can implement methods allowing sensor event access by timestamp proximity, which the **SENSOR FUSION**/pose calculation component can use to match frames and IMU data timestamps before notifying the renderer (but see the final paragraph of Subsection 6.4.1 for some issues with video frame timestamping).

If using **SENSOR FUSION** then it can be implemented either as a separate task, or part of the main task depending on the computational requirements of the fusion algorithms. For most locational systems the requirements are low enough for implementation in the main task, although if using a SLAM style optimisable pose graph (see Subsection 6.4.8) then a separate task may be required.

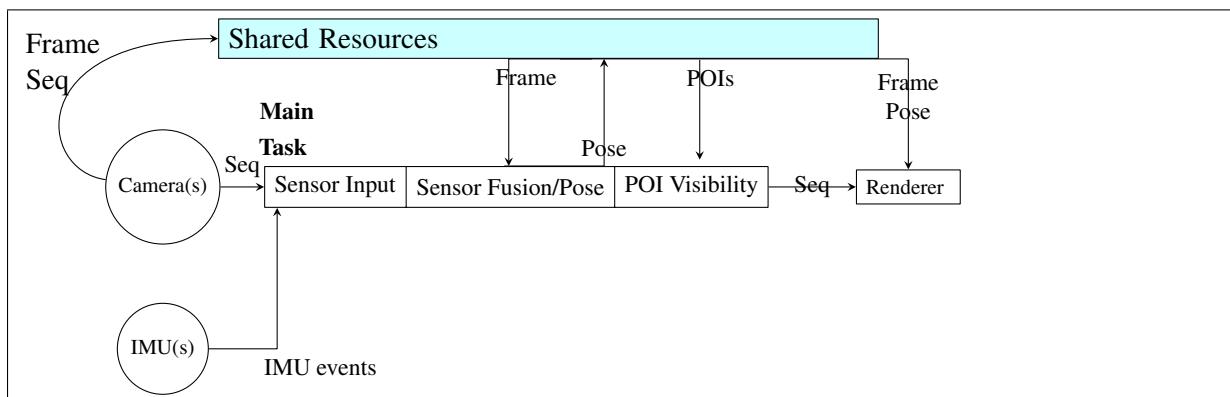


Figure 6.3: High level architectural design for a feature based MAR system utilising the **STRUCTURAL TASK GRAPH** pattern described in Section 5.5.1

Figure 6.3 illustrates a potential locational MAR architecture. Note the POIs are assumed to be obtained using the patterns described in Chapter 8 while the renderer is also assumed to be implemented using patterns from Chapter 7. The **SENSOR FUSION** could also be implemented in a separate task which is not illustrated here, however in that case there would also be a notification with the frame sequence number from the fusion task to the renderer when a pose has been calculated for a given frame.

6.4 Patterns for Vision Based MAR

The following patterns form the vocabulary for a vision based pattern language. Both feature based and SLAM MAR are covered.

SLAM is assumed to be keyframe based (see Subsection 4.4.8), as almost all modern SLAM systems are, as it is difficult to uncouple the older filter based approach into concurrent tasks or threads. Modern SLAM systems are also differentiated on whether they are sparse feature based or dense/semi-dense optical flow based. Because of the computational complexity involved with processing large arrays of

pixels, particularly on modern devices with high resolution cameras, the dense/semi-dense approach is less popular for MAR. There are more patterns for sparse SLAM documented, but dense/semi-dense SLAM is also covered both in the patterns and the PL. Several patterns also apply to both types of SLAM.

From a hardware perspective most of the patterns apply to monocular vision, although the POINT CLOUD POSE pattern (see Subsection 6.4.7) for finding correspondences and pose from point clouds is documented (it can also be used on 3D maps generated from SLAM). This is because since the demise of Google’s project Tango the support for RGB-D and other specialised point cloud generating hardware for mobile devices has all but disappeared. The patterns would however support stereo cameras through using epipolar geometry to provide binocular vision.

6.4.1 *Acquire Frame (OS Idiom)*

Context: Efficiently and timeously acquire video frames from the hardware camera sensor(s).

Problem: In order to utilise video frames for CV tasks, the frames need to be acquired from the hardware video stream and converted to processable formats. In addition in order to reduce latency effects this process needs to be performed as rapidly as possible, as even a few milliseconds of latency can make a difference to the reality of the AR experience. A related issue is that the video frames need to be accurately timestamped when used in a visual-inertial system as the video and IMU data timestamps need to be matched by the visual-inertial filter.

Solution: Solutions to this problem (and many other efficiency specific problems) are by nature somewhat OS specific. In general terms the gist of the solution is to make use as much as possible of the hardware capabilities of the device to perform the acquisition and conversions. This normally means making use of parallel GPU cores, and the solution, presented as an OS level idiom (the name is in italics to denote an idiom rather than a pattern), describes an Android solution using Renderscript (Google’s Java/Kotlin equivalent of GPU APIs such as OpenCL or CUDA) [Android API Guides, 2019].

The reasons for the use of Renderscript are:

- Heterogeneous GPU APIs include CUDA which is NVidia specific and OpenCL which is an open standard implemented by the major graphic card vendors such as NVidia⁶ and AMD. OpenCL would then seem to be a candidate solution, however its implementation is not standardised across Android devices so it is not always available, and when available is not necessarily accessible in a standardised way. Renderscript, however is standardised and available on all Android devices that

⁶Although benchmarks appear to show CUDA faster than OpenCL on NVidia hardware.

have GPUs;

- OpenCL/CUDA APIs are implemented in C, while Android camera API support at the NDK level is still not fully feature complete, although much improved in recent times. Renderscript is a Java/Kotlin API and can thus be used to interface to the Java Camera APIs directly and then pass the results on to lower level NDK C/C++ layers for CV processing; and
- Renderscript is used internally in the Android OS for video processing so it already contains several library routines for video conversion.

It should be noted that another alternative to Renderscript would be to use OpenGL/Vulkan compute shaders which are usable from both Java/Kotlin and C/C++ NDK, although this approach is not examined in greater detail in this solution. Apple iOS supports heterogeneous GPU programming through Metal, and presumably legacy OpenGL, compute shaders.

Video hardware usually encodes frames in some variant of a hardware friendly format known as YUV. The YUV format encodes the luminance as the Y channel and two chrominance (colour) components U and V. Different variants of YUV differ in memory layouts for the different channels, usually grouping the bytes from each channel linearly as opposed to RGB where the colour information for each pixel is combined per pixel in memory. This allows for faster transfer of video data to memory. The task of the acquisition component is to read the YUV from the camera and convert it to a RGB format that the CV software can operate on. In addition, as several CV feature detectors operate on greyscale images, converting incoming frames to greyscale is often required. In such cases it is more efficient to perform this conversion at the same time.

The following Renderscript listing highlights the basic detail of the solution. The code listed is a GPU kernel, that is, the GPU invokes the same kernel per pixel simultaneously (or in batches depending on the number of available cores). In this case the code uses Renderscript supplied YUV primitives to retrieve the Y, U and V components and then combines them and writes them to a RGB memory location and also to a separate greyscale output location for each location based on input parameters x and y for a 2D image. These YUV primitives abstract obtaining YUV colour and intensity bytes so the complexities of using different variations of YUV is minimised. This would not be the case for OpenCL or OpenGL/Vulkan/Metal compute shaders where the developer would have to write code to retrieve the requisite byte from the correct location for the type of YUV in use.

```
rs_allocation YUVinput;
rs_allocation greyOutput;

uchar4 __attribute__((kernel)) YUVtoRGBAGrey(uint32_t x, uint32_t y)
{
```

```

uchar py = rsGetElementAtYuv_uchar_Y(YUVinput, x, y);
uchar pu = rsGetElementAtYuv_uchar_U(YUVinput, x, y);
uchar pv = rsGetElementAtYuv_uchar_V(YUVinput, x, y);
uchar4 out = yuvToRGBA4(py, pu, pv);
uchar grey = 0.299 * out.r + 0.587 * out.g + 0.114 * out.b;
rsSetElementAt_uchar(greyOutput, grey, x, y);
return out;
}

static uchar4 yuvToRGBA4(uchar y, uchar u, uchar v)
{
    short Y = ((short)y) - 16;
    short U = ((short)u) - 128;
    short V = ((short)v) - 128;
    short4 p;
    p.x = (Y * 298 + V * 409 + 128) >> 8;
    p.y = (Y * 298 - U * 100 - V * 208 + 128) >> 8;
    p.z = (Y * 298 + U * 516 + 128) >> 8;
    p.w = 255;
    if(p.x < 0) p.x = 0;
    if(p.x > 255) p.x = 255;
    if(p.y < 0) p.y = 0;
    if(p.y > 255) p.y = 255;
    if(p.z < 0) p.z = 0;
    if(p.z > 255) p.z = 255;
    return (uchar4){p.x, p.y, p.z, p.w};
}

```

The Java/Kotlin code to set up the Renderscript is somewhat lengthy involving input and output allocations linked via the Camera API. See <https://github.com/android/renderscript-samples/> for sample source code. It should be noted that the Java callbacks occur within the context of an OS thread so further parallelisation is not required.

Video frame timestamping

Providing video timestamps for the output video is a related problem, which unfortunately is difficult to fully resolve. The camera hardware has a constant exposure time per frame, and on Android for example, the minimal exposure time is dictated by the minimum frame rate setting for the Camera API, so if the frame rate is set to 50 fps then the exposure time is 20ms. The hardware then needs to transfer the sensor data to memory which may take up to 5ms, followed by the OS driver in turn having to supply the copied memory to the client software. The result is that by the time the software can timestamp the frame it is already at least 25ms old and quite possibly even older. By contrast the IMU readings have much less overhead, both in not requiring an exposure time as well as the amount of data created so IMU data timestamps are more accurate, and more importantly, will not match the video timestamps. A hardware solution for timestamped video frames would be optimal, but in the absence of this, a possible solution is to attempt to measure the average video latency using techniques such as those of Chen et al. [2015] and use this to pre-date the timestamp.

6.4.2 FEATURE EXTRACTION

Context: Extracting unique recognisable features from images that are repeatable across multiple images.

Problem: Computer vision tasks such as image comparison, object detection and pose determination require that distinctive recognisable features can be identified in images. The requirements include:

- Features should be uniquely defined so that identifiably different points in the image map on to different features;
- The features should be repeatable so that the same point in different images result in features that when compared result in equality. In particular features should be scale and rotation invariant, at least to some degree; and
- Features should be resistant as far as possible to the effects of changes of lighting as well as various types of image noise such as speckling and blurring.

In addition to extracting the features there is also a requirement to create feature descriptors which can be used when comparing features in different images. The descriptors are required to encode feature information such as scale and dominant orientation, in such a way that it is easy for applications to compare the feature descriptors rapidly and accurately.

Solution: Use one of the many feature extractors and descriptors that have been developed over the course of the last twenty or more years of Computer Vision research in this area. The techniques used for feature extraction have been refined from Hessian based detectors in the 1970's through the then revolutionary SIFT extractor [Lowe, 2004] and on to the more recent detectors such as ORB [Rublee et al., 2011] and AKAZE [Alcantarilla et al., 2013]. The theory behind object detection was briefly discussed in Section 4.4.1.1. Grauman and Leibe [2011] also provide a tutorial and historical overview of the field, while numerous texts describing the usage of feature extraction for specific CV libraries and toolkits exists of which Kaehler and Bradski [2016] is an example for the OpenCV library. This solution section will not go into the specifics which are available in many programming texts, but will instead concentrate on broader principles and best practises.

The ability to test an application using multiple feature detectors is important, as different detectors have differing performance and accuracy attributes. For example AKAZE provides very distinctive features but is slow even on a desktop system. Other detectors, for example SIFT, are encumbered by

patents that may preclude their use in a given application⁷, but may still allow for their use during testing. Using the **STRATEGY** pattern to provide generic access to different detectors provides a solution where the detector used can be switched at compile time or runtime, or in the case of a patented detector can be removed from the final link of the application. Recent versions of OpenCV for example, provide the generic `cv::Feature2D` interface, with the detector implementations implementing the interface. In cases where toolkit or libraries do not apply a strategy pattern, the application can be designed to add this functionality. OpenCV users working with non-OpenCV detectors can also implement a `cv::Feature2D` wrapper for the detector to make it interoperable with existing OpenCV detectors.

When utilising the **STRATEGY** pattern approach described in the previous paragraph, it should be noted that most detectors are parameterised and the parameters are usually specific to a detector, or in some cases a class of detectors. A **FACTORY** [see Gamma et al., 1995, p. 107], **ABSTRACT FACTORY** [see Gamma et al., 1995, p. 87] or **DYNAMIC FACTORY** (see Subsection 8.4.7.3) may be used to facilitate the creation of explicit instantiations of detectors.

In most cases detectors and descriptors are interoperable⁸ so the output from one detector can be described by several different descriptor algorithms. Mixing and matching detectors and descriptors provides a further means to optimize a given application by utilising detectors and descriptors with different attributes and testing which combinations give best results. The **STRATEGY** pattern is again of value in abstracting the process.

6.4.3 FEATURE MATCHING

Context:

- Matching the descriptors of detected features between images; and
- Using the features for object recognition.

Problem: Once features are available for two (or more) images a means of matching features between the images is required. The resulting matches are used in several places for example object recognition and pose determination so accurate matching is important. A solution should be robust in the face of possible mismatches where descriptor comparison yield equality but the points are actually different (Figure 6.4).

⁷Although the SIFT patent applies only to commercial use (in the US) and not to academic and open source usage.

⁸This is not always the case; some detectors have custom dedicated descriptors so its a good idea to read the detector documentation before using this technique.

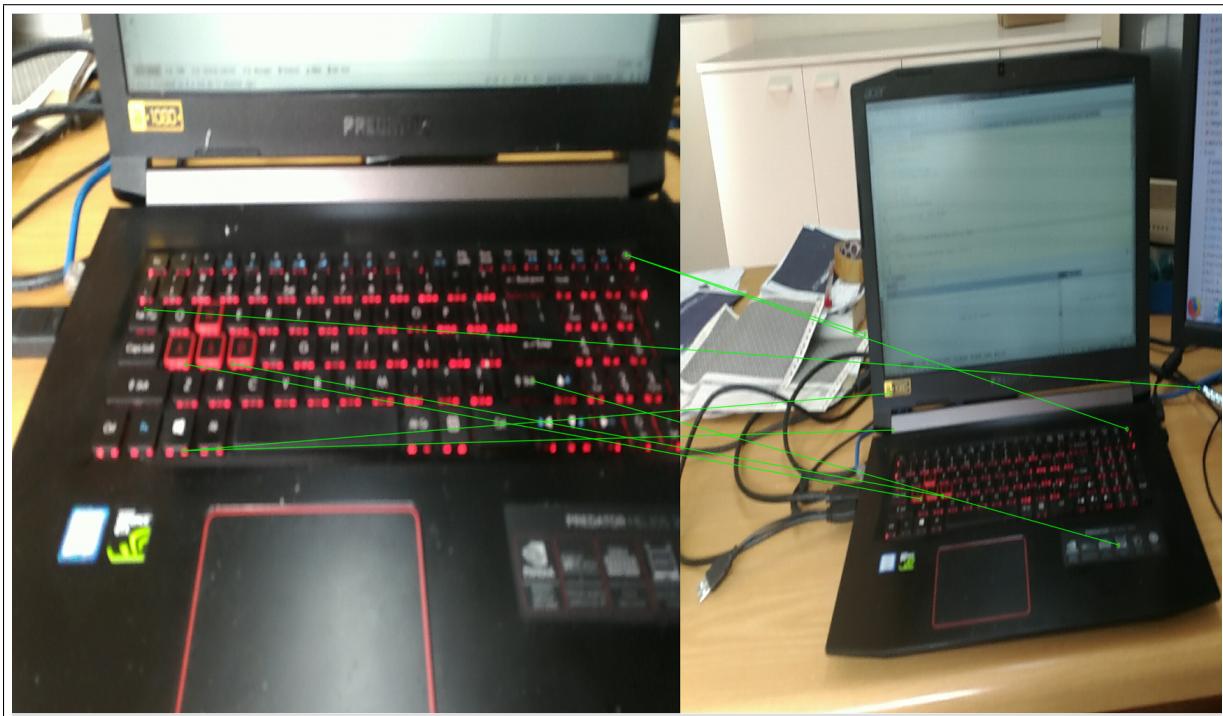


Figure 6.4: Example where some detected features are mismatched

Solution: Before describing a solution some notation will be clarified. In what follows, the training set will be the original (first) image or in the case of object recognition the object from the AR database, while the query set refers to the second image or in the case of object recognition the input video frame. The way in which descriptors are matched depends on how the descriptor metric is encoded. There are two approaches in common use, the first encodes the descriptor as a vector of floating point values (for the SIFT descriptor for example the values are the output from 128 orientation histograms surrounding the feature) and the comparator distance is computed as an L2 (or L1) Euclidean distance. Binary bit based descriptors encode the descriptor in a bit string and the distance is calculated using Hamming distances [Kelbert and Suhov, 2013].

Several techniques are combined to minimise the effect of mismatches and outliers amongst the detected feature points. The first step is choosing the descriptor matching algorithm to find the closest feature points in terms of the distance metric to the training feature point. The simplest approach is a brute force distance matching algorithm where, for every feature in the training set, the feature is matched against all features in the query set. This approach provides the most comprehensive solution but at a high cost in time and computing resources. When the number of features and/or the dimensionality of the features increases then the brute force approach becomes impractical. One alternative is the use of

clustering to speed up nearest neighbour searches, although unlike the brute force approach this approach is not guaranteed to always return the closest match, instead it returns the best match with some algorithm-specified probability and will otherwise return matches that are at least “close enough”.

Algorithms providing this functionality include KD trees [Samet, 2006], hierarchical K-means clustering trees [Arai and Barakbah, 2007] or Locality Sensitive Hashing (LSH) forests [Bawa et al., 2005]. OpenCV for example utilises the approach of Muja and Lowe [2009] which searches for a best choice from the above algorithms and concomitant algorithm parameters given a set of feature points. The nanoflann⁹ single header template based C++ KD-tree implementation provides a more lightweight KD-tree only implementation useful for testing and prototyping as well as general purpose K-D tree use. After finding the K nearest neighbour matches, a common heuristic to further refine the matches is to select features only where the ratio of best match to next best match is less than some fraction usually in the range 0.75–0.8 [Lowe, 2004]. Liu et al. [2014] provides further useful techniques for refining feature matches.

Another commonly used matching algorithm is the Visual Bag of Words, which originated from a text indexing algorithm known as Bag of Words (BoW) in which words are indexed by the count of their occurrences in different documents or locations. A similar approach can be used for indexing feature points using their descriptors, however visual BoW has been further optimised by Nister and Stewenius [2006] to utilise hierarchical K-means to build trees known as vocabulary trees comprising the feature words. The implementation of such a scheme encompasses building the trees from training images by extracting features and adding them to the tree. The query image features can then be extracted and looked up in the vocabulary tree.

A popular open source BoW implementation is available from <https://github.com/dorian3d/DBoW2>. One possible issue that should be taken note of is the k-means implementations used to build a BoW assume the use of an Euclidean metric, therefore if a descriptor implemented as a bit string is used then an extra translation step is required to use Hamming distance instead, see Grana et al. [2013] for details. A potential shortcoming of visual BoW is the spatial relationships between features disappears during indexing, Zhang and Mayo [2010] describe an enhancement which attempts to work around this issue. Support Vector Machines (SVM) [Campbell and Ying, 2011] can also be used in conjunction with BoWs to improve classification [Tian et al., 2007].

After the matching step, a further verification of whether the features matches are correct can

⁹<https://github.com/jlblancoc/nanoflann>

be carried out. This step is used to eliminate outliers and other feature mismatches. The most common verification utilises RANSAC (see Subsection A.4.1) which iteratively eliminates outliers and bad matches by fitting features to a model. The most commonly used model assuming the features are coplanar is that of the homography [Dubrofsky, 2009]. Other non-coplanar models such as an essential/fundamental matrix (See Subsection A.3) can also be used. In the case when it is unknown whether the points are coplanar, that is whether to use a homography or essential/fundamental matrix model, a heuristic is to calculate both, possibly in different threads, and then if the ratio $m_H/(m_H + m_F) \geq [0.4, 0.5]$ where m_H is the number of homography matches and m_F is the number of essential/fundamental matrix matches then use the homography result [Mur-Artal and Tardós, 2017].

The RANSAC process starts by randomly selecting the minimum number of feature points required for the model and uses these points to generate matching points in the query image. The reprojection error using these points is then calculated and points that fall within a given error distance are added to a set of inliers while the rest are added to the outlier set. This process continues iteratively until the necessary termination conditions are met as described in A.4.1. The Generalised Hough Transform have also been used to refine matches [Lowe, 1999], however this approach does not scale well to large numbers of features.

The geometric model used in the verification is an output from this step and can be utilised later, for example when calculating pose. Computer Vision API often do the verification as part of an API to calculate the geometric model, for example the OpenCV findHomography API takes a verification method parameter specifying whether to use RANSAC, Least Median Squares (LMEDS) or none. Selecting none would enable the customisation of the verification step by using your own RANSAC related verifier, before calling findHomography. Apart from the OpenCV internal RANSAC, several open source libraries can be used to supply RANSAC implementations. The Theia computer vision library¹⁰ [Sweeney, 2015] provides easily customisable RANSAC code, which is also easy to extract and use independently of the rest of the library. Theia provides standard RANSAC as well as several variants [Raguram et al., 2008] such as PROSAC [Chum and Matas, 2005] which appeared subsequent to the publication of RANSAC. A simpler C++ header only template library is also available from <https://github.com/donaldmunro/TemplRansac>.

When using feature matching for object detection a method of selecting which feature points to use in the training image can be useful as it allows for the manual selection of coplanar feature points as well

¹⁰<http://theia-sfm.org/>

as testing the selected points across multiple query images with the manual elimination of weak feature points. One such application is available from <https://github.com/donaldmunro/PlanarTrainer> which provides an UI for keypoint detection, selection, matching and saving the selected keypoints/descriptors and associated data to an AR database. A comparison of some of the more popular feature detection algorithms is available from Tareen and Saleem [2018] and Karami et al. [2017].

6.4.4 OBJECT TRACKING

Context: Tracking object movement between frames.

Problem: Once objects have been detected in a video frame, a method that can track the object between frames as the object or camera moves is needed. The tracking solution should be able to use information about the objects current location to optimise tracking in the next frame, particularly if the architecture specifies that the tracker speed dictates the application frame rate. It should also be robust enough to be able to cope with some degree of occlusion of the tracked object during the tracking process, and should also be able to re-acquire objects that disappear for a few frames, while also being able to decide when the object has been lost so as to notify other components.

Solution: Tracking, as with detection, has been a subject of much research in the CV community and a large variety of different tracking algorithms have been developed. The theory behind some of these algorithms are described in Section 4.4.2. Most trackers require only an initial bounding box to initiate tracking, possibly combined with detected features. There are exceptions however, some machine learning and neural network based trackers require offline learning with multiple images or even videos, while a few other trackers require access to object specific information such as contours or 3D models.

The simplest approach is to cut out the part of the image that is to be tracked and use it as a template to be searched for in the query image. This approach is known as template matching with the template from the training image being convolved with every possible position in the query image while using the sum of square differences for comparison. Apart from being very slow, this technique cannot account for even small changes in scale, rotation or lighting to the tracked object.

The next simplest approach is to use the methods of FEATURE EXTRACTION and FEATURE MATCHING (Section 6.4.3) for detecting features in every frame and matching the features to the training object features. This approach can be optimised by assuming some maximum distance the object can move between frames and restricting the detection to a bounding box defined by this distance instead of the whole frame (restricting search to a region known as a Region of Interest or ROI tracking is also

used by other more sophisticated trackers). Once the object is detected its centroid can be calculated and used to specify the next bounding box. While simple, this approach can easily be enhanced, for example sampling the initial detection box at multiple scales and detecting within the scale pyramid subsequently. For an example of the simple keypoint based approach optimised so as to be useful in real-world cases see Bergmann et al. [2019].

Predicting the next location given the current location is a classic filter application, indeed one could envisage a Kalman filter [Chui and Chen, 2017] added to the simple detector from the previous paragraph as another way of increasing its sophistication. Kalman filter based trackers have been used for object tracking for some time [Lee et al., 1995; Weng et al., 2006], although as the default Kalman filter is designed only for linear systems, tracking applications need to consider using more recent adaptations of the Kalman filter such as the extended Kalman filter [Cuevas et al., 2005] and the unscented Kalman filter [Chen et al., 2018]. Saho [2018] provides a modern overview of the design process for a tracking Kalman filter, including issues such as the noise model. OpenCV provides a linear Kalman filter class¹¹ (`cv::KalmanFilter`) and an unscented Kalman filter (`cv::tracking::UnscentedKalmanFilter`) either of which the user can extend to provide implementation specific models. The KFilter library (<http://kalman.sourceforge.net/>) also provides an extended Kalman filter. The MATLAB computer vision toolbox also supports Kalman filtering which is useful for prototyping purposes.

The particle filter [Doucet and Johansen, 2009; Chen, 2003] is an alternative to the Kalman filter which does not depend on linearity or a Gaussian probability distribution. Particle systems generate random “particles” combined with noise for a posterior distribution, given the current (prior) distribution, which, in a tracking context would mean particles for each possible location in the query frame. The posterior along with the measurement then becomes the input prior for the next frame. Unlike Kalman filters there are no modelling constraints such as linearity or a normal distribution for particle filters and the implementation of particle filters allow for high concurrency in multi-core systems. They can however eventually suffer concentration of more and more particles into a smaller sample area every iteration leading to eventual tracking loss, and the large number of particles required can consume excessive memory¹². An example of a successful particle based tracking system is provided by Breitenstein et al. [2009] in the form of a pedestrian tracking system. Recent versions of OpenCV provide a particle filter tracker (`cv::TrackerSamplerPF`). The Mobile Robot Programming Toolkit (<http://www.mrpt.org>) also provides

¹¹ Although it can be used as an extended Kalman filter as described at <https://sites.google.com/site/timecontroll/tutorials/extended-kalman-filtering-with-opencv>

¹² See http://www.cs.cmu.edu/~16831-f14/notes/F11/16831_lecture04_tianyul.pdf for some pros and cons of particle filters.

a particle filter implementation that can be modified for use outside of a robotics context. MATLAB also provides a particle filter (see <https://www.mathworks.com/help/control/ref/particlefilter.html>), but it is located in the control system toolkit, and not the computer vision one.

Optical flow, as described in Subsection 4.4.2.1 is still a popular tracking option despite its age. Optical flow can also be further refined to use filters to improve performance particularly in regard to occlusion [Shantaiya et al., 2015]. An implementation point to note when using (sparse) optical flow is that the detected object feature descriptors as found using FEATURE EXTRACTION (Subsection 6.4.2) are usually in one of the popular descriptor formats such as SIFT or ORB, however LK optical flow is optimised to work with the “good features to track” as defined by Shi and Tomasi [1993] and extended by Tommasini et al. [1998]. It is possible to use the keypoints associated with the SIFT/ORB descriptors but tracking performance appears to be better when using the “good features” [Nourani-Vatani et al., 2012]. As the “good features” are processed much faster than descriptors as they don’t need to perform scale and rotation invariance and gradient histogram calculations, adding an extra “good features” detection step in the detector component does not present a problem as the detector is not required to run at frame rate and the resulting features can then be utilised in the tracking component.

Mean-shift and CAMshift as described in Subsection 4.4.3 provides a tracking option which makes use of colour information of the tracked box represented as a probability distribution derived from a colour histogram. These methods are best used where the colour of the tracked item differs from that of the background. Hidayatullah and Konik [2011] describes an approach to improve CAMshift tracking resilience in the presence of less colour differentiation between the tracked object and the background and to support multiple object tracking. Iswanto and Li [2017] describe an approach which switches between CAMshift and a Kalman or Particle filter depending on the colour contrast. As would be expected OpenCV supports both methods, although not as part of the new Tracking framework. The MATLAB computer vision toolbox also supports CAMshift through the *HistogramBasedTracker*.

Many of the more recent tracker algorithms are based on discriminative correlation filters (see Subsection 4.4.5) frequently combined with machine learning techniques. These trackers sample the tracking bounding box or the entire image for positive and negative samples or generate permutation samples from the bounding box for training. These trackers include MOSSE, KCF and CSRDCF, with recent versions of OpenCV having implementations of these as part of the new tracking API which provides a generic interface for different trackers.

Another newer tracker which is not based on correlation filters is the Median Flow tracker described

in Subsection 4.4.4. This tracker works well provided that the tracked object is not moving at a high speed, that is the movement does not have big jumps between frames (or the frame rate is high enough to not have big jumps for faster moving objects). It also does not support occlusion well, but it does have good loss of tracking reporting. The OpenCV implementation for the median flow tracker is part of the new tracking API (`cv::TrackerMedianFlow`).

The Tracking-Learning-Detection (TLD) tracker [Kalal et al., 2012] provides a machine learning approach that attempts to learn from tracking errors. It uses another tracker to implement the tracking itself. In the OpenCV implementation (`cv::TrackerTLD`) the median flow tracker is used (although experimentation with using other newer trackers may improve its performance).

As with the FEATURE EXTRACTION implementations, using the STRATEGY [see Gamma et al., 1995, page 315] pattern to provide generic access to different trackers is important to allow for experimentation to find the best fit solution in a mobile context. This is particularly important for tracking as the overall frame rate of the application is dependent on the tracker component. Allowing compile time, or preferably run time selection of the tracker algorithm makes it easier to benchmark the application and select the best method. For OpenCV unfortunately not all the tracker solutions are part of the new tracker interface, for example both optical flow and CAMshift do not conform to the interface so the BRIDGE [see Gamma et al., 1995, page 151] pattern may need to be utilised to standardise their interfaces.

As an example of the importance of being able to select different tracking algorithms, informal surveys¹³ of several newer trackers including TLD, CSR-DCF, MOSSE, TLD and median flow appeared to show CSR-DCF (named CSRT in OpenCV) having the best mix of speed and accuracy, MOSSE the fastest but less accurate than CSRT/CSR-DCF while KCF lies somewhere in between in terms of accuracy/performance. This was on a desktop however, so more testing would be required on much slower mobile devices not having the benefit of a large number of GPU cores (OpenCV utilises CUDA or OpenCL for GPU use on desktops).

6.4.5 RELATIVE POSE

Context: Finding change in position and orientation (pose) between two images when:

- in a feature based MAR system an object from an AR database is detected and the image containing the object in the database has a known initial pose then the change in pose suffices to determine the pose used when rendering the object; or

¹³<https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking/>, <https://ehsangazar.com/object-tracking-with-opencv-fd18ccdd7369>

- in a feature based MAR system when the initial pose was found by absolute means (Section 6.4.6), subsequent poses can be found by relative pose between the image and the initial image which has an associated pose; or
- in a SLAM system relative pose changes between frames and keyframes is required when building the pose graph.

Problem:

- In feature based MAR, pose is required in order to align the world and camera coordinate systems for rendering the virtual content into the camera frame correctly. When the pose information is available for another image containing the object then a method of obtaining the change in pose is required; or
- In SLAM based MAR the pose change between frames is required during tracking.

Solution: Use the position of feature points which constitute the 2D projections of 3D points to calculate the pose. The methods used vary depending on whether the points are coplanar. For coplanar features where a homography has been found, as described in FEATURE MATCHING (Subsection 6.4.3), the homography matrix can be decomposed to extract the pose [Malis and Vargas, 2007]. For the more general case where features are not coplanar several relative pose methods based on epipolar geometry (see Appendix A.3) have been formulated. The original eight point solution [Longuet-Higgins, 1987] was successively refined to 7 points for uncalibrated cameras [Hartley and Zisserman, 2004], six points [Pizarro et al., 2003] and five points [Nistér, 2003; Stewénius et al., 2006]. Of these the most recent five point method of Stewénius et al. [2006] (a refinement of Nistér [2003] using Gröbner bases for numerical solution of polynomials) normally provides the best results although in the case of forward motion the eight point solution may be slightly better [Brückner et al., 2008].

OpenCV provides both a homography decomposition (`cv::decomposeHomographyMat`) and five point relative pose through a combination of `cv::findEssentialMat` and `cv::recoverPose/cv::decomposeEssentialMat`. Theia [Sweeney, 2015] provides numerous relative pose implementations including for 8, 7, 6 and 5 points as does the OpenGV library [Kneip and Furgale, 2014]. As an implementation note, these techniques can result in multiple solutions, so a further step to eliminate non-visible solutions and choose the best solution is also required [Brückner et al., 2008].

The above methods dealt with computer vision only relative pose detection. It is possible to utilise IMU sensors in mobile devices to simplify pose determination by utilising extra information available from the IMU. Fraundorfer et al. [2010] provide a relative pose estimation method that views the essential

matrix \mathbf{E} in terms of the composition of separate rotation matrices for the Euler angles yaw, pitch and roll $\mathbf{E} = [\vec{t}]_x(\mathbf{R}_y\mathbf{R}_p\mathbf{R}_r)$. They then use the epipolar constraint to provide a linear system of equations based on simplification by setting the roll and pitch matrices to \mathbf{I} . They then provide a linear 5-point solution where the solution has to be corrected by rectifying the resultant essential matrix to satisfy the essential matrix constraints. They also provide a 4-point and minimal 3-point solution resulting in cubic and quartic polynomials providing a maximum of three and four separate solutions respectively.

A more recent gravity assisted relative pose solution is provided by Sweeney et al. [2014] who use a combination of reformulating the essential constraint as $-(\vec{x}' \times \mathbf{R}\vec{x}) \cdot \vec{t} = 0$ and parameterise the rotation matrix using a quaternion representation $[c, \vec{g}]$ as $\mathbf{R} \approx 2(\vec{g}\vec{g}^T + c[\vec{g}]_x) + (c^2 - 1)\mathbf{I}$ where \vec{g} is the normalised gravity vector and c is the unknown $\cos \frac{\theta}{2}$. They substitute the rotation approximation into the essential constraint to get a quadratic in c . They then stack these quadratics in the form $(c^2 M + cC + K) \cdot t = 0$ which can be solved as a quadratic eigenvalue problem [Tisseur and Meerbergen, 2001]. The solution provides 4 real-valued solutions which can then be substituted into the rotation matrix parametrisation. The implementation of the algorithm of Sweeney et al. [2014] is available at <http://theia-sfm.org> [Sweeney, 2015].

The algorithms of Fraundorfer et al. [2010] and Sweeney et al. [2014] above apply for calculating the pose between two images with no extra pose information available such as might be the case with an object from a feature based AR database. See Appendix B for a gravity assisted relative pose calculation where it is assumed one image is a object from an AR database with associated information.

An alternative which does not require FEATURE EXTRACTION or FEATURE MATCHING for finding the relative pose between frames was proposed in Engel et al. [2013]. This approach, which was implemented in LSD-SLAM [Engel et al., 2014] and is normally used in a SLAM context, uses a direct pixel derived approach which minimises a cost function for pixel intensity with the pose as a parameter. Engel et al. [2013] and Engel et al. [2014] reduce the high computational complexity by only using pixels near high gradient areas. They also use the Huber norm which is less sensitive to outliers instead of the Euclidean norm and apply variance normalisation.

6.4.6 ABSOLUTE POSE

Context: Finding the position and orientation (pose) of a 2D projection of an object in an image with respect to a 3D geometric model of the object

- in a feature based MAR system an object from an AR database is detected and the the object in the database has an associated 3D model; or

- in a SLAM system when using the 3D map to find the pose (in this case the 3D points are synthesised during SfM processing).

Problem:

- In feature based AR a detected objects pose is required in order to align the world and camera coordinate systems for rendering the object into the camera frame correctly. When the object has associated 3D model data, then a method of obtaining the pose from correspondences between 2D feature points and 3D model points is required; or
- In a SLAM system pose must be obtained from matches between previously encountered points which have been triangulated and added to the 3D map as 3D points and 2D feature points in newly acquired frames.

Solution: Use one of the absolute pose algorithms described in Subsection 4.4.7.2 (Page 58) to calculate the pose. Alternately utilise one of the IMU assisted approaches described later in this subsection.

The fastest algorithms for absolute pose are the class of non-iterative P3P solutions. The P3P problem has a long history [Haralick et al., 1994], with most of the classic solutions being variations on building polynomials using the law of cosines applied to the edges joining three 3D points to the camera origin, resulting in four possible solutions. Kneip et al. [2011], Ke and Roumeliotis [2017] and Banno [2018] provide more modern approaches that reduce error and improve performance. The main use of the P3P algorithms however, is iterative as a RANSAC (see Appendix A.4.1) model when processing more than three points.

The multiple point PnP class of solutions can be classified as iterative or non-iterative. Among the non-iterative approaches EPnP [Lepetit et al., 2009] and UPnP [Kneip et al., 2014] are freely available in CV libraries. Finally the iterative PnP solution utilising least squares minimisation with Levenberg-Marquardt optimisation (Appendix A.4.2) provides the highest accuracy but at a cost in speed.

OpenCV supports P3P and PnP through a single interface, namely the solvePnP call (also variants solvePnPRansac which performs RANSAC and solvePnPGeneric which returns all possible solutions). The particular method used is passed as a parameter to solvePnP. For P3P the solution of Gao et al. [2003] (SOLVEPNP_P3P) is supported, and later versions also support Ke and Roumeliotis [2017] (SOLVEPNP_AP3P). The P3P solvePnP/solvePnP Ransac calls require 4 points instead of 3 with the extra point to disambiguate the correct (visible) solution. For the PnP non-iterative case, OpenCV implements both EPnP (SOLVEPNP_EPnP) and UPnP (SOLVEPNP_UPnP). Finally the full iterative solution is supported using SOLVEPNP_ITERATIVE. When using the solvePnP variants, it should be noted that

they throw a C++ exception for errors such as invalid numbers of points supplied, so adding exception handling at some point in the call stack to the pose handlers is important.

OpenGV [Kneip and Furgale, 2014] has an implementation of Kneip et al. [2011], Kneip et al. [2014] and Gao et al. [2003] and several other PnP solutions. Theia [Sweeney, 2015] also supports numerous absolute pose algorithms.

When experimenting using multiple pose algorithms from multiple sources, the **S T R A T E G Y** [see Gamma et al., 1995, page 315] may be employed to provide a generic pose interface. The **B R I D G E** [see Gamma et al., 1995, page 151] pattern may also be used to translate between object-oriented interfaces from different providers.

It can be difficult to create AR databases for matching features from **F E A T U R E M A T C H I N G** and 3D points. Software that allows such manual matching of points in a point clouds to detected features is available at <https://github.com/donaldmunro/PnPTrainer>.

As with relative pose, the use of a gravity vector can assist during pose determination. Kukelova et al. [2011] provided one of the first gravity sensor assisted absolute pose estimation methods. They used the gravity vector to find the rotation as Euler angles around the X and Z axes leaving the Y axis as the only rotational unknown (a single reading from a gravity sensor can only provide two out of three Euler angles). They solve the equation $\lambda\vec{u} = \lambda(u_x, u_y, 1) = \mathbf{K}(\mathbf{R}\vec{X} + \vec{t})$ by taking the cross product of both sides of $\lambda\vec{u} = [\mathbf{R} \ \vec{t}]^T\vec{X}$ with \vec{u} resulting in the left hand side being zeroed. They then solve the resulting quadratic system to get the tangent of the Y axis rotation angle and then solve for the translation by back substitution which results in a linear system. As the system is quadratic two possible solutions are obtained.

Sweeney et al. [2015] solve for the depths of two points λ_1 and λ_2 (using two point correspondences) utilising two constraints namely that two points must have the same distance between them in the camera and world coordinate systems and the vector between the two points projected onto the vertical (gravity determined) vector will be the same. These constraints again lead to a quadratic system with two solution sets of λ_1 and λ_2 . The rotation around the gravity vector axis is then obtained using the solutions for λ_1 and λ_2 and the translation is then found by subtraction of the rotated points in the world and camera systems. The implementation of Sweeney et al. [2014] is available at <http://theia-sfm.org> [Sweeney, 2015].

For a gravity assisted PnP solution that utilises extra pose information available such as might be the case with an object from a feature based AR database, (see Appendix B).

6.4.7 POINT CLOUD POSE

Context: Finding correspondences and pose between two 3D point clouds.

Problem: When utilising a RGB-D sensor that can return 3D point clouds a means of matching points between two point clouds and finding the 3D pose using the matches is required.

Solution: Use the Iterative Corresponding Point (ICP) algorithm [Chen and Medioni, 1991]. The algorithm minimises the pose difference using a cost function to find the best matches between points. Yang et al. [2016] provide a more recent incarnation named Go-ICP using branch and bound to search the Euclidean space and reduce issues with local minima encountered by the original ICP.

The C++ source for Go-ICP is available from <https://github.com/yangjiaolong/Go-ICP>, with a Python version also available from <https://pypi.org/project/py-goicp/>. The Point Cloud Library <https://github.com/PointCloudLibrary/pcl> also provides an implementation.

6.4.8 BUNDLE ADJUSTMENT

Context: Inferring 3D coordinates and refining relative pose from multiple image points occurring in multiple images.

Problem: In a SLAM MAR system a 3D map of the encountered environment must be built using relative pose differences between keyframes gathered from 2D camera projections of the 3D points. Because of small errors (and larger outliers) drift may accumulate so the entire map as well as the poses needs to be optimised when new keypoints are added or a loop is detected (that is, when a previously visited location is revisited).

Solution: Represent keyframe poses in a Factor Graph [Kschischang et al., 2001; Dellaert and Kaess, 2017] and perform inference on the model to optimise both the stored poses and the 3D positions.

A factor graph can be defined as [Dellaert and Kaess, 2017]:

... a bipartite graph $F = (\mathcal{U}, \mathcal{V}, \mathcal{E})$ with two types of nodes: **factors** $\phi_i \in \mathcal{U}$ and **variables** $x_j \in \mathcal{V}$. Edges $e_{ij} \in \mathcal{E}$ are always between factor nodes and variables nodes. The set of variable nodes adjacent to a factor ϕ_i is written as $\mathcal{N}(\phi_i)$, and we write X_i for an assignment to this set. With these definitions, a factor graph F defines the factorization of a global function $\phi(X)$ as $\phi(X) = \prod_i \phi_i(X_i)$. In other words, the independence relationships are encoded by the edges e_{ij} of the factor graph, with each factor ϕ_i a function of only the variables X_i in its adjacency set $\mathcal{N}(\phi_i)$.

In Figure 6.5a below, solid circles will represent factors while hollow ones represent variables (although other graphical notations for factor graphs exist, for example denoting factors as squares).

For each keyframe the pose estimate x_i can be found in terms of the previous keyframes pose estimate x_{i-1} and a measurement z_i , $x_i = f_i(x_{i-1}, z_i) + \mathcal{N}(\mu, \sigma^2)$ where $\mathcal{N}(\mu, \sigma^2)$ is Gaussian noise. In a MAR visual SLAM system the x_i 's are 3D rotations \mathbf{R}_i and translations $\tilde{\mathbf{t}}_i$ and the measurement is a set of found feature correspondences associated with 3D points (or landmarks) in the 3D map and the current frame (the pose graph does not directly store the measurements, instead they are held in the 3D map and are referenced when processing the graph).

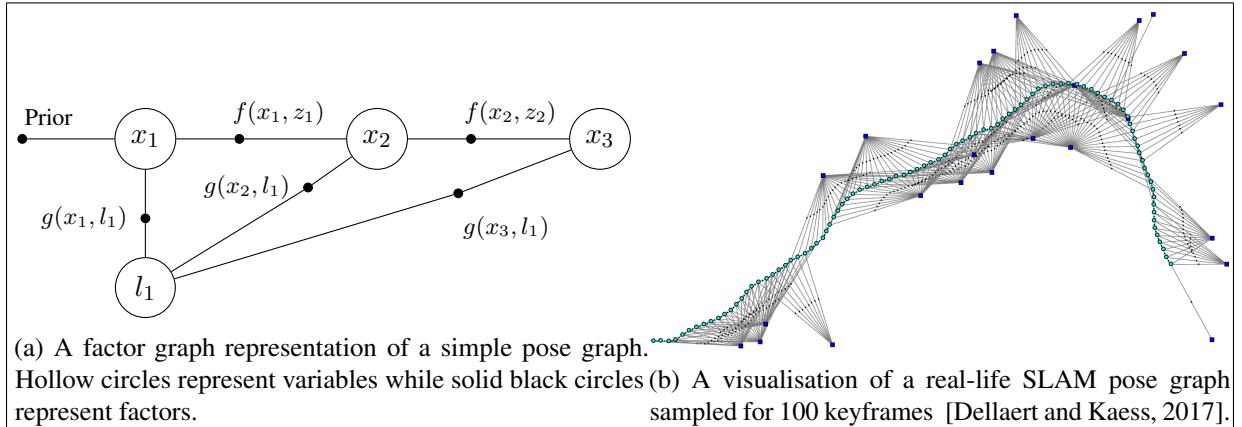


Figure 6.5: Factor graphs.

A factor graph representation of a pose graph has the unknowns, in this case the poses and 3D points, as variable nodes while the measurement functions are represented by factors. Figure 6.5a depicts a much simplified pose graph with three keyframes and a single 3D point or landmark. Figure 6.5b shows graph visualisation for a more realistic pose graph.

Factor graphs in general allow the representation of factorisable functions, and when applied in a

probabilistic setting for marginalisation and factorisation of multiple random variables using the marginalisation and chain rules of probability. If the poses are modelled probabilistically in a factor graph expressing conditional probabilities then the total probability can be represented as $\Pr(X, L | Z)$ where $X \triangleq \{x_i\}$ and L is the set of 3D points in the map. The factor graph representation makes it possible to extract a computationally feasible representation of the maximum a posteriori (MAP) estimate:

$\hat{\theta} \triangleq \operatorname{argmax}_{\theta} P(X, L | Z) = \operatorname{argmax}_{\theta}(P(X, L, Z)) = \operatorname{argmin}_{\theta}(-\log P(X, L, Z))$ where $\theta \triangleq (X, L)$ is the set of unknowns. The minimisation occurs over the reprojection error of the observed 2D image location and the corresponding reprojected point where the reprojection is calculated using the pose at that keyframe. A further simplification is that because the probability distribution is Gaussian the minimisation reduces to least squares which is amenable to optimisation methods such as Levenberg-Marquardt (see Appendix A.4.2).

The inference process itself involves performing variable elimination on the nodes to convert the factor graph to a chordal Bayes net which is converted to a tree from which the actual inference is then trivial [Kaess et al., 2012]. Another useful feature of the factor graph is that methods allowing online updates to the factor graph have been developed [Kaess et al., 2012]. This means that the graph can be added to as new frames are processed. Typically these updates are highly efficient as very small parts of the tree are changed. Unlike traditional bundle adjustment techniques factor graphs are not restricted to modelling projective geometry but can instead model anything where factors can be written as functions of random variables, for example IMU measurements could be added to a factor graph.

It is also possible to numerically solve the inference problem using linear algebraic means as the factor graph can also be expressed as a sparse matrix which can be factored using Cholesky decomposition. The matrix representation however is less flexible which makes it difficult to dynamically update the estimate. In addition a linear algebraic approach is by definition limited to representing linear systems (for SLAM the first order Taylor term is used), while factor graphs can represent non-linear systems.

Two types of bundle adjustment are normally performed at different stages during the SLAM lifecycle. Local bundle adjustment occurs when a new keyframe is added. Only keyframes visible from the new keyframe and keyframes that share 3D points with the visible keyframes are processed. Global bundle adjustment, where all keyframes are processed, occurs when a previously encountered location is re-encountered (loop closing).

Implementing a factor graph for SLAM from first principles would be a large and time consuming undertaking. Fortunately several open source libraries have been developed in recent years. Of these

possibly the most popular is g2o¹⁴ [Kümmerle et al., 2011] which is used by both ORB-SLAM2 [Mur-Artal and Tardós, 2017] and LSD-SLAM [Engel et al., 2014]. Georgia Institute of Technologies provide GTSAM¹⁵, which also provides IMU support and both MATLAB and Python frontends for testing and prototyping. SLAM++¹⁶ [Ila et al., 2017] provide another option that also supports matrix based optimisation. The MATLAB navigation toolbox also provides an *optimizePoseGraph* call which can use g2o or an internal MATLAB optimiser.

6.4.9 KEYFRAME IDENTIFICATION

Context: Selecting keyframes from incoming camera frames.

Problem: When the incoming frame rate is sufficiently high, many incoming frames will be very similar. It is inefficient to perform expensive operations such as bundle adjustment for SLAM or object detection for feature based AR on every incoming frame. Instead it is preferable to only perform these operations for frames that are identifiably different, or some other heuristically defined criterion is matched. A method to identify new keyframes is therefore required.

Solution: Keyframes were first used in video compression schemes such as MPEG, in which a keyframe is identified and stored. Subsequent frames are then stored only in terms of their differences from the keyframe [Richardson, 2010] until a new keyframe is initiated. The selection criteria for video compressor keyframes is very simple as they are selected at regular time intervals defined by the compression level. This is because the recording process must be implementable in hardware and secondly the stream must be recorded in a standardised way so a decoder can play it back. MAR systems however, can be more flexible as they do not need to follow a standard.

For sparse MAR systems utilising FEATURE MATCHING (Subsection 6.4.3), the number of feature matches can be utilised to decide when the keyframes should be updated. Dense non-feature based MAR systems utilise the number of dense correspondences between the current keyframe and the current frame instead.

There are other criteria that may apply for keyframe selection. For example, the user could rotate the device without moving the camera in which case the current frame may still match the keyframe, but the pose will be incorrect. Because of this a heuristic approach to keyframe selection is usually followed for real-world applications. A good example would be the approach of ORB-SLAM2 which has a general

¹⁴<https://github.com/RainerKuemmerle/g2o>

¹⁵<https://github.com/borglab/gtsam>

¹⁶<https://sourceforge.net/p/slam-plus-plus/wiki/Home/>

policy of inserting keyframes frequently to ameliorate problems with faraway points in outdoor AR.

ORB-SLAM2 applies the following heuristics to choose whether to switch:

- More than one second of frames at frame rate have been processed since the last new keyframe;
- More than a parametrised minimum number of frames have been processed since the last new keyframe when the local mapping task is idle; or
- if the number of feature matches drops below 90% (25% for RGB-D camera) and there are at least 15 matches

Other possible heuristics for keyframe switches used in other SLAM systems include:

- Rotation of new frame compared to keyframe more than 15%;
- Average translation of all 2D feature pixel points exceeds 20% of image width; or
- Number of feature correspondences below 30% of a global average.

Because the cost of running `BUNDLE ADJUSTMENT` is dependent on the number of keyframes, pruning unnecessary keyframes periodically is also important. A common heuristic is to delete keyframes for which a high percentage (90% or more) of mapped 3D points visible from the keyframe are also visible in three other keyframes.

Once a new keyframe is identified it should be placed on a shared queue or related data structure so other components such as `BUNDLE ADJUSTMENT` can access it.

6.4.10 DIRECT 3D MAP

Context: Building a 3D map when implementing a dense or semi-dense SLAM system.

Problem: Direct dense or semi-dense SLAM system must directly use pixel intensities in frames to build a 3D depth map instead of building a 3D features (landmarks) map that sparse SLAM does.

Solution: Create a per-pixel depth map when creating a new keyframe, using projected points from the previous keyframe (or randomised for the initial keyframe). The depth map represents the probabilistic depth of the pixel which is accompanied by a variance for the depth. Subsequent frames, until the next keyframe, are used to refine the keyframe depth map using epipolar geometry, or possibly multi view geometry, with the variance associated with the depth being reduced every iteration. The refinement, as with the dense/semi-dense approach described in the final paragraph of Subsection 6.4.5, is an iterative optimisation process. The method is described in Engel et al. [2013] and implemented in Engel et al. [2014].

6.4.11 LOOP CLOSURE

Context: Detection of revisited locations for pose and map validation in SLAM MAR.

Problem: After exploring an environment the user may return to a previously visited location, thereby affording the system the possibility of further optimisation and drift elimination. Techniques for detecting loops are therefore required as well as actions that should be performed when a loop closure is detected.

Solution: Loop closing is usually run in a separate task as it is computationally expensive and would slow the frame rate down if run as part of the main task. The loop closing task should monitor the keyframe queue for new keyframes and check these keyframes for loop closure.

For sparse feature based MAR systems a common approach is to utilise FEATURE MATCHING (see Subsection 6.4.3) to find possible matches amongst all keyframes based on their BoW similarity score. A more optimal approach if a visibility list is maintained of all keyframes visible from a given keyframe, is to find a minimum score for all keyframes visible from the current keyframe and use this as a minimum score to find candidate matches with a better score in the global BoW. For each such candidate check consistency in a similar way for its visibility list up to a given depth. Finally for each candidate try and find a similarity transformation (rotation, translation and isotropic scaling) using RANSAC and select the best solution, or reject the loop closure if the matches are not close enough.

For dense image based systems such as LSD-SLAM [Engel et al., 2014] a BoW approach is not possible so another approach of checking for a similarity transformation on keyframes from a visibility list is performed using direct image alignment and depth maps [Engel et al., 2013].

Once a loop is detected the positions of all common 3D points between the loop candidate and the keyframe must be reconciled. Global bundle adjustment must then be performed to re-optimize based on the reconciled 3D points. The global bundle adjustment can also be started in a new task to allow the loop closure task to continue evaluating incoming keyframes.

6.4.12 DETECT/TRACK ARCHITECTURE

Context: Define a concurrent architecture for vision based detection, tracking and registration that can be used within the overall architectural framework defined in Chapter 5.

Problem: Detection and tracking are the most time-sensitive components in a MAR system, consequently it is important that they have optimal access to computing resources.

Solution: Compartmentalise components so they can run concurrently. The architectural approach advocated in the **STRUCTURAL TASK GRAPH** pattern (Section 5.5.1) details a task based decomposition where the tasks can be represented as nodes in a DAG.

The following discussion will start by focusing on the use of parallelism in SLAM systems as this is where the explicit use of concurrent architectures is documented. This concurrent architecture will then be extended to feature based visual MAR.

The original use of a concurrent architecture for SLAM is described in Klein and Murray [2007] in which a mapping thread builds a Structure from Motion (SfM) 3D map while a tracking thread matches features to the latest map, calculates the latest pose using 2D feature to 3D point PnP and renders the frame. The mapping thread computation is more time-intensive and thus is not required to match the frame rate.

Later SLAM implementations refined this approach to use more threads. For example, Mur-Artal and Tardós [2017] adds a third Loop Closing thread and periodically utilises a fourth thread for bundle adjustment (see Section 4.4.8 for an overview of SLAM). An example, where this approach is adapted to the **STRUCTURAL TASK GRAPH** pattern is illustrated in Figure 6.6.

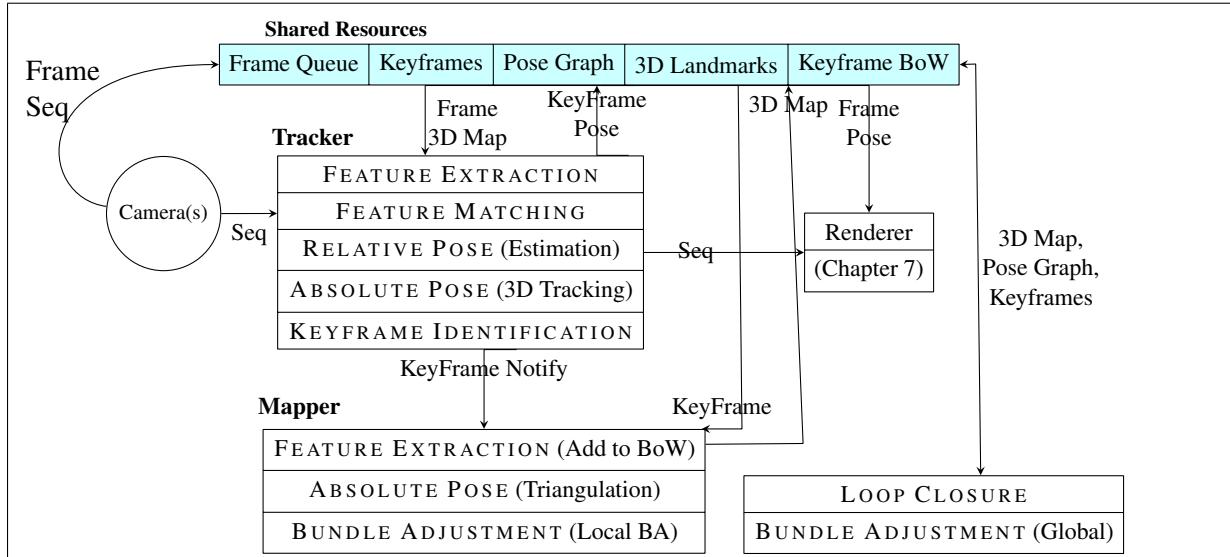


Figure 6.6: High level architectural design for a SLAM system utilising the **STRUCTURAL TASK GRAPH** pattern described in Section 5.5.1

While concurrent architectures for SLAM systems have been explicitly documented more frequently, the same principles apply to feature based MAR. For example feature detection and object recognition using the detected features typically operates on a full image, but it may not be critical that new objects are

instantaneously detected, while tracking¹⁷ typically operate on smaller regions of interest (ROI) within the image and can be fast enough to operate at the best case frame rate for the application, allowing tracking and pose calculations for previously detected objects to occur at the same rate that rendering occurs at. Another task could also concurrently process sensor data in order to provide approximate pose when there are no visible recognisable predefined objects in the frame .

An example of a such a concurrent design, which also incorporates the keyframe concept into a feature based MAR system is illustrated in Figure 6.7. In this architecture, object detection is only performed on new keyframes with the detector adding detected objects detected using an object database to a tracked object list. The new keyframe detector decides whether an incoming frame qualifies as a new keyframe and if it does, updates the current keyframe in the repository and notifies the detector. The tracker executes on incoming frames and attempts to track objects in the tracked list, updating their position if trackable or removing them from the tracked list if not. The tracker also perform pose registration and updates the current pose, which is utilised by the renderer along with the current frame and tracked objects to render AR content. Sensor data is acquired in a separate concurrent task and pose data is extracted from the orientation sensors to supply an alternate pose approximation in case there are no visible objects to use for vision based pose determination¹⁸. The geographic location is also updated and POIs and other geographic data may then be retrieved as described in Chapter 8.

6.5 Pattern Language

In this section valid pattern sequences for various design scenarios will be described. The pattern sequences will be further illustrated using railroad-style syntax diagrams [Braz, 1990]. The detection/tracking components designed here will slot into the larger architectural design specified by the task graph architectural pattern (see Section 5.5). The pattern sequence descriptions will describe any concurrency detail regarding individual patterns that affect its representation within the architectural pattern as a separate concurrent task or part of an existing task.

6.5.1 Locational MAR

An overall concurrent architecture is described by LOCATIONAL ARCHITECTURE (see Subsection 6.3.5), although simple locational MAR application may elect to use a simpler architecture. The first choice to make for a locational MAR artefact is the coordinate system to use. It is possible to use WGS-

¹⁷The terminology differs from the SLAM case as in SLAM the tracking thread does do feature detection (but not object recognition) for frame to keyframe comparison.

¹⁸The pose data may also be fused with vision based pose in a visual-inertial filter.

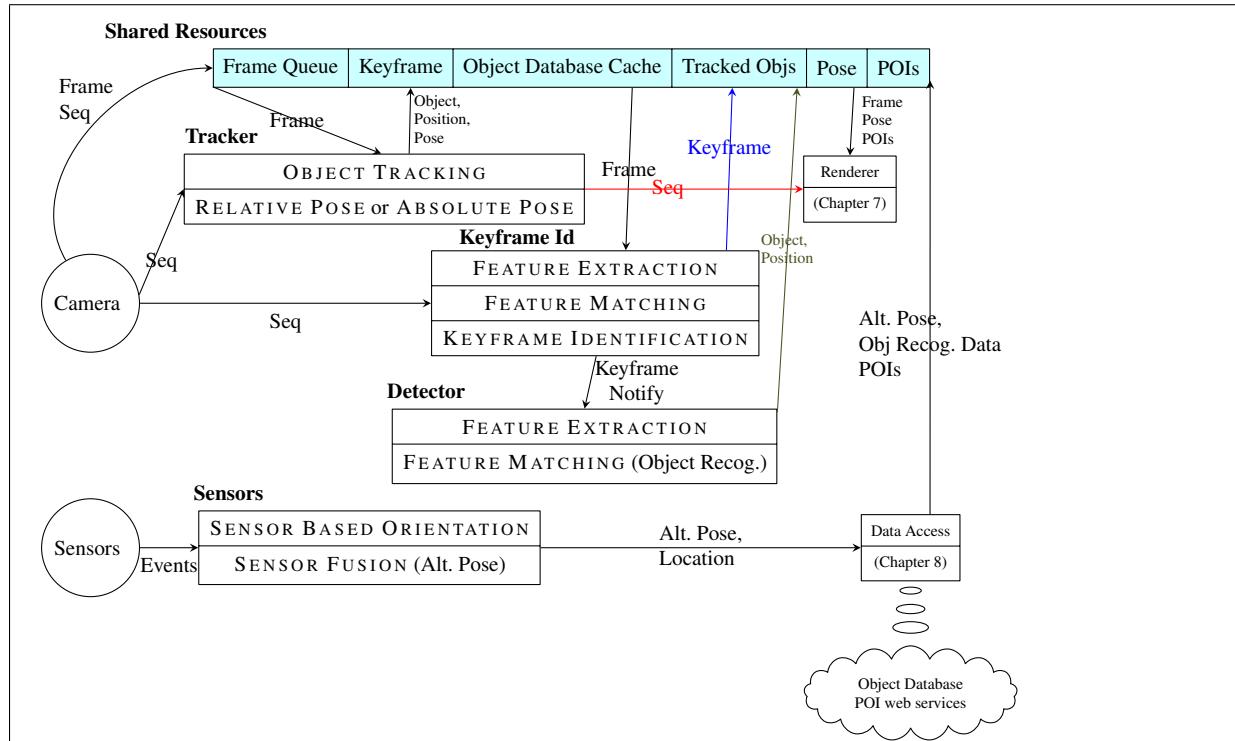


Figure 6.7: High level architectural design for a feature based MAR system utilising the STRUCTURAL TASK GRAPH pattern described in Section 5.5.1

84 (latitude and longitude) coordinates as reported by the GPS sensors, however this choice can make it difficult to perform visibility checks and distance calculations on nearby POIs. The alternative is to use the COMMON LOCATIONAL COORDINATES pattern to convert GPS coordinates to a common Cartesian system in which calculations are simplified. If choosing to use COMMON LOCATIONAL COORDINATES a further choice can also be made between using a local ENU coordinate system centred around the devices current position or a global ECEF coordinate system as described by Michel [2017].

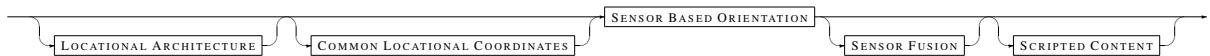
Incoming video frames can be converted to RGB using the *Acquire Frame* idiom and placed in a shared location using the SHARED RESOURCE pattern (Subsection 5.5.2). Even when not using *Acquire Frame* placing the frames in a shared location reduces subsequent expensive copy operations.

Next utilise SENSOR BASED ORIENTATION to use the device IMU sensors to obtain pose information. This usually entails making use of mobile device sensor APIs to obtain IMU data normally through event callbacks from the device OS. The sensor data can then optionally be used by a SENSOR FUSION component to filter and combine the various IMU device information in order to obtain improved accuracy. Finally if the artefact requires scripted content then add the SCRIPTED CONTENT

pattern.

Chapter 8 describes POI and other locationally keyed network data access, while Chapter 7 describes MAR rendering and interaction in general, most of which can also be applied to locational MAR. Note chapter 8 Subsection 8.4.10 also contains POI rendering data-access linked patterns specifically for locational MAR which was placed in the data access chapter as it is more tightly linked to networked data access.

A rail diagram for valid sequences through the locational patterns can then be defined as:



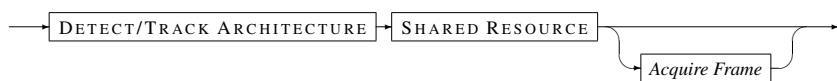
6.5.2 Vision Based MAR

DETECT/TRACK ARCHITECTURE in Subsection 6.4.12 describes in generic terms an overall architecture for vision based MAR (both feature based and SLAM) in terms of a task based decomposition for use in the **STRUCTURAL TASK GRAPH** pattern described in Chapter 5. The exact nature of the decomposition is left to the designer, as it is impossible to foresee all possibilities. Subsection 6.4.12 does contain some example architectures, the SLAM architecture for example is a high level conversion of ORB SLAM [Mur-Artal and Tardós, 2017] to a task based architecture. The following pattern languages for feature based and SLAM artefacts will assume a high level architectural design as the initial pattern in the sequence, although the designer can experiment and further improve the pattern to task mapping as the design unfolds.

6.5.2.1 Feature Based MAR

Implementing the **SHARED RESOURCE** pattern (Subsection 5.5.2) for sharing large resources such as camera frames is the first non-architectural step in the PL. The shared resource should provide atomic access to common data accessed concurrently by multiple tasks.

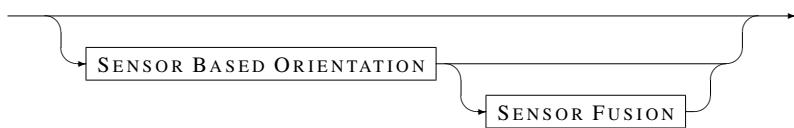
Acquiring camera frames to store as a shared resource is the next step. The use of the *Acquire Frame* (see Subsection 6.4.1) idiom (an OS level idiom for Android utilising which may also be implementable in iOS using computational shaders) provides an optimised solution for simultaneously obtaining colour and greyscale frames using the device GPU. If this approach is not feasible then more conventional image grabbing and conversion methods may be used, for example OpenCV provides APIs for converting YUV to RGB.



A decision also needs to be made as to whether the artefact will combine the use of vision based methods and IMU sensors for pose. If this is the case then two possibilities should be considered:

1. the IMU pose functions as a backup which will only be used when there are no trackable or detectable visual objects in the scene; and
2. the IMU pose is fused with the visual pose [Alatise and Hancke, 2017], which also implies that it reverts to 1 when no visual objects are available.

The latter case may benefit from implementing the fusion as a separate task. If IMU sensors are to be used then **SENSOR BASED ORIENTATION** (see Subsection 6.3.2) and optionally **SENSOR FUSION** (See Subsection 6.3.3) should be added.



While an explicit pattern-task mapping is not defined by **DETECT/TRACK ARCHITECTURE**, for feature based MAR placing object tracking and object detection in different tasks where tracking is assumed to be faster than detection is recommended. If this recommendation is followed then the object detection will be implemented as a separate task in which the processing time for detecting objects can exceed the frame rate. The object detection task extracts features using **FEATURE EXTRACTION** (see Subsection 6.4.2), and then **FEATURE MATCHING** (See Subsection 6.4.3) uses the features for matching and detection (Subsection 6.4.3 discusses various object detection techniques such as visual BoW). Once an object is detected it should be added to a data structure in the **SHARED RESOURCE** used for maintaining trackable objects along with the objects current bounding structure. Object data such as feature descriptors and other information linked to an object that the detector should search for should also be accessible to the detector. Exactly where the object detection information comes from is artefact dependent. An example might be a local cache queried by locational (GPS) coordinates from a global database as described in chapter 8, or in a small artefact all objects could be available on-device as part of the artefacts resources. Once detected the object in the local cache should be marked as detected so the detector does not search for it again until the tracker loses it.

The detector may also be required to implement an interface that restricts object detection to a small region of interest (ROI) instead of the entire image when:

- using a simple tracking by detection scheme where tracking is implemented by an object detector attempting to find the specific object being tracked in a small ROI near where it was last detected.

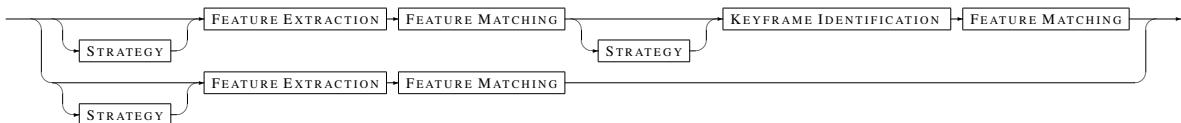
ted/tracked; or

- the tracker only returns a bounding box for a successfully tracked object, but the pose determination requires feature points.

As described in **FEATURE EXTRACTION** and **OBJECT TRACKING, STRATEGY** [see Gamma et al., 1995, page 315] can optionally be used to provide generic access to different detectors and trackers. This may be particularly important during development while experimenting to find optimal performance.



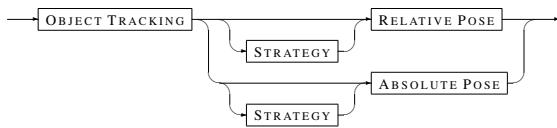
It is also possible to use the Keyframe concept from SLAM in a feature based MAR artefact. If using keyframes then doing object detection on keyframes reduces unnecessary object searches as frames subsequent to the keyframe have insignificant differences from the keyframe. This means that when using keyframes, the keyframe detection should happen before object detection, with the keyframe selection detector notifying the object detector when a new keyframe is available. The object detector should use the latest keyframe when it starts processing the next keyframe so the **SHARED RESOURCE** should have a double ended queue or stack for keyframes. **KEYFRAME IDENTIFICATION** (See Subsection 6.4.9) identifies the keyframes, and is dependent on **FEATURE EXTRACTION** and **FEATURE MATCHING** for features used to make the new keyframe decision. It may also optionally be dependent on **RELATIVE POSE** (See Subsection 6.4.5) if the criteria for keyframe change depends on the relative pose between the current keyframe and the new frame. The keyframe change policy can possibly also be more aggressive than for SLAM (in the sense that the keyframe does not need to be changed as frequently as is the case for SLAM) because the keyframe is not being used to build a 3D map of the surroundings (unless a hybrid SLAM/features system is being designed). Making the keyframe change criteria selectable at runtime can make experimentation in this regard easier. **KEYFRAME IDENTIFICATION** can run in its own task or in the main task with the tracker. The keyframe feature descriptors may also be stored with the keyframe so the detector does not have to redetect the features. **KEYFRAME IDENTIFICATION** can also be strategised in order to make the keyframe change policy runtime selectable.



OBJECT TRACKING (See Subsection 6.4.4) usually runs in the context of the main task and dictates the frame rate. Rendering (Chapter 7) for 3D APIs such as OpenGL which depend on rendering occurring in a special GUI thread, should also run in the main task or use a render queue monitored in the

main task (Vulkan/Metal do not have such a drawback). The tracker tracks objects from a data structure that the detector add detected objects to. When the tracker loses an object, it should remove it from the trackable data structure and mark the object as undetected in the object cache so the detector can start attempting to find it again.

Once the tracker has found a trackable object the pose can be obtained using either **RELATIVE POSE** or **ABSOLUTE POSE** (see Subsection 6.4.6) depending on the type of data in the object database. For 2D matches relative pose is used, while if 3D modelling data or point clouds for the object are available then **ABSOLUTE POSE** can be used instead. **STRATEGY** can also be applied to the pose algorithms, to allow different algorithms to be selected.



The result of assembling the individual PL sequences is shown in Figure 6.8.

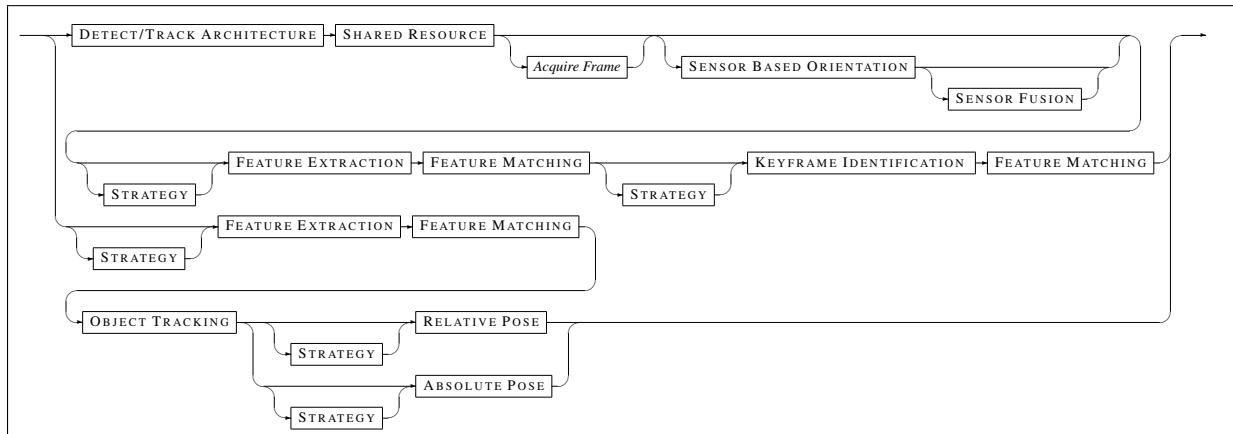
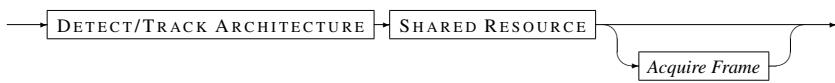


Figure 6.8: The pattern language for detection and tracking for feature based MAR.

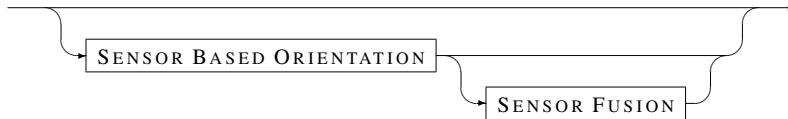
6.5.2.2 SLAM Based MAR

Similarly to the feature based MAR from the previous subsection (6.5.2.1), the PL starts by implementing **SHARED RESOURCE** and optionally *Acquire Frame* if feasible on the mobile OS in use. As in 6.5.2.1 the shared resource should provide atomic access to common data accessed concurrently by multiple tasks.



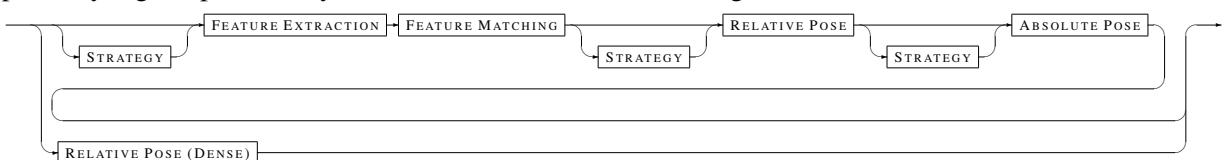
If IMU sensors are to be used in a visual-inertial SLAM system then **SENSOR BASED ORIENTATION** can be added to obtain sensor readings. The readings can then be combined with visual data in

the pose graph maintained by **BUNDLE ADJUSTMENT** (later versions of GTSAM support IMU data). It is also possible to use an extended Kalman filter for implementing **SENSOR FUSION**, although the filter based approach has lost popularity as optimisation through bundle adjustment became mainstream.



The tracker runs as the main task dictating the frame rate. For sparse SLAM (as epitomised by ORB-SLAM) the tracker applies **FEATURE EXTRACTION** to extract features and **FEATURE MATCHING** to match between keyframe and frame features based on their projected position derived from a constant velocity motion model. If the frame is successfully tracked then **RELATIVE POSE** is applied to the feature matches. Alternately if the number of correspondences is too low then visual BoW from **FEATURE MATCHING** is used to select the closest keyframe based on 3D **ABSOLUTE POSE** matching, and the process continues with the selected keyframe. The pose is then refined using **ABSOLUTE POSE** between the reprojected points and the 3D map. Finally **KEYFRAME IDENTIFICATION** using the matched features is performed, where **STRATEGY** can also be applied to **KEYFRAME IDENTIFICATION** to make the keyframe change policy runtime selectable.

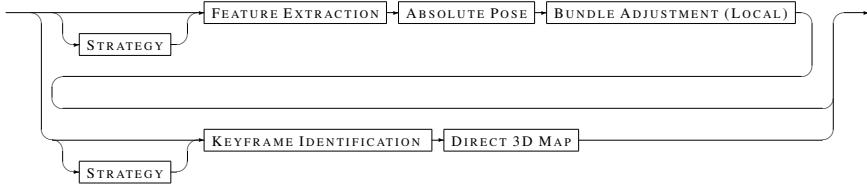
For semi-dense SLAM (as epitomised by LSD-SLAM), **FEATURE EXTRACTION** and **FEATURE MATCHING** are not used instead the pose is calculated from the optical flow between the keyframe and the frame as explained in the final paragraph of Subsection 6.4.5. While omitting the feature processing stage reduces the apparent complexity, these methods do iterative minimisation on all or many pixels (semi-dense methods only use pixels near high gradient areas) so the computational complexity is probably higher, particularly on modern devices which have high resolution camera frames.



The SLAM mapper task for feature based SLAM is notified of new keyframes by the tracker, and performs **FEATURE EXTRACTION** for providing features to add to the BoW database of keyframes. The 3D map is updated using **ABSOLUTE POSE** and triangulation, and finally local **BUNDLE ADJUSTMENT** is performed on all keyframes that are visible from the new keyframe.

For semi-dense SLAM the mapper task identifies new keyframes and initialises their depth maps or refines existing keyframe 3D depth maps using **DIRECT 3D MAP** (See Subsection 6.4.10). When a

new keyframe is identified, the current keyframe and its associated depth map is archived in the 3D map.



Finally the loop closure/optimisation task implements **LOOP CLOSURE** (see Subsection 6.4.11) and (global) **BUNDLE ADJUSTMENT**:



The complete PL collected from the individual sequences with the dense/semi-dense SLAM path shown in **light grey** above is shown in Figure 6.9.

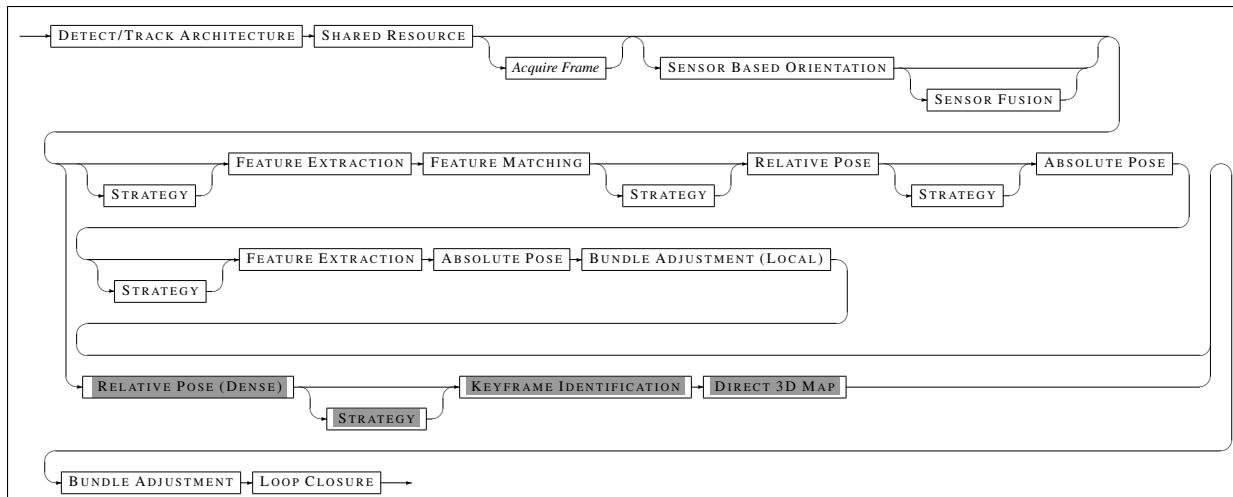


Figure 6.9: The pattern language for detection and tracking for SLAM based MAR (the path taken by dense/semi-dense SLAM is shown in **grey**).

6.6 Summary

Section 6.1 provided a brief introduction highlighting why detection and tracking is an important part of a MAR artefact. Section 6.2 provided some background on different coordinate systems used for detection and tracking in both locational and vision based MAR, with references to Appendix A.1 for greater depth. Section 6.3 described patterns for Locational MAR, although several of the sensor specific patterns from this section also apply to hybrid vision based MAR. Section 6.4 commenced by defining the scope of MAR systems for which the patterns applied, followed by a vocabulary of patterns for vision based MAR forming the bulk of the chapter. Finally in Section 6.5, Subsection 6.5.1 combined the locational patterns into a pattern language, followed by Subsection 6.5.2 doing the same for vision based MAR, while also

CHAPTER 6: DETECTION AND TRACKING

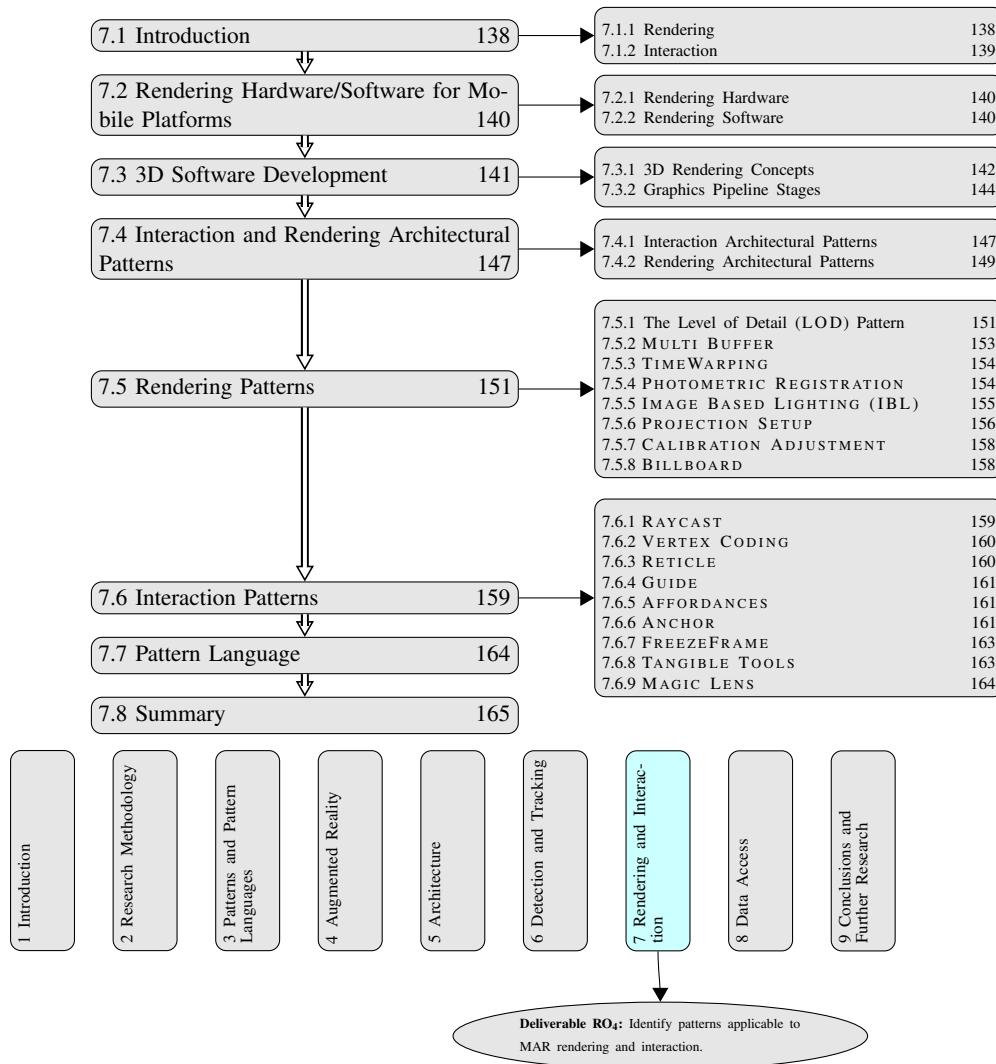
utilising some locational patterns for hybrid MAR systems.

The research output resulting from this chapter comprises:

1. The MAR detection and tracking patterns identified in Sections 6.3 and 6.4; and
2. The pattern language described in Section 6.5.

7

Rendering and Interaction



7.1 Introduction

The previous chapter discussed algorithms and techniques to detect and track real-world objects, and described the patterns that can be used in the design of AR detection and tracking components. This chapter will describe how the objects detected using the methods of the previous chapter are presented to the user by graphically interlacing real and virtual objects and also enabling interaction with the objects, that is finding a solution for RO₄, which states:

Identify patterns and a pattern language applicable to MAR rendering and interaction.

The applicable research process stage from Subsection 2.4.3 is:

Analyse & Design the System.

The output will be:

1. A set of patterns applicable to MAR interaction and rendering; and
2. A pattern language built on these patterns.

This section will provide an overview of the main topics of this chapter, namely MAR rendering and interaction. The following two sections will provide more detail of 3D rendering from both a hardware and software perspective which may be required for an understanding of the rendering patterns described later. The rest of the chapter will provide a detailed presentation of design patterns for both components.

The theoretical background of MAR rendering and presentation has already been covered in Subsection 4.4.10, while that of interaction was covered in Subsection 4.4.9. The following subsections highlight the importance of these components in the overall scheme of MAR.

7.1.1 Rendering

The rendering component in a MAR system is tasked with producing the final display which the end user sees and interacts with. It is therefore crucial that the output that is produced convincingly blends real-world objects and virtual content as the human visual system can be very sensitive to small inconsistencies. In order to convincingly place virtual entities in the real-world, the renderer needs to include various visual cues including:

- Correct perspective, that is farther away objects exhibit a decrease in apparent size;
- Correct shading and illumination based on the position of real light sources, if possible;
- Shadows that are as realistic as possible, as the human visual system appears to be especially reliant on shadows [Kersten et al., 1996], although Gooch et al. [1999] show that the visual system can also utilise less accurate shadow renditions correctly;

- Motion parallax where (virtual) objects that are further away appear to move slower than real or virtual objects that are nearby, and
- Occlusion where nearby objects can cover or partially cover more distant objects. Occlusion remains a particularly difficult problem in an AR context, unless a full 3D map of the surrounding exist (and the occluding objects are stationary).

Section 4.4.9 in Chapter 4 describe these cues in more detail.

In addition to visual accuracy, the renderer also has to deal with significant real-time constraints. In particular latency between the current camera position and what appears on the rendered frame can reduce the perceived reality of the AR experience, particularly as the human visual system is capable of detecting latencies as small as 5–10 milliseconds. Given the amount of processing that is required to capture a camera frame, track objects in the frame and calculate the camera pose using the limited processing power available from mobile devices, achieving such low latencies is extremely difficult¹. Fortunately, unlike for VR where latency can cause motion sickness, some compromises are possible for camera based MAR, although achieving even 20–30ms latencies can be quite difficult. The latency problem is further complicated by a lag time between the GPU rendering the frame to a framebuffer (which is what most GPU profilers report) and the frame being physically displayed as the output first has to be rasterised and then drawn line by line to the display.

As the renderer component is at the end of the AR flow graph/pipeline (see Chapter 5 Section 5.4), it is sensitive to both the throughput and accuracy of preceding components. If the tracking data and pose from the detection/tracking components are inaccurate then the renderer output is compromised, and if this data is out of date it will be perceived as latency. The underlying architecture of the MAR system as a whole can therefore also affect the final render quality.

7.1.2 Interaction

While the rendering component presents the AR world to the end user, a means of allowing the user to interact with the AR world may also be required. Some theoretical background for AR and MAR interactivity was previously discussed in Section 4.4.9, therefore this chapter will concentrate on patterns for AR interaction. As interaction modifies the scene being rendered the rendering and interaction components dovetail architecturally.

As user interaction could also be seen to include movement of the device, the interaction component may also handle sensor data pertinent to device movement. This could include fusion of sensor data from

¹Alternative designs that perform detection and tracking on a server incur possibly even greater network latency.

different sensors to improve the sensor accuracy. See the Pattern Language section (Section 7.8) for more detail on assigning responsibilities with regard to sensor handing and fusion.

7.2 Rendering Hardware/Software for Mobile Platforms

7.2.1 Rendering Hardware

Early desktop 3D graphics development was accomplished entirely in software, however 3D rendering is CPU intensive and frequently trivially parallelisable [Crockett, 1997]. Early hardware support came in the form of line and triangle rasterisation implemented in graphics cards, followed by early GPUs from NVidia which supported a complete hard coded graphics pipeline. Later GPUs evolved to support programmable shaders that allowed developers to customise the hardware rendering using shader programs running simultaneously in multiple GPU processor cores. In order to support the programmable shader model, GPU cores evolved from being fixed function processing units into simplified but massively parallel CPUs which could be allocated for use anywhere in the graphics pipeline or used for non-graphics work such as calculating Fourier transforms or powering neural networks.

Employing GPUs on mobile devices can be problematic given that GPU cores are both power intensive and generate large amounts of heat, and the cards themselves are frequently heavier than the motherboards of the desktop computers in which they are installed. This has not stopped hardware vendors from designing limited GPU versions designed to use less power and produce less heat, however these GPUs are limited both in terms of the number of cores available and the performance of the cores.

7.2.2 Rendering Software

As rendering hardware evolved, software APIs capable of utilising the hardware also evolved in parallel. OpenGL was one of the earliest such API with its first release in 1992 as a multi-platform open specification for a 2D and 3D API which could be implemented by hardware vendors [Scali's OpenBlog, 2012]. Some hardware vendors also introduced proprietary APIs, for example 3dfx's Glide which was widely used in games in the early- to mid-nineties before fading into obscurity. OpenGL's main rival however, was Microsoft's Direct3D which was OS proprietary but could be implemented by various hardware vendors. Direct3D was introduced in 1996 for Windows 95 [Kerin, 2016].

The constant improvements to rendering hardware resulted in new versions of the software APIs which tracked these improvements². Early changes included the switch from hard coded fixed function shading to programmable shaders, while later changes included support for new types of programmable

²OpenGL also supports vendor supplied extensions which can be queried for availability and used if available.

shaders applicable at different points within the graphics pipeline as well as general purpose computing “shaders”.

OpenGL and Direct3D are high level APIs which attempt to reduce the complexity of 3D development by hiding much of the complexity such as tracking state, managing memory and synchronising tasks in the graphics driver. This approach has made developers lives easier but at a performance cost for applications. This cost was exacerbated as the number of both CPU and GPU cores increased on modern hardware, leading to the use of multithreading to improve performance by fully utilising the available hardware. The high level 3D API drivers proved to be mostly incompatible with a multithreaded programming model due to having to maintain state for multiple graphics objects, and internal driver management of synchronisation meant developers could not optimally utilise synchronisation between CPU threads and GPU cores for control and memory transfer. This incompatibility led to the introduction of new lower-level API, of which the main contenders are Vulkan [Sellers and Kessenich, 2016], an OpenGL descendent from Khronos the owner of the OpenGL specification, and Metal, an Apple specific low level API [Apple Developer, 2018]. In addition Direct3D 12 [Luna, 2016], the latest incarnation of Direct3D, has changed the Direct3D programming model to support low-level development equivalent to Vulkan and Metal.

As OpenGL is an open specification it rapidly established itself as the only 3D API available for mobile devices³. Desktop OpenGL proved to be much too large with many deprecated features such as the pre-shader immediate mode, which are still in place to support legacy software, to be ported to mobile devices so a slimmer embedded version of OpenGL known as OpenGL ES was ported by vendors such as NVidia and Qualcomm instead. While having a smaller footprint, OpenGL ES is still a high level API, so more recently Vulkan has been ported to Android while Metal has become available for iOS⁴.

7.3 3D Software Development

Development of 3D software is centred around a graphics pipeline which begins with 3D models defined in a local coordinate system and ends with a 2D scene representation of all visible objects represented in 2D, as seen through a virtual camera in a hardware specific pixel based coordinate system. In an AR setting the virtual camera and the real camera must be coincident. Before describing the graphics pipeline, some of the concepts used in 3D rendering will be briefly described.

³Although Direct3D did appear, albeit briefly, in the ill-fated Windows Phone OS.

⁴It is also possible to use Vulkan on Apple devices by utilising the open source MoltenVK emulator which translates Metal API calls to Vulkan.

7.3.1 3D Rendering Concepts

This subsection provides a brief summary of 3D rendering concepts. See Akenine-Moller et al. [2018] for a comprehensive API agnostic introduction, Kessenich et al. [2016] for OpenGL and Sellers and Kessenich [2016] for Vulkan.

7.3.1.1 Geometry

The geometric vertex is the lowest level primitive in 3D graphics. Models are built by combining vertices into higher level shapes such as lines, triangles or quadrilaterals and then building 3D models out of these shapes. Image textures are frequently also mapped onto the shapes. Complex 3D models are usually created using 3D modelling software such as Blender or Maya. Simpler models such as spheres can be generated programmatically using parametric equation, while even simpler models such as cubes can be created by hand. Other low level primitives include colour, surface normal vectors and texture coordinates used to map textures onto shapes. These primitives are specified per vertex, and are normally combined with the vertex when the model is stored.

The vertices in these 3D models are specified in a local coordinate system specific to the model, for example, the coordinate system origin might lie in the centre of the model. The 3D rendering code has to define a 3D world coordinate system and then transform all the models that are required to be rendered into their current positions in the world coordinate system. These transforms are typically affine transform such as translation, rotation and scaling and are specified as homogenous matrices although internally quaternions can also be used for rotation⁵. As homogenous matrices are used, it is possible to group multiple transformations into a single matrix by multiplying the individual transform matrices together.

In order for the scene to be displayed on a 2D surface such as the screen of a device, the world coordinate system representing the scene must be projected onto a 2D plane in a visually convincing way. In reality, the projection does not project to a plane but instead to a unit cube with the new coordinates known as Normalised Device Coordinates (NDC) [Kessenich et al., 2016]. The most common projection transform in 3D graphics is perspective projection, a technique first discovered by Italian Renaissance era artists, in which objects seem smaller as their distance from the camera increases and parallel lines converge at infinity. Another frequently used projection is orthographic projection where, unlike perspective projection, parallel lines remain parallel and object sizes are unchanged. The GPU hardware uses the

⁵See Chapter 6 for a description of the mathematics of homogenous matrices and quaternions.

NDC to clip points that are not within the cube and then converts the NDC coordinates to 2D viewport coordinates (it is possible to have multiple viewports in a display window) and then finally converts the viewport coordinates to pixel coordinates for display.

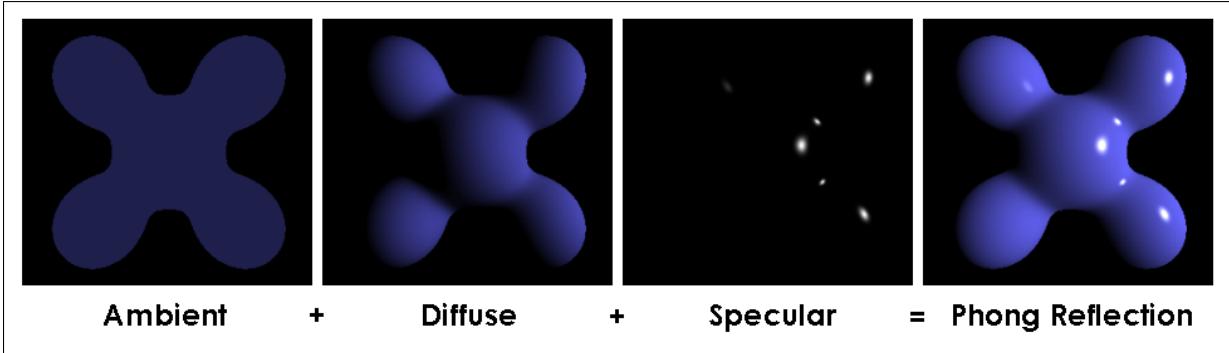


Figure 7.1: Lighting contributions from different components of the Phong shading model (Wikimedia Commons, Brad Smith [CC BY-SA 3.0]).

7.3.1.2 Shading

The appearance of the rendered objects is primarily defined by what is known as the shading model which calculates colours intensity based on the direction of light sources in the scene and the camera position relative to the model. An example of a simple but widely used shading model is the Phong shading model [Phong, 1975] (with further optimisations by Blinn [1977]) which calculates shade and lighting based on diffuse reflection or scattering and specular reflection or shininess. Both reflection types are dependent on position of the light sources, while specular reflection is also dependent on the camera position. A background or ambient lighting term is also added to the result of the lighting calculations to obtain a final result (see Figure 7.1). The simplest lighting model applies only a diffuse and ambient term calculated per-vertex rather than per pixel, however with such a simple model vertices become apparent and the objects do not appear smooth. More complex models include specular effects and are calculated at pixel level.

More advanced physics based shading models, also known as physically based rendering, have also become possible as the underlying rendering hardware has became more powerful. Such models are commonly defined by a Bidirectional Scattering Distribution Function (BSDF) which still uses the diffuse and specular concepts, but models surfaces as consisting of small randomly aligned faces that reflect light at different angles. The BSDF also accounts for shadowing and masking of one face by another. The extent to which the faces are angled towards the camera are specified by a roughness parameter. A full description of physically based rendering is beyond the scope of this document, see Akenine-Moller et al.

[2018] and Guy and Agopian [2018] for more detail.

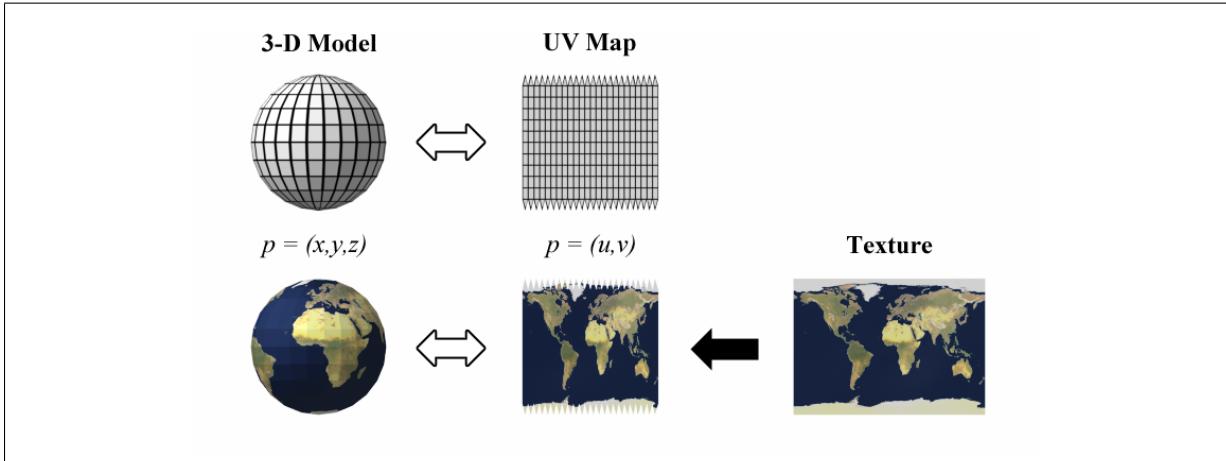


Figure 7.2: Texturing a 3D globe using uv coordinates (Wikimedia Commons, Tschmits [CC BY-SA 3.0]).

7.3.1.3 Texturing

While combining geometries composed of vertices with shading provides a convincing illusion of three dimensions, it cannot fully simulate the structure of materials in the real-world. In order to do so, one approach is to use images of material stitched onto the vertices with the colours from the image replacing RGB colours that would be specified for un-textured models. The prototypical example would be a brick wall where the bricks are represented by a repeated image of a brick and neighbouring mortar. Other forms of texturing exist which performs other types of per-pixels modifications, for example bump mapping makes small alterations to surface normals interpolated across vertices in order to provide surface roughness [Akenine-Moller et al., 2018, Chapter 6], as is also provided by physically based rendering, but at a lower computational cost.

In order to implement texturing, each vertex also has texture coordinates, typically known as uv coordinates, specified in addition to its positional coordinates. The texture coordinates differ from regular coordinates in that they vary between 0 and 1. The GPU interpolates a u,v pair for a given pixel based on the texture coordinates specified for a given vertex, and then the uv pair is mapped to a location in the image and the colour at that location in the image is then applied to the pixel location (see Figure 7.2).

7.3.2 Graphics Pipeline Stages

The graphic pipeline comprises several programmable stages, although internally these stages are further subdivided and parallelised. Additionally a number of hardware level fixed function stages link the

programmable stages. The main programmable stages comprise the application stage, the vertex processing stage, the tessellation and geometry shader stage and finally fragment processing at pixel level. For example Figure 7.3 provides a simplified view of the OpenGL pipeline.

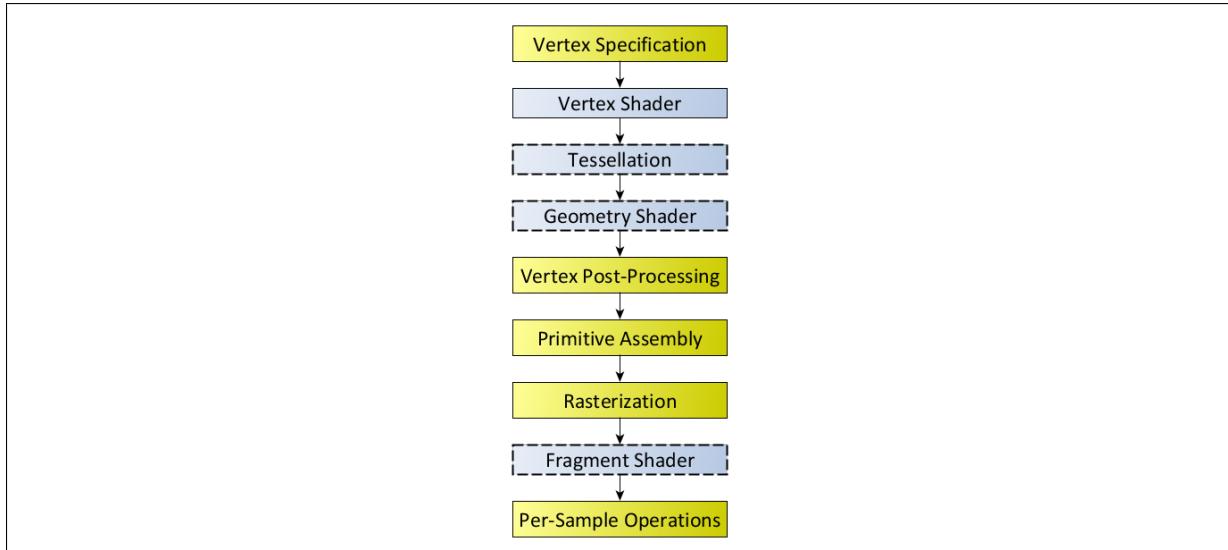


Figure 7.3: Simplified OpenGL graphics pipeline (Khronos <https://www.khronos.org/opengl/wiki/File:RenderingPipeline.png>).

7.3.2.1 Application Stage

The application stage is directly controlled by the developer in software running on the computer CPU. All the tasks required to select the 3D models, extract the model geometry specified as vertices in triangles, quadrilaterals or lines to be rendered and place them in a common coordinate system occur here. The vertices, colours, normals and texture coordinates are loaded into CPU memory in order to be moved to GPU memory, while the transforms required to position the models are specified as matrices and are placed in CPU memory for subsequent transfer to the GPU. The texture images themselves are also placed in CPU memory for transfer to the GPU.

Various other application level calculations also occur in this stage, for example culling of 3D points that would not be visible based on a global model such as a scene graph, sorting the order of rendering where transparency may occur, collision detection between 3D objects, loading textures (images) into CPU memory and ray casting for 3D user interaction. As the shader programming model improved however, many tasks such as shadow generation that used to be performed in the application stage have gradually moved to the GPU. It is also during this stage that the value of multithreading friendly APIs such as Vulkan becomes apparent as these APIs make it possible to perform these operations on different

CPU threads which can individually interact with the GPU, albeit at the cost of greater complexity.

7.3.2.2 Vertex Processing Stage

The vertex processing stage is the first stage executed on the GPU. The vertices are transferred to GPU memory if they are not already resident from a previous render cycle, and these vertices are then streamed to developer created shader programs for transformation into projective space using transformation matrices uploaded to GPU memory. For cases where per-vertex lighting is performed the lighting calculations are also done in the shader, although applications requiring more realistic lighting effects delay lighting calculation until later. If surface normals are supplied then they may also be transformed to projective space for later lighting calculations or used in the per-vertex lighting calculations. The shaders run in parallel on potentially hundreds of different GPU cores with each shader program processing a single vertex in a Single Instruction Multiple Data fashion.

7.3.2.3 Tesselation/Geometry Stages

Modern GPUs optionally support a tesselation shader which can be used to subdivide pixel patches, and a geometry shader which takes the output from a vertex or tesselation shader and edits it by either outputting new vertices not specified in the application stage or outputting no vertices as a method of editing the vertex shader output.

7.3.2.4 Intermediate Fixed Function Stages

The GPU provides several post vertex processing stages which are not programmable and are used to transform and rasterise the projective space vertices to screen space fragments, logically representing a geometry level object such as a triangle as a series of pixels in screen space, with interpolated properties such as colours where necessary.

7.3.2.5 Fragment Processing Stage

Fragment Processing is the final programmable stage in the pipeline. Texture colour values are extracted from texture images using a GPU texture sampler where texturing is used. If texturing is not in use for the object being rendered, then a colour value interpolated from the per-vertex colour supplied in the vertex shader is used. Per-pixel shading calculation is carried out using the selected colour value combined with an interpolated normal and output to the final fixed function GPU display stage (the fragment shader can also elect to discard its output).

7.3.2.6 Final Display

Depth tests using a Z-buffer are used to compare pixel depths to decide which pixels overlay others. If enabled then Scissor and stencil tests are also used to eliminate pixels. Finally blending and anti-aliasing are carried out before writing the pixel to the display.

7.4 Interaction and Rendering Architectural Patterns

This section describes patterns applicable to MAR rendering/interaction architectural patterns for the combined renderer/interaction components.

7.4.1 Interaction Architectural Patterns

7.4.1.1 MVC

Context: Defining an architecture for MAR interaction and rendering.

Problem:

Rendering and interaction are parts of a feedback loop which need to be architecturally integrated.

Solution:

MVC is a long established interaction pattern which certainly cannot be described as a proto-pattern. The MVC pattern is really a compound pattern, which is composed from several classic patterns such as **OBSERVER**, **STRATEGY** and **COMPOSITE**. The MVC acronym stands for Model-View-Controller, with the Model being responsible for the state of a displayable logical entity, the View displaying a representation of the model and the Controller responding to user interaction and updating the model. MVC has been somewhat exhaustively documented [Krasner and Pope, 1988; Buschmann et al., 1996], therefore the pattern itself will not be described in detail, instead its use in an AR scenario will be expanded on.

A consequence of using MVC for MAR is the liberal use of events for updates of all kinds, not just user interaction. This can have performance implications unless the effects of event processing on the AR system as a whole is not taken into consideration during design. When used within the **STRUCTURAL TASK GRAPH** architecture (see Section 5.5.1), one approach would be to incorporate the events as data streams between nodes in the task graph. Batching events where possible may also be helpful. The model may also need to cache data from external data sources when the object it represents is based on externally accessed data, for example a map from an Internet map service as described in Chapter 8.

7.4.1.2 MVC Variants

MVC was designed for use within a traditional 2D desktop GUI context which does not always transfer seamlessly to 3D AR interaction. The architecture circumscribed by MVC requiring separate models and controllers for renderable items can also lead to a plethora of small objects which may have both performance and concurrency side effects.

Variations of MVC addressing some of these issues have also appeared. Multimodal Architecture and Interfaces (MMI) is a W3C open standard proposal [W3C Multimodal Interaction Working Group, 2017], which modifies MVC to support multimodal input and distributed MVC via an underlying interaction manager, although it is primarily directed towards web based use allowing “input via speech, handwriting, touchscreens and keystrokes, with output presented via displays, pre-recorded and synthetic speech, audio” [W3C Multimodal Interaction Working Group, 2017, MMI Mission Statement].

Hesenius and Gruhn [2013] propose Model-View-Interaction-Controller (MVIC), a variation of MVC that adds a dedicated interaction component that handles interaction including data from sensors. The interaction component is multimodal as it supports multiple interaction handlers within the component. This also allows new functionality or sensors to be added without altering existing functionality.

7.4.1.3 DPI

Context: Defining an architecture for MAR interaction and rendering.

Problem:

Rendering and interaction are parts of a feedback loop which need to be architecturally integrated.

Solution:

DPI or Data-Processing-Interaction pattern [Bi et al., 2011] provides a presentation/interaction architecture pattern for 3D graphics and simulation specific applications. As this is a new pattern without many known uses, it should be regarded as a meta-pattern. DPI is a specialisation of the LAYERS architectural pattern, and comprises three layers of interaction, processing and data. These layers will be described below within the context of a STRUCTURAL TASK GRAPH architecture.

The data layer, as its name suggests, holds all the data used for rendering. In a STRUCTURAL TASK GRAPH architecture, the data could be accessed through the SHARED RESOURCE, either as individual components per renderable or as a monolithic data container for all data. As with the MVC model, the data layer may need to connect to external sources to receive the data. When the data is accessed via individual components, which is preferable at least for more complex items, a common

interface for data access can be exposed using EXPLICIT INTERFACE. The interaction layer detects and interprets user interactions, filters them and forwards them to the processing layer. The processing layer receives interaction events either directly from the interaction layer or via a task graph stream. It can access and update the data layer before invoking the main renderer component with updated view of the renderable item.

7.4.2 Rendering Architectural Patterns

7.4.2.1 The ABSTRACT RENDERER Pattern

Context: Enable support and runtime selection of multiple renderers within a MAR artefact.

Problem: It may be desirable for a MAR artefact to support multiple renderer implementations. Examples include:

- Providing support for more than one 3D API such as OpenGL or Vulkan;
- Providing support for more than one 3D engine such as Unreal or Unity;
- Providing support for renderers with different levels of abstraction such as higher level scene graph based renderers or low level abstraction free renderers providing maximal performance.

Solution: Use STRATEGY [see Gamma et al., 1995, page 315] to define a common interface that renderers for the artefact are required to implement. The specific renderer to use can then be created at runtime using FACTORY METHOD [see Gamma et al., 1995, page 107].

The renderer implementation plays the *Concrete Strategy* role as an implementor of the *Abstract Strategy* role. The render node from the STRUCTURAL TASK GRAPH Pattern (see Subsection 5.5.1) plays the *Context* role when invoking a renderer through the *Abstract Strategy* role interface.

The concrete implementation of some APIs such as OpenGL are only allowed to render in the context of an application main thread, while others, for example Vulkan, support parallelisation. In order to support both types, it may be necessary to provide an interface which supports both direct rendering and queued rendering with a separate single thread render method that is invoked in the context of the main thread and which renders de-queued frames.

7.4.2.2 The SCENE GRAPH Pattern

Context: High level representation of relative spatial positions of virtual objects in an AR scene defining an architecture for the renderer.

Problem: Virtual objects in an AR scene need to be placed in a world coordinate system. This requires every objects position to be updated every frame, with a transformation matrix needing to be calculated for every object. The result is complex and hard to maintain code which is potentially less efficient.

Solution: Describe a 3D scene as a tree where leaf nodes represent renderable objects and intermediate level nodes represent positions, transformations or properties that are applied to all descendants of the node. This tree, which is frequently implemented a DAG, is known as a scene graph. The nodes in the tree make use of the COMPOSITE [see Gamma et al., 1995, page 163] pattern, in order to allow intermediate level nodes to compose children nodes, which in turn can also be compositions or leaf nodes representing actual objects. Intermediate node types include affine transforms, custom transforms, lighting and material nodes. For example, in the excerpt from a scene graph in Figure 7.4, the light and the two geometry nodes are all transformed by the encapsulating transform, while only the material from the “Material 1” node is used for the “Geometry 1” renderable while the material from the “Material 2” node is used for the “Geometry 2” node.

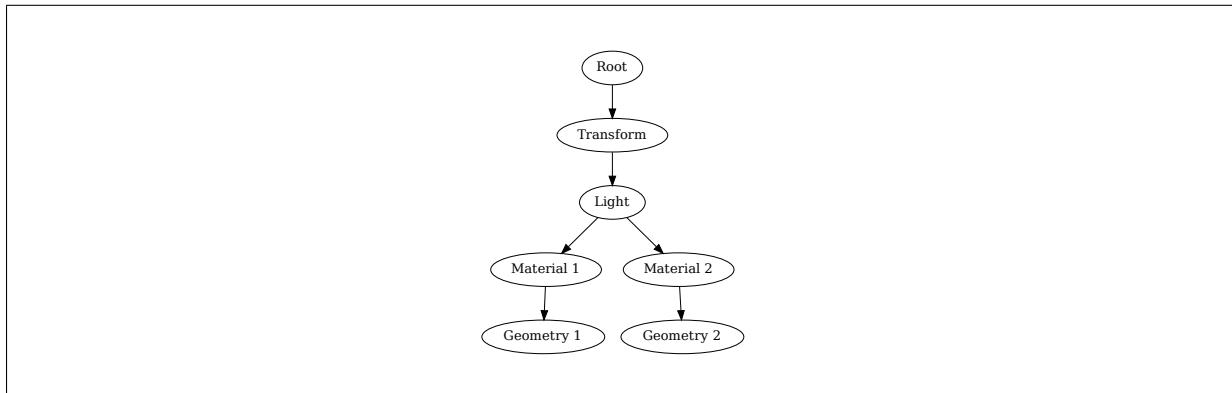


Figure 7.4: Example of a scene graph.

Operations on the scene graph are carried out by utilising the VISITOR pattern and traversing the graph having each node invoke the appropriate *Visitor*. For example there would usually be a render *Visitor* and possibly other visitor types such as for visible node culling.

Nodes in the scene graph can be *Composite* with descendants nodes or leaf *Nodes* from which the *Composite* nodes are descended. All *Nodes* can accept *Visitors* as described in the VISITOR pattern. The scene graph may include a front-end *Facade* as described in FACADE to facilitate operations on the scene graph.

Some high level 3D engines such as Unity and UnReal have built in scene graphs. Several open source scene graphs are available, for example Open Scene Graph (OSG) [Burns and Osfield, 2000], a

cross platform OpenGL scene graph.

Language specific optimisations for the **VISITOR** pattern also exist. For example, C++ can use a combination of templates and the **CURIOUSLY RECURRING TEMPLATE PATTERN** [Alexandrescu, 2001] to provide a generic *Visitor*.

Coelho et al. [2004] describe an AR specific scene graph that attempts to take into account pose registration errors that result in mispositioned virtual objects, by associating a covariance matrix with each transformation node. The transformation is then seen as a probability distribution around a mean of the specified transformation parameter value. A Level of Error (LOE) scene graph node can be used to select different augmentations based on the error probabilities. Coelho et al. also provide a open source implementation derived from OSG.

Another variation of the scene graph for AR is provided by Echtler et al. [2008]. Echtler et al. [2008] propose a variation of the scene graph known as a Spatial Relation Graph (SRG). Nodes in a SRG represent coordinate systems and edges are transformations between these coordinate systems. Nodes include different camera coordinates systems, screen coordinate systems as well as nodes encapsulating a normal scene graph which is rendered in the given coordinate system.

7.5 Rendering Patterns

This section will introduce rendering patterns within the context of a PL for MAR. Note that two further locational data access specific rendering patterns are described in Chapter 8 Section 8.4.10, as their description utilises data access patterns.

7.5.1 The Level of Detail (L OD) Pattern

Context: Vary the level of detail for virtual objects in a scene based on some criteria in order to improve rendering throughput.

Problem: Rendering detailed 3D objects is computationally expensive, and when combined with AR specific processes such as detection and tracking, can lead to poor performance and high latency.

Solution: Reducing the complexity of 3D models, that is, rendering models consisting of fewer geometric elements such as triangles, accelerates rendering and decreases the number of GPU cores required per model. The most common criteria used to select which models are to be displayed with less detail is Euclidean distance from camera in the 3D Camera coordinate system. The rationale for using distance as a criteria being that objects that are further away will be projected to a smaller screen space and the extra detail from complex geometry will be scaled down to the point of invisibility, therefore rendering a

reduced complexity model incurs no cost but does result in a performance gain. A related criteria is using the size of the 3D projection of the model or a bounding box of the model, with the same rationale as that given for distance from camera.

Other criteria that are less frequently used include:

- The angular velocity of a moving model, with the rationale that faster moving objects cannot be discerned at the same level of detail as stationary or slow moving ones; and
- Peripherally placed models are also candidates for detail removal as the humans eye focusses most attention on items in the centre of the field of vision (this assumes the user is looking at the centre of the display or a eye or head tracking system is in use).

LOD can be implemented in three ways:

- Discrete LOD where multiple models at different levels of complexity are separately created using modelling software and stored as assets to be used by the rendering software. At the point of switching between models a form of blending between the models has to be performed to mask the switch from the viewer. Due to its simplicity and efficiency this is still the most common form of LOD in everyday use;
- Progressive or continuous LOD which is based on the concept of progressive meshes introduced by Hoppe [1996]. The progressive mesh encodes the fully detailed model as a specialised form of a triangle mesh and then applies a decimation algorithm to reduce the number of triangles continuously in real-time but at a relatively high memory cost; and
- View dependent LOD which is based on progressive LOD, but optimises the selection of the detail level using the current view parameters. It is however more CPU intensive and incurs an even higher memory cost than progressive LOD.

LOD can be provided as a node in a scene graph which switches between different models based on one of the previously mentioned criteria. Open Scene Graph (see Subsection 7.4.2.2), for example has a LOD node for discrete LOD based on distance to camera.

A simplified form of LOD can be provided using only textures. Such textures have an extra coordinate in addition to the standard uv texture coordinates. Multiple textures corresponding to different levels of detail are uploaded as a single texture with the extra coordinate (sometimes known as w) used to select the texture to display. This technique is known as mipmapping, and can also be used in conjunction with standard LOD by rendering models at a higher level of detail and switching to textures for lower detail. Mipmapping is supported in hardware by all modern GPUs.

A full treatment of LOD is beyond the scope of this document, Luebke et al. [2003] provide a textbook dedicated solely to this topic, while Cudworth [2018] describe its use in a mixed reality setting.

7.5.2 MULTI BUFFER

Context: Provide an atomic update view for readers of items that are updated sequentially by multiple writers, possibly in parallel.

Problem: Renderers update a GPU memory buffer called a framebuffer with the final binary representation of the pixels that the display hardware converts to what is seen on the screen. The renderer updates are done in parallel, and sometimes the same area can be written to more than once at different times, for example when implementing transparency effects. The hardware display conversion however, is done sequentially scan line by scan line, and if updates occur to the framebuffer while the display hardware is updating the screen a visual effect known as tearing occurs. Alternately the framebuffer can be locked while the hardware updates the screen (known as VSync) which compromises performance.

There are also non graphics related problem areas addressed by this pattern. For example, it may be necessary to have several threads update an object, but only accept the changes if all are correct. An example from the UI domain is efficiently providing a cancel option on a dialog with numerous options representing application properties.

Solution: Use multiple framebuffers and display from one while rendering to the others. Double buffering provides a single render buffer to write to while the other buffer is being displayed by the hardware, with the hardware using page flipping to switch between the framebuffers⁶. Triple buffering allows for two write buffers while the third is being rendered from, while quad buffering is double buffering for stereo displays such as AR glasses.

Non-graphic problems that MULTI BUFFER addresses normally require only double buffering. For example an object the client has a reference to could be copied, the changes made and if the changes are affected then the clients reference could be swapped atomically and the old copy discarded or used for future buffering.

OpenGL usually provides a high level double buffering interface using OS windowing specific calls to switch buffers. Later versions of OpenGL and OpenGL ES also support lower level FrameBuffer objects with a default framebuffer which is always used for rendering by the higher level methods, however extra framebuffers can be created, rendered to and switched to by the rendering software.

⁶In pre-GPU days a CPU RAM buffer would be written to and copied to video display memory when ready, using VSync to synchronise the copy.

Vulkan has no concept of a default framebuffer, instead it uses a swap chain containing a queue of framebuffers that need to be created and attached to the swap chain by the rendering software. The software can then render to framebuffers, and submit framebuffers for physical rendering. The rendering software has to perform all synchronisation on the GPU using Vulkan fence and semaphore objects. This is in contrast to the OpenGL framebuffer rendering which hides the synchronisation complexity from the software. The low level Vulkan approach requires greater sophistication from the developer, but allows greater flexibility and optimisability.

In an AR architecture defined using the **STRUCTURAL TASK GRAPH** pattern as described in Section 5.5.1, a form of multi-buffering can also be used in addition to the renderer GPU buffering. This approach uses render tokens as a form of buffer which are drawn from a fixed size queue to throttle the task graph stream, which would otherwise not be limited by the maximum number of frames that can be pre-rendered.

7.5.3 TIMEWARPING

Context: Update pose while rendering.

Problem: The pose for a frame may have changed.

Solution: At end of rendering to a framebuffer, retrieve the latest pose and, if the change is large enough, apply a rotational warp (known as timewarping). While this is not as accurate as rendering with the correct pose, for small pose changes it does usually provide a better visual result with less jitter.

In an AR architecture defined using the **STRUCTURAL TASK GRAPH** using the **SHARED RESOURCE** pattern (Section 5.5.2) in order to minimise any performance penalties, the pose can be updated to GPU memory at the same time the shared resource pose is updated. The warp can then be done in a GPU shader.

7.5.4 PHOTOMETRIC REGISTRATION

Context: Direction of primary light source is required for 3D shading and shadows.

Problem: In order to create convincing augmentations, virtual objects must appear to share the same lighting characteristics. For example shadows for real objects should appear to be cast in approximately the same direction as real objects.

Solution: Detecting light origins and direction from a camera image or multiple images precisely is a complex problem which is currently not fully solved. One approach is to place one or more special

light probes such as Lambertian spheres into a location and then use the known lighting properties of the probe to estimate the directions of multiple light sources [Zhang and Yang, 2001; Aittala, 2010]. Another approach is to use known objects in the scene whose lighting characteristics have been previously recorded (faces [Knorr and Kurz, 2014] and eyeballs [Tsumura et al., 2003] for example), and use this information to infer the lighting. Methods using 3D point clouds such as Boom et al. [2015] have also emerged in recent years as RGBD devices became more common. Lopez-Moreno et al. [2013] provide a method based on using surface normal vectors on a contour in the image and iteratively adding light sources to the scene to find the best match to the actual lighting.

As light discovery is still an open research topic and the solutions given are not necessarily usable in real-time, a few lower-cost alternatives are described. Most modern mobile devices have an ambient light sensor which returns a scalar value for the light intensity for the front face of the device which is used to change the brightness of the display. These values could be combined with vision or sensor data to find the approximate direction of maximum lighting, although it is dependent on the user moving the device around the scene. Soulier et al. [2017] also describe a DIY approach by building a low cost light sensor utilising an array of photo-resistors which can be interfaced with the device or a desktop server.

For outdoor applications it is possible to calculate the position of the sun given the GPS location and time of day⁷. This information can be combined with output from the ambient light sensor to determine the strength of illumination [Barreira et al., 2013]. Clements and Zakhor [2014, page 3367] take this method a step further by “constructing a model of the sun-earth system to determine all shadows possible at a given approximate latitude”.

As a last resort or a first approximation before more information becomes available, a light source can be placed in the top left of the scene as, in the absence of other cues, the human visual system appears to assume lighting from above-left [Mamassian and Goutcher, 2001].

7.5.5 IMAGE BASED LIGHTING (IBL)

Context: Precomputed lighting for known locations.

Problem: Correct lighting by real-time rendering techniques is difficult to achieve and, even if technically feasible, is also computationally expensive.

Solution: Use precomputed lighting stored in an environment map. This approach dates back to 1976 when Blinn and Newell [1976] introduced spherical radiosity mapping. Modern GPU implementations

⁷<http://pveducation.org/pvcdrom/properties-of-sunlight/suns-position>

have replaced spherical maps with cube mapping performed in GPU hardware. Cube maps, introduced by Greene [1986], have the advantage of being view independent, and the projection function used to look up a radiosity value is more uniform than any of the spherical maps which tend to cluster values.

The textures themselves are also now stored as High Dynamic Range (HDR) images⁸ to allow for more realistic lighting. There are several ways of creating the maps:

- Using spherical light probes as described in Subsection 7.5.4 and illustrated in Figure 7.5;
- Use of fisheye lens cameras which can take panoramic images; and
- Panoramas stitched together from images taken by specialised HDR cameras.

Using IBL usually means the locations visited by the AR user are predefined so an environment map can be pre-created, and some form of localisation such as GPS or bluetooth beacons used to identify the location and select the correct map to use. Recently some progress has also been made in creating the maps in real time as the user moves around [Monroy et al., 2018], although the lack of HDR capability would reduce the realism compared to a pre-created scene.

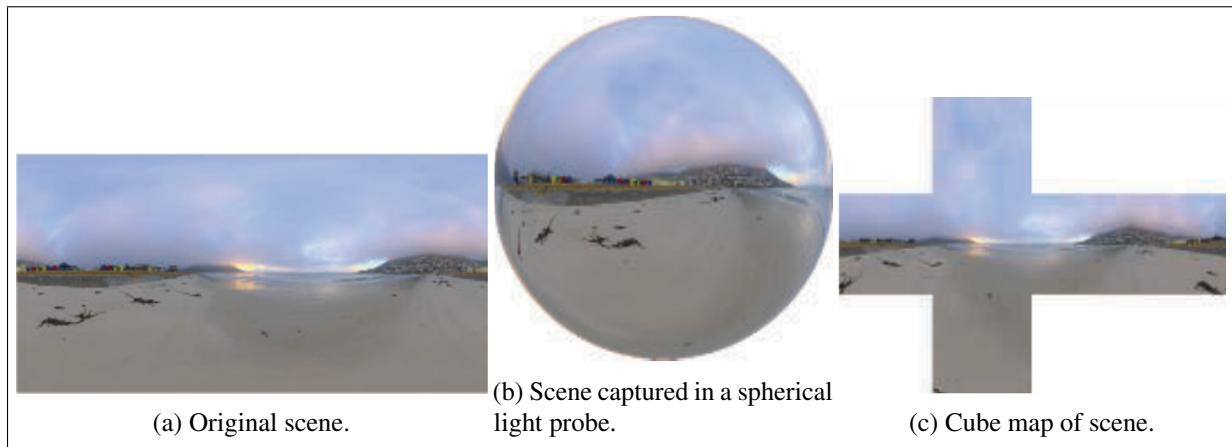


Figure 7.5: Capturing an IBL cube map (from [Akenine-Moller et al., 2018]).

7.5.6 PROJECTION SETUP

Context: Setting up the 3D projection transformation matrix using the camera intrinsics parameters.

Problem: In order to correctly place virtual objects into the 2D screen projection, the projection matrix must be constructed using the cameras intrinsic parameters such as the focal length and centre of projection.

⁸The same image captured using multiple exposure times so as to capture the full range of luminosity and then combined into a single image with color component representation as floating point values and not RGB bytes.

Solution: Most Computer Vision texts show a projection matrix projecting to a 2D plane, which is the expected behaviour for projection, however the 3D rendering APIs project to a standardised cube in order to retain depth information used during the final rasterisation phases. For OpenGL the cube has dimensions from -1 to 1, while for Vulkan (and DirectX) the x and y range is still -1 to 1 but the depth range is from 0 to 1.

The generalised OpenGL projection matrix is given by [Kessenich et al., 2016]:

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad (7.1)$$

where l, r, t, b, n and f are the left, right, top, bottom, near and far planes of the perspective frustum.

To adapt the projection for AR, the fact that the centre of projection is offset o_x, o_y , and the projection itself does not lie on the Z axis has to be taken into account. In order to make the projection matrix generic the x, y and z bounds are explicitly stated using $x_{max}, x_{min}, y_{max}, y_{min}, z_{max}, z_{min}$, rather than being assumed 1 and -1 as in the OpenGL matrix.

The transformation comprises a translation on the Z axis of $-\frac{fn(z_{max}-z_{min})}{f-n}$ which remains unchanged from OpenGL (after substituting 1 and -1 for z_{max} and z_{min}). The matrix diagonal (shown in red) performs scaling to within the specified bounds with the third entry scaling the Z axis to be between the near and far values remains the same, while the X and Y axis scaling are now based on the focal lengths f_x, f_y . The first and second entries of the third column (shown in blue) perform a shear transformation based on the camera centre offset. The projection itself is accomplished by the -1 in the fourth row which has the effect of dividing the x and y coordinates by the -z coordinate⁹.

$$\begin{pmatrix} \frac{f_x(x_{max}-x_{min})}{w} & 0 & 1 - \frac{o_x(x_{max}-x_{min})}{w} & 0 \\ 0 & \frac{f_y(y_{min}-y_{max})}{h} & \frac{o_y(y_{max}-y_{min})}{h} - 1 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{fn(z_{max}-z_{min})}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

For Vulkan, substituting 0 and 1 for z_{min} and z_{max} respectively fixes the Z coordinate, however Vulkan also differs from OpenGL in that the origin of its NDC is at the top left and not the bottom left so the Y transformations need to be flipped:

⁹See http://www.songho.ca/opengl/gl_projectionmatrix.html for a derivation of the OpenGL projection matrix.

$$\begin{pmatrix} \frac{2f_x}{w} & 0 & 1 - \frac{2o_x}{w} & 0 \\ 0 & \frac{2f_y}{h} & 1 - \frac{2o_y}{h} & 0 \\ 0 & 0 & -\frac{f}{f-n} & -\frac{fn}{2(f-n)} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Alternately for systems supporting both APIs, the OpenGL projection matrix can be premultiplied by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

to give the Vulkan one. Recent versions of Vulkan may also support an extension known as “KHR_VK_maintainance1” which, if explicitly enabled, allows a negative viewport height which will also have the effect of flipping the Y axis.

7.5.7 CALIBRATION ADJUSTMENT

Context: Adjusting the camera calibration to the display resolution.

Problem: Camera calibration is done at a specific resolution which should normally be recorded along with the calibration details (focal lengths and offset). The AR system may however display using a different resolution, resulting in incorrect calibration.

Solution: The calibration matrix must be scaled to the correct value by multiplying it with a diagonal scaling matrix. Given the original resolution at which the calibration was performed of $r_x \times r_y$ and the display resolution of $r'_x \times r'_y$ and original calibration focal lengths of f_x, f_y and offsets o_x, o_y the scaling can be performed by:

$$\begin{pmatrix} \frac{r'_x}{r_x} & 0 & 0 \\ 0 & \frac{r'_y}{r_y} & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{pmatrix}$$

7.5.8 BILLBOARD

Context: Ensure rendered content always faces towards the user or a designated direction.

Problem: How can it be ensured that some 3D virtual content is directly visible to the user or is always oriented in some given direction regardless of the camera position in world coordinates ?

Solution: Use a 3D rendering technique known as billboarding [Akenine-Moller et al., 2018, Section 13.6]. Several different types of billboarding are possible based on different configurations for the desired normal and up vectors. Screen-aligned billboards use the camera’s up vector and a normal opposite to the

vector to the projected view plane, resulting in the “classic” billboard used for text which is always screen aligned. World-oriented billboards rotate with the camera by constructing a rotation transform using the world up vector. Axial billboards rotate around an absolute world axis, but transform to face the user as much as possible within their range of motion. Many available scene graphs and 3D engines will have built in support for billboarding.

7.6 Interaction Patterns

This section describes patterns for MAR interaction. As mentioned in Subsection 7.1.2 the interaction and rendering components are coupled to some degree, as user initiated changes during interaction change the scene being rendered.

7.6.1 RAYCAST

Context: Virtual item selection.

Problem: A means of selecting a 3D object using the touchscreen is required. This may also include selection of UI items such as buttons.

Solution: Cast a 3D ray into the scene and check for intersections between the geometry of virtual objects¹⁰ and the ray. In order to construct a ray the following steps, comprising the reverse of the usual rendering transform, are followed:

1. The touchscreen viewport coordinates are converted to NDC coordinates, that is coordinates normalised to a range of -1 – 1. The Y coordinate also needs to be reversed as the viewport origin is at the top left.
2. Convert the above NDC coordinates to homogenous 4D coordinates by setting z to -1 (the camera in the world system is looking down the -z axis).
3. Multiply the homogenous coordinates by the inverse of the projection matrix to get the eye-space coordinates reset z and w to -1 and 0.
4. Now multiply the eye-space coordinates by the inverse of the Model-View Matrix to get the world coordinates.

The final step is to test for intersections between the world-space ray and the geometry in the scene. To do this the geometry to be tested is surrounded by some bounding volume such as a sphere or bounding box, and then the ray points are tested for intersection with the bounding volume. This process can be

¹⁰For cases where the geometry of real-world objects is known (for example from CAD files), and the object has been registered in the scene, then the pattern would also work with such objects.

optimized by maintaining a K-D tree or octtree data structure for the vertices in the geometry [Samet, 2006]. Scene graphs implementations may also provide bounding volumes for nodes. Chapter 22 of Akenine-Moller et al. [2018] discusses in great depth the various approaches and possible optimisations.

In order to make it easier for end users to obtain feedback on where they are pointing, the ray itself can be rendered while the user is interacting with the touchscreen. For large virtual worlds the Go-GO technique of Poupyrev et al. [1996] can be used where the extension of the ray is initially linear and then becomes exponential.

7.6.2 VERTEX CODING

Context: Virtual item selection.

Problem: A means of selecting a 3D object using the touchscreen is required. This may also include selection of UI items such as buttons.

Solution: Assign a specific code to the vertex data per geometry and then use it to identify the geometry the vertex belongs to when a user selection is made. A separate framebuffer is used to store a non-display texture that contains the code for those pixels where the geometry projects to in viewport space. This framebuffer is created in a separate render pass from the one that creates the display. The fragment shader for the render pass writes the code representing the geometry to its location in the texture. The selection test can then bind the texture framebuffer, get the touchscreen coordinates and read the pixels from the framebuffer at the point (it may be necessary to change the Y coordinate to reflect a top left origin). This eliminates having to check all vertices against a ray as described in Subsection 7.6.1, although reading pixels back from the GPU can also be expensive.

7.6.3 RETICLE

Context: Indication of a virtual item's focus status.

Problem: When a user is attempting to interact with a virtual item and several items are selectable, a means to show the user which item is currently pointed at is required.

Solution: Provide a visual indicator of which item the user is currently pointing at. Examples include drawing a circular target or other indicator similar to a mouse pointer, or more advanced options such as outlining the entire virtual object geometry in a different colour. In the former case the shape of the pointer can change based on the current context of the interaction.

7.6.4 GUIDE

Context: Spatial guidance for navigating the AR world.

Problem: Some AR applications provide services such as finding a location that may be required to guide a user to a destination.

Solution: Render an onscreen guide which moves to the left or right of the screen to show the user in which direction to move. The simplest examples can display a Google Maps style position marker, which is generally well known to users. More advanced variations can show a circular tunnel effect composed of intermediate waypoints on the route to the destination. Textual advice or simple voice synthesis of phrases such as “go left” can also be used for further reinforcement. Even more sophisticated versions could use particle effects to create glittering translucent guides which capture the users attention.

7.6.5 AFFORDANCES

Context: Manipulation hint through rendered properties of a virtual object.

Problem: It can be difficult for a user to determine if a virtual object can be manipulated, and how to initiate the manipulation.

Solution: An affordance is “the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could possibly be used” [Norman, 1988, p. 9]. For example the handle of the ubiquitous teapot used in almost every 3D rendering demonstration, is an example of an affordance, as most humans will be aware that teapots can be manipulated using their handle. Affordances, as UI widgets, often have some form of validation that occurs naturally, for example a cars steering wheel cannot be turned beyond the maximum that the wheels themselves can turn.

In the context of this pattern, the solution then is to render some form of virtual affordance onto manipulatable objects. For example items that can be unscrewed could have a bottlecap texture, while objects than can be opened may have some equivalent of a door handle. Even simple affordances can be useful, for example after selection of an object, render a wireframe cube around the selected object with the cube wireframe lines drawn with an obvious grabbable area that is slightly thicker than the rest of the line and possibly a different colour. Grabbable areas with differing properties may then be used for translation, scaling and rotation of the object.

7.6.6 ANCHOR

Context: Attach a virtual object or informational content to a real-world object or location.

Problem:

1. Developers of location sensor based AR applications may wish to attach informational content to a sensor location.
2. Developers may wish to attach informational content to detected real-world objects in a non-SLAM or hybrid SLAM AR system where the object detection features reside in a database.
3. Users may want to fix a virtual object at a detected real-world objects location where the object may be from a feature database or an arbitrary generalised object such as a plane.
4. Developers can use recognised real-world object locations for SLAM (see Subsection 4.4.8) loop closure and to monitor pose and registration drift.

Solution: For location only based systems convert the current sensor location into a convenient global or local coordinate system, then find the heading from the sensor location to the anchor location. If the anchor is visible from the current location place the content in the direction of the heading. GPS based system often use the global geocentric ECEF (Earth-Centered, Earth-Fixed) or the local ENU (Local East, North, Up) coordinate systems [Pryss et al., 2016]. Indoor sensor based systems such as Bluetooth beacon or WiFi localisation can use arbitrary coordinate systems, for example local building, floor or room.

Non-SLAM or hybrid SLAM vision based systems can detect features from a database, while both SLAM and non-SLAM systems can add features of generically recognisable objects such as planes found during detection (Subsection 4.4.1). Then when these features are detected or tracked, place the associated virtual object or content at or near the detected location. In a SLAM system the known location of the anchor can be used during loop closing [McDonald et al., 2013]. This applies to all three types of anchor with the third type being SLAM system generated anchors that have no displayable content. In a SLAM system, the world coordinates of an anchor can also be improved as the world map is enhanced during user navigation.

Information content displayed at anchored positions include:

1. Simple textual annotations which typify many informational augmented browsing AR applications. Textual annotations require the renderer supports outputting text as 3D APIs do not support this by default and require code to rasterize truetype or other fonts. In addition simply placing text close to the target object may occlude other salient features. Rosten et al. [2005] describe automating positioning of textual annotations to minimize invasivity.
2. 2D and 3D graphs or other image based visualisations.
3. Multimedia content such as video.

4. Virtual 3D objects.
5. See the MAGIC LENS (Section 7.6.9) pattern for an alternative type of informational embellishment.

7.6.7 FREEZE FRAME

Context: Provide a static frame for interaction or provide an alternative exocentric view.

Problem:

1. Handheld AR, as typified by MAR, can be difficult to interact with, as the user has only one free hand to interact with, and the hand holding the device may not be stable during interaction. An approach that makes it possible to stabilise the display during interaction is required.
2. It may be necessary to switch to an exocentric view (see Subsection 4.4.9) to provide more information.

Solution:

Provide a UI widget, gesture or voice command that freezes the display on the current frame so the user can interact with objects that are static and thus easier to manipulate. The underlying application must utilise the positions of objects within the frozen frame when performing the interactions. Applications can however continue processing incoming frames and sensor data without rendering them in order to keep the system up to date.

Another use of this pattern is to enable displaying related 2D information, for example a 2D map of the current location, as mobile device screens are too small to provide multi-modal displays of 2D and 3D views simultaneously.

7.6.8 TANGIBLE TOOLS

Context: Providing an intuitive UI which allows users to directly interact with virtual objects rather than through the device touchscreen.

Problem: Users may find it difficult to interact through a touch screen compared to manipulating objects directly.

Solution: Provide a Tangible User Interface (TUI) (see Subsection 4.4.9). This is implemented by recognising some manipulator in the camera feed, and using the manipulator to perform the interaction. The most intuitive manipulator is the human hand, but other simpler options exist, for example Kawashima et al. [2001] provided a paddle with a fiducial marker for recognition. If hand recognition is used [Datcu

and Lukosch, 2013], more advanced detection and tracking methods are required (the Viola-Jones facial detection algorithm [Viola and Jones, 2001] is frequently also used on hands, see Chapter 6 for more on detection and tracking). The use of hand recognition also allows for gesture recognition [Houssem and Mahmoud, 2018], although at a cost of still greater complexity.

When using this technique with monocular cameras with no depth detection, it can be difficult to determine what object the user is attempting to interact with without additional aid from the user (for example highlighting candidate objects based on results from the RAYCAST pattern).

7.6.9 MAGIC LENS

Context: Providing drill-down information of visible real-world objects.

Problem: How to switch between a hidden internal view and a normal everyday view of a real-world object ?

Solution: MAGIC LENS is an extension of the ANCHOR (Section 7.6.6) pattern, as it provides an alternative view for a detected real-world object whenever it is viewable. Instead of rendering a object or informational annotation, an overlay of the real-world object with a 2D (texture) or 3D display of the internals of the object is rendered. For example a medical application may display a skeleton or underlying tissue for a body [State et al., 1996], while Schall et al. [2009] display underground infrastructure such as pipes to aid maintenance. MAR applications may also add a capability to toggle the magic lens on and off to allow the user to switch between seeing the unadorned real-world object and the enhanced view.

7.7 Pattern Language

At an architectural level the rendering and interaction subsystems will be instantiated within task graph nodes (see Section 5.5). The rendering node will normally be the final node in the tree and will directly or indirectly process output from the detection, tracking and interaction components. The design of the renderer is affected by the choice of architectural pattern for interaction, as, for example, if the MVC pattern is selected then the renderer plays the role of the view, and view components have to be created in the renderer (scene graph implementations may need to implement views as scene graph nodes). The initial pattern choice is thus to choose an interaction architectural pattern (Section 7.4).

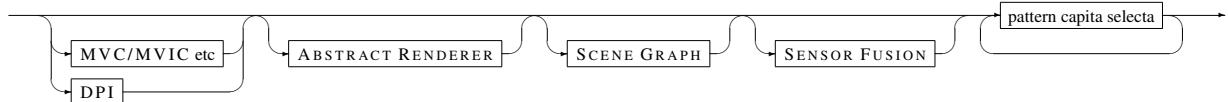
After selection of the interaction architectural pattern, the next step is to decide whether multiple renderers will be supported through the use of ABSTRACT RENDERER. If multiple renderers are supported than a common ABSTRACT RENDERER must be defined and FACTORY METHODS or ABSTRACT FACTORY used to instantiate the selected renderer.

The next step is to select an architecture for the renderer (or to decide not to use an architecture). Currently the only architectural pattern specified for the renderer is SCENE GRAPH, therefore the choice is whether to use a scene graph (it is also possible to support both scene graph and non-scene graph renderers via the ABSTRACT RENDERER interface).

It is possible to consider all device sensors to belong in the interaction domain, and therefore to implement SENSOR FUSION (see Section 6.3.3) in the interaction component. On the other hand, SENSOR FUSION can also be performed in the detection and tracking components (see Chapter 6). In particular for SLAM system, it may be more natural to implement SENSOR FUSION during detection/tracking. For non-vision sensor only based MAR, placing SENSOR FUSION in the interaction component or as a standalone component may be better. For hybrid vision/sensor systems the choice is not clear cut and best left to the discretion of the designer. The choice then is whether to use sensor fusion, and if so whether it should be implemented within the interaction component.

Beyond these architectural guidelines, it is difficult to lay down any further sequences for lower level patterns as the order in which they are applied may depend on the technology and tools used. For example many scene graphs and higher level 3D engines may include components implementing many of the patterns as scene graph nodes or embeddable components.

The rail diagram for the subsystem PL is then:

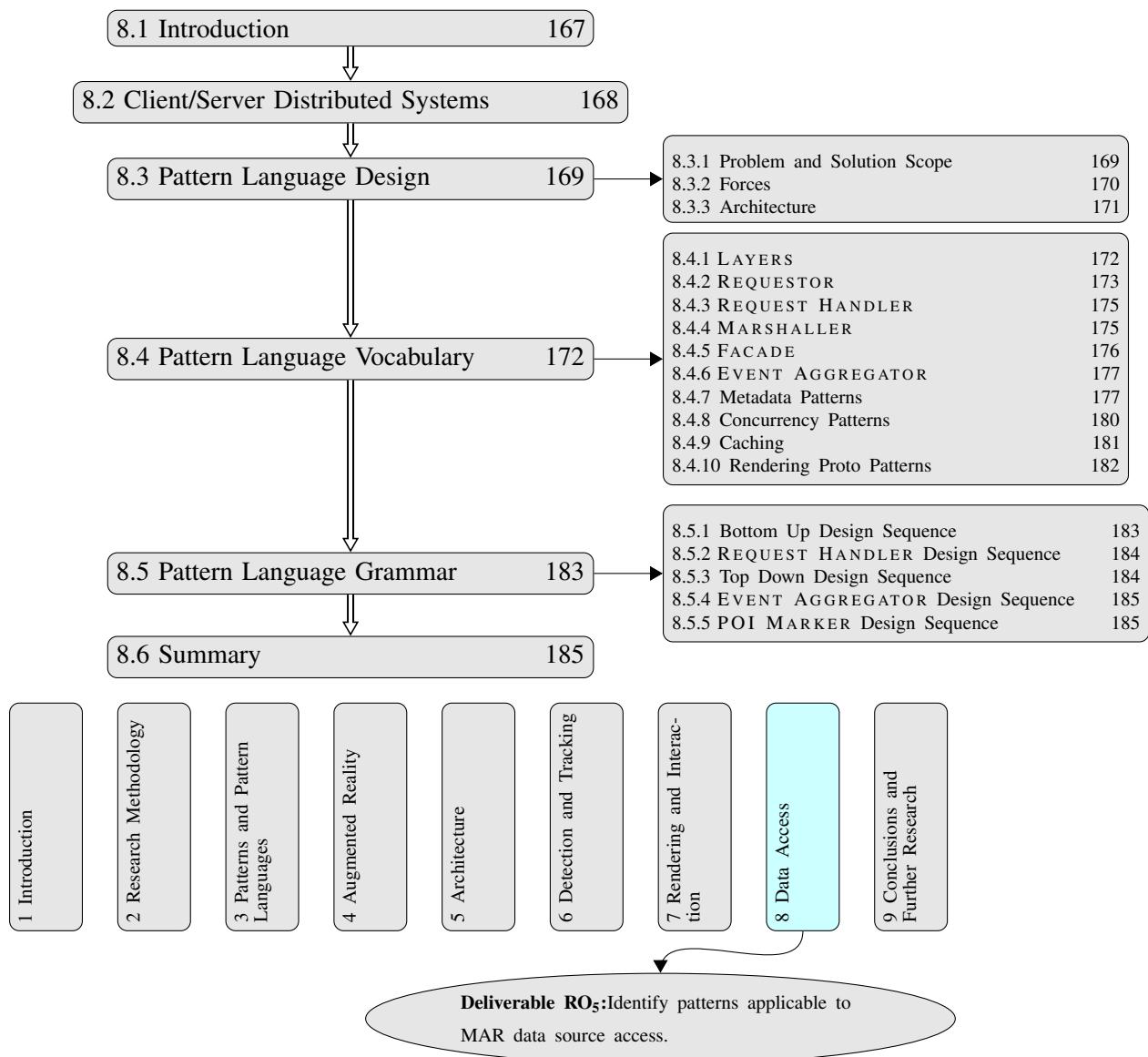


7.8 Summary

The chapter commenced with a brief review of 3D rendering hardware and software including the evolution of hardware GPUs and APIs used to create software for both software and GPU 3D rendering. A few technical concepts underlying the development of 3D software were then described, before moving on to cataloging some architectural patterns relevant to rendering and interaction components. Pattern catalogs for rendering and interaction were then covered in the following section. Finally possible pattern sequences starting from the architectural patterns were discussed in Section 7.7.

8

Data Access



8.1 Introduction

Modern mobile devices are equipped with multiple sensors including Global Positioning System (GPS) sensors, gyroscopes and compasses/magnetometers, while also supporting several networking options such as WiFi and Bluetooth. The use of these sensors allows for access to relevant AR data based on the current location, for example, retrieving data keyed by a given radius or bounding box of the current location. Which sensors are used depend on whether the device is located where GPS signals are accessible, mainly outdoors. For indoor Mobile Augmented Reality (MAR) WiFi signal triangulation, WiFi Round Trip Time (RTT) estimation¹ [Want et al., 2018] or using Bluetooth beacons [Statler, 2016] may be used instead.

The type of data required by MAR applications vary widely, for example:

1. Internal use, for example non-SLAM vision based applications (see Chapter 6) can retrieve detection features for objects that are within range of detection, held in a LAN database and cached in the device;
2. Local data on visible objects for annotation, which is held in a Local Area Network (LAN), and can again be cached on-device; and
3. Remote data held on Wide Area Network (WAN) Internet servers.

In particular item 3 above requires further elucidation. The last decade has seen the increased availability of geographical keyed data in public and private Geographical Information System (GIS) databases, Spatial Data Infrastructure (SDI) GIS Web services [Masser, 2005], Semantic Web Linked Open Data (LOD) datasets [Reynolds et al., 2010], general information for example DBpedia [Auer et al., 2007] and Web-based mapping resources such as Google maps [Miller, 2006] and OpenStreetMap [Haklay and Weber, 2008]. A related area is that of Big Data [Mayer-Schönberger and Cukier, 2013] visualisation where large quantities of remote data held on multiple servers can be visualised using AR [Bermejo et al., 2017].

These heterogeneous data sources provide an opportunity for AR applications to access a much larger variety of location-based data. Accessing such a variety of sources is complex, and patterns and pattern languages provide a means of providing guidance in the design of this facet of a MAR application without being prescriptive as to the exact details or level of complexity of the solution.

The rest of this chapter will concentrate on finding a solution for RO₅, which states:

Identify patterns and a pattern language applicable to MAR data source access.

¹IEEE 802.11mc

The applicable research process stage from Subsection 2.4.3 is:

Analyse & Design the System.

The output will be:

1. A set of patterns applicable to MAR data access; and
2. A pattern language built on these patterns.

The next section (Section 8.2) provides a brief overview of the field of client/server distributed systems, as the underlying model for the PL is based on this field. Section 8.3 describes the design of a PL covering data access for MAR, and is followed by Section 8.4 that describes the individual patterns making up the vocabulary of the PL. Finally Section 8.5 defines the PL grammar by connecting all the patterns into pattern sequences.

8.2 Client/Server Distributed Systems

Provision of AR data from heterogeneous sources can be seen as a subset of the more general domain of client/server distributed systems. Tanenbaum and Van Steen [2007] characterise distributed systems as a collection of independent components available on different computers that interact to provide a system that the end-user perceives as being a single system. Some of the most complex computer applications in use today are distributed, for example Google's search servers accessing distributed search data [Dean, 2010].

Distributed systems are inherently complex, particularly with regard to latency, concurrency and error handling. Participating processes in distributed systems run concurrently on separate machines and are connected by networks where the network latency may vary or the network may fail altogether. Timeouts between components may lead to partial errors which complicate error handling considerably. While it is possible to develop such systems using low level networking APIs, the complexity involved has led to a layered approach being widely adopted in which a mid-level structured communication layer and a higher level middleware layer are utilised [Buschmann et al., 2007b]. The structured communication layer encapsulates bits and byte level detail while providing a simplified wrapper around specific protocols such as XML-RPC [Winer, 2003]. Middleware provides further abstraction as it attempts to make the underlying network access as invisible as possible, allowing the developer to operate on local and remote resources in a similar fashion. It is however, not always possible to fully hide the networked character of a distributed API, particularly given that errors or network timeouts can occur in multiple locations while processing a distributed call.

Client/server distributed systems utilise several communication methods between participants listed below: [Voelter et al., 2004]

1. The simplest and oldest approach is simple file transfer, for example the FTP and TFTP protocols;
2. Message passing, for example HTTP requests and responses;
3. Remote Procedure Calls (RPC) where clients invoke functions or methods on remote components analogous to local function provided by conventional program languages;
4. Distributed objects which are middleware built on top of message passing or RPC, and consist of local or remote objects providing callable interfaces where invoking methods from the interface is network transparent. Most distributed object systems also support metadata about the interfaces which may be queried by clients;
5. Queue based where communication occurs through the placement and removal of messages from a commonly accessible queue. This includes single sender/receiver as well as broadcasting of messages to multiple recipients;
6. Shared repositories where clients place requests in a commonly accessible location, for example a database; and
7. Streaming, where interaction between clients and servers take the form of a continuous stream of data.

In the next section a PL will be described which attempts to provide guidance in the design of such systems in the restricted domain of AR data access. The PL does make some simplifying assumptions for the specific problem domain which makes the problem somewhat more tractable as a PL covering the design of a generalised distributed system would require an entire book (for example Buschmann et al. [2007b]).

8.3 Pattern Language Design

This section will describe the overall design of a PL for MAR data access. Subsection 8.3.1 describes the problem in more detail. Subsection 8.3.2 then describes the forces at play in the design of the PL. Finally the high level architecture is discussed in Subsection 8.3.3.

8.3.1 Problem and Solution Scope

The problem that must be addressed is to provide guidance for the design of client-side distributed layers that runs on the mobile device and provides AR data access for other components in the AR system. This may involve guidance in the design of client-side layers utilising low-level protocols or interaction with

existing remote server middleware layers.

Given the complexity involved in the design of distributed systems, it is fortunate that several simplifications apply for the AR data source domain. Only the client side of a distributed system needs to be considered, as the server side comprises predefined web services and databases. Of the communication methods enumerated in Section 8.2, only simple message passing (usually HTTP), RPC and simplified distributed objects are considered as the other methods are currently rare or non-existent in AR data sources. The PL could also be extended to support textual queue based systems, for example those using the text based STOMP protocol [Mesnil, 2014].

The common denominator amongst the supported communication methods are that parameters are textual. HTTP uses HTML, only textual RPC such as XML-RPC and textual XML based distributed objects such as SOAP [Newcomer, 2002] are supported. Database parameters for protocols such as JDBC [Fisher et al., 2003] are assumed to use SQL text parameters and receive responses in text. This means that the complexity involved in the marshalling of binary parameters between client and server is avoided. In particular the difficulties involved in passing parameters for binary distributed object systems such as CORBA [Balen et al., 2000] or DCOM [Thai, 1999] are not encountered.

8.3.2 Forces

The various criteria that affect a design comprise the pattern and PL forces. Forces include desired properties as well as constraints. The forces a PL for the domain should address include:

1. A large variety of different data-sources need to be supported;
2. The sources are heterogeneous, that is they are available from multiple sources, use multiple protocols and are provided in different data formats;
3. New types of sources should be easy to add;
4. Accessing the sources may be complex, so providing optional simplified interfaces may be desirable;
5. Mobile device user interfaces are adversely affected by synchronous operations which block the user interface thread so support for asynchronous operations are crucial;
6. Mobile device networking can be subject to intermittent failure so the ability to preload and/or cache relevant data is necessary; and
7. As data from various sources may be complementary, it may be necessary to merge data from different sources.

8.3.3 Architecture

A large body of literature already exists describing the use of patterns in the design of distributed systems [Buschmann et al., 2007b; Voelter et al., 2004]. Selected patterns from this literature can, therefore, be used to form the basis for the design of an AR data-source pattern language.

The first decision to be made concerns defining an overall architecture for the PL. The **BROKER** pattern which is defined by Buschmann et al. [2007b, p. 237] to '*[...]separate and encapsulate the details of the communication infrastructure in a distributed system from its application functionality*' provides a good starting point. The **BROKER** pattern was further refined by Voelter et al. [2004] to form a PL comprising eight different primary patterns and several ancillary ones. The client specific patterns from this compound with their context modified where necessary will be used as a foundation for a MAR data-source.

The **BROKER** pattern itself has an underlying architectural structure in that the patterns it defines provide functionality at different levels of abstraction, that is, they conform to an architecture defined by the **LAYERS** pattern which "*[...] helps to structure applications that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction*" [see Buschmann et al., 1996, p. 31].

The **FACADE** pattern plays approximately the same role as **CLIENT PROXY** [Buschmann et al., 1996] pattern does in **BROKER**. The emphasis differs slightly though, as instead of providing a client interface matching a binary server component interface, it provides a simplified interface to a **REQUESTOR**, that is, its role is that of a **FACADE** [Gamma et al., 1995] or **WRAPPER FACADE** [Schmidt et al., 2000] depending on whether the **REQUESTOR** interacts with a document based source such as a web data source or a procedural data source such as a XMLRPC server. As these patterns all provide the same functionality of providing a facade that hides the underlying complexity, the term **FACADE** will be used generically.

An addition to the **BROKER** pattern specific to the AR data PL is to, optionally, add a higher level **EVENT AGGREGATOR** pattern [Fowler, 2004] which can aggregate several data-sources such as **FACADE** or **REQUESTOR** in order to provide clients with coalesced results from various sources where the query results are asynchronous events.

While client side rendering is obviously not part of a distributed system, two further rendering patterns are also described which are specific to locational mobile applications namely **POI MARKER** and **POI RADAR** (Section 8.4.10). These patterns are described in this chapter instead of Chapter 7 as

they depend on the data access PL patterns.

A high level overview of the patterns is provided in Figure 8.1. Some pattern categories, for example Configuration or Concurrency are grouped together. A dotted box for a pattern means it is not part of the layered architecture and can be applied at any level. Figure 8.1 illustrates both pattern dependencies in the form of arrows between patterns as well as the layering architecture by having rows of patterns correspond to architectural layers.

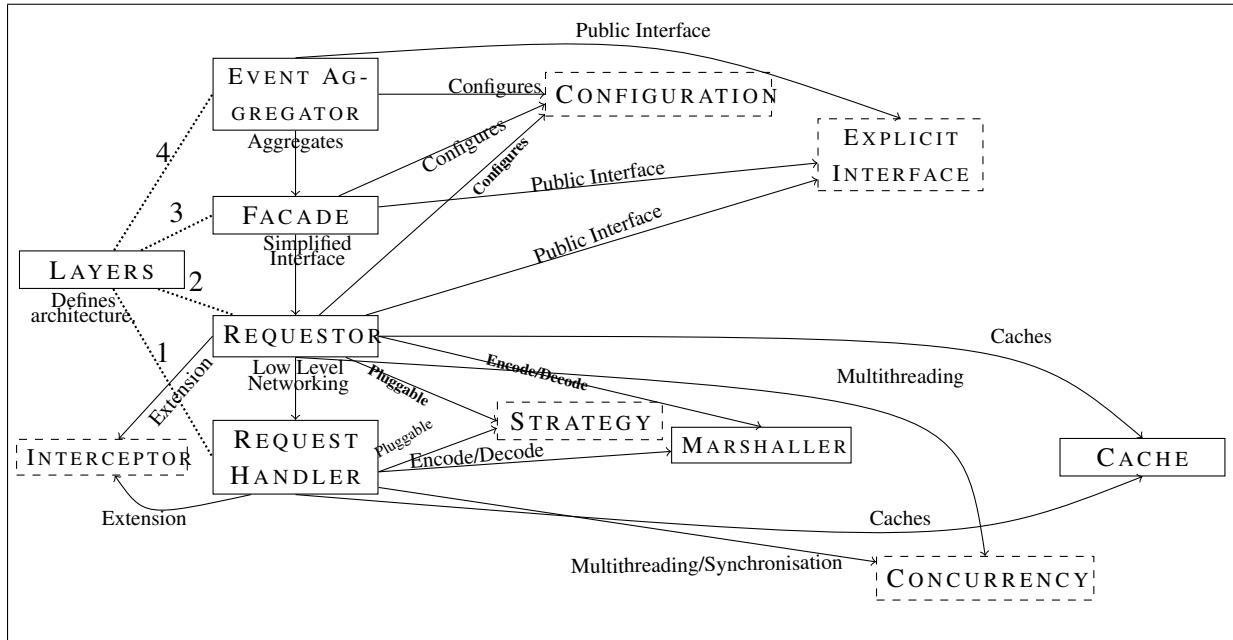


Figure 8.1: Pattern Dependencies and Layered Architecture.

8.4 Pattern Language Vocabulary

The individual patterns that were briefly described in terms of their use in an overall architecture in Section 8.3.3 are described in greater detail in this section. The context and forces applicable to these patterns and their use of other complementary patterns, is narrowed down to fit within the setting in which they are used within the PL as opposed to their more generic description when used as standalone patterns. After describing the individual patterns, Section 8.5 describes their use as part of the PL grammar, that is, how the individual patterns can be combined into different sequences depending on the requirements of the artefact being designed.

8.4.1 LAYERS

Context: Decomposition of PL components into different levels of abstraction.

Problem: The PL constituent patterns vary from providing a high level simplified interfaces for accessing a data-source to implementing low level networking protocols and providing multi-threading and concurrency. How should these patterns be organised so as to be understandable and maintainable while also providing good performance and scalability?

Solution: Organise the PL patterns into layers by level of abstraction with each layer only being able to access patterns in the same or a lower layer as in the LAYERS pattern [Buschmann et al., 1996]. Such layering exhibits high cohesion which improves maintainability, reusability and understandability. Isolating low level functionality such as network communication protocols, means changes to protocols, bug fixes or the introduction of new protocols can be done without affecting higher level functions. Multiple layers also make testing easier which promotes reliability and performance, while also makes it easier to refactor the design to accommodate late changes in design requirements.

Having LAYERS as the initial pattern in the pattern language allows an artefact designer to make a decision on the desired complexity of the artefact by deciding how many layers will suffice for the required level of complexity. Once this decision is made the designer can start fleshing out the rest of the design by adding patterns to the different levels. In a full deployment of the PL, four layers exist as seen in Figure 8.1.

Different configurations are possible depending on the desired complexity of the artefact. For example the REQUESTOR (See Subsection 8.4.2) could handle all low level network operations instead of using a REQUEST HANDLER (see Subsection 8.4.3) in which case layer 2 becomes layer 1 in a three layer configuration. The simplest possible configuration would comprise only a REQUESTOR providing both an interface to a data-source and the low level protocol and concurrency support.

8.4.2 REQUESTOR

Context: Accessing heterogeneous data-sources requires protocols, techniques and parameters that are specific to a given source.

Problem: Requiring clients to create data-source specific requests for every type of source, encode parameters, establish the connection and decode results would be onerous and error prone. Software artefacts designed in this way would be difficult to maintain and require low level expertise from developers.

Solution: Provide a REQUESTOR for different data-sources. This REQUESTOR is responsible for all communication with its particular data source. Examples of data-source types include SDI web services such as WFS, SPARQL endpoints, dedicated services such as Google Maps and JDBC databases.

The granularity at which REQUESTORS are created is left to the designer using the PL to create an

artefact. For example, a single REQUESTOR could be created to handle all SOAP requests by creating service-method calls dynamically for different services or different REQUESTORS could be used for different SOAP services at different URLs.

Mobile devices UI require that long-running operations support concurrency so as not to have the UI being blocked for long periods of time. Recent versions of Android, for example, will throw an Application Not Responding (ANR) exception after only 5 seconds of blocking. REQUESTORS should, therefore, support concurrency as outlined in Section 8.4.8 unless concurrency is supported at a lower level in the REQUEST HANDLER or at a higher level in the FACADE. The recommended point at which to apply concurrency to the design is in the REQUESTOR.

If the REQUESTOR may repeat access to the same large resources frequently, caching (Section 8.4.9) may be added to the design.

Parameters to REQUESTORS comprise both application-specific parameters, for example, a location specified in a coordinate system and data-source specific configuration parameters such as an Internet host address. The data-source specific configuration can be specified as parameters to the relevant operations along with the domain specific ones. Alternatively the data-source specific parameters may, as described in Section 8.4.7, be obtained by using one or more applicable configuration metadata supplier patterns.

In some situations, it may be required to choose a particular REQUESTOR from a number of related ones. For example, when implementing a JDBC REQUESTOR, the developer may have to provide support for several different SQL dialects specific to different databases for spatial queries. In order to allow the REQUESTOR to vary independently of its clients the STRATEGY pattern [Gamma et al., 1995] may be used. An EXPLICIT INTERFACE containing common functionality may be used to play the Strategy role in defining the REQUESTOR interface while the various REQUESTORS play the *Concrete Strategy* roles. A FACADE, or artefact code in the case of not using FACADEs, plays the Context role in selecting and using a particular *Concrete Strategy*.

For black box frameworks, the REQUESTOR can allow for user extension by using the INTERCEPTOR pattern [Schmidt et al., 2000]. White box frameworks can specify framework “hot spots” (points of variation) as overridable TEMPLATE METHODS [Gamma et al., 1995] of the REQUESTOR class.

8.4.3 REQUEST HANDLER

Context: Factoring out commonalities and abstracting complexity in low level implementations of network protocols.

Problem: When implementing REQUESTORS, in many cases they may share common code. For example HTTP request methods such as GET or POST can be implemented generically with only the parameters used when invoking the request differing.

Having multiple REQUESTORS reimplementing the same code is inefficient and leads to code that is hard to maintain. Additionally, mobile devices are battery constrained and network access requires a relatively high power use, so unnecessary network connections can lead to mobile applications being “power hogs” utilising unnecessary amounts of battery power.

Solution: Create a specialised REQUEST HANDLER whose responsibility it is to perform common low-level networking functionality that can be shared between multiple REQUESTORS. As a secondary function, the REQUEST HANDLER can also batch multiple requests together as a method to reduce power usage.

For example, a REQUEST HANDLER for message-oriented data-sources could be used by multiple REQUESTORS. The REQUEST HANDLER could handle opening HTTP connections, placing HTTP GET or POST requests and batching requests to the same address together by using the HTTP Keep-Alive header, while the higher level REQUESTOR handles the message content.

In order to support concurrency, the recommended approach is to implement the REQUEST HANDLER code to be threadsafe and have it execute in the context of the calling REQUESTOR which provides the concurrency support. If this is not possible or desirable, then the concurrency patterns outlined in Section 8.4.8 may be used.

If the REQUEST HANDLER accesses the same large resources frequently, caching (Section 8.4.9) may be added to the design. The designer should decide whether it is better to provide caching at the REQUESTOR or REQUEST HANDLER level.

8.4.4 MARSHALLER

Context: Encoding and decoding textual markup languages such as XML or JSON.

Problem: When implementing REQUESTORS or REQUEST HANDLERS, parameters for data-source requests often need to be encoded and results decoded from text-based formats specified by standards defined for the particular type of source.

Encoding the request parameters correctly can be a complicated process. Sometimes it involves having to adhere to strict rules and protocols with incorrect encoding resulting in errors being returned from data-source requests. Similar sources may also share the same parameter encoding rules so encapsulating the processes involved in encoding parameters allows their reuse.

Solution: Create a MARSHALLER with the responsibility to encode parameters for requests to specific types of data-sources. For example a SOAP MARSHALLER could create the XML required to send a SOAP request and decode the XML returned as the result to extract the return values. An HTTP REQUESTOR using HTTP POST methods to send HTML forms, could use a MARSHALLER to encode/decode URL-encoded requests and responses.

8.4.5 FACADE

Context: Providing high level interfaces for interaction with data-sources.

Problem: Data-sources represented by REQUESTORS (Section 8.4.2) may require complex interaction requiring many data-source specific parameters, domain-specific parameters, protocol-specific processing and result and error handlers. This requires that the client have an in-depth knowledge on how each source is used and where it is located. From a design, use and maintenance perspective, it would be preferable if the client only needed domain specific knowledge to access a source through a REQUESTOR. Interacting directly with REQUESTORS makes client implementation complex and harder to debug and may lead to possibly unnecessary coupling between the client and the REQUESTOR.

Solution: Provide a class that encapsulates as much of the complexity involved in dealing with a data-source REQUESTOR as possible. The amount of low-level knowledge about the source represented by the REQUESTOR should be minimised by providing a simplified interface that the client interacts with. The interface implementation should be responsible for supplementing client supplied parameters with data-source specific ones, perhaps by using the patterns specified in Section 8.4.7 to read source-specific configuration metadata from an external source. The FACADE should also be able supply an EXPLICIT INTERFACE to receive concurrent results or errors and interpret the results for the client. The FACADE should then be able to demultiplex the concurrently supplied results and provide them to the client on the client's GUI thread.

Advanced clients that require a greater degree of control can still use the REQUESTOR by interacting directly with it. Similarly, an artefact designed using the PL can choose to skip the FACADE layer altogether or only implement FACADES for some REQUESTORS.

8.4.6 EVENT AGGREGATOR

Context: Combining asynchronous results from multiple FACADES.

Problem: Clients may need to issue the same query to several data-sources via their FACADES or REQUESTORS. An example would be to return all points of interest, AR feature recognition vectors and map data within a given radius of the current location. It would be more convenient if results from the sources were aggregated so that a single request could be issued and a single, conglomerate response received. Another frequent requirement is that these updates be continually issued as the devices location changes so that the MAR application does not need to manually reissue requests based on location changes of the device.

Solution: Provide a EVENT AGGREGATOR [Fowler, 2004] which can aggregate multiple FACADES and/or REQUESTORS. The client can then issue commands to all the contained sources and receive a response containing combined results from all its FACADES. As the results are asynchronous they are decoupled from requests so incoming results from different FACADES can be treated as events from multiple sources. This pattern also helps to resolve force 7 from Section 8.3.2. Where necessary the EVENT AGGREGATOR can also aggregate mobile OS events such as location updates which can then either be combined with data source results or supplied as separate client updates.

8.4.7 Metadata Patterns

From the client's perspective, a source of complexity revolves around the parameters required by different data-sources. For example, a particular source might require a combination of domain-level parameters such as a location in latitude and longitude, in combination with several source-specific parameters such as the Internet host address of a web service. In many cases, the abovementioned source-specific parameters can be treated as configuration metadata which may be held in configuration files or databases existing at specified locations [Duval et al., 2002]. Several patterns exist that provide guidance when designing configuration providers will be discussed in the context of the PL in this section.

8.4.7.1 SERIALIZER

Context: Reading predefined configuration for FACADES or REQUESTORS from a file or database into a static publicly accessible class.

Problem: Static configuration classes containing parameters accessed as properties, provide a simple method to obtain parameters. Initialising the content of these classes in such a way that there is no dependence on the type of media the content is stored on or the format it is stored in is required.

Solution: **SERIALIZER** [Martin et al., 1997] describes a solution which allows an abstract interface for serialisation to and from multiple data types such as CSV or JSON files or a database. Some languages such as Python and Java provide built in support for simple flat-file based implementations of this pattern.

8.4.7.2 REGISTRY

Context: Obtaining predefined global configuration metadata for **FACADE** or **REQUESTOR**.

Problem: Access to potentially complex, globally defined data-source metadata, the contents of which can be defined separately from a particular software artefact.

Solution: **REGISTRY** [Sommerlad and Rüedi, 1998] provides a central registry object that is initialised by *Installer* objects with *Element* instances that are associated with a well-known name and can thus be accessed by name.

8.4.7.3 DYNAMIC FACTORY

Context: Dynamic instantiation of configuration classes that implement some **EXTERNAL INTERFACE** where the concrete types of these classes should be determined at runtime.

Problem: It may be necessary or convenient to allow the addition of new configuration class satisfying a given configuration interface without needing to recompile the artefact.

Solution: **DYNAMIC FACTORY** [Welicki et al., 2008] utilises a Dynamic Factory class instance to create concrete configuration classes based on information obtained from a metadata-reader class. The metadata-reader class provides metadata about the concrete classes, which in the PL context is metadata about metadata.

8.4.7.4 ANNOTATED METADATA Proto Pattern-/Idiom

This pattern/idiom is described in greater detail as it is not formally documented elsewhere so a reference to detailed discussion cannot be used. As it is specific to languages that support source code annotations to provide run-time information, it should probably be viewed as an idiom rather than a pattern. Nevertheless, in the context of the PL², it provides a much simplified means to provide configuration metadata. It is therefore documented as an optional idiom in the PL so designers planning to implement using annotation supporting languages can utilise it.

Context: A framework designed using the PL, must allow clients to define metadata about data-sources by providing standalone classes that are independent of any framework architecture, classes or interfaces.

²Annotations in general have many other uses, for example stereotyping (tagging) methods and classes, data validation, dependency injection and in aspect-oriented programming (AOP).

Problem: Typically, frameworks achieve reuse by allowing clients to provide specialised implementations of pre-defined points of variability called hotspots using, for example, TEMPLATE METHOD. When specifying metadata though, this approach can be cumbersome and unintuitive.

Solution: Let the framework user specify metadata by using annotations in the source code of a simple standalone class which has no relationship to any classes in the framework.

For example, consider defining metadata about a database table and its columns and primary key. An EXPLICIT INTERFACE based approach might provide an abstract interface for clients to override:

```
interface IEntity
{
    String tableName();
    String[] columnNames();
    SqlType columnTypes();
    String[] primaryKey();
}
```

The client application would then need to provide implementations for all tables and the framework would need to provide associated classes providing values for all the defined columns when querying the database.

An annotation based approach would allow a user to define a standalone class that uses annotations to define the metadata (metadata definition is preceded by an @ sign), for example:

```
@Table(name='person')
public class Person
{
    @Id public long id;
    @Column public String name;
    @Column(Type=SqlType.SHORT) public int age;
}
```

Several widely used languages such as Java and C# do have annotation support, while scripting languages have similar capabilities by using dynamic programming techniques. Statically compiled languages such as C++³ could emulate some annotation features by using the pre-processor, in particular the #pragma pre-processor directive (for example the C++ ORM ODB uses pragmas to annotate classes that map onto database tables).

The Enterprise Java Beans (EJB) framework [Panda et al., 2014] and Hibernate [Bauer et al., 2015]

³C++ 11 does have partial support for annotations which are known as generalised attributes.

use annotations for database persistence in Java while NHibernate [Kuaté et al., 2009] does the same for C#. JUnit [Tahchiev et al., 2011] uses annotations to simplify unit testing.

8.4.8 Concurrency Patterns

As the REQUESTOR could be accessed simultaneously by multiple clients and/or proxy/facades and the CLIENT REQUEST HANDLER, in turn, could be accessed by multiple REQUESTORS, some form of concurrency support may need to be implemented for one or both of these patterns (although it would be preferable to restrict the concurrency support to one or the other and not both). Asynchronous operation should support a single client making multiple asynchronous requests or multiple client threads making single or multiple requests.

The ACTIVE OBJECT pattern [Schmidt et al., 2000] provides a means to concurrency by decoupling method invocation from method execution. ACTIVE OBJECT provides a synchronised queue onto which method invocations are added while a scheduler running in another thread invokes the methods after removing them from the list. The default ACTIVE OBJECT uses one thread for the scheduler and the object on which the method is invoked and another for the invoker.

This configuration is suitable when there is one REQUESTOR instance per client so the individual operations from the client are executed asynchronously, but in the same order as they were submitted by the client (depending on the implementation, the scheduler can also invoke method requests in a different order from which they were received, for example, by using a priority queue). When providing a REQUESTOR which can service multiple clients, an ACTIVE OBJECT variant which uses a thread-pool for method execution can be used.

For the REQUEST HANDLER, the simplest synchronisation technique would be to implement it in a threadsafe manner, that is, do not maintain any state between method calls and only use parameters and local variables within its methods. In this way all concurrency support can be implemented in the REQUESTOR with the REQUEST HANDLER code running in the context of the calling REQUESTORS thread. If this is not possible then a MONITOR OBJECT [Schmidt et al., 2000] which ensures that only a single thread can execute within the REQUEST HANDLER can be used to synchronise entire individual methods or just sensitive sections of code with the method.

On mobile devices with possibly limited computing resources it may be desirable to be able to limit the number of threads running simultaneously. One approach to this issue involves making a decision as to the granularity of the ACTIVE OBJECT. For example by using a single global ACTIVE OBJECT instance that uses a thread pool with the desired maximum number of threads for all classes that require

concurrency support.

8.4.9 Caching

In general, the use of caching as a pattern is described in Chapter 3 of Kircher and Jain [2004]. While this pattern describes caching in general terms, the PL is concerned with caching of heterogeneous AR related resources such as AR features, 3D models, annotations, 2D images and maps and visualization for mobile devices. The mobility of the cache client as well as the heterogeneity of the cachable resources complicate cache implementations in this domain. Rathore and Prinja [2007] provides a mobile specific overview of caching, while Chan et al. [2001] details an example framework for mobile caching of databases.

8.4.9.1 Cache Implementation

The cache can be integrated at either the REQUESTOR or REQUEST HANDLER level dependent on the abstraction level of the cache. A framework could also provide an INTERCEPTOR [Schmidt et al., 2000] in the REQUESTOR/REQUEST HANDLER to allow a framework user to inject cache accessing (or other) code before doing a network request instead of using a built-in cache.

In some cases specific protocols may already have some form of caching defined. For example the HTTP protocol supports HTTP caching [Internet Engineering Task Force (IETF), 2014] as part of the protocol which may, therefore, be used either as an entire cache solution or as a first level cache.

Traditional application caches are key-value pairs with discrete-valued keys. Caching the results of spatial queries specified by a locational key such as latitude/longitude may be problematical, as most spatial queries involve a WHERE clause specifying a polygon with multiple locational keys. Attempting to develop a cache for such a scheme requires “reinventing the wheel” as the cache indexing scheme would have to be similar to the indexing scheme already implemented by spatial databases. In such a case, an idiomatic solution may be to implement the cache as a local spatial database on the mobile device by using a lightweight spatial database such as SpatiaLite [Furieri, 2013] which is available for both Android and iOS.

8.4.9.2 Cache Clustering

When querying objects within spatial regions, for example inside a given radius or within a bounding box, ping-pong effects are possible for objects on the borders of the region. Clustering such objects and retrieving entire clusters for objects within the region bounds is a solution. If the database being queried is local or otherwise under the control of the AR developer, the clustering can be done server side. For generally available spatial databases, however, this will not be the case, so clustering can be handled as

part of the caching process as the cache has access to previously queried data and can add newly queried objects to existing or new clusters.

8.4.10 Rendering Patterns based on Data Access

While rendering patterns are covered in Chapter 7, two such patterns that apply specifically to locational data and are tied to the access of location data are described here. These patterns are location specific and do not rely on vision based technologies.

8.4.10.1 POI MARKER

Context: POI representations and information must be displayed and continually updated as the user moves.

Problem: MAR artefacts display information on nearby POIs returned from datasources where these POIs are within the current field of view of the device. Keeping track of these POIs and updating their position in the virtual world increases the complexity of the artefact.

Solution: Provide two EXPLICIT INTERFACES, the first provides rendering for a POI, and the second for receiving location and visibility updates (visibility refers to whether the POI is within the current field of view of the device and within a distance range). The POI MARKER which implements these interfaces is responsible for using the location and visibility information from the update interface to choose if and how to render the POI. The MAR artefacts rendering component is then decoupled from the underlying details of POI representation, and need only request that the POIs be rendered through a generalised rendering interface. In the context of the PL, an EVENT AGGREGATOR (Section 8.4.6) can be used to provide updates for multiple POI MARKER'S through their location interface as the location changes (where the EVENT AGGREGATOR also aggregates location events provided by the mobile OS).

8.4.10.2 POI RADAR

Context: The position of nearby POIs relative to the user must be visualisable.

Problem: In many cases, for example in cities, there may be many POIs in the vicinity of the user. This can prove confusing for a user attempting to find a particular POI.

Solution: Provide a “radar” display with the device at the centre so the user can monitor her location relative to the surrounding POIs. The solution mechanics closely follows that of the POI MARKER in using an update and rendering interface, however the EVENT AGGREGATOR needs to aggregate all visible POIs and pass them to the update interface.

8.5 Pattern Language Grammar

This section will describe the how the individual patterns can be combined into valid sequences which can be used when designing artefacts in the PL solution space. Valid pattern grammar constructs will be described in terms of sequences of patterns illustrated by railroad-style syntax diagrams [Braz, 1990].

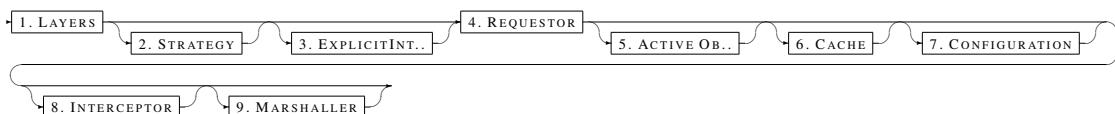
The grammar will be presented piecemeal starting with the PL entry pattern in a sequence similar to that which most design processes would take, rather than by trying to present the entire PL in a single complex diagram. To keep the diagrams readable multiple concurrency, caching and configuration patterns will be grouped together.

Most pattern languages have an initial or entry pattern where the design is assumed to start. With this PL, the initial pattern is **LAYERS** as it decides the architecture of the rest of the artefact design. The following pattern is **REQUESTOR**, because it forms the central component between the lower-level patterns such as **REQUEST HANDLER** and higher-level **FACADES** and **EVENT AGGREGATORS**, therefore, its interface defines the design of the levels below and above it. In addition the **REQUESTOR** will always appear even in the simplest artefact incarnation.

After completing **REQUESTOR** the designer can choose to complete the low level design of **REQUEST HANDLERS** and concurrency patterns in a bottom up approach or design the higher level **FACADES** and **EVENT AGGREGATORS** which provide front ends to the **REQUESTOR** in a top down approach. In the following sub-sections the PL sequences are described.

8.5.1 Bottom Up Design Sequence

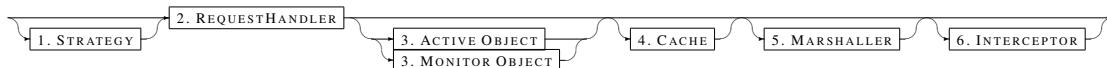
To initiate a design using a bottom up approach the following sequences can be used.



1. Apply layers for overall design.
2. If there are multiple related **REQUESTORS** which need to be interchangeable apply **STRATEGY**.
3. Optionally add a **EXPLICIT INTERFACE** if the **REQUESTOR** must adhere to a public interface.
4. Add the **REQUESTOR**.
5. Optionally add concurrency support for the **REQUESTOR** using **ACTIVE OBJECT**.
6. Optionally add caching support for the **REQUESTOR**.
7. Optionally add support for reading configuration metadata for the **REQUESTOR** by using the **CONFIGURATION PATTERNS**.

8. Optionally allow the REQUESTOR to be transparently extended when used in a framework by applying INTERCEPTOR.
9. If a REQUEST HANDLER is to be used to provide low level access to the data-source then proceed to REQUEST HANDLER design (Section 8.5.2) otherwise optionally add a MARSHALLER to directly encode/decode requests/results.

8.5.2 REQUEST HANDLER Design Sequence



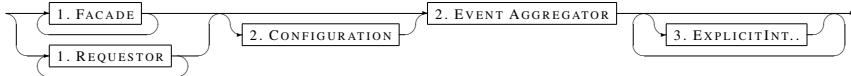
1. If there are multiple related REQUEST HANDLERS which need to be interchangeable apply STRATEGY.
2. Add REQUEST HANDLER.
3. Optionally support concurrency by using ACTIVE OBJECT or MONITOR OBJECT if it is not possible to make the REQUEST HANDLER threadsafe.
4. Optionally support caching by using CACHE.
5. Optionally support marshalling of requests and results using MARSHALLER.
6. Optionally allow the REQUEST HANDLER to be transparently extended when used in a framework by applying INTERCEPTOR.

8.5.3 Top Down Design Sequence



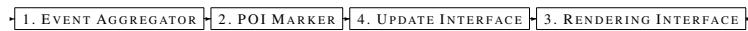
1. Add LAYERS.
2. Add REQUESTOR (see Section 8.5.1).
3. Optionally add an EXPLICIT INTERFACE if the REQUESTOR must adhere to a public interface.
4. Optionally add a FACADE.
5. Add EXPLICIT INTERFACEes to the FACADE that the REQUESTOR can asynchronously call with results and/or errors.
6. Optionally add CONFIGURATION PATTERNS to obtain configuration metadata to use when invoking methods on the REQUESTOR.

8.5.4 EVENT AGGREGATOR Design Sequence



1. Add one or more FACADEs and/or REQUESTORS to be aggregated.
2. Optionally add CONFIGURATION PATTERNS to obtain configuration metadata for the FACADE/REQUESTORS.
3. Add an EVENT AGGREGATOR which will aggregate the FACADE/REQUESTORS.
4. Optionally add EXPLICIT INTERFACEes to the EVENT AGGREGATOR that the domain level code can use for standardised access.

8.5.5 POI MARKER Design Sequence



1. Add an EVENT AGGREGATOR to provide POI information as well as locational updates.
2. Add the POI MARKER.
3. Add EXPLICIT INTERFACEes for POI updates and rendering.

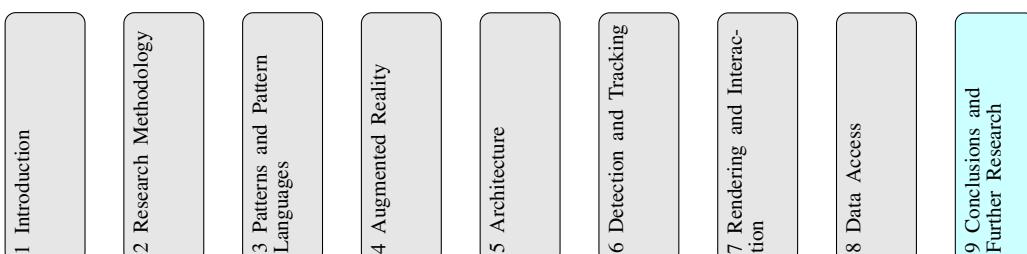
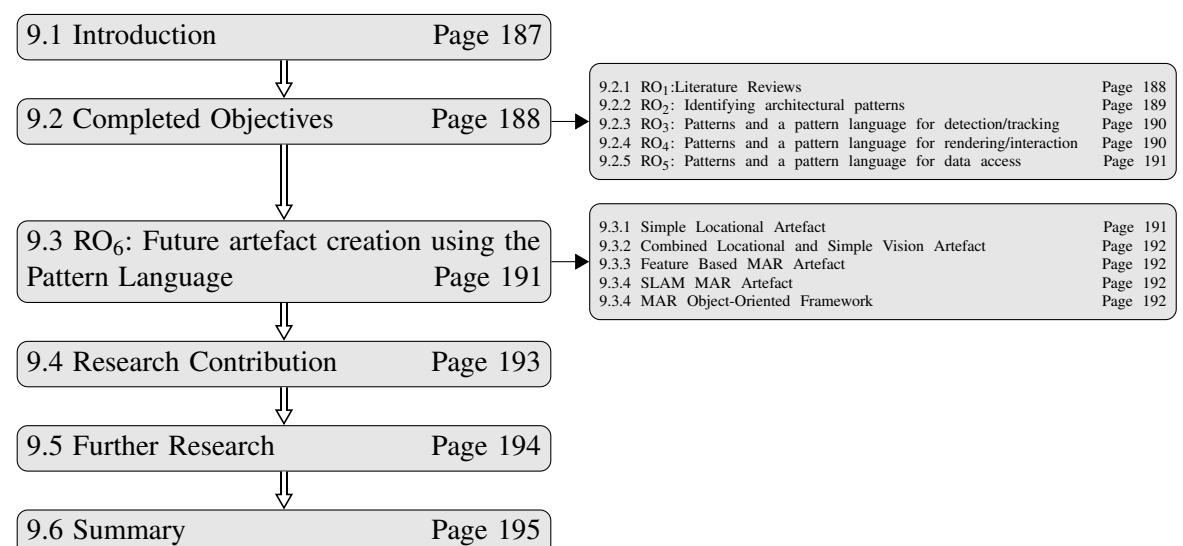
8.6 Summary

Data access for MAR applications can add a good deal of complexity to the already complex field of MAR development. The objective of this chapter has been to document a design structure in the form of a PL describing a solution space for MAR data access which a developer can customise to suit the unique requirements specific to the AR application under construction. It may be noted though, that the data access PL is generic enough to be used in many other types of applications for mobile devices.

Because the underlying model for AR data access in the PL is as a distributed system, the chapter commenced with a brief overview of client/server distributed systems in Section 8.2. Section 8.3 then described the design of the PL in terms of the problem, the forces to be address and the overall architecture being derived from existing PLs for distributed systems. The core of the chapter providing descriptions of the patterns used by the PL, that is the PL vocabulary, were covered by Section 8.4. Finally Section 8.5 linked the patterns from the PL vocabulary by defining valid sequences of patterns to be used in a design.

9

Conclusions and Further Research



Deliverable RO₆: Designate MAR software artefacts whose future design and development would cover a wide enough continuum in the solution space of the PLs to allow for both the evaluation of the PLs and probably also lead to improvements and extensions to the PLs.

9.1 Introduction

The development of Mobile Augmented Reality (MAR) software presents a significant challenge due to the highly complex nature of the components constituting a MAR artefact. These components such as visual object detection, tracking and rendering of virtual objects into the real-world, rely on composing many technologies including computer graphics, computer vision, machine learning, mobile device programming and an eclectic mix of mathematical fields. Because of the technological nature of MAR, most research concentrates on dealing with technological issues, with much less emphasis on architecture and design.

The goal of this research has been to address the high levels of complexity associated with Mobile Augmented Reality (MAR) software design and architecture through the use of patterns and pattern languages. Section 1.2 described this issue in detail and summarised the research problem to be:

The design and implementation of Mobile Augmented Reality software artefacts is difficult with minimal or no guidelines available on an overall architecture and fragmented low level guidance for individual components.

Section 1.4 provided a main research objective to address the aforementioned problem as:

RO_M: Specify one or more Pattern Languages providing a fully generalised, non-prescriptive design aid for MAR detailing and documenting the overall architecture and individual components required for various types and levels of complexity of MAR software artefacts.

Several sub-objectives were then extracted from the main objective, being:

1. **RO₁:** Conduct a literature review of Mobile Augmented Reality, Patterns and Pattern Languages;
2. **RO₂:** Identify software architectures applicable to MAR and document them as architectural patterns;
3. **RO₃:** Identify patterns and a pattern language applicable to MAR detection and tracking;
4. **RO₄:** Identify patterns and a pattern language applicable to MAR rendering and interaction;
5. **RO₅:** Identify patterns and a pattern language applicable to MAR data source access; and
6. **RO₆:** *Designate MAR software artefacts whose future design and development would cover a wide enough continuum in the solution space of the PLs to allow for both the evaluation of the PLs and probably also lead to improvements and extensions to the PLs.*

The use of Design Science Research (DSR) methodology as applied to meta-artefacts [Iivari, 2015] was combined with the research process of Nunamaker et al. [1990], with the resulting process to object-

ive mapping outlined below:

Stage	Description	Objective
Construct a Conceptual Framework	1. State research questions/objectives. 2. Understand the underlying theory (literature review).	RO ₁
Develop a System Architecture	Architectural pattern identification	RO ₂
Analyse & Design the System	Pattern identification and Pattern Language Development	RO ₃ , RO ₄ , RO ₅
Build the (Prototype) System	This stage is not applicable to a pattern language as it targets an entire solution space and not a single artefact	
Observe & Evaluate the System	Designate candidate artefacts for implementation using the pattern languages.	RO ₆

Of the above objectives, objective 1–5 have already been addressed in preceding chapters and Section 9.2 will summarise how each objective was reached. The final objective (RO₆) will be addressed in section 9.3. The research contribution of the thesis will then be described in Section 9.4, followed by Section 9.5 covering possible further research. The chapter concludes with a summary (Section 9.6).

9.2 Completed Objectives

This section will provide a summary of each research objective briefly outlining the objective and the solution. The objectives are mapped to individual chapters as follows:

Objective	Description	Chapter	Page	Subsection
RO ₁	Patterns and pattern languages literature review.	3	24	9.2.1
RO ₁	Mobile Augmented Reality literature review.	4	42	9.2.1
RO ₂	Identifying architectural patterns.	5	76	9.2.2
RO ₃	Patterns and a pattern language for detection and tracking.	6	92	9.2.3
RO ₄	Patterns and a pattern language for rendering and interaction.	7	137	9.2.4
RO ₅	Patterns and a pattern language for data access.	8	166	9.2.5

9.2.1 RO₁: Literature Reviews

Chapter 3 reviewed the literature for software reuse in general including procedural libraries (Subsection 3.2.1), Object-oriented libraries (Subsection 3.2.2), Object-oriented frameworks (Subsection 3.2.4) and Product-line Architectures (Subsection 3.2.5) before concentrating on patterns (Section 3.3). Pattern attributes and pattern interactions such as aggregation composition and sequences were discussed in greater detail based on pattern literature of which the most influential are the original “gang of four” book [Gamma et al., 1995] and the Pattern-Oriented Software Architecture (POSA) series [Buschmann et al., 1996; Schmidt et al., 2000; Buschmann et al., 2007a]. Pattern languages were then reviewed in terms of

attributes such as their domain, their use of individual patterns utilised within the context of the language forming the vocabulary while the legal pattern sequences form the grammar. Several references provided greater insight of which the final volume in the POSA series (POSA 5) [Buschmann et al., 2007a], was the most influential, while Buschmann et al. [2007b] provided a good example of a large pattern language that had already been used to build real-world applications.

Chapter 4 provided an overview of MAR research commencing by placing AR in context as part of a spectrum of varying realities called mixed reality before briefly reviewing the history of AR hardware and software (Subsection 4.2). The primary focus of the chapter however, was on covering MAR software development (Section 4.4). In this section object detection (Subsection 4.4.1), tracking (Subsection 4.4.2), pose registration (Subsection 4.4.7), Simultaneous Localisation and Mapping (SLAM) (Subsection 4.4.8), rendering (Subsection 4.4.10) and interaction (Subsection 4.4.9) were comprehensively covered by utilising a large array of up-to-date references. Various software development topics for MAR such as prototyping (Subsection 4.4.12) and existing AR frameworks (Subsection 4.4.13) including Google's ARCore and Apple's ARKit were also covered.

9.2.2 RO₂: Identifying architectural patterns

Chapter 5 differentiated good from bad or non-existent software architectures in terms of attributes such as modularity leading to separation of concerns and low coupling combined with encapsulation (Subsection 5.2.1). Next the use of architectural patterns to describe architectural styles (Subsection 5.2.2) and architectural level design decisions was introduced (Subsection 5.2.3).

The processing of streams of video and sensor data in real-time was highlighted as one of the main characteristics of MAR systems, therefore an architecture capable of dealing with multiple data streams was deemed important. Various possible architectural patterns were considered (Subsection 5.3.2) including PIPES AND FILTERS, PARALLEL PIPELINE and the Software Architecture for Immersipresence (SAI) [Fran ois, 2003] adaption of PIPES AND FILTERS, but in the end a modified version of the TASK GRAPH [Miller, 2010] pattern dubbed the STRUCTURAL TASK GRAPH pattern was chosen in conjunction with the SHARED RESOURCE pattern to provide a suitable architecture for MAR, as implemented on modern concurrency supporting hardware (Section 5.4). Some examples of libraries and frameworks such as Threading Building Blocks and OpenVX, capable of implementing such an architecture was described (Subsection 5.4.1). It was also noted that other architectural patterns could be used within individual component PLs in combination with the STRUCTURAL TASK GRAPH/SHARED RESOURCE architectural patterns, in line with the concept of a heterogeneous styled architecture [Abd-

Allah, 1996]. Finally, in Section 5.5 the relevant architectural patterns (**STRUCTURAL TASK GRAPH** and **SHARED RESOURCE**) were formally documented as architectural patterns for MAR.

9.2.3 RO₃: Patterns and a pattern language for detection and tracking

Chapter 6 was the most extensive chapter of the thesis covering probably the most complicated topics of detection and tracking which provide the underlying machinery on which AR and MAR is based. While the main thrust of the chapter was on vision based patterns as this is where the greatest complexity lies, pattern identification started with several patterns intended for locational MAR (Section 6.3) including an architectural pattern for locational MAR based on the **STRUCTURAL TASK GRAPH** architectural pattern from chapter 5.

The chapter continued with the task of identifying and documenting detection and tracking patterns for vision based MAR, starting with general purpose patterns applicable to all types of MAR. Several SLAM specific patterns were then identified. Next an architecture, again derived from the **STRUCTURAL TASK GRAPH**, was presented with example architectures for both a feature based and a SLAM MAR system. In conclusion a pattern language making use of the identified patterns was constructed. The pattern language was presented in a piecemeal fashion in the order it would be applied to reduce the complexity, with the full language also supplied separately for reference.

9.2.4 RO₄: Patterns and a pattern language for rendering and interaction

Chapter 7 provided background on how 3D rendering hardware has been increasingly influenced over the past decade by the emergence of the Graphical Processing Unit (GPU), and how this led to major changes in both 3D rendering software and later in general purpose high performance concurrent computing. High level details on 3D rendering such as geometry, shading, texturing and the graphics pipeline were introduced to provide background to the patterns introduced later.

High level architectural patterns for both interaction and rendering components were then introduced. These patterns are meant to be used at the component (rendering or interaction) level only in conjunction with the primary architecture patterns from Chapter 5 which define the overall architecture.

Next the individual rendering (Subsection 7.5) and interaction (Subsection 7.6) patterns were identified and documented. Finally in Section 7.7 a simple pattern language for rendering was presented. The pattern language however could not be as detailed as some of the other pattern languages, as which rendering patterns were utilised and in what order was very much dependent on the design of the individual renderer and particular sequences could not be prescribed without loss of generality.

9.2.5 RO₅: Patterns and a pattern language for data access

Chapter 8 described a PL for data access which, while it was documented in a MAR context, could also be used as a general pattern language for mobile locational data access. Because the PL is based on a distributed system using the BROKER pattern compound [Voelter et al., 2004], distributed systems in general and the BROKER in particular were outlined, and the reduced scope of the client only BROKER used in the PL was introduced (Section 8.3). The additional use of the LAYERS pattern and optional higher level EVENT AGGREGATOR pattern in the language architecture were also described (Subsection 8.3.3).

After providing the above-mentioned component level architectural details, the individual patterns were introduced. The patterns are a combination of client-side BROKER component patterns and various patterns identified for use within the context of the MAR pattern language (Section 8.4). Finally the pattern language itself is documented (Section 8.5), following the same piecemeal approach taken for the detection and tracking pattern language.

9.3 RO₆: Future artefact creation using the Pattern Language

Pattern languages provide a solution space:

An essential property of stand-alone patterns is that they provide a solution space for resolving a given problem, rather than just a single solution. . . . Pattern languages similarly spawn a design space for building a particular type of system, system aspect, or system part, rather than a single, fixed, one-size-fits-all solution [see Buschmann et al., 2007a, p. 285].

This solution space is much larger than that provided by a single pattern as it combines all the constituent pattern solution spaces into a whole which is larger than its parts. Because of this, exhaustively testing a pattern language is difficult. It is however possible by utilising knowledge of the domain covered by the pattern language to provide a representative list of prospective test implementations that will cover a range of the solution space varying from the simple to the generic which will be the aim of this section.

9.3.1 Simple Locational Artefact

A locational MAR artefact avoids the complexities of vision based detection and tracking, using sensor data instead. An efficient architecture is still preferred as the artefact has to deal with video and sensor data streams, however without the pressure of dealing with real-time visual detection and tracking, the overall implementation is much simplified.

A possible design for the artefact would be to start with the STRUCTURAL TASK GRAPH and

SHARED RESOURCE architectural patterns and then utilise the locational patterns described in Section 6.3 as described in the pattern language for location MAR (Subsection 6.5.1). Additionally the data access patterns and language (Sections 8.4, 8.5) could be utilised to retrieve points of interest (POI) information, for example using Google places and maps. The renderer could utilise patterns and the pattern language from Chapter 7.

9.3.2 Combined Locational and Simple Vision Artefact

In order to provide a slightly more ambitious but still relatively simple implementation, the artefact above (Subsection 9.3.1) could be extended to do simple visual detection and tracking, perhaps using fiducial markers instead of natural features for objects to detect and track. The design would be similar to that of Subsection 9.3.1, except that vision based detection patterns from Chapter 6 would also be included.

9.3.3 Feature Based MAR Artefact

A feature based MAR artefact that uses feature based object detection and tracking provides a mid-level challenge. This option also requires some form of object features database which can either be held entirely in the artefact, or be accessed using the data access pattern language based on location, with the latter approach also serving to increase the complexity. This also implies some effort is required in order to create the feature database.

A candidate design would initiate with STRUCTURAL TASK GRAPH/SHARED RESOURCE architectural patterns, followed by a pattern sequence from the detection/tracking language possibly utilising both vision and locational patterns if data access is to be used. If data access is used then the requisite patterns defined by the data access language would also be required. Finally the relevant rendering patterns could be combined with a relevant pose registration pattern (Subsections 6.4.5 or 6.4.6) from the detection language. See Figure 6.7 for a high level diagram of a possible design.

9.3.4 SLAM MAR Artefact

A SLAM MAR artefact provides another mid-level challenge, arguably at a slightly higher level than that of Subsection 9.3.3. Further options could be added to increase the complexity to high, for example making the artefact hybrid SLAM/feature based, or fusing sensor data in the pose graph to create a Visual/Inertial SLAM artefact.

Possible designs would resemble the design described in the previous Subsection (9.3.3), apart from also using SLAM specific patterns from the detection language. See Figure 6.6 for a high level diagram depicting a potential design.

9.3.5 MAR Object-Oriented Framework

Creating an Object-oriented framework [Fayad and Schmidt, 1997] as a meta-artefact enabling the development of MAR artefacts in a specific subdomain by inversion of control (also known as the Hollywood principle) would provide a high complexity challenge. The level of complexity would be governed by the subdomain of choice as well as the type of framework (black box where customisation is implemented by inheritance versus white-box where it is implemented by composition). The framework designer would be responsible for creating classes that are used by developers creating MAR artefacts, with these classes being designed using the patterns and pattern languages.

A framework created using the pattern languages is itself a meta-artefact covering a subset of the pattern language solution space. Because of this the evaluation of such a framework would itself encounter the issue of not being directly evaluable, but instead would also require a selection of candidate artefacts be developed for evaluation.

9.4 Research Contribution

The output from this thesis comprises:

1. An architecture for MAR defined by an architectural pattern compound.
2. Individual design patterns and idioms identified in the MAR sub-domains of Detection/Tracking, Rendering/Interaction and Data Access.
3. Patterns languages for all three sub-domains mentioned above.

The pattern languages cover a large solution space and may be combined or used individually to design a wide variety of MAR artefacts. From the research methodology perspective covered in Chapter 2, the output is that of a generic design theory for MAR software development (see Subsection 2.3.1.1).

As mentioned in Section 1.2, most research literature for MAR describe solutions to specific technological problems and do not address architecture and design for MAR. A search of academic web resources such as Google Scholar return very few results pertaining to design or architecture, and the results that are returned are dated. This research strives to fill this vacuum in research into design for MAR by providing a theoretically sound backbone on which MAR artefacts of varying complexities may be built.

9.5 Further Research

Pattern languages are always evolving, as underlying technologies change and new technologies are introduced. This is particularly the case for MAR, which is a young and dynamic field. Buschmann et al. [2007a] summarise this theme:

... all pattern languages, from the rawest to the most mature, should always be considered *works in progress* that are subject to continuous revision, enhancement, refinement, completion, and sometimes even complete rewriting.

Patterns and pattern languages are best viewed as community owned [see Buschmann et al., 2007a, ch. 14], so the results of this study should be made available publicly so as to enable participation by others in the MAR community. Examples of public pattern repositories include Berkeley's *Our Pattern Language*¹ and the Portland Pattern Repository².

A possible area of expansion for the detection and tracking pattern language could be in the area of dense/semi dense SLAM. As most of the current MAR implementations use sparse SLAM, and the author is also better acquainted with sparse SLAM, it is possible that more dense/semi dense SLAM patterns could be identified (in addition to the existing ones such as DIRECT 3D MAP from Subsection 6.4.10) and added to the pattern language. This may also become more important as the power of mobile devices improve to better handle the high computational requirements for the dense/semi-dense approach to SLAM when applied to high resolution frames.

As alluded to in Subsection 4.4.1.2, while neural network have come to dominate visual detection for desktop computers with powerful GPUs, the same cannot be said for mobile devices. The reasons for this, duplicated from Subsection 4.4.1.2, are:

1. The rate of miniaturisation (Moore's law) is approaching asymptote, and the cost of miniaturisation is increasing.
2. The amount of heat dissipated by GPUs is extreme, for example a typical NVIDIA 1080 based GPU has two fans and a large heat sink and frequently weighs more than the desktop motherboard it is plugged into, which makes it hard to imagine a similar powered GPU in a "mobile" device.
3. The more powerful the GPU is, the more power is required to run the GPU, which, for mobile devices means rapid battery drainage.

¹<https://patterns.eecs.berkeley.edu/>

²<https://c2.com/ppr/>

4. The amount of GPU and CPU (for transfer to GPU) memory required by the very deep CNNs used for image processing, given that mobile devices trail desktops both in quantity and speed of memory.

This means that mobile device GPUs have far less cores than desktop ones and in most cases many of those cores are already in use by the OS rendering components (and the MAR renderer if a MAR artefact is running). It is possible that this situation may change in the future, possibly through the use of FPGAs instead of GPUs, in which case the detection/tracking patterns and pattern language would require updating.

The pattern languages are currently restricted to generic mobile devices with cameras, for example cellphones and tablets. Other forms of wearable AR, for example Google Glass or Microsoft Hololens, may become more important in the future and may need to be catered for too, either by extending the existing patterns/pattern languages or adding new ones.

9.6 Summary

The chapter commenced by reviewing completed research objectives (RO₁–RO₅) by the chapter in which they were covered. Where applicable the underlying concepts used in the solution were briefly summarised, followed by details on the solution provided.

Next a solution to the single outstanding objective RO₆ was undertaken in Section 9.3. A selection of alternative artefacts of varying complexity were proposed for future work in evaluating the pattern languages. The research contribution made by the research was then outlined, followed by suggestions for future research.

Appendices



Mathematics for Computer Vision Overview

Several mathematical fields, particularly Linear Algebra and Projective Geometry, are used in various Computer Vision topics in the text. While the reader is assumed to be broadly familiar with these fields, this section reviews a few key concepts and techniques used in later sections and related references with more in-depth coverage are supplied.

A.1 Coordinates and Cameras

Coordinates such as points in 3D space and their projected locations in images are represented by vectors. For image coordinates these could be 2D pixel locations, for example (623, 125) or three component homogeneous coordinates, for example (623, 125, 1) or (934.5, 187.5, 1.5). Three dimensional coordinates are similarly represented as three component or homogeneous four component vectors. Homogeneous coordinates represent n-dimensional points with multiple corresponding points in $n + 1$ dimensions. Coordinates $1 \dots n - 1$ are then multiples of the n th coordinate, for example (mx, my, m) would be a homogeneous representation of (x, y) as would (nx, ny, n) and $(x, y, 1)$ or any scalar multiple of the vector $(x, y, 1)$. Homogeneous coordinates provide a convenient tool when dealing with 3D to 2D projections and are an important part of Projective Geometry [Goldman, 2009].

The pinhole camera model uses three coordinate systems comprising the 3D world coordinate system, the 3D camera coordinate system with its origin at the cameras' optical centre and the 2D projective image coordinate system. Extrinsic parameters map world coordinates onto camera coordinates by using rigid transforms (rotation \mathbf{R} and translation \mathbf{t}) while intrinsic parameters project the camera coordinates onto image coordinates. Intrinsic parameters are camera specific and include focal length, image origin offset and skew and are represented by the camera calibration matrix \mathbf{K} . When using the pinhole camera model the projection of a 3D world point \vec{X} to image coordinates is given by

$$\lambda \vec{u} = \lambda(u_x, u_y, 1) = \mathbf{K}(\mathbf{R}\vec{X} + \vec{t}) \quad (\text{A.1})$$

where the image coordinate \vec{u} is expressed in 3D homogeneous coordinates (see the following paragraph) with the third component being 1 and λ is the homogeneous scaling constant. Hartley and Zisserman [2004] provides an in depth description of the pinhole model.

When dealing with 2D image coordinates it is often useful to transform these coordinates into 3D camera coordinate directions (also known as rays) by multiplying the homogeneous image coordinate vector by the inverse of the calibration matrix:

$$\hat{\vec{u}} = \mathbf{K}^{-1}\vec{u} \quad (\text{A.2})$$

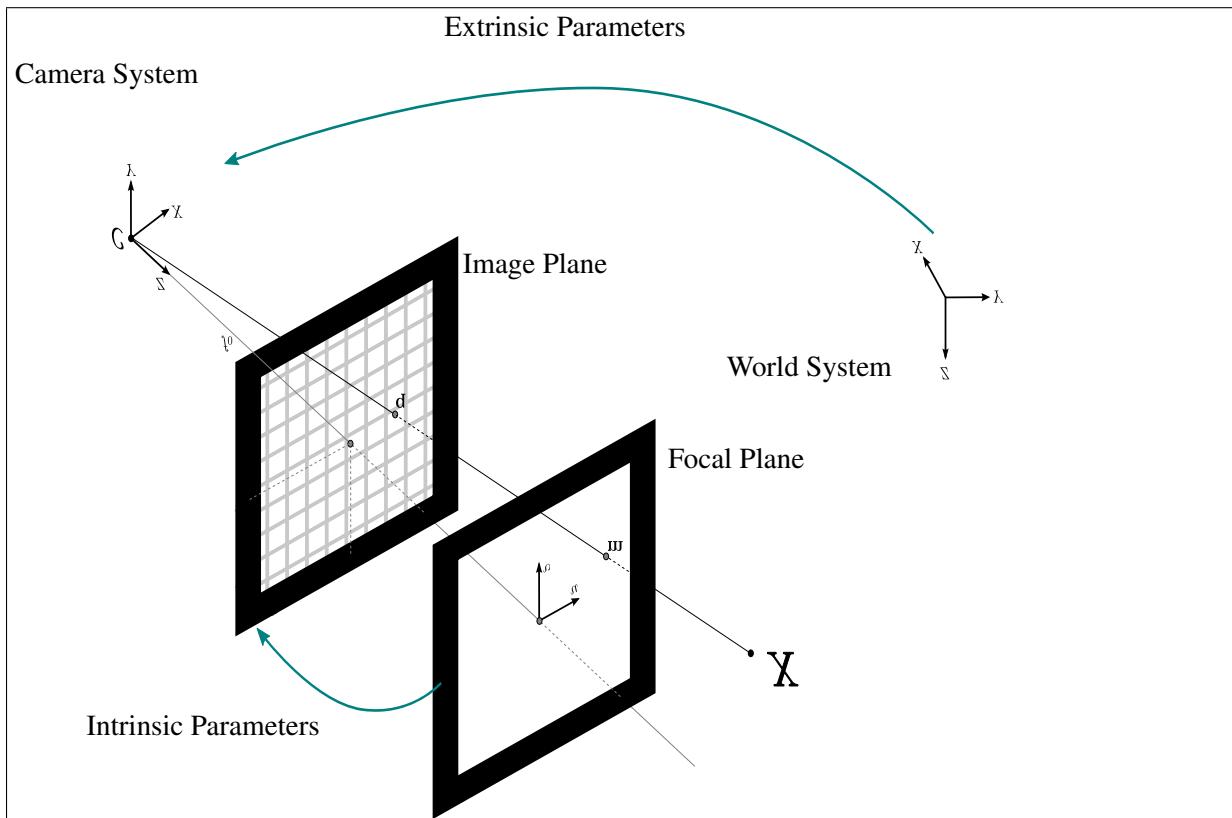


Figure A.1: Pinhole camera model (Source: <http://fa.mobivap.uva.es/~fradelg/phd/tracking/camera.html>.)

In some cases the ray also needs to be normalised to produce a unit vector.

Two vectors can be multiplied either by a standard inner product (or dot product) which produces a scalar number or a cross product which produces another vector. The vector resulting from a cross product is known as a *pseudovector* and is similar to a normal (polar) vector except when it is transformed by a transformation matrix that does not preserve orientation such as an improper rotation (The determinant $|T| < 0$ where T is the transformation matrix). It is sometimes convenient to express the vector cross product in terms of multiplication between a matrix and a vector instead of a vector-vector cross product. In order to do this the skew-symmetric matrix representation of a vector (a_1, a_2, a_3) can be used:

$$[\vec{a}]_x = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}_x = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \quad (\text{A.3})$$

A.2 Transforms and Pose

Pose describes the transformation of objects in coordinate systems in terms of rigid transforms (transforms which preserve distance between points). Such transformations can be described in terms of rotation and translation, usually in an orthogonal Cartesian coordinate system. Translation can be described as a vector \vec{t} with each component specifying the distance a point is moved in the coordinate system axis represented by the component. Rotation can be described by an orthogonal $n \times n$ rotation matrix R where n is the number of dimensions. The rotation matrix is orthogonal so $R^{-1} = R^T$ and the determinant $|R| = \pm 1$ (in the case of -1 it is an improper rotation such as a reflection). The combined rigid transform of a

3D point \vec{X} in world coordinates to camera coordinates by a rotation \mathbf{R} and translation \vec{t} is given by $\vec{x} = (\mathbf{R}\vec{X} + \vec{t})$. The projection of the point \vec{X} to image coordinates is given by Equation A.1 can also be expressed in terms of a single 4×4 matrix multiplication $\begin{bmatrix} \mathbf{R} & 0 & \vec{t} \\ 0 & 0 & 0 & 1 \end{bmatrix} \vec{X}_h$ with \vec{t} forming the first three elements of the fourth column of the matrix and \vec{X}_h being a four component homogeneous representation of \vec{X} .

Three dimensional rotation can be represented more compactly by a quaternions. Quaternions have, as their name would suggest, four components comprising a real component and three imaginary components $\mathbf{q} = (w, x\hat{i}, y\hat{j}, z\hat{k})$ with $w, x, y, z \in \mathbb{R}$. While quaternion algebra is defined in terms of imaginary numbers, the three imaginary components can also be seen as a vector, that is $\mathbf{q} = [w, \vec{v}] = [w, (x, y, z)]$.

The algebra of quaternions comprises scalar multiplication $n(w, x\hat{i}, y\hat{j}, z\hat{k}) = (nw, nx\hat{i}, ny\hat{j}, nz\hat{k})$ $n \in \mathbb{R}$, addition where corresponding components are added and multiplication.

The multiplication rules governing quaternions are defined as: $\hat{i}^2 = \hat{j}^2 = \hat{k}^2 = \hat{i}\hat{j}\hat{k} = -1$
 $\hat{i}\hat{j} = \hat{k}, \hat{j}\hat{k} = \hat{i}, \hat{k}\hat{i} = \hat{j}, \hat{j}\hat{i} = -\hat{k}, \hat{k}\hat{j} = -\hat{i}, \hat{i}\hat{k} = -\hat{j}$

The result of multiplying two quaternions after applying the above rules and expressing the result in terms of vector dot (\cdot) and cross products (\times) is:

$[w_1, \vec{v}_1][w_2, \vec{v}_2] = [w_1w_2 - \vec{v}_1 \cdot \vec{v}_2, w_1\vec{v}_2 + w_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2]$. Like matrix multiplication, quaternion multiplication is non-commutative, that is $\mathbf{pq} \neq \mathbf{qp}$.

In order to represent rotation by quaternions, the point to be rotated is represented as a pure quaternion $\mathbf{p} = [0, \vec{p}] = [0, (x, y, z)]$. A rotation quaternion of angle θ around a normalised axis of rotation \vec{a} is defined as $\mathbf{q} = [\cos \frac{\theta}{2}, \sin \frac{\theta}{2}\vec{a}]$ and the rotation is then represented by pre-multiplying the point by the quaternion and post-multiplying with the conjugate of the quaternion (The conjugate of a quaternion \mathbf{q} is defined as $\mathbf{q}^{-1} = [w, (-x, -y, -z)]$):

$$\mathbf{qpq}^{-1} = [0, (1 - \cos \theta)(\vec{a} \cdot \vec{p})\vec{a} + \cos \theta \vec{p} + \sin \theta \vec{a} \times \vec{p}]$$

The rotated point is then given by the vector part of this product. A quaternion can also be converted to a rotation matrix and vice versa, see the references specified in the next paragraph for more details.

A full discussion of quaternions is beyond the scope of this paper. An introduction from a computer graphics/vision perspective is provided by Vince [2011] while a more advanced approach with a more geometrically oriented re-derivation of quaternion operations can be found in Goldman [2010]. A comprehensive reference on the various methods of representing rotation is provided by Diebel [2006].

A quaternion represents rotation only. In order to do a full rigid transform the vector resulting from the quaternion rotation calculation has to be added to the translation vector. An alternate approach that allows for the full rigid transform to be represented by quaternions is the dual quaternion [Jia, 2013].

A dual quaternion is composed of two normal quaternions $\eta = \mathbf{p} + \epsilon \mathbf{q}$ where \mathbf{p} and \mathbf{q} are quaternions and ϵ is the dual unit [Klawitter, 2014] with $\epsilon^2 = 0$. Dual quaternion algebra include scalar multiplication $(s(\mathbf{p} + \epsilon \mathbf{q})) = s\mathbf{p} + s\epsilon \mathbf{q}$, addition $((\mathbf{p}_1 + \epsilon \mathbf{q}_1) + (\mathbf{p}_2 + \epsilon \mathbf{q}_2)) = ((\mathbf{p}_1 + \mathbf{p}_2) + \epsilon(\mathbf{q}_1 + \mathbf{q}_2))$ and multiplication $((\mathbf{p}_1 + \epsilon \mathbf{q}_1) \times (\mathbf{p}_2 + \epsilon \mathbf{q}_2)) = \mathbf{p}_1 \mathbf{p}_2 + \epsilon(\mathbf{p}_1 \mathbf{q}_2 + \mathbf{q}_1 \mathbf{p}_2)$. Dual quaternions have no less than three conjugates defined, but for the purposes of rigid transform representation only $\eta^\diamond = \mathbf{p}^{-1} - \epsilon \mathbf{q}^{-1}$ is required.

In order to represent a rigid transform we start by defining a rotation quaternion $\mathbf{q}_r = [\cos \frac{\theta}{2}, \sin \frac{\theta}{2}\vec{a}]$ and a dual quaternion representation of the rotation with a zero real part and the rotation quaternion for the dual part $\eta_R = \mathbf{q}_r + \epsilon[0, \vec{0}]$. Next a translation quaternion is defined with a zero real component and the translation for the vector component: $\mathbf{q}_t = [0, \vec{t}]$ and a corresponding translation dual quaternion $\eta_t = [1, \vec{0}] + \epsilon \mathbf{q}_t$. A rigid transform can then be represented by multiplying these dual quaternions: $\eta_T = \eta_t \eta_R = \mathbf{q}_r + \frac{\epsilon}{2} \mathbf{q}_t \mathbf{q}_r$.

To transform a point using a rigid transform dual quaternion, the point \vec{p} is represented as a quaternion $\mathbf{p}_q = [0, \vec{p}]$ and then a dual quaternion $\eta_p = 1 + \epsilon \mathbf{p}_q$. The transform is then accomplished in

much the same way it is for normal quaternions:

$$\eta_T \eta_p \eta_T^\diamond = [1, \vec{0}] + \epsilon(\mathbf{q}_r \mathbf{p}_q \mathbf{q}_r^* + \mathbf{q}_t) \quad (\text{A.4})$$

The vector part of the dual part of the resulting dual quaternion is then the transformed point.

A.3 Epipolar Geometry

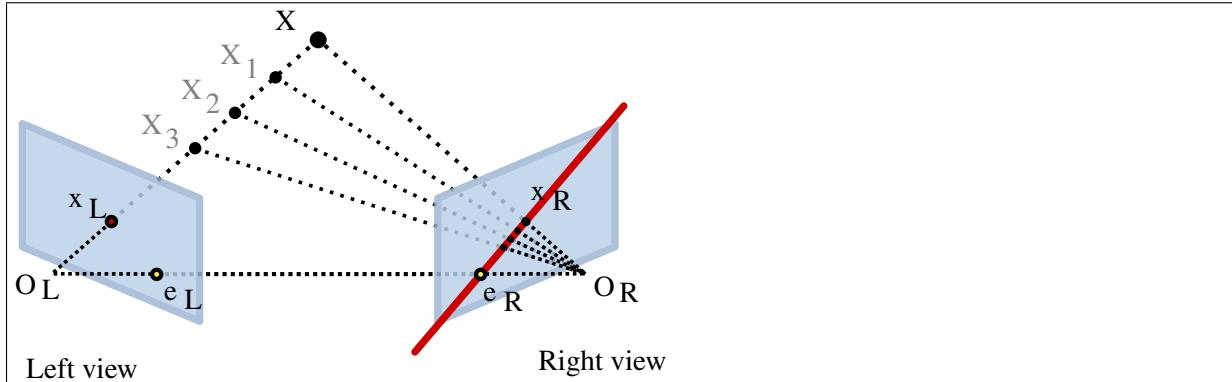


Figure A.2: Epipolar geometry for planar pose (Source: Arne Nordmann https://en.wikipedia.org/wiki/Epipolar_geometry)

The relative planar pose between two different images of a scene or two simultaneous images from two cameras in a two camera stereo system are elegantly related using epipolar geometry. Given left \vec{x}_L and right \vec{x}_R projections of a 3D point, then the relationship between the two points can be expressed as the epipolar constraint:

$$\vec{x}_R^T \mathbf{E} \vec{x}_L = 0 \quad (\text{A.5})$$

where \vec{x}_R and \vec{x}_L are the rays (Equation A.2) corresponding to \vec{x}_R and \vec{x}_L and \mathbf{E} is the *Essential matrix* $\mathbf{E} = [t]_x \mathbf{R}$ (using Equation A.3 to express the cross product of translation and rotation as a matrix multiplication). Referring to Figure A.2, assume the origin of the coordinate system is at the first image camera centre O_L , then the pose of the second image relative to the first one is the rotation and translation between O_L and the second image camera centre at O_R (in the figure there is only translation). The epipoles e_L and e_R are the intersections of the line joining O_L and O_R and the images. Assume \vec{X} is a 3D point visible to both cameras and \vec{x}_L and \vec{x}_R are the image projections of \vec{X} , however only \vec{x}_L is known. O_L , O_R and X form a 3D plane, and the line of intersections of this plane with the right hand image form the epipolar line with the points on the line corresponding to rays through all the possible image projections for \vec{x}_R . Equation A.5 is known as the epipolar constraint and provides an important tool when estimating image relative pose. See [Hartley and Zisserman, 2004] for derivations and a description of the related Fundamental matrix.

A.4 Optimisation

A.4.1 RANSAC

A commonly used optimisation method in CV that may be used in feature point matching and pose refinement for example, is known as RANdom Sample Consensus (RANSAC) [Fischler and Bolles, 1981]. A RANSAC based pose estimate process iteratively eliminates outliers and bad matches by fitting points to a model. The process randomly selects the minimum number of points required by the model and uses these points to generate a solution. The error using that solution is then calculated and points

that fall within a given error distance are added to a set of inliers while the rest are added to the outlier set. Record is kept of the result with the most inliers. The process is repeated iteratively until the inlier size exceeds a threshold or the maximum number of iterations is exceeded. The maximum number of iterations can also be dynamically updated using $P = I/N$ with P being the current probability, I the current inlier count and N the number of samples with $\text{iterations} = \frac{\log(1-P)}{\log(1-p^N)}$ where p is the probability that some given point is an inlier. A more detailed description of the algorithm can be found in §4.7.1 of Hartley and Zisserman, 2004.

A.4.2 Levenberg-Marquardt

Various CV tasks such as pose estimation and bundle adjustment use numerical methods from traditional mathematical optimisation techniques. One of the most common method used is called Levenberg-Marquardt which itself is derived from the Gauss-Newton method which can be applied to over-determined multivariate least squares problems. The Gauss-Newton method utilises the constraint that the function to be minimised is a difference of squares to approximate the second derivative matrix (the Hessian \mathbf{H}) in terms of the first order matrix (the Jacobian \mathbf{J}) $\mathbf{H} \approx 2\mathbf{J}^T\mathbf{J}$ leading to an iterative process:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{s}^k \text{ with } \mathbf{s}^k = -(\mathbf{J}^T(\tilde{\mathbf{x}})\mathbf{J}(\tilde{\mathbf{x}}))^{-1}\mathbf{J}^T(\tilde{\mathbf{x}})\mathbf{r}(\tilde{\mathbf{x}})$$

where $\mathbf{r}(\tilde{\mathbf{x}})$ is the residual (least square difference) function. Levenberg-Marquardt improves on the Gauss-Newton approach by combining it with gradient descent (a simpler first order method where each iteration is in the direction of negative gradient) after scaling the gradient according to the curvature (obtained from the second order Hessian approximation) leading to the following iteration update:

$$\mathbf{s}^k = -[\mathbf{J}^T(\tilde{\mathbf{x}})\mathbf{J}(\tilde{\mathbf{x}}) + \lambda \text{diag}(\mathbf{J}^T(\tilde{\mathbf{x}})\mathbf{J}(\tilde{\mathbf{x}}))]^{-1}\mathbf{J}^T(\tilde{\mathbf{x}})\mathbf{r}(\tilde{\mathbf{x}})$$

where diag denotes the diagonal of a matrix and λ is a constant determining the step size (the learning rate in machine language terms) which is dynamically updated while the algorithm is executing. For a more in depth description of the Levenberg-Marquardt algorithm see Moré [1978] and Treiber [2013].

Optimisation in the context of pose estimation can be seen as constrained optimisation if the rotation is being optimised. This is because there is a constraint imposed that the rotation matrix must always be orthogonal (or the rotation quaternion must always be a unit quaternion). This can make optimisation difficult and have led to methods to reduce the problem to unconstrained optimisation using Lie Algebras to represent infinitesimal rotation on a smooth manifold [Eade, 2017; Tapp, 2016].



Gravity2Gravity Relative and Absolute Pose

This appendix will describe an algorithm for a gravity assisted PnP solution that utilises extra information available from an AR database for an object that has been identified using object detection techniques. The extra information assumed to be held in the database is the gravity vector at the time when the training image was captured and optionally the depth to the centre of the predominant plane of the object (this approach would work best on planar or close to planar objects that are close enough to measure). In the case of relative pose storing depth allows the approximation of the translation as opposed to many other relative pose algorithms which only return translation up to an unknown scale factor. The algorithm uses the rotation between the stored and current gravity vector to find the rotation and the depth to approximate translation.

The accuracy of any method utilising gravity vectors relies on the underlying accuracy of the IMU. On most current devices the gravity vector is obtained by filtering readings from an accelerometer sensor. Android for example uses a Butterworth filter [Cohen Tenoudji, 2016] to calculate both a gravity vector and a separate linear acceleration vector where the gravity component is removed from the fused accelerometer and gyroscope readings. The filtering process does introduce some latency, however the sampling rate of modern MEMS sensors is high enough to compensate for most applications. For example the LSM6DSL MEMS chip used by current Samsung devices has a maximal data rate of 6664 Hz resulting in the filter receiving a reading every 0.15 milliseconds, which should be fast enough, even when including any filter latency, to provide several readings per video frame.

A study on the accuracy of the gravitational component extraction from accelerometers in real-life situations such as medical biofeedback have also found that “static angle errors due to accelerometer noise and drift are practically negligible” [Kos et al., 2016].

B.0.1 Rotation between gravity vectors

Let the stored gravity vector be $\vec{G}_t = (g_{tx}, g_{ty}, g_{tz})$ and the incoming video frame gravity vector be $\vec{G}_q = (g_{qx}, g_{qy}, g_{qz})$. The rotation between the stored image and the incoming image can be determined by finding the shortest arc rotation between \vec{G}_t and \vec{G}_q as a rotation quaternion (that is applying the quaternion to \vec{G}_t results in \vec{G}_q). The procedure, which handles small angle deltas that occur when processing real-time video frames correctly, is described in Melax [2000] and will be briefly outlined.

Assume that \vec{G}_t and \vec{G}_q have been normalised. Let

$$\vec{G}_t \times \vec{G}_q = (c_x, c_y, c_z) = |\vec{G}_t||\vec{G}_q| \sin \theta \vec{n} = \sin \theta \vec{n} \quad (\text{B.1})$$

be the cross product of \vec{G}_t and \vec{G}_q (\vec{n} is a unit vector $\perp G_t$ and G_q) and

$$d = \vec{G}_t \cdot \vec{G}_q = |\vec{G}_t||\vec{G}_q| \cos \theta = \cos \theta$$

be the standard inner product of \vec{G}_t and \vec{G}_q where θ is the (unknown) angle between \vec{G}_t and G_q . We also have that the rotation quaternion is of the form

$$\mathbf{q} = [\cos \frac{\theta}{2}, \sin \frac{\theta}{2}(q_x, q_y, q_z)] \quad (\text{B.2})$$

then because equations B.1 and B.2 are both normal to the plane of \vec{G}_t and \vec{G}_q

$$\frac{(q_x, q_y, q_z)}{\sin \frac{\theta}{2}} = \frac{(c_x, c_y, c_z)}{\sin \theta} \implies (q_x, q_y, q_z) = (c_x, c_y, c_z) \frac{\sin \frac{\theta}{2}}{\sin \theta}$$

By employing the substitutions $\sin \frac{\theta}{2} = \sqrt{\frac{1-\cos \theta}{2}}$, $\sin \theta = \sqrt{1 - \cos^2 \theta}$ and $d = \cos \theta$ we get

$$\begin{aligned} \frac{\sin \frac{\theta}{2}}{\sin \theta} &= \frac{\sqrt{(1-d)}}{\sqrt{2}\sqrt{1-d^2}} = \sqrt{\frac{1-d}{2(1+d)(1-d)}} = \frac{1}{\sqrt{2(1+d)}} \\ \therefore (q_x, q_y, q_z) &= \frac{(c_x, c_y, c_z)}{\sqrt{2(1+d)}}, q_w = \cos \frac{\theta}{2} = \sqrt{\frac{1+d}{2}} = \frac{\sqrt{2(1+d)}}{2} \end{aligned}$$

where both expressions are simplified to contain $\sqrt{2(1+d)}$, which therefore only needs to be calculated once. The implementation of this procedure also needs to handle special cases, see for example `getRotationTo` in OGRE3D Team [2018]. MATLAB 2015 and above also implement this method as “`vrrotvec`” and the Eigen Linear Algebra library also has an implementation.

B.0.2 Relative translation up to scale

All relative pose finding techniques are affected by the depth/speed scaling ambiguity resulting from the inability to discern the difference between distant large objects moving rapidly and nearby small objects moving slowly. Without extra information the best that can be done is to derive a translation unit vector using techniques such as the epipolar constraint. For example, if the translation is $(10, 5, 0)$ then the unit vector translation would be $(0.89443, 0.44721, 0)$ and

$$11.18034(0.89443, 0.44721, 0) = (10, 5, 0).$$

The rotation obtained as in Section B.0.1 is converted to a rotation matrix \mathbf{R} . Let $\vec{t} = (t_x, t_y, t_z)$ be the translation vector then the essential matrix is the product of the skew symmetric matrix form of \vec{t} and \mathbf{R} :

$$\mathbf{E} = [t]_x \mathbf{R} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} r_{20}t_y - r_{10}t_z & r_{21}t_y - r_{11}t_z & r_{22}t_y - r_{12}t_z \\ -r_{20}t_x + r_{00}t_z & -r_{21}t_x + r_{01}t_z & -r_{22}t_x + r_{02}t_z \\ r_{10}t_x - r_{00}t_y & r_{11}t_x - r_{01}t_y & r_{12}t_x - r_{02}t_y \end{bmatrix}$$

where r_{xy} are components of the rotation matrix \mathbf{R} . The epipolar constraint $\vec{x}_q^T E \vec{x}_t = \vec{0}$ where $\vec{x}_q = (x_q, y_q, 1)$ is a known point in the query frame and $\vec{x}_t = (x_t, y_t, 1)$ is a known point in the training

image¹, may then be used to ascertain the coefficients of t_x , t_y and t_z for a homogeneous system of linear equations:

$$\begin{aligned} \vec{x}_q^T E \vec{x}_t &= at_x + bt_y + ct_z = 0 \\ a &= (r_{12} + r_{10}x_t + r_{11}y_t - r_{22}y_q + (-r_{20}x_t - r_{21}y_t)y_q) \\ b &= (-r_{02} - r_{00}x_t + r_{22}x_q + r_{20}x_t x_q - r_{01}y_t + r_{21}x_q y_t) \\ c &= (-r_{12}x_q - r_{10}x_t x_q - r_{11}x_q y_t + (r_{02} + r_{00}x_t + r_{01}y_t)y_q) \end{aligned}$$

Solving the system numerically using the Singular Value Decomposition (SVD) yields the unit translation. A minimum of three correspondences are required as there are three unknowns.

B.0.3 Relative pose with full translation approximation

In order to find the real translation, additional information is required, such as the actual distance between two detected points or the depth of the points. It is assumed that we are dealing with semi-planar objects, therefore the measured depth from the camera to the plane of the training object can serve as an approximation which is not too onerous to measure. The feature database would then contain the image features, the gravity vector and the measured training image depth to the plane of the object. It is however important that the angle to the plane of the training object is as close to perpendicular as possible so that the measured depth (Z coordinate) is as close as possible to being the same for all points on the plane.

We assume we have calculated the rotation quaternion $\mathbf{q} = [q_w, (q_x, q_y, q_z)]$ as in Section B.0.1 from which we can obtain the rotation matrix \mathbf{R} . Let $\vec{p}_0 = (x_0, y_0, 1)$ be a training image feature point ray with $\vec{p}_1 = (x_1, y_1, 1)$ the corresponding query image feature point ray in the camera coordinate system. We also have d_0 the measured depth to the training object plane and d_1 the unknown depth of the query object plane. The rigid transform

$$d_1 \vec{p}_1 = \mathbf{R} d_0 \vec{p}_0 + \vec{t} \quad (\text{B.3})$$

transforms \vec{p}_0 to \vec{p}_1 where $\vec{t} = (t_x, t_y, t_z)$ is the translation. Taking the cross product of both sides of (B.3) with \vec{p}_1 :

$$(\mathbf{R} d_0 \vec{p}_0 + \vec{t}) \times \vec{p}_1 = 0 \quad (\text{B.4})$$

Expanding equation (B.4) and moving t_x, t_y, t_z to the left hand side and the rest of the terms to the right:

$$\begin{bmatrix} 0 & 1 & -y_1 \\ -1 & 0 & x_1 \\ y_1 & -x_1 & 0 \end{bmatrix} = \begin{bmatrix} d_0(r_{20}x_0y_1 + r_{21}y_0y_1 - r_{10}x_0 - r_{11}y_0 + r_{22}y_1 - r_{12}) \\ -d_0(r_{20}x_0x_1 + r_{21}x_1y_0 - r_{00}x_0 + r_{22}x_1 - r_{01}y_0 - r_{02}) \\ d_0(r_{10}x_0x_1 + r_{11}x_1y_0 - r_{00}x_0y_1 - r_{01}y_0y_1 + r_{12}x_1 - r_{02}y_1) \end{bmatrix} \quad (\text{B.5})$$

where $r_{00}...33$ are the zero-base indexed components of \mathbf{R} . The equations in (B.5) form a linear system which can be solved numerically to obtain the translation using three or more correspondences.

B.0.4 Absolute Pose

The 3D PnP problem uses correspondences between points in a 3D model and image coordinates in the query image to estimate pose. With various hardware assisted point cloud capable devices and APIs becoming available such as Google Tango devices, Intel RealSense cameras and ZED cameras, the ability to record 3D model pointclouds is becoming available at a reasonable cost. From an AR database perspective, the gravity vector associated with the recording of a pointcloud could again be stored along with the pointcloud for use in object recognition and pose estimation.

The rotation quaternion $\mathbf{q} = [q_w, (q_x, q_y, q_z)]$ is again obtained as in Section B.0.1 from which we

¹The 2D points are assumed to have been converted to unit norm rays (Equation A.2)

can obtain the rotation matrix \mathbf{R} . Let $\vec{p} = (x_1, y_1, 1)$ be an image ray in the query image (See equation A.2) and $\vec{P} = (X_1, Y_1, Z_1)$ be the corresponding 3D point from a 3D model in the feature database and $\vec{t} = (t_x, t_y, t_z)$ be the unknown translation. The full transform is then given by $\vec{p} = \mathbf{R}\vec{P} + \vec{t}$. Taking the cross product of both sides with \vec{p} results in $(\mathbf{R}\vec{P} + \vec{t}) \times \vec{p} = \vec{0}$. Keeping the unknown translation on the left and moving the known terms to the right we get:

$$\begin{bmatrix} 0 & -1 & y_1 \\ 1 & 0 & -x_1 \\ -y_1 & x_1 & 0 \end{bmatrix} = \begin{bmatrix} -X_1r_{20}y_1 - Y_1r_{21}y_1 - Z_1r_{22}y_1 + X_1r_{10} + \\ Y_1r_{11} + Z_1r_{12} \\ \dots \\ X_1r_{20}x_1 + Y_1r_{21}x_1 + Z_1r_{22}x_1 - X_1r_{00} - \\ Y_1r_{01} - Z_1r_{02} \\ \dots \\ -X_1r_{10}x_1 - Y_1r_{11}x_1 - Z_1r_{12}x_1 + X_1r_{00}y_1 + \\ Y_1r_{01}y_1 + Z_1r_{02}y_1 \end{bmatrix}$$

where $r_{00\dots 33}$ are the zero base indexed components of \mathbf{R} .

B.1 Implementation

This section describes some specific detail which may be required in order to implement the pose methods. Subsection B.1.1 describes steps and guidelines that may be followed when creating the software code. A reference implementation is available at <https://github.com/donaldmunro/Gravity2Gravity>.

B.1.1 Programming Details

In all cases the minimum number of correspondences required is three so the algorithms are well suited to be implemented using RANSAC (as described in Section A.4.1). The estimated pose can also be further refined using the Levenberg-Marquardt method. The refinement can be done only for translation, assuming that the IMU rotation is correct (within reasonable error bounds), resulting in an unconstrained optimisation which is much simpler to implement than the constrained optimisation required when optimising rotation. Alternately the refinement can be done for both rotation and translation, possibly using a quaternion specific approach such as the one described by Schmidt and Niemann [2001].

All three methods find the translation by solving a linear system numerically. These systems may be solved using different techniques supplied by Linear Algebra libraries. The techniques provide different tradeoffs between speed and accuracy with SVD based methods such as the JacobiSVD class provided by the Eigen library being the most accurate while QR decomposition, particularly the Householder reflection method provides a good balance between speed and accuracy Lee [2012].

Implementors of the depth approximation method described in Section B.0.3 should take note of equation B.3, in which the point to be projected is converted to an image ray and then multiplied by the known training image depth obtained from the AR database before being multiplied by the rotation matrix. An example for projecting a point would then read:

```

input :K the camera calibration matrix
      image coordinate  $\vec{u}$  eg (640,480)
      depth
output :Projected point
 $\vec{r} \leftarrow \mathbf{K}^{-1}(u_x, u_y, 1);$ 
 $\vec{r} \leftarrow \frac{\vec{r}}{\vec{r}(3)};$  // Divide by homogeneous coordinate
 $\vec{r} \leftarrow \vec{r} * \text{depth};$ 
return  $\mathbf{K}(\mathbf{R}\vec{r} + \vec{t});$ 

```

B.2 Verification

B.2.1 Planar Relative Pose

The two planar 2D methods are tested using both synthesised and real image data. For the depth assisted method where translation is approximated, the pose results are compared with OpenCV PnP estimates for the same models using measured 3D data for the model.

B.2.1.1 Synthesised data

Training points used in the synthesised tests are generated randomly (for the planar tests the depth or Z axis remains constant for a given test but is generated randomly per test). These points are then transformed by the test rotations and randomly generated translations to create corresponding query points. The rotations are determined by the two gravity vectors used in each test, therefore, in order to avoid using the same code used in the pose calculations to obtain the test rotations, several rotations based on predefined known rotation between gravity vectors are used in the initial tests. For example a rotation between gravity vector $(0, 9.80665, 0)$ and $(9.80665, 0, 0)$ is known in Euler angles to be a roll and pitch of zero and a yaw of ninety, so a rotation matrix or quaternion can be constructed from the Euler angles.

The pose algorithm is then applied to the generated matches. For the depth assisted method the reprojection error between the resulting points and the generated query points are calculated. The reprojection result for the initial tests was in the region 1×10^{-14} , that is zero within floating point error bounds. For the non-depth assisted method, reprojection is not feasible as the translation is to scale and not absolute. In this case the same test data as used above in the depth case but with known non-random translations is used. The resulting translation should be the normalised vector of the translation vector returned from the depth supplied algorithm. For example one of the test cases resulted in a translation of $(-10, -5, 3)$ where depth was supplied and the translation as determined by the the non-depth algorithm using the same data was $(-0.863868, -0.431934, 0.259161) = \frac{(-10, -5, 3)}{\|(-10, -5, 3)\|}$.

Additionally in order to extend the synthetic tests to some more extreme rotations, the above procedure of selecting predefined gravity vectors to generate rotations was bypassed for later tests, with the rotation between vectors being calculated externally using MATLAB. Using this procedure a few instances with extreme angles encountered during testing with a device held in unconventional positions resulted in slightly larger errors, although the reprojection errors were still well below 1 pixel. For example the rotation from $(-0.001095810, 7.897157669, 5.814230919)$ to $(-1.700023532, 6.482434750, 7.159492493)$ resulted in a maximum error of 0.0143 and a mean error of 0.0098.

In order to further expand the test coverage multiple random vectors having norms of 9.8 were also generated, and the rotations were generated from pairs of these vectors using Eigen's `setFromTwoVectors` method to set the rotation quaternion. These rotations were combined with random translations to create further synthetic tests. Again all the the results were close to zero.

The reference implementation includes the synthetic tests in the source file `src/synth2d.cc`.

B.2.1.2 Image Data

In order to test the algorithm with real images an experiment using a 3D model and several A4 pages were used. Measurements (in metres) were made of an object used as a 3D model, in this case a clock/penholder. The clock base provides a planar rectangle for use with the planar algorithms while other non-planar points are also defined for use by PnP style methods. The origin of the world coordinate system is set at the top left corner of the base and the coordinate system itself is right handed with the positive Y axis vertically down, X to the right and Z positive away from the observer (see Figure B.1a). The camera used is an Asus Zenfone AR, which, as a Google Tango device, has its calibration parameters accessible via the Tango API (the calibration parameters used are $f_x = 1479.832887819$, $f_y = 1477.564891301$, $c_x = 976.255980955$

APPENDIX B: GRAVITY2GRAVITY RELATIVE AND ABSOLUTE POSE

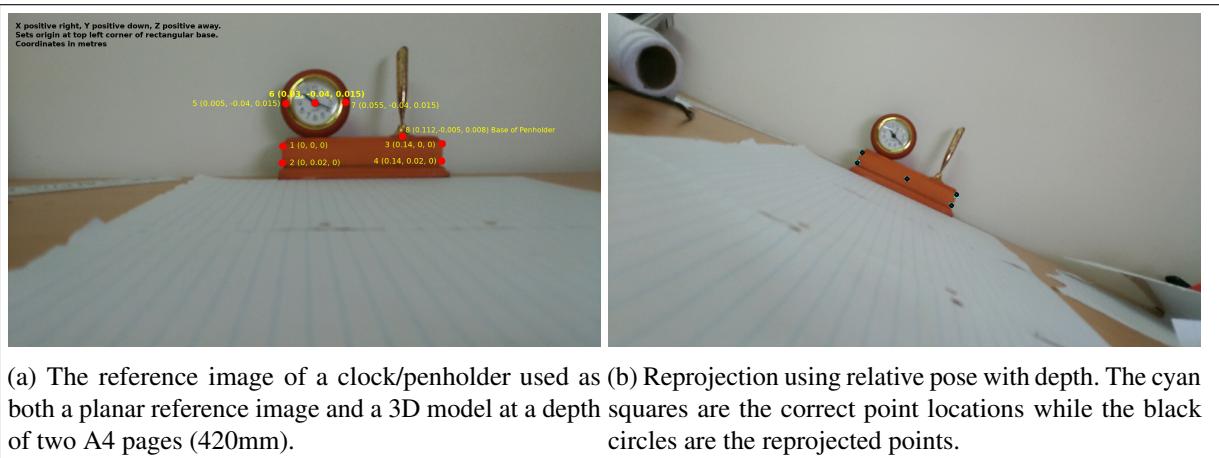


Figure B.1: Reference and reprojeciton images.

and $c_y = 538.286875345$ with images of size 1080×1920 pixels). The gravity vectors for the training image and the query images were also captured by using an application that records the gravity vector at the time the image is recorded.

A4 pages have dimensions of $210\text{mm} \times 297\text{mm}$ and are used to easily define and show the depth or Z coordinates. The reference image is with the camera in the centre and a depth of two A4 pages (420mm). Different images are taken at both extremes of the page, both level and rotated, while testing at different depths is done by using a distance of one or three A4 pages representing nearer or further scenarios respectively. Points matching the marked points in the reference image were manually defined, with the mappings specified in text files that serve as input to the test program (the manual matching is probably slightly noisy as human error in marking the matches is possible, providing a further test of the pose estimation). In order to evaluate the pose the reprojection error was measured and also visualised by drawing original query points and reprojected points onto the query image. One shortcoming of the experiment is that there is no easy way to standardise the rotations used in the test images so the rotations are relatively random within a range of $\pm 15^\circ$ to $\pm 25^\circ$.

In order to provide some comparison, the OpenCV PnP API which provides a front end to various PnP pose algorithms was used on the same data (although different text files mapping 3D points to points in the query image were used and additional correspondences were added as some PnP algorithms are not compatible with co-planar points). In particular the accurate but slow iterative Levenberg-Marquardt method, as implemented in OpenCV was used. It should be noted that the comparison is not perfect in that relative pose measures pose between images while PnP pose measures pose between an image and a 3D model, nonetheless the results were similar after taking into account that the relative pose is specified in terms of the reference image, while for 3D PnP it is relative to the 3D model location.

To illustrate the process an example is described. In this example the reference image is compared to a “further” image, that is a depth of 630mm (three A4 pages), translated to the right and rotated. The results for a rotation between gravity vectors $(0.0982511, 9.74825, 1.06408)$ and $(4.2009, 8.85038, 0.440082)$ with a training image depth of 0.42m (the query image depth of 0.63m is an unknown) was a rotation in Euler angles of Roll 3.63602° , Pitch -1.9699° and Yaw 24.7282° and a translation of $(-0.0430363, 0.0269885, 0.220236)$ (the Z translation is relative to the reference image, that is $\approx 0.63 - 0.42$ or 0.21). The maximum reprojection error was 2.14697 pixels and the mean error was 1.12551. The OpenCV iterative PnP solution was a rotation of Roll 172.367° , Pitch -170.638° and Yaw -156.408° with a translation of $(-0.0624, -0.0390, 0.6147)$ (Z relative to the model) resulting in a maximum error of 7.60276 and a mean error of 4.63662.

Note the apparent discrepancy between the rotations is due to the same 3D rotation being expressible in multiple ways; one way of comparing rotations is to multiply the quaternion of the first rotation with the conjugate of the quaternion of the second² and see how close to 1 the real (w) component of the resulting quaternion is (in this case it is 0.9595 so the rotations are close to being the same). See Figure B.1b for an illustration of the reprojection errors.

The rest of the results are summarised in Tables B.1 and B.2 (abbreviations GP = Gravity2Gravity method, It = OpenCV Iterative Levenberg-Marquardt Method) with the average reprojection error of the gravity based method (1.2547) comparing favourably to that of the OpenCV iterative PnP method (2.3680). The tests can be run using a command line interface of the reference implementation, with a script to run the tests available in the directory *test-planar/* along with all the data and image files used.

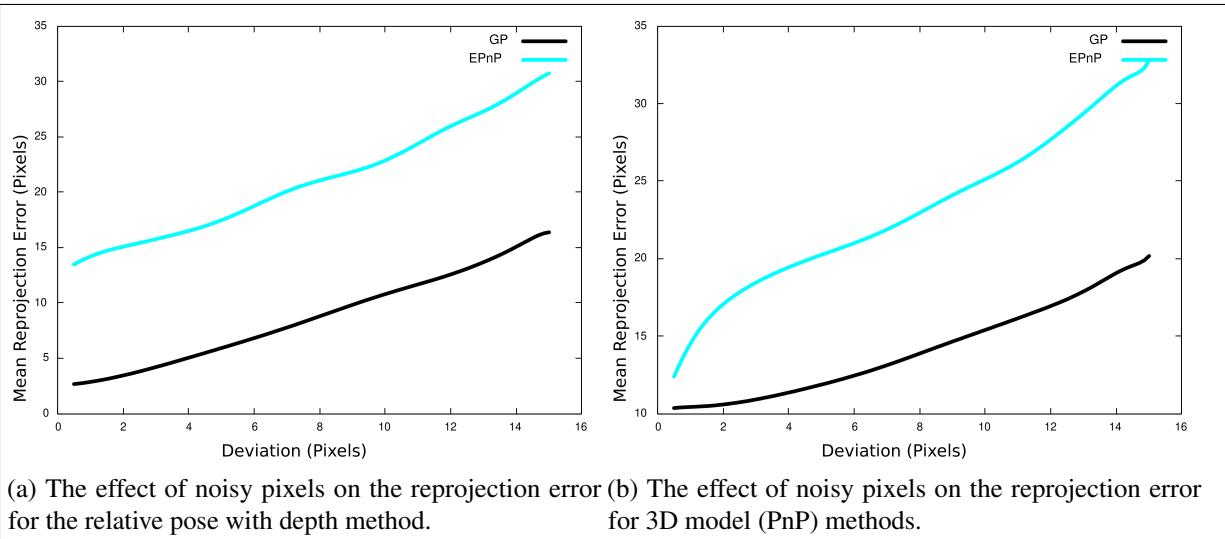


Figure B.2: The effects of pixel noise.

Another metric of interest is resistance to pixel noise (random deviations from the “correct” location). In order to test this the real-world image data was again used, instead of synthesised data in order to provide more insight into the effects of noise on real images requiring calibration data for projection. In the test Gaussian noise was added to all query image points, with the deviation σ ranging from 0.5 to 15 pixels. The results are summarised in Figure B.2a (the abbreviations are the same as in Table B.1 except the OpenCV EPnP Lepetit et al. [2009] method was used as the iterative method did not seem to handle noise on all pixels well).

B.2.2 3D Model (PnP) Pose

The 3D method was tested using both synthesised and real image data. In the latter case the reprojection errors were also compared to those of the OpenCV iterative PnP method.

B.2.2.1 Synthesised Data

Synthesised data is again used as a first step towards verifying the method, with 3D points being generated randomly, transformed using known and generated rotations between gravity vectors combined with random translations and projected to 2D query point matches before invoking the pose methods on the generated matches as described in Section B.2.1.1. In almost all cases the reprojection errors were within floating point accuracy limits of zero, the only exception being the same extreme gravity vector

²Or multiply the rotation matrix of the first with the transpose/inverse of the second and compare to the identity matrix

combination that was alluded to in Section B.2.1.1, although even for this case the reprojection error, while slightly larger than the 2D case, was still well within one pixel with a maximum error of 0.0771 and a mean error of 0.0436.

The reference implementation includes the 3D synthetic tests in the source file `src/synth3d.cc`.

B.2.2.2 Image Data

The same experimental setup described in Subsection B.2.1.2 was again used except that this time the text files mapped 3D model points in the training image to 2D image points in a test (query) image. The tests were again run using the Gravity2Gravity method and the OpenCV iterative method. The results of the tests are summarised in Table B.2. The comparisons in this test are more direct as all methods tested are PnP type and the OpenCV iterative method does now provide better results, with the averages being 2.5483 for the OpenCV iterative method and 4.2619 for the gravity pose method. The 3D tests can be run using a command line interface of the reference implementation, with a script to run the tests available in the directory `test-3d/` along with all the data and image files used.

The effects of noise on the image data was also tested for the 3D method, with Gaussian noise being added to all query image points, with deviation σ ranging from 0.5 to 15 pixels. The results are plotted in Figure B.2b.

Depth	Location	Method	Mean Error (Pixels)
1 A4 0.21m (Near)	Left	GP	7.0016
	It		3.62549
	Left (Rotated)	GP	8.3756
	It		4.75257
2 A4 0.42m (Mid)	Right	GP	8.5320
	It		12.161
	Right (Rotated)	GP	9.6172
	It		21.161
3 A4 0.63m (Far)	Left	GP	2.2051
	It		4.51083
	Left (Rotated)	GP	2.6547
	It		9.06991
2 A4 0.42m (Mid)	Centre (Up)	GP	2.6286
	It		5.50653
	Right	GP	2.3212
	It		4.6181
3 A4 0.63m (Far)	Right (Rotated)	GP	2.4871
	It		10.3103
	Left	GP	1.2042
	It		3.66194
3 A4 0.63m (Far)	Left (Rotated)	GP	2.7797
	It		4.69803
	Right	GP	1.5273
	It		3.63867
3 A4 0.63m (Far)	Right (Rotated)	GP	1.1046 *
	It		4.63662 *

Table B.1: Reprojection mean error results for gravity vector planar pose with depth. The reprojections for the row in the table marked with an * are illustrated in Figure B.1b.

Depth	Location	Method	Mean Error (Pixels)
1 A4 0.21m (Near)	Left	GP	8.6181
	It		3.62549
	Left (Rotated)	GP	7.1687
	It		4.75257
2 A4 0.42m (Mid)	Right	GP	25.0813
	It		12.161
	Right (Rotated)	GP	24.8512
	It		21.161
2 A4 0.42m (Mid)	Left	GP	13.2350
	It		6.36485
	Left (Rotated)	GP	10.8183
	It		9.6328
3 A4 0.63m (Far)	Centre (Up)	GP	9.3534
	It		5.3187
	Right	GP	11.2224
	It		5.3970
3 A4 0.63m (Far)	Right (Rotated)	GP	10.8817
	It		9.2962
	Left	GP	8.3450
	It		4.5863
3 A4 0.63m (Far)	Left (Rotated)	GP	9.7257
	It		8.3627
	Right	GP	6.7306
	It		3.6387
3 A4 0.63m (Far)	Right (Rotated)	GP	6.3737
	It		4.9724

Table B.2: Reprojection mean error results for gravity vector PnP based 3D model pose.

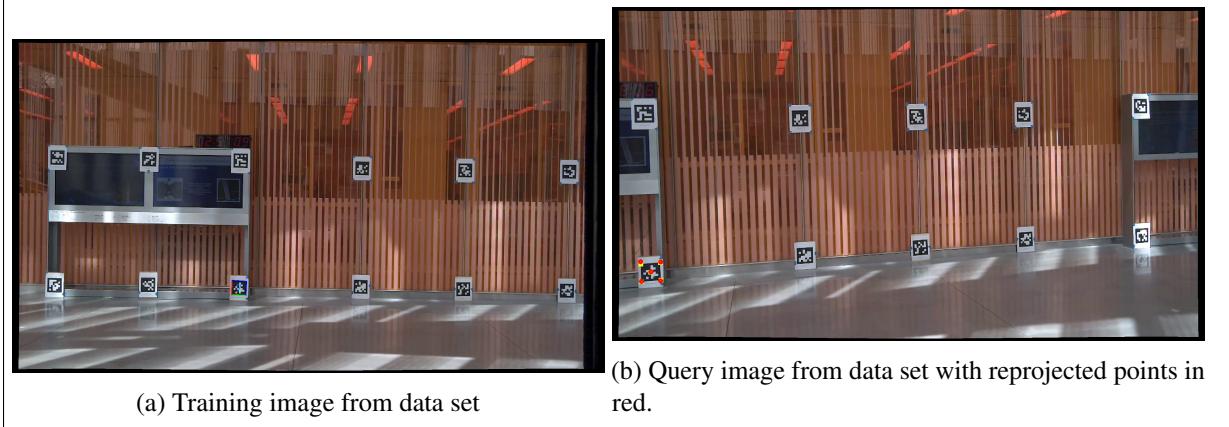


Figure B.3: Reprojection test for images with ground truth from PennCOSYVIO data set [Pfrommer et al., 2017]

B.2.3 Data Set with Ground Truths

Recently several CV data sets having images, IMU data and ground truth data obtained using specialised hardware have appeared. In order to further test the Gravity2Gravity pose method, the data set created by Pfrommer et al. [2017] was utilised. First two images extracted from one of the data set videos were selected as training and query images. The images contain several AprilTag fiducial markers [Wang and Olson, 2016], and one of these were selected as the training object. A difficulty in using the data set (or any other similar dataset) is that a depth to a selected training object cannot be physically measured so the depth was estimated using the ground truth translation. The timestamps for the images in the video were used to obtain the closest matching IMU data in order to extract a gravity vector which was also checked by using the ground truth rotation matrix. The planar pose method was then run on the two images resulting in a mean reprojection error of 2.592 (see Figure B.3).

In order to compare the results with the ground truth a MATLAB program was utilised. Firstly the matching image point rays from the fiducial markers was used to calculate the train and query points using the ground truth rotation and translation, and then the pose method rotation and translation was applied to the train image rays to obtain the query point which was then transformed into the ground truth coordinate system³ resulting in a ground truth position vector of (65.92197, 30.19891, 1.03989) versus the pose method one of (65.0576, 30.57619, 1.06875).

Only the planar pose method was tested using the data set as physically measuring 3D points for a training object is not possible for a remote location.

³The ground truth coordinate system is the conventional mathematical system with positive Z pointing up, while the relative pose coordinate system is one typically used for the camera coordinate system in CV with Y positive pointing down.

C

Pattern Index

C.1 Architectural Patterns

Pattern	Context	Page
STRUCTURAL TASK GRAPH	Defining a base architecture for MAR where the design is required to process multiple data streams in real-time.	88
SHARED RESOURCE	When the size of the data that will have to be passed between task nodes in a task graph is too large to be efficiently copied or when data has to be shared between nodes.	90
LOCATIONAL MAR	Architecture for locational MAR artefacts that can be used within the overall task graph architecture	103
ARCHITECTURE DETECT/TRACK ARCHITECTURE	Define a concurrent architecture for vision based detection, tracking and registration that can be used within the overall task graph architecture	126
MVC/DPI	MAR interaction sub-architectures that can be used within the overall task graph architecture	147
BROKER/LAYERS	Data access architecture patterns used within the overall task graph architecture	171

C.2 Detection Patterns

C.2.1 Locational MAR

Pattern	Context	Page
COMMON	Provide a common geographically derived local coordinate system in which POIs, annotations and other augmentable items can be placed.	98
LOCATIONAL COORDINATES		
SENSOR BASED	Using device sensors for orientation of the MAR scene.	100
ORIENTATION		
SENSOR FUSION	Improving the accuracy of sensor based orientation derivation.	101
SCRIPTED	Providing MAR content or markers at pre-defined pre-scripted locations.	102
CONTENT		

C.2.2 Vision Based MAR

Pattern	Context	Page
ACQUIRE FRAME	Efficiently and timeously acquire video frames from the hardware camera sensor(s).	105
FEATURE EXTRACTION	Extracting unique recognisable features from images that are repeatable across multiple images.	108
FEATURE MATCHING	Matching the descriptors of detected features between images and using the features for object recognition.	109
OBJECT TRACKING	Tracking object movement between frames.	113
RELATIVE POSE	Finding change in position and orientation (pose) between two images.	116
ABSOLUTE POSE	Finding the position and orientation (pose) of a 2D projection of an object in an image with respect to a 3D geometric model of the object.	118
POINT CLOUD POSE	Finding correspondences and pose between two 3D point clouds.	121
BUNDLE ADJUSTMENT	Inferring 3D coordinates and refining relative pose from multiple image points occurring in multiple images.	121
KEYFRAME IDENTIFICATION	Selecting keyframes from incoming camera frames.	124
DIRECT 3D MAP	Building a 3D map when implementing a dense or semi-dense SLAM system.	125
LOOP CLOSURE	Detection of revisited locations for pose and map validation in SLAM MAR.	126

C.3 Rendering Patterns

Pattern	Context	Page
ABSTRACT RENDERER	Support and runtime selection of multiple renderer technologies within a MAR artefact.	149
SCENE GRAPH	High level representation of relative spatial positions of virtual objects in an AR scene.	149
LOD (Level of Detail)	Vary the level of detail for virtual objects in a scene based on some criteria in order to improve rendering throughput.	151
MULTI BUFFER	Provide an atomic update view for readers of video buffers) that are updated sequentially by multiple writers, possibly in parallel.	153
TIMEWARPING	Update pose while rendering.	154
PHOTOMETRIC REGISTRATION	Direction of primary light source is required for 3D shading and shadows.	154
IBL (Image Based Lighting)	Precomputed lighting for known locations.	155
PROJECTION SETUP	Setting up the 3D projection transformation matrix using the camera intrinsics parameters.	156
CALIBRATION ADJUSTMENT	Adjusting the camera calibration to the display resolution.	158
BILLBOARD	Ensure rendered content always faces towards the user or a designated direction.	158

C.4 Interaction Patterns

Pattern	Context	Page
RAYCAST	Virtual item selection using ray intersections.	159
VERTEX	Virtual item selection using pre-coded vertex properties.	160
CODING		
RETICLE	Indication of a virtual item's focus status.	160
GUIDE	Spatial guidance for navigating the AR world.	161
AFFORDANCES	Manipulation hint through rendered properties of a virtual object.	161
ANCHOR	Attach a virtual object or informational content to a real-world object or location.	161
FREEZEFRAME	Provide a static frame for interaction or provide an alternative exocentric view.	163
TANGIBLE	Providing an intuitive UI which allows users to directly interact with virtual objects	163
TOOLS	rather than through the device touchscreen.	
MAGIC LENS	Providing drill-down information of visible real-world objects.	164

C.5 Data Access Patterns

Pattern	Context	Page
REQUESTOR	Accessing heterogeneous MAR data sources requires protocols, techniques and parameters that are specific to a given data sources.	173
REQUEST	Factoring out commonalities and abstracting complexity in low level implementations of network protocols.	175
HANDLER		
MARSHALLER	Encoding and decoding textual markup languages such as XML or JSON.	175
FACADE	Providing high level interfaces for interaction with data-sources.	176
EVENT	Combining asynchronous results from multiple FACADES.	177
AGGREGATOR		
SERIALIZER	Reading predefined configuration for FACADE's or REQUESTOR's from a file or database.	177
REGISTRY	Obtaining predefined global configuration metadata for FACADE or REQUESTOR.	178
ANNOTATED	Definition of configuration metadata in program source instead of externally.	178
METADATA		
Concurrency Patterns for Data Access	Various patterns for concurrency and parallelism used within the data access PL.	180
POI MARKER	POI representations and information must be displayed and continually updated as the user moves.	182
POI RADAR	The position of nearby POIs relative to the user must be visualisable.	182

Bibliography

- Abd-Allah, A. A. (1996). ‘Composing Heterogeneous Software Architectures’. PhD thesis. Los Angeles, CA, USA. ISBN: 0-591-11786-X (cited on pp. 79, 189).
- Agusanto, K., Li, Li, Chuangui, Zhu and Sing, Ng Wan (2003). Photorealistic Rendering for Augmented Reality using Environment Illumination. *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 208–216. DOI: 10.1109/ISMAR.2003.1240704 (cited on: p. 68).
- Aittala, M. (2010). Inverse Lighting and Photorealistic Rendering for Augmented Reality. *The Visual Computer*, vol. 26, pp. 669–678. DOI: 10.1007/s00371-010-0501-7 (cited on: p. 155).
- Akenine-Moller, T., Haines, E., Hoffman, N., Pesce, A., Iwanicki, M. and Hillaire, S. (2018). *Real-Time Rendering*. 4th ed. CRC Press. ISBN: 978-1138627000 (cited on pp. 66, 142–144, 156, 158, 160).
- Alatise, M. and Hancke, G. (2017). Pose Estimation of a Mobile Robot Based on Fusion of IMU Data and Vision Data Using an Extended Kalman Filter. *Sensors*, vol. 17 no. 10 (cited on: p. 131).
- Alcantarilla, P. F., Bartoli, A. and Davison, A. J. (2012). KAZE Features. *European Conference on Computer Vision (ECCV)*. Fiorenze, Italy (cited on: p. 50).
- Alcantarilla, P., Nuevo, J. and Bartoli, A. (2013). Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. *Proceedings of the British Machine Vision Conference*. Bristol, UK: BMVA Press (cited on pp. 50, 108).
- Aldinucci, M., Danelutto, M., Kilpatrick, P., Meneghin, M. and Torquati, M. (2011). Accelerating Code on Multi-Cores with FastFlow. *European Conference on Parallel Processing*. Springer, pp. 170–181. URL: <http://calvados.di.unipi.it/dokuwiki/doku.php/ffnamespace:about> (cited on: p. 86).
- Alexander, C. (1979). *The Timeless Way of Building*. Center for Environmental Structure Berkeley, Calif: Center for Environmental Structure series. Oxford University Press. ISBN: 0-19-502402-8 (cited on pp. 3, 17, 27, 31).
- Alexander, C., Ishikawa, S. and Silverstein, M. (1968). *A Pattern Language which Generates Multi-service Centers*. Center for Environmental Structure (cited on: p. 27).
- Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. and Angel, S. (1977). *A Pattern Language: Towns, Buildings, Construction*. Center for Environmental Structure Berkeley, Calif: Center for Environmental Structure series. Oxford University Press. ISBN: 9780195019193 (cited on pp. 27, 28).

BIBLIOGRAPHY

- Alexander, C., Silverstein, M., Angel, S., Ishikawa, S. and Abrams, D. (1975). *The Oregon Experiment*. Center for Environmental Structure Berkeley, Calif: Center for Environmental Structure series. Oxford University Press. ISBN: 9780195018240 (cited on: p. 27).
- Alexandrescu, A. (2001). *Modern C++ Design: Generic Programming and Design Patterns Applied*. C++ in-depth series. Addison-Wesley. ISBN: 9780201704310 (cited on: p. 151).
- Aljafari, R. and Khazanchi, D. (2013). On the Veridicality of Claims in Design Science Research. *System Sciences (HICSS), 2013 46th Hawaii International Conference on*. IEEE, pp. 3747–3756 (cited on: p. 13).
- Altwaijry, H., Veit, A. and Belongie, S. J. (2016). Learning to Detect and Match Keypoints with Deep Architectures. *The British Machine Vision Association (BMVA)*. URL: <http://www.bmva.org/bmvc/2016/papers/paper049/paper049.pdf> (cited on: p. 52).
- Android API Guides (2019). *RenderScript Overview*. URL: <http://developer.android.com/guide/topics/rsrenderscript/compute.html> (cited on: p. 105).
- Apple Developer (2018-08). *Metal: Accelerating Graphics and Much More*. URL: <https://developer.apple.com/metal/> (cited on: p. 141).
- Arai, K. and Barakbah, A. (2007). Hierarchical K-means: An Algorithm for Centroids Initialization for K-means. *Reports of the Faculty of Science and Engineering*, vol. 36, pp. 25–31 (cited on: p. 111).
- Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R. and Ives, Z. (2007). *Dbpedia: A Nucleus for a Web of Open Data*. Springer (cited on: p. 167).
- Azuma, R. (1997). *A Survey of Augmented Reality*. URL: <http://www.cs.unc.edu/~azuma/ARpresence.pdf> (cited on pp. 44, 45).
- Balen, H., Elenko, M., Jones, J., Palumbo, G. and Hoffman, B. (2000). *Distributed Object Architectures with CORBA*. SIGS reference library series. Cambridge University Press. ISBN: 9780521654180 (cited on: p. 170).
- Banno, A. (2018). A P3P Problem Solver Representing all Parameters as a Linear Combination. *Image and Vision Computing*, vol. 70, pp. 55–62. DOI: 10.1016/j.imavis.2018.01.001. URL: <https://reader.elsevier.com/reader/sd/pii/S0262885618300027> (cited on: p. 119).
- Bar-Yam, Y. (1997). *Dynamics of Complex Systems*. Addison-Wesley Reading, MA. ISBN: 0813341213. URL: <https://necsi.edu/dynamics-of-complex-systems> (cited on: p. 13).
- Barreira, J., Magalhães, L. and Bessa, M. (2013). A Sensor Based Approach to Outdoor Illumination Estimation for Augmented Reality Applications on Mobile Devices. *Eurographics 2013*. The Eurographics Association. URL: <http://diglib.eg.org/EG/DL/conf/EG2013/posters/PDF/003-004.pdf> (cited on pp. 69, 155).
- Baskerville, R. and Pries-Heje, J. (2010). Explanatory Design Theory. *Business and Information Systems Engineering*, vol. 2 no. 5, pp. 271–282. DOI: 10.1007/s12599-010-0118-4 (cited on: p. 16).
- Baskerville, R. and Pries-Heje, J. (2015). Projecting the Future for Design Science Research: An Action-Case Based Analysis. *New Horizons in Design Science: Broadening the Research Agenda*. Ed. by B. Donnellan, M. Helfert, J. Kenneally, D. VanderMeer, M. Rothenberger and R. Winter. Springer International Publishing, pp. 280–291. ISBN: 978-3-319-18714-3. DOI: 10.1007/978-3-319-18714-3_18 (cited on: p. 17).
- Batra, D., Nabbe, B. C. and Hebert, M. (2007). An Alternative Formulation for Five Point Relative Pose Problem. *2007 IEEE Workshop on Motion and Video Computing (WMVC'07)*. DOI: 10.1109/WMVC.2007.6 (cited on: p. 58).
- Bauer, C., King, G. and Gregory, G. (2015). *Java Persistence with Hibernate*. 2nd ed. Manning Greenwich. ISBN: 9781617290459 (cited on: p. 179).
- Bawa, M., Condie, T. and Ganesan, P. (2005). LSH Forest: Self-Tuning Indexes for Similarity Search. *Proceedings of the 14th International Conference on World Wide Web*. WWW '05. Chiba, Japan:

BIBLIOGRAPHY

- ACM, pp. 651–660. ISBN: 1-59593-046-9. DOI: 10.1145/1060745.1060840. URL: <http://infola.b.stanford.edu/~bawa/Pub/similarity.pdf> (cited on: p. 111).
- Bay, H., Ess, A., Tuytelaars, T. and Van Gool, L. (2008). Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, vol. 110 no. 3, pp. 346–359. URL: <http://www.vision.ee.ethz.ch/~surf/eccv06.pdf> (cited on: p. 49).
- Beardsley, P. A., Torr, P. H. S. and Zisserman, A. (1996). 3D Model Acquisition from Extended Image Sequences. *Proceedings of the 4th European Conference on Computer Vision*. Vol. 2. ECCV '96. Berlin, Heidelberg: Springer-Verlag, pp. 683–695. ISBN: 3-540-61123-1 (cited on: p. 62).
- Beck, K. (1997). *Smalltalk: Best Practice Patterns*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. ISBN: 0-13-476904-X (cited on: p. 35).
- Beck, K. (2002). *Test Driven Development: By Example*. Addison-Wesley. ISBN: 0321146530 (cited on: p. 72).
- Bergmann, P., Meinhardt, T. and Leal-Taixé, L. (2019). Tracking Without Bells and Whistles. *The Computing Research Repository*, vol. 1903 no. 5625. URL: <https://arxiv.org/pdf/1903.05625.pdf> (cited on: p. 114).
- Bermejo, C., Huang, Z., Braud, T. and Hui, P. (2017). When Augmented Reality Meets Big Data. *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 169–174. DOI: 10.1109/ICDCSW.2017.62 (cited on: p. 167).
- Berning, M., Yonezawa, T., Riedel, T., Nakazawa, J., Beigl, M. and Tokuda, H. (2013). pARnorama: 360 Degree Interactive Video for Augmented Reality Prototyping. *Proceedings of the 2013 ACM conference on Pervasive and Ubiquitous Computing Adjunct Publication*, pp. 1471–1474. URL: <http://ubicomp.org/ubicomp2013/adjunct/adjunct/p1471.pdf> (cited on: p. 72).
- Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A. and Torr, P. H. (2017). End-to-End Representation Learning for Correlation Filter Based Tracking. *CVPR 2017*. Honolulu, Hawaii, pp. 5000–5008. URL: <https://www.robots.ox.ac.uk/~luca/cfnet.html> (cited on: p. 55).
- Bi, Y., Dominguez, C. and Hassan, H. (2011). A General Design Pattern for Programs of Scene Graph and its Application in a Simulation Instance. *Proceedings of the International Conference on Computer Graphics and Virtual Reality (CGVR 2011)*. Santander, Spain. URL: <https://pdfs.semanticscholar.org/7122/d6334ad7231561fbb3f126ace69705c60036.pdf> (cited on: p. 148).
- Billinghurst, M., Clark, A. and Lee, G. (2015). A Survey of Augmented Reality. *Foundations and Trends in Human-Computer Interaction*, vol. 8 no. 3, pp. 73–272. DOI: 10.1561/1100000049 (cited on: p. 45).
- Billinghurst, M., Kato, H. and Poupyrev, I. (2001). Tangible Augmented Reality Interfaces. *HCI International 2001* (cited on: p. 64).
- Billinghurst, M., Piumsomboon, T. and Bai, H. (2014). Hands in Space: Gesture Interaction with Augmented-Reality Interfaces. *IEEE Computer Graphics and Applications*, vol. 34 no. 1, pp. 77–80. DOI: 10.1109/MCG.2014.8 (cited on: p. 65).
- Bimber, O. and Raskar, R. (2005). *Spatial Augmented Reality: Merging Real and Virtual Worlds*. Natick, MA, USA: A. K. Peters, Ltd. ISBN: 1568812302. URL: <http://www.uni-weimar.de/medien/ar/SpatialAR/download.php> (cited on: p. 46).
- Blinn, J. F. (1977). Models of Light Reflection for Computer Synthesized Pictures. *ACM SIGGRAPH Conference*, vol. 11 no. 2, pp. 192–198. DOI: 10.1145/965141.563893 (cited on: p. 143).
- Blinn, J. F. and Newell, M. E. (1976). Texture and Reflection in Computer Generated Images. *Communications of the ACM*, vol. 19 no. 10, pp. 542–547. DOI: 10.1145/360349.360353 (cited on: p. 155).

BIBLIOGRAPHY

- Bolme, D. S., Beveridge, J. R., Draper, B. A. and Lui, Y. M. (2010). Visual Object Tracking using Adaptive Correlation Filters. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2544–2550 (cited on: p. 54).
- Booch, G. (2008). Architectural Organizational Patterns. *IEEE Software*, vol. 25 no. 3, pp. 18–19. DOI: 10.1109/MS.2008.56 (cited on: p. 77).
- Boom, B. J., Orts-Escolano, S., Ning, X. X., McDonagh, S., Sandilands, P. and Fisher, R. B. (2015). Interactive Light Source Position Estimation for Augmented Reality with an RGB-D Camera. *Computer Animation and Virtual Worlds*, vol. 28 no. 1. DOI: 10.1002/cav.1686 (cited on: p. 155).
- Bouguet, J. (2000). Pyramidal Implementation of the Lucas Kanade Feature Tracker. *Intel Corporation, Microprocessor Research Labs*, URL: http://robots.stanford.edu/cs223b04/algo_affine_tracking.pdf (cited on: p. 53).
- Braga, R. T. V. and Masiero, P. C. (2002). The Role of Pattern Languages in the Instantiation of Object-Oriented Frameworks. *Advances in Object-Oriented Information Systems*. Ed. by J. Bruel and Z. Bellahsene. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 122–131. ISBN: 978-3-540-46105-0 (cited on: p. 40).
- Braz, L. M. (1990). Visual Syntax Diagrams for Programming Language Statements. *ACM SIGDOC Asterisk Journal of Computer Documentation*, vol. 14 no. 4, pp. 23–27 (cited on pp. 128, 183).
- Breen, D. E., Whitaker, R. T., Rose, E. and M., Tuceryan (1996). Interactive Occlusion and Automatic Object Placement for Augmented Reality. *Computer Graphics Forum*, vol. 15 no. 3, pp. 11–22. URL: <https://www.cs.drexel.edu/~david/Papers/eg96.pdf> (cited on: p. 67).
- Breitenstein, M. D., Reichlin, F., Leibe, B., Koller-Meier, E. and Gool, L. V. (2009). Robust Tracking-by-Detection Using a Detector Confidence Particle Filter. *2009 IEEE 12th International Conference on Computer Vision*. Kyoto, Japan, pp. 1515–1522. DOI: 10.1109/ICCV.2009.5459278. URL: <http://web-info8.informatik.rwth-aachen.de/media/papers/breitenstein-iccv09.pdf> (cited on: p. 114).
- Brückner, M., Bajramovic, F. and Denzler, J. (2008). Experimental Evaluation of Relative Pose Estimation Algorithms. *Proceedings of the Third International Conference on Computer Vision Theory and Applications*. Madeira, Portugal, pp. 431–438. URL: https://pub.inf-cv.uni-jena.de/pdf/Bruckner_08:EEO (cited on: p. 117).
- Buckl, S., Matthes, F., Schneider, A. W. and Schweda, C. M. (2013). Pattern-based Design Research: An Iterative Research Method Balancing Rigor and Relevance. *Proceedings of the 8th International Conference on Design Science at the Intersection of Physical and Virtual Design*. Ed. by J. vom Brocke, R. Hekkala, S. Ram and M. Rossi. DESRIST'13. Helsinki, Finland, pp. 73–87. ISBN: 978-3-642-38826-2. DOI: 10.1007/978-3-642-38827-9_6 (cited on pp. 17, 18).
- Burdak, L Wikimedia Commons (2017). *Bhimbetka rock painting*. URL: <https://commons.wikimedia.org/w/index.php?curid=380566> (visited on 23/07/2017) (cited on: p. 2).
- Burns, D. and Osfield, R. (2000). *OpenSceneGraph*. URL: <http://www.openscenegraph.org> (cited on: p. 150).
- Buschmann, F., Henney, K. and Schmidt, D. (2007a). *Pattern Oriented Software Architecture: On Patterns and Pattern Languages*. Wiley Software Patterns Series. John Wiley & Sons. ISBN: 0471486485 (cited on pp. 3, 4, 18–20, 28, 29, 40, 188, 189, 191, 194).
- Buschmann, F., Henney, K. and Schmidt, D. (2007b). *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing*. Wiley Software Patterns Series. John Wiley & Sons. ISBN: 9780470059029 (cited on pp. 29, 37, 90, 168, 169, 171, 189).
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M. (1996). *Pattern-oriented Software Architecture: A System of Patterns*. Wiley Software Patterns Series. John Wiley & Sons, Inc. ISBN: 0-471-95869-7 (cited on pp. 29, 30, 32, 33, 36, 81, 147, 171, 173, 188).

BIBLIOGRAPHY

- Calian, D. A., Mitchell, K., Nowrouzezahrai, D. and Kautz, J. (2013). The Shading Probe: Fast Appearance Acquisition for Mobile AR. *SIGGRAPH Asia 2013 Technical Briefs*. Hong Kong, Hong Kong. ISBN: 978-1-4503-2629-2. DOI: 10.1145/2542355.2542380. URL: http://www0.cs.ucl.ac.uk/staff/d.calian/preprints/ShadingProbe_SIGGRAPH_Asia_2013.pdf (cited on: p. 68).
- Campbell, C. and Ying, Y. (2011). *Learning with Support Vector Machines*. Morgan & Claypool Publishers. ISBN: 9781608456161 (cited on: p. 111).
- Caudell, T.P. and Mizell, D.W. (1992). Augmented Reality: An Application of Heads-up Display Technology to Manual Manufacturing Processes. *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on*. no. 2, pp. 659–669 (cited on: p. 45).
- Chan, B. Y., Si, A. and Leong, H. V. (2001). A Framework for Cache Management for Mobile Databases: Design and Evaluation. *Distributed and Parallel Databases*, vol. 10 no. 1, pp. 23–57 (cited on: p. 181).
- Chen, H., Wei, C., Song, M., Sun, M. and Lau, K. (2015). Capture-to-display Delay Measurement for Visual Communication Applications. *APSIPA Transactions on Signal and Information Processing*, vol. 4. DOI: 10.1017/AT SIP.2015.21 (cited on: p. 107).
- Chen, J., Turk, M. and MacIntyre, B. (2012). A Non-Photorealistic Rendering Rramework with Temporal Coherence for Augmented Reality. *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 151–160. DOI: 10.1109/ISMAR.2012.6402552 (cited on: p. 69).
- Chen, X., Wang, X. and Xuan, J. (2018). *Tracking Multiple Moving Objects Using Unscented Kalman Filtering Techniques*. URL: <https://arxiv.org/pdf/1802.01235.pdf> (cited on: p. 114).
- Chen, Y. and Medioni, G. (1991). Object Modeling by Registration of Multiple Range Images. *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pp. 2724–2729. DOI: 10.1109/ROBOT.1991.132043. URL: <https://graphics.stanford.edu/~smr/ICP/comparison/chen-medioni-align-rob91.pdf> (cited on: p. 121).
- Chen, Z. (2003). Bayesian Filtering: From Kalman Filters to Particle Filters, and Beyond. *Statistics*, vol. 182. DOI: 10.1080/02331880309257. URL: <http://140.113.144.123/EnD98/Bayesian%20filtering-%20from%20Kalman%20filters%20to%20Particle%20filters%20and%20beyond.pdf> (cited on: p. 114).
- Chowdhury, S. A., Arshad, H., Parhizkar, B. and Obeidy, W. K. (2013). Handheld Augmented Reality Interaction Technique. *Advances in Visual Informatics*. Ed. by H. B. Zaman, P. Robinson, P. Olivier, T. K. Shih and S. Velastin. Lecture Notes in Computer Science. Springer International Publishing, pp. 418–426. ISBN: 978-3-319-02957-3. DOI: 10.1007/978-3-319-02958-0_38 (cited on: p. 65).
- Chui, C.K. and Chen, G. (2017). *Kalman Filtering: With Real-Time Applications*. Springer International Publishing. ISBN: 9783319476124 (cited on pp. 93, 114).
- Chum, O. and Matas, J. (2005). Matching with PROSAC - Progressive Sample Consensus. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. San Diego, CA, USA, pp. 220–226. DOI: 10.1109/CVPR.2005.221 (cited on: p. 112).
- Clements, M. and Zakhor, A. (2014). Interactive Shadow Analysis for Camera Heading in Outdoor Images. *2014 IEEE International Conference on Image Processing (ICIP)*, pp. 3367–3371. DOI: 10.1109/ICIP.2014.7025681 (cited on: p. 155).
- Coelho, E. M., Julier, S. J. and MacIntyre, B. (2004). OSGAR: A Scene Graph with Uncertain Transformations. *Third IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 6–15. DOI: 10.1109/ISMAR.2004.44 (cited on: p. 151).
- Cohen Tenoudji, F. (2016). *Analog and Digital Signal Analysis: From Basics to Applications*. Springer. ISBN: 9783319423821 (cited on pp. 56, 93, 202).

BIBLIOGRAPHY

- Comaniciu, D. and Meer, P. (1999). Mean Shift Analysis and Applications. *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*. ICCV '99. IEEE Computer Society. ISBN: 0-7695-0164-8 (cited on: p. 53).
- Craig, A. B. (2013). *Understanding Augmented Reality: Concepts and Applications*. Burlington, MA: Elsevier (cited on: p. 44).
- Crockett, T. W. (1997). An Introduction to Parallel Rendering. *Parallel Computing*, vol. 23 no. 7, pp. 819–843. DOI: 10.1016/S0167-8191(97)00028-8 (cited on: p. 140).
- Cudworth, A.L. (2018). *Extending Virtual Worlds: Advanced Design for Virtual Environments*. CRC Press. ISBN: 9781482261172 (cited on: p. 153).
- Cuevas, E., Zaldivar, D. and Rojas, R. (2005). Kalman Filter for Vision Tracking. *Measurement*, vol. 33. URL: https://refubium.fu-berlin.de/bitstream/handle/fub188/19186/2005_12.pdf?sequence=1 (cited on: p. 114).
- Curley, M., Kenneally, J. and Ashurst, C. (2013). Design Science and Design Patterns: A Rationale for the Application of Design-Patterns within Design Science Research to Accelerate Knowledge Discovery and Innovation Adoption. *Design Science: Perspectives from Europe: European Design Science Symposium, EDSS 2012, Leixlip, Ireland, December 6, 2012, Revised Selected Papers*. Ed. by M. Helfert and B. Donnellan. Springer International Publishing, pp. 29–37. ISBN: 978-3-319-04090-5. DOI: 10.1007/978-3-319-04090-5_4 (cited on pp. 17, 18).
- Dai, J., Li, Y., He, K. and Sun, J. (2016). R-FCN: Object Detection via Region-based Fully Convolutional Networks. *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., pp. 379–387. ISBN: 978-1-5108-3881-9 (cited on: p. 55).
- Datcu, D. and Lukosch, S. (2013). Free Hands Interaction in Augmented Reality. *Proceedings of the ACM Symposium on Spatial User Interaction (SUI 2013)*. Los Angeles, CA, USA. DOI: 10.1145/2491367.2491370 (cited on: p. 163).
- Davidson, P. and Piché, R. (2017). A Survey of Selected Indoor Positioning Methods for Smartphones. *IEEE Communications Surveys and Tutorials*, vol. 19 no. 2, pp. 1347–1370. DOI: 10.1109/COMST.2016.2637663 (cited on: p. 96).
- De Sá, M. and Churchill, E. (2012). Mobile Augmented Reality: Exploring Design and Prototyping Techniques. *Proceedings of the 14th International Conference on Human-computer Interaction with Mobile Devices and Services*, pp. 221–230 (cited on pp. 70, 71).
- Dean, J. (2010-11). *Building Software Systems at Google and Lessons Learned*. URL: <https://static.googleusercontent.com/media/research.google.com/en//people/jeff/Stanford-DL-Nov-2010.pdf> (cited on: p. 168).
- Dellaert, F. and Kaess, M. (2017). Factor Graphs for Robot Perception. *Foundations and Trends® in Robotics*, vol. 6, pp. 1–139. URL: <http://www.cs.cmu.edu/~kaess/pub/Dellaert17fnt.pdf> (cited on pp. 121, 122).
- Devernay, F. and Faugeras, O. (2001). Straight Lines Have to Be Straight: Automatic Calibration and Removal of Distortion from Scenes of Structured Enviroments. *Machine Vision and Applications*, vol. 13 no. 1, pp. 14–24. DOI: 10.1007/PL00013269. URL: <https://hal.inria.fr/inria-00267247/file/distcalib.pdf> (cited on: p. 98).
- Diebel, J. (2006). *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors*. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.110.5134&rep=rep1&type=pdf> (cited on: p. 199).
- Dijkstra, E. W. (1972). The Humble Programmer. *Communications of the ACM*, vol. 15 no. 10, pp. 859–866. DOI: 10.1145/355604.361591. URL: <https://www.cs.utexas.edu/~EWD/transcripts/EWD03xx/EWD340.html> (cited on: p. 26).

BIBLIOGRAPHY

- DiVerdi, S. and Höllerer, T. (2006). Image-space Correction of AR Registration Errors Using Graphics Hardware. *IEEE Virtual Reality Conference (VR 2006)*, pp. 241–244. DOI: 10.1109/VR.2006.82 (cited on: p. 67).
- Doucet, A. and Johansen, A. M. (2009). A Tutorial on Particle Filtering and Smoothing: Fifteen Years Later. *Handbook of Nonlinear Filtering*, vol. 12 no. 3, pp. 656–704. URL: https://www.stats.ox.ac.uk/~doucet/doucet_johansen_tutorialPF2011.pdf (cited on: p. 114).
- Draper, N.R. and Smith, H. (2014). *Applied Regression Analysis*. Wiley Series in Probability and Statistics. Wiley. ISBN: 9781118625682. URL: <https://books.google.co.za/books?id=uSReBAAQBAJ> (cited on: p. 54).
- Dubrofsky, E. (2009). *Homography Estimation*. URL: https://www.cs.ubc.ca/grads/resources/thesis/May09/Dubrofsky_Elan.pdf (cited on: p. 112).
- Duval, E., Hodgins, W., Sutton, S. and Weibel, S.L (2002). Metadata Principles and Practicalities. *D-lib Magazine*, vol. 8 no. 4 (cited on: p. 177).
- Dyson, G. (2012). *Turing's Cathedral: The Origins of the Digital Universe*. Knopf Doubleday Publishing Group. ISBN: 9780307907066 (cited on: p. 13).
- Eade, E. (2017). *Lie Groups for 2D and 3D Transformations*. URL: <http://ethaneade.com/lie.pdf> (cited on: p. 201).
- Echtler, F., Huber, M., Pustka, D., Keitler, P. and Klinker, G. (2008). Splitting the Scene Graph: Using Spatial Relationship Graphs Instead of Scene Graphs in Augmented Reality. *Proceedings of the 3rd International Conference on Computer Graphics Theory and Applications (GRAPP)* (cited on: p. 151).
- Edelman, A. (2017-07). *Circulant Matrices*. URL: <http://web.mit.edu/18.06/www/Spring17/CirculantMatrices.pdf> (visited on 19/03/2018) (cited on: p. 54).
- Engel, J., Koltun, V. and Cremers, D. (2018). Direct Sparse Odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40 no. 3, pp. 611–625. DOI: 10.1109/TPAMI.2017.2658577. URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7898369> (cited on: p. 63).
- Engel, J., Schöps, T. and Cremers, D. (2014). LSD-SLAM: Large-Scale Direct Monocular SLAM. *Computer Vision – ECCV 2014*. Ed. by D. Fleet, T. Pajdla, B. Schiele and T. Tuytelaars. Springer International Publishing, pp. 834–849. ISBN: 978-3-319-10605-2. URL: https://vision.in.tum.de/_media/spezial/bib/engel14eccv.pdf (cited on pp. 63, 118, 124–126).
- Engel, J., Sturm, J. and Cremers, D. (2013). Semi-dense Visual Odometry for a Monocular Camera. *2013 IEEE International Conference on Computer Vision*. Sydney, Australia, pp. 1449–1456. DOI: 10.1109/ICCV.2013.183. URL: <https://jsturm.de/publications/data/engel2013iccv.pdf> (cited on pp. 118, 125, 126).
- Faugeras, O. (1993). *Three-dimensional Computer Vision: A Geometric Viewpoint*. Cambridge, MA, USA: MIT Press. ISBN: 0-262-06158-9 (cited on: p. 96).
- Fayad, M. and Schmidt, D. (1997). Object-oriented Application Frameworks. *Communications of the ACM*, vol. 40 no. 10, pp. 32–38. DOI: 10.1145/262793.262798. URL: https://www.researchgate.net/profile/Mohamed_Fayad/publication/215617675_Object-Oriented_Application_Frameworks/links/0912f50bfd0be0a85f000000.pdf (cited on pp. 4, 193).
- Feichtenhofer, C., Pinz, A. and Zisserman, A. (2017). Detect to Track and Track to Detect. *IEEE International Conference on Computer Vision (ICCV)*. Venice, Italy. URL: <https://www.robots.ox.ac.uk/~vgg/research/detect-track/> (cited on: p. 55).
- Feichtinger, J., Fillafer, F. and Surman, J. (2018). *The Worlds of Positivism: A Global Intellectual History, 1770–1930*. ISBN: 978-3-319-65761-5. DOI: 10.1007/978-3-319-65762-2 (cited on: p. 10).

BIBLIOGRAPHY

- Feiner, S., MacIntyre, B., Höllerer, T. and Webster, A. (1997). A Touring Machine: Prototyping 3D Mobile Augmented Reality Systems for Exploring the Urban Environment. *Personal Technologies*, vol. 1 no. 4, pp. 208–217. DOI: 10.1007/BF01682023 (cited on: p. 45).
- Feiner, S., Macintyre, B. and Seligmann, D. (1993). Knowledge-based Augmented Reality. *Communications of the ACM (Special issue on computer augmented environments)*, vol. 36 no. 7, pp. 53–62 (cited on: p. 45).
- Feiner, S. and Olwal, S. (2003). The Flexible Pointer: An Interaction Technique for Selection in Augmented and Virtual Reality. *Conference supplement of ACM symposium on user interface software and technology*, pp. 81–82 (cited on: p. 64).
- Ferraz, L., Binefa, X. and Moreno-Noguer, F. (2014). Leveraging Feature Uncertainty in the PnP Problem. *British Machine Vision Conf. (BMVC)*. University of Nottingham. URL: <http://www.bmva.org/bmvc/2014/files/paper065.pdf> (cited on: p. 60).
- Fischler, M. A. and Bolles, R. C. (1981). Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, vol. 24 no. 6, pp. 381–395. DOI: 10.1145/358669.358692 (cited on pp. 50, 200).
- Fisher, M., Ellis, J. and Bruce, J. C. (2003). *JDBC API Tutorial and Reference*. 3rd ed. Pearson Education. ISBN: 0321173848 (cited on: p. 170).
- Foote, B. and Yoder, J. (1999). Big Ball of Mud. *Pattern Languages of Program Design 4*. Ed. by B. Foote, H. Rohnert and N. Harrison. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., pp. 29–37. ISBN: 0201433044 (cited on: p. 77).
- Forster, C., Carbone, L., Dellaert, F. and Scaramuzza, D. (2015). IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation. *Robotics: Science and Systems*. Rome, Italy. DOI: 10.15607/rss.2015.xi.006. URL: <http://www.roboticsproceedings.org/rss11/p06.pdf> (cited on: p. 63).
- Forster, C., Carbone, L., Dellaert, F. and Scaramuzza, D. (2017). On-Manifold Preintegration for Real-Time Visual-Inertial Odometry. *IEEE Transactions on Robotics*, vol. 33, pp. 1–21 (cited on: p. 102).
- Fourati, H., Manamanni, N., Afifal, L. and Handrich, Y. (2011). A Nonlinear Filtering Approach for the Attitude and Dynamic Body Acceleration Estimation Based on Inertial and Magnetic Sensors: Bio-Logging Application. *IEEE Sensors Journal*, vol. 11. DOI: 10.1109/JSEN.2010.2053353. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.393.4064&rep=rep1&type=pdf> (cited on: p. 102).
- Fowler, M. (2004). *Event Aggregator*. URL: <http://martinfowler.com/eaaDev/EventAggregator.html> (visited on 09/2004) (cited on pp. 171, 177).
- Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R. and Stafford, R. (2002). *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0321127420 (cited on: p. 29).
- Frakes, W. B. and Isoda, S. (1994). Success Factors of Systematic Reuse. *IEEE Software*, vol. 11 no. 6, pp. 14–19. DOI: 10.1109/52.311045 (cited on: p. 25).
- François, A. (2003). Software Architecture for Computer Vision. *Emerging Topics in Computer Vision*. Ed. by G. Medioni and S. B. Kang. Upper Saddle River, NJ, USA: Prentice Hall PTR, pp. 585–653. ISBN: 0131013661 (cited on pp. 82, 83, 189).
- Fraundorfer, F., Tanskanen, P. and Pollefeys, M. (2010). A Minimal Case Solution to the Calibrated Relative Pose Problem for the Case of Two Known Orientation Angles. *Proceedings of the 11th European Conference on Computer Vision: Part IV*. ECCV'10. Heraklion, Crete, Greece: Springer-Verlag, pp. 269–282 (cited on pp. 61, 117, 118).
- Furieri, A. (2013). *SpatialLite*. URL: <https://www.gaia-gis.it/fossil/libspatialite/index> (cited on: p. 181).

BIBLIOGRAPHY

- Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-63361-2 (cited on pp. 3, 18, 27, 33–37, 101, 109, 116, 120, 132, 149, 150, 171, 174, 188).
- Gao, X., Hou, X., Tang, J. and Cheng, H. (2003). Complete Solution Classification for the Perspective-Three-Point Problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25 no. 8, pp. 930–943. DOI: 10.1109/TPAMI.2003.1217599 (cited on pp. 59, 119, 120).
- Gauch, H. G (2003). *Scientific Method in Practice*. Cambridge University Press (cited on: p. 9).
- Gervautz, M. and Schmalstieg, D. (2012). Anywhere Interfaces Using Handheld Augmented Reality. *Computer*, vol. 45 no. 7, pp. 26–31. DOI: 10.1109/MC.2012.72. URL: http://data.icg.tugraz.at/~dieter/publications/Schmalstieg_228.pdf (cited on: p. 64).
- Goldman, R. (2009). *An Integrated Introduction to Computer Graphics and Geometric Modeling*. Chapman and Hall/CRC Computer Graphics, Geometric Modeling, and Animation Series. CRC Press. ISBN: 9781439803356 (cited on pp. 97, 197).
- Goldman, R. (2010). *Rethinking Quaternions: Theory and Computation*. Morgan and Claypool (cited on: p. 199).
- Gooch, B., Sloan, P. J., Gooch, A., Shirley, P. and Riesenfeld, R. (1999). Interactive Technical Illustration. *Proceedings of the 1999 Symposium on Interactive 3D Graphics*. I3D '99. Atlanta, Georgia, USA: ACM, pp. 31–38. ISBN: 1-58113-082-1. DOI: 10.1145/300523.300526 (cited on: p. 138).
- Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep Learning*. MIT Press. URL: <http://www.deeplearningbook.org> (cited on: p. 51).
- Google Developers (2017). <https://developers.google.com/kml>. URL: <https://developers.google.com/kml> (cited on: p. 102).
- Graham, I. (2002). *A Pattern Language for Web Usability*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201788888 (cited on: p. 37).
- Grana, C., Borghesani, D., Manfredi, M. and Cucchiara, R. (2013). A Fast Approach for Integrating ORB Descriptors in the Bag of Words Model. Vol. 8667. San Francisco, California, USA. DOI: 10.1117/12.2008460 (cited on: p. 111).
- Grauman, K. and Leibe, B. (2011). Visual Object Recognition. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 5 no. 2. DOI: 10.2200/S00332ED1V01Y201103AIM011 (cited on: p. 108).
- Greene, N. (1986). Environment Mapping and Other Applications of World Projections. *IEEE Computer Graphics and Applications*, vol. 6 no. 11, pp. 21–29. DOI: 10.1109/MCG.1986.276658 (cited on: p. 156).
- Guba, E.G. (1990). *The Paradigm Dialog*. SAGE Publications. ISBN: 9780803938229 (cited on: p. 10).
- Guerra, E. and Nakagawa, E. Y. (2015). Relating Patterns and Reference Architectures. *Proceedings of the 22nd Conference on Pattern Languages of Programs*. PLoP '15. Pittsburgh, Pennsylvania: The Hillside Group. ISBN: 978-1-941652-03-9 (cited on: p. 40).
- Guy, R. and Agopian, M. (2018). *Physically Based Rendering in Filament*. URL: <https://google.github.io/filament/Filament.html> (cited on: p. 144).
- Haklay, M. and Weber, P. (2008). Openstreetmap: User-generated Street Maps. *IEEE Pervasive Computing*, vol. 7 no. 4, pp. 12–18 (cited on: p. 167).
- Haller, M., Drab, S. and Hartmann, W. (2003). A Real-time Shadow Approach for an Augmented Reality Application Using Shadow Volumes. *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*. VRST '03. Osaka, Japan, pp. 56–65. ISBN: 1-58113-569-6. DOI: 10.1145/1008653.1008665. URL: <http://www.amire.net/DocumentosWeb/publicaciones/Conferencias/fh@vrst03.pdf> (cited on: p. 69).

BIBLIOGRAPHY

- Haralick, B. M., Lee, C., Ottenberg, K. and Nölle, M. (1994). Review and Analysis of Solutions of the Three Point Perspective Pose Estimation Problem. *International Journal of Computer Vision*, vol. 13 no. 3, pp. 331–356. DOI: 10.1007/BF02028352. URL: https://haralick-org.torahcode.us/journals/three_point_perspective.pdf (cited on: p. 119).
- Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press. ISBN: 0521540518. DOI: 10.1017/CBO9780511811685 (cited on pp. 57, 58, 117, 197, 200, 201).
- Hemayed, E. E. (2003). A Survey of Camera Self-Calibration. *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance*. AVSS '03. Washington, DC, USA: IEEE Computer Society, pp. 351–357. ISBN: 0-7695-1971-7 (cited on: p. 57).
- Henney, K. (2006). Context Encapsulation — Three Stories, A Language, and Some Sequences. *EuroPLoP'2005, Tenth European Conference on Pattern Languages of Programs*. Ed. by A. Longshaw and U. Zdun. Irsee, Germany: UVK - Universitaetsverlag Konstanz. ISBN: 978-3-87940-805-4 (cited on: p. 39).
- Henriques, J. F., Caseiro, R., Martins, P. and Batista, J. (2015). High-Speed Tracking with Kernelized Correlation Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, pp. 583–596 (cited on: p. 54).
- Hesenius, M. and Gruhn, V. (2013). MVIC – An MVC Extension for Interactive, Multimodal Applications. *Software Architecture*. Ed. by K. Drira. Springer Berlin Heidelberg, pp. 324–327. ISBN: 978-3-642-39031-9 (cited on: p. 148).
- Hevner, A. R. (2007). A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, vol. 19 no. 2, pp. 87–92. URL: http://www.uio.no/studier/emner/jus/afin/FINF40_02/v13/hefner-design.pdf (cited on pp. 15, 16).
- Hevner, A. R., March, S. T., Park, J. and Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, vol. 28 no. 1, pp. 75–105. URL: http://wise.vub.ac.be/thesis_info/design_science.pdf (cited on pp. 13, 15).
- Hidayatullah, P. and Konik, H. (2011). CAMSHIFT Improvement on Multi-Hue and Multi-Object Tracking. *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics*. DOI: 10.1109/ICEEI.2011.6021825. URL: <https://hal-ujm.archives-ouvertes.fr/ujm-00634639/document> (cited on: p. 115).
- Hopkins, R. and Jenkins, K. (2008). *Eating the IT Elephant: Moving from Greenfield Development to Brownfield*. IBM Press. Pearson Education. ISBN: 9780132704267 (cited on: p. 78).
- Hoppe, H. (1996). Progressive Meshes. *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. ACM, pp. 99–108 (cited on: p. 152).
- Hornik, K., Stinchcombe, M. and White, H. (1989). Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, vol. 2 no. 5, pp. 359–366 (cited on: p. 51).
- Houssem, L. and Mahmoud, N. (2018). Hand Gesture Recognition Method Based On HOG-LBP Features for Mobile Devices. *Procedia Computer Science*, vol. 126, pp. 254–263. DOI: <https://doi.org/10.1016/j.procs.2018.07.259> (cited on: p. 164).
- Hugues, O., Fuchs, P. and Nannipieri, O. (2011). New Augmented Reality Taxonomy: Technologies and Features of Augmented Environment. *Handbook of Augmented Reality*. Ed. by B. Furht. Springer New York, pp. 47–63. ISBN: 978-1-4614-0063-9. DOI: 10.1007/978-1-4614-0064-6_11 (cited on pp. 44, 63).
- Hurst, W. and Van Wezel, C. (2011). Multimodal Interaction Concepts for Mobile Augmented Reality Applications. *Proceedings of the 17th International Conference on Advances in Multimedia Modeling - Volume II*. MMM'11. Taipei, Taiwan: Springer-Verlag, pp. 157–167. ISBN: 978-3-642-17828-3.

BIBLIOGRAPHY

- URL: http://www.staff.science.uu.nl/~hurst101/Publications/2011_mmm-2.pdf (cited on: p. 65).
- Iivari, J. (2015). Distinguishing and Contrasting Two Strategies for Design Science Research. *European Journal of Information Systems*, vol. 24 no. 1, pp. 107–115 (cited on pp. 17, 18, 22, 187).
- Ila, V., Polok, L., Solony, M. and Istenic, K. (2017). Fast Incremental Bundle Adjustment with Covariance Recovery. *2017 Fifth International Conference on 3D Vision*. Qingdao, China: Institute of Electrical and Electronics Engineers. ISBN: 978-989-8425-47-8. DOI: 10.1109/3DV.2017.00029. URL: https://www.fit.vut.cz/research/publication-file/11542/egpaper_final.pdf (cited on: p. 124).
- Internet Engineering Task Force (IETF) (2014). *RFC7234: Hypertext Transfer Protocol (HTTP/1.1): Caching*. URL: <https://tools.ietf.org/html/rfc7234> (cited on: p. 181).
- Ishii, H. and Ullmer, B. (1997). Tangible Bits: Towards Seamless Interfaces Between People, Bits and Atoms. *Proceedings of the ACM SIGCHI Conference on Human factors in Computing Systems*. CHI '97. Atlanta, Georgia, USA: ACM, pp. 234–241. ISBN: 0-89791-802-9. DOI: 10.1145/258549.258715. URL: <http://web.media.mit.edu/~anjchang/ti01/ishii-chi97-tangbits.pdf> (cited on: p. 64).
- Iswanto, I. A. and Li, B. (2017). Visual Object Tracking Based on Mean-Shift and Particle-Kalman Filter. *Procedia Computer Science*, vol. 116, pp. 587–595. DOI: 10.1016/j.procs.2017.10.010. URL: <https://reader.elsevier.com/reader/sd/pii/S1877050917320471> (cited on: p. 115).
- Jazayeri, M., Ran, A. and van der Linden, F. (2000). *Software Architecture for Product Families: Principles and Practice*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-69967-2 (cited on: p. 40).
- Jerath, R., Crawford, M. W. and Barnes, V. A. (2015). Functional Representation of Vision Within the Mind: A Visual Consciousness Model Based in 3D Default Space. *Journal of Medical Hypotheses and Ideas*, vol. 9 no. 1, pp. 45–56. DOI: <https://doi.org/10.1016/j.jmhi.2015.02.001>. URL: <http://www.sciencedirect.com/science/article/pii/S2251729415000051> (cited on: p. 66).
- Jia, Y. (2013). Dual Quaternion. *Com S*, vol. 477. URL: <http://web.cs.iastate.edu/~cs577/handouts/dual-quaternion.pdf> (cited on: p. 199).
- Kaehler, A. and Bradski, G. (2016). *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. 1st. O'Reilly Media, Inc. ISBN: 9781491937990 (cited on pp. 49, 60, 108).
- Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. and Dellaert, F. (2012). iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree. *The International Journal of Robotics Research*, vol. 31 no. 2, pp. 216–235. DOI: 10.1177/0278364911430419. URL: https://www.cs.cmu.edu/~k_aess/pub/Kaess12ijrr.pdf (cited on: p. 123).
- Kalal, Z., Mikolajczyk, K. and Matas, J. (2010). Forward-Backward Error: Automatic Detection of Tracking Failures. *2010 20th International Conference on Pattern Recognition*, pp. 2756–2759 (cited on: p. 53).
- Kalal, Z., Mikolajczyk, K. and Matas, J. (2012). Tracking-Learning-Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34 no. 7, pp. 1409–1422. DOI: 10.1109/TPAMI.2011.239. URL: <http://epubs.surrey.ac.uk/713800/1/Kalal-PAMI-2011%281%29.pdf> (cited on: p. 116).
- Kaplan, E.D. and Hegarty, C. (2017). *Understanding GPS/GNSS: Principles and Applications*. 3rd ed. GNSS technology and applications series. Artech House Publishers. ISBN: 9781630814427 (cited on pp. 94, 95).
- Karami, E., Prasad, S. and Shehata, M. S. (2017). Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images. *The Computing Research Repository*, vol. abs/1710.02726. URL: <http://arxiv.org/abs/1710.02726> (cited on: p. 113).

BIBLIOGRAPHY

- Kato, H. and Billinghurst, M. (1999). Marker Tracking and HMD calibration for a Video-based Augmented Reality Conferencing System. *IEEE and ACM International Workshop on Augmented Reality (IWAR '99)*. San Francisco, CA, USA, pp. 85–94 (cited on pp. 45, 48).
- Katz, M. (2002). *Introduction to Geometrical Optics*. World Scientific. ISBN: 9789812382245 (cited on: p. 98).
- Kawashima, T., Imamoto, K., Kato, H., Tachibana, K. and Billinghurst, M. (2001). Magic Paddle: A Tangible Augmented Reality Interface for Object Manipulation. *Proceedings of the IEEE and ACM International Symposium on Augmented Reality*. New York, NY, USA, pp. 111–119 (cited on: p. 163).
- Ke, T. and Roumeliotis, S. I. (2017). An Efficient Algebraic Solution to the Perspective-Three-Point Problem. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, Hawaii, US, pp. 4618–4626. DOI: 10.1109/CVPR.2017.491. URL: <https://arxiv.org/pdf/1701.08237.pdf> (cited on: p. 119).
- Kelbert, M. and Suhov, Y. (2013). *Information Theory and Coding by Example*. New York, NY, USA: Cambridge University Press. ISBN: 9780521139885 (cited on: p. 110).
- Kempe, V. (2011). *Inertial MEMS: Principles and Practice*. Cambridge University Press (cited on: p. 56).
- Kerin, A. (2016). The History of Graphics: Software's Sway Over Silicon. *Handbook of Visual Display Technology*. Ed. by J. Chen, W. Cranton and M. Fihn. Springer International Publishing, pp. 3381–3388. ISBN: 978-3-319-14346-0. DOI: 10.1007/978-3-319-14346-0_156 (cited on: p. 140).
- Kersten, D., Knill, D. C., Mamassian, P. and Bülthoff, I. (1996). Illusory Motion from Shadows. *Nature*, vol. 379 no. 31 (cited on: p. 138).
- Kessenich, J., Sellers, G. and Shreiner, D. (2016). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V*. New York, NY, USA: Pearson Education. ISBN: 9780134495538 (cited on pp. 142, 157).
- Keutzer, K. and Mattson, T. (2011). *Our Pattern Language (OPL) : A Design Pattern Language for Engineering (Parallel) Software*. URL: <https://patterns.eecs.berkeley.edu/> (cited on: p. 83).
- Khronos (2015). *OpenVX*. URL: <https://www.khronos.org/openvx/> (cited on: p. 87).
- Kircher, M. and Jain, P. (2004). *Pattern-Oriented Software Architecture: Patterns for Resource Management*. Wiley Software Patterns Series. John Wiley & Sons. ISBN: 0470845252 (cited on: p. 181).
- Kirk, D. S. (2005). ‘Understanding Object-oriented Frameworks’. PhD thesis. University of Strathclyde, Glasgow. (cited on: p. 40).
- Klawitter, D. (2014). *Clifford Algebras: Geometric Modelling and Chain Geometries with Application in Kinematics*. Springer Wiesbaden. ISBN: 9783658076184 (cited on: p. 199).
- Klein, G. and Murray, D. (2007). Parallel Tracking and Mapping for Small AR Workspaces. *Proceedings of the Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*. Nara, Japan (cited on pp. 62, 127).
- Kneip, L. and Furgale, P. (2014). OpenGV: A Unified and Generalized Approach to Real-time Calibrated Geometric Vision. *Proceedings of The IEEE International Conference on Robotics and Automation (ICRA)*. Hong Kong, China (cited on pp. 98, 117, 120).
- Kneip, L., Li, H. and Seo, Y. (2014). UPnP: An Optimal O(n) Solution to the Absolute Pose Problem with Universal Applicability. *European Conference of Computer Vision (ECCV)*. Zurich, Switzerland (cited on pp. 60, 119, 120).
- Kneip, L., Scaramuzza, D. and Siegwart, R. (2011). A Novel Parametrization of the Perspective-three-point Problem for a Direct Computation of Absolute Camera Position and Orientation. *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR '11. Washington, DC, USA: IEEE Computer Society, pp. 2969–2976. ISBN: 978-1-4577-0394-2. DOI: 10.1109/CVPR.2011.5995464 (cited on pp. 59, 119, 120).

BIBLIOGRAPHY

- Knorr, S. B. and Kurz, D. (2014). Real-time Illumination Estimation from Faces for Coherent Rendering. *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 113–122. DOI: 10.1109/ISMAR.2014.6948416 (cited on: p. 155).
- Kok, M., Hol, J. D. and Schön, T. B. (2018). Using Inertial Sensors for Position and Orientation Estimation. *Foundations and Trends in Signal Processing*, vol. 11. URL: <https://arxiv.org/pdf/1704.06053.pdf> (cited on: p. 102).
- Kos, A., Tomažič, S. and Umek, A. (2016). Suitability of Smartphone Inertial Sensors for Real-Time Biofeedback Applications. *Sensors*, vol. 16 no. 3. DOI: 10.3390/s16030301. URL: <http://www.mdpi.com/1424-8220/16/3/301> (cited on: p. 202).
- Krasner, G. E. and Pope, S. T. (1988). A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80. *J. Object Oriented Program.*, vol. 1 no. 3, pp. 26–49 (cited on: p. 147).
- Kruppa, E. (1913). Zur ermittlung eines objektes aus zwei perspektiven mit innerer orientierung. *Sitz.-Ber. Akad. Wiss.*, vol. 122, pp. 1939–1948 (cited on: p. 58).
- Kschischang, F. R., Frey, B. J. and Loeliger, H. (2001). Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, vol. 47 no. 2, pp. 498–519. DOI: 10.1109/18.910572. URL: <http://web.cs.iastate.edu/~honavar/factorgraphs.pdf> (cited on: p. 121).
- Kuaté, P. H., Bauer, C., King, G. and Harris, T. (2009). *NHibernate in Action*. 1st ed. Manning Greenwich. ISBN: 9781932394924 (cited on: p. 180).
- Kuhn, T. S. (1970). *The Structure of Scientific Revolutions*. University of Chicago Press (cited on: p. 10).
- Kukelova, Z., Bujnak, M. and Pajdla, T. (2011). Closed-form Solutions to Minimal Absolute Pose Problems with Known Vertical Direction. *Proceedings of the 10th Asian Conference on Computer Vision*. Vol. 2. ACCV'10. Queenstown, New Zealand: Springer-Verlag, pp. 216–229. ISBN: 978-3-642-19308-8 (cited on pp. 61, 120).
- Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K. and Burgard, W. (2011). G2o: A General Framework for Graph Optimization. *2011 IEEE International Conference on Robotics and Automation*. Shanghai, China, pp. 3607–3613. DOI: 10.1109/ICRA.2011.5979949. URL: <http://ais.informatik.uni-freiburg.de/publications/papers/kuemmerle11icra.pdf> (cited on: p. 124).
- Lavric, T. (2017). MPEG ARAF : A Language for Expressing Augmented Reality Experiences. *Workshop on Standards for Mixed and Augmented Reality (ISMAR 2017)*. Nantes, France. URL: <https://hal.archives-ouvertes.fr/hal-01691543> (cited on: p. 103).
- Lechner, M. (2013). ARML 2.0 in the Context of Existing AR Data Formats. *6th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS 2013)*, pp. 41–47. DOI: 10.1109/SEARIS.2013.6798107 (cited on: p. 103).
- Lee, D. (2012). *Numerically Efficient Methods for Solving Least Squares Problems*. URL: <http://www.math.uchicago.edu/~may/REU2012/REUPapers/Lee.pdf> (cited on: p. 205).
- Lee, G.A., Yang, U., Kim, Y., Jo, D., Kim, K., Kim, J. and Choi, J. (2009). Freeze-Set-Go interaction method for handheld mobile augmented reality environments. *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology*. VRST '09. Kyoto, Japan: ACM, pp. 143–146. ISBN: 978-1-60558-869-8. DOI: 10.1145/1643928.1643961 (cited on: p. 65).
- Lee, J. W., Kim, M. S. and Kweon, I. S. (1995). A Kalman Filter Based Visual Tracking Algorithm for an Object Moving in 3D. *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*. Vol. 1. Pittsburgh, PA, USA, pp. 342–347. DOI: 10.1109/IROS.1995.525818 (cited on: p. 114).
- Lepetit, V., Moreno-Noguer, F. and Fua, P. (2009). EPnP: An Accurate O(N) Solution to the PnP Problem. *International Journal of Computer Vision*, vol. 81 no. 2, pp. 155–166. DOI: 10.1007/s11263-008-0152-6 (cited on pp. 60, 119, 208).

BIBLIOGRAPHY

- Leutenegger, S., Chli, M. and Siegwart, R. (2011). BRISK: Binary Robust Invariant Scalable Keypoints. *International Conference on Computer Vision (ICCV)*, pp. 2548–2555. URL: <http://www.robots.ox.ac.uk/~vgg/rg/papers/brisk.pdf> (cited on: p. 49).
- Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R. and Furgale, P. T. (2015). Keyframe-Based Visual-Inertial Odometry using Nonlinear Optimization. *The International Journal of Robotics Research*, vol. 34, pp. 314–334 (cited on: p. 102).
- Lewens, T. (2016). *The Meaning of Science: An Introduction to the Philosophy of Science*. Basic Books. ISBN: 9780465097494 (cited on: p. 9).
- Li, L. (2014). Time-of-Flight Camera: An Introduction. *Technical White Paper (Texas Instruments)*, no. SLOA190B (cited on: p. 67).
- Liu, Z., Monasse, P. and Marlet, R. (2014). Match Selection and Refinement for Highly Accurate Two-View Structure from Motion. *Computer Vision – ECCV 2014*. Ed. by D. Fleet, T. Pajdla, B. Schiele and T. Tuytelaars. Zurich, Switzerland: Springer International Publishing, pp. 818–833. URL: <https://hal-enpc.archives-ouvertes.fr/hal-01153274/document> (cited on: p. 111).
- Longuet-Higgins, H. (1987). Readings in Computer Vision: Issues, Problems, Principles, and Paradigms. Ed. by M. A. Fischler and O. Firschein. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Chap. A Computer Algorithm for Reconstructing a Scene from Two Projections, pp. 61–62. ISBN: 0-934613-33-8. URL: <https://cseweb.ucsd.edu/classes/fa01/cse291/hchl/SceneReconstruction.pdf> (cited on: p. 117).
- Lopez-Moreno, J., Garces, E., Hadap, S., Reinhard, E. and Gutierrez, D. (2013). Multiple Light Source Estimation in a Single Image. *Computer Graphics Forum*, vol. 32 no. 8, pp. 170–182. DOI: 10.1111/cgf.12195 (cited on: p. 155).
- Lowe, D. (1999). Object Recognition from Local Scale-invariant Features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, pp. 1150–1157. DOI: 10.1109/ICCV.1999.790410. URL: <http://www.cs.ubc.ca/~lowe/papers/iccv99.pdf> (cited on pp. 49, 112).
- Lowe, D. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, vol. 60 no. 2, pp. 91–110. DOI: 10.1023/B:VISI.0000029664.99615.94 (cited on pp. 108, 111).
- Lucas, B. D. and Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision. *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'81. Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., pp. 674–679 (cited on: p. 53).
- Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B. and Huebner, R. (2003). *Level of Detail for 3D Graphics*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 9780080510118 (cited on: p. 153).
- Lukezic, A., Vojír, T., Cehovin, L., Matas, J. and Kristan, M. (2017). Discriminative Correlation Filter with Channel and Spatial Reliability. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4847–4856 (cited on: p. 55).
- Luna, F. (2016). *Introduction to 3D Game Programming with DirectX 12*. USA: Mercury Learning & Information. ISBN: 9781942270065 (cited on: p. 141).
- MacWilliams, A., Reicher, T., Klinker, G. and Bruegge, B. (2004). Design Patterns for Augmented Reality Systems. *International Workshop Exploring the Design and Engineering of Mixed Reality Systems - MIXER 2004*. URL: <http://ceur-ws.org/Vol-91/paperE4.pdf> (cited on: p. 80).
- Madgwick, S. O. H., Harrison, A. J. L. and Vaidyanathan, R. (2011). Estimation of IMU and MARG Orientation Using a Gradient Descent Algorithm. *2011 IEEE International Conference on Rehabilitation Robotics*. DOI: 10.1109/ICORR.2011.5975346 (cited on: p. 102).

BIBLIOGRAPHY

- Madsen, C. B., Jensen, T. and Andersen, M. S. (2006). Real-time Image-Based Lighting for Outdoor Augmented Reality Under Dynamically Changing Illumination Conditions. *International Conference on Graphics Theory and Applications, GRAPP 2006*, pp. 364–371 (cited on: p. 69).
- Madsen, C. B. and Laursen, R. (2007). A Scalable GPU-Based Approach to Shading and Shadowing for Photo-Realistic Real-Time Augmented Reality. *Proceedings: GRAPP 2007*. Ed. by J. Braz, P. Vasquez and J. Pereira. Institute for Systems, Technologies of Information, Control and Communication, pp. 252–261. ISBN: 978-972-8865-71-9 (cited on: p. 69).
- Maier, M. W., Emery, D. and Hilliard, R. (2001). Software Architecture: Introducing IEEE Standard 1471. *Computer*, vol. 34 no. 4, pp. 107–109. DOI: 10.1109/2.917550 (cited on: p. 77).
- Mair, E., Hager, G. D., Burschka, D., Suppa, M. and Hirzinger, G. (2010). Adaptive and Generic Corner Detection Based on the Accelerated Segment Test. *Computer Vision–ECCV 2010*. Springer, pp. 183–196. URL: <http://www6.in.tum.de/Main/Publications/Mair2010c.pdf> (cited on: p. 49).
- Malis, E. and Vargas, M. (2007). *Deeper Understanding of the Homography Decomposition for Vision-Based Control*. Research Report. URL: <https://hal.inria.fr/inria-00174036v1/document> (cited on: p. 117).
- Mamassian, P. and Goutcher, R. (2001). Prior Knowledge on the Illumination Position. *Cognition*, vol. 81 no. 1, pp. B1–B9 (cited on: p. 155).
- March, S.T. and Smith, G.F. (1995). Design and Natural Science Research on Information Technology. *Decision Support Systems*, vol. 15 no. 4, pp. 251–266 (cited on: p. 17).
- Martin, R. C., D. Riehle and F. Buschmann, eds. (1997). *Pattern Languages of Program Design 3*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-31011-2 (cited on: p. 178).
- Masser, I. (2005). *GIS Worlds: Creating Spatial Data Infrastructures*. Vol. 338. ESRI press Redlands, CA (cited on: p. 167).
- Mayer-Schönberger, Viktor and Cukier, Kenneth (2013). *Big Data: A Revolution that will Transform how we Live, Work, and Think*. Houghton Mifflin Harcourt (cited on: p. 167).
- McCool, M., Reinders, J. and Robison, A. (2012). *Structured Parallel Programming: Patterns for Efficient Computation*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 9780123914439 (cited on: p. 81).
- McDonald, J., Kaess, M., Cadena, C., Neira, J. and Leonard, J. J. (2013). Real-time 6-DOF Multi-session Visual SLAM over Large-scale Environments. *Robotics and Autonomous Systems*, no. 10, pp. 1144–1158. DOI: 10.1016/j.robot.2012.08.008 (cited on: p. 162).
- McKay, J. and Marshall, P. (2007). Science, Design, and Design Science: Seeking Clarity to Move Design Science Research Forward in Information Systems. *Australasian Conference on Information Systems*. Toowoomba, Australia, pp. 604–614. URL: <http://aisel.aisnet.org/cgi/viewcontent.cgi?article=1055&context=acis2007> (cited on: p. 13).
- Melax, S. (2000). Game Programming Gems. Ed. by M. DeLoura. Charles River Media. Chap. The Shortest Arc Quaternion, pp. 214–218 (cited on: p. 202).
- Mesnil, J. (2014). *Mobile and Web Messaging: Messaging Protocols for Web and Mobile Devices*. O'Reilly Media. ISBN: 9781491944783 (cited on: p. 170).
- Meszaros, G. and Doble, J. (1997). Pattern Languages of Program Design 3. Ed. by R. C. Martin, D. Riehle and F. Buschmann. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. Chap. A Pattern Language for Pattern Writing, pp. 529–574. ISBN: 0-201-31011-2 (cited on: p. 37).
- Michel, T. (2017). ‘On Mobile Augmented Reality Applications based on Geolocation’. PhD thesis. URL: <https://hal.inria.fr/tel-01651589/document> (cited on pp. 98, 100, 102, 129).
- Milgram, P. and Kishino, F. (1994a). A Taxonomy of Mixed Reality Visual Displays. *IEICE Trans. Information Systems*, vol. E77-D no. 12, pp. 1321–1329. URL: http://vered.rose.utoronto.ca/people/paul_dir/IEICE94/ieice.html (cited on pp. 2, 3).

BIBLIOGRAPHY

- Milgram, P. and Kishino, F. (1994b). A Taxonomy of Mixed Reality Visual Displays. *IEICE Trans. Information Systems*, vol. E77-D no. 12, pp. 1321–1329. URL: http://vered.rose.utoronto.ca/people/paul_dir/IEICE94/ieice.html (cited on pp. 43, 44).
- Miller, A. (2010). The Task Graph Pattern. *Proceedings of the 2010 Workshop on Parallel Programming Patterns*. ParaPLOP '10. Carefree, Arizona, USA: ACM. ISBN: 978-1-4503-0127-5. DOI: 10.1145/1953611.1953619. URL: http://www.ademiller.com/tech/reports/paraplop_2010_the_task_graph_pattern_workshop_submission.pdf (cited on pp. 83, 189).
- Miller, C. (2006). A Beast in the Field: The Google Maps Mashup as GIS/2. *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 41 no. 3, pp. 187–199 (cited on: p. 167).
- Mohr, R., Quan, L. and Veillon, F. (1995). Relative 3D Reconstruction Using Multiple Uncalibrated Images. *International Journal of Robotics Research*, vol. 14 no. 6, pp. 619–632. DOI: 10.1177/027836499501400607 (cited on: p. 62).
- Mohring, M., Lessig, C. and Bimber, O. (2004). Video See-Through AR on Consumer Cell-Phones. *IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR 2004*. Arlington, VA, USA, pp. 252–253 (cited on: p. 45).
- Monroy, R., Hudon, M. and Smolic, A. (2018). Dynamic Environment Mapping for Augmented Reality Applications on Mobile Devices. *Proceedings of the Conference on Vision, Modeling, and Visualization*. EG VMV '18. Stuttgart, Germany: Eurographics Association, pp. 21–28. DOI: 10.2312/vmv.20181249 (cited on: p. 156).
- Moré, J. J. (1978). The Levenberg-Marquardt Algorithm: Implementation and Theory. *Numerical Analysis*. Springer, pp. 105–116. URL: <https://www.osti.gov/scitech/servlets/purl/7256021> (cited on: p. 201).
- Mossel, A., Venditti, B. and Kaufmann, H. (2013a). 3DTouch and HOMER-S: Intuitive Manipulation Techniques for One-Handed Handheld Augmented Reality. *Laval Virtual VRIC '13*. URL: http://publik.tuwien.ac.at/files/PubDat_217297.pdf (cited on: p. 65).
- Mossel, A., Venditti, B. and Kaufmann, H. (2013b). DrillSample: Precise Selection in Dense Handheld Augmented Reality Environments. *Laval Virtual VRIC '13*. URL: http://publik.tuwien.ac.at/files/PubDat_217298.pdf (cited on: p. 65).
- Mourikis, A. I. and Roumeliotis, S. I. (2007). A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 3565–3572 (cited on: p. 102).
- Muja, M. and Lowe, D. (2009). Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. *Proceedings of the 4th International Conference on Computer Vision Theory and Applications*. Vol. 1. Lisbon, Portugal, pp. 331–340 (cited on: p. 111).
- Muja, M. and Lowe, D. (2014). Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (cited on: p. 50).
- Mullen, T. (2011). *Prototyping Augmented Reality*. John Wiley & Sons. ISBN: 978-1-118-03663-1 (cited on: p. 71).
- Mulloni, A., Dünser, A. and Schmalstieg, D. (2010). Zooming interfaces for Augmented Reality browsers. *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*. MobileHCI '10. Lisbon, Portugal: ACM, pp. 161–170. ISBN: 978-1-60558-835-3. DOI: 10.1145/1851600.1851629 (cited on: p. 65).
- Munro, D., Calitz, A. P. and Vogts, D. (2017). A Mobile Augmented Reality Emulator for Android. *South African Computer Journal*, vol. 29, pp. 172–190 (cited on: p. 72).
- Mur-Artal, R. and Tardós, J. D. (2017). ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics*, vol. 33 no. 5, pp. 1255–1262. DOI:

BIBLIOGRAPHY

- 10.1109/TRO.2017.2705103. URL: <https://arxiv.org/pdf/1610.06475.pdf> (cited on pp. 112, 124, 127, 130).
- Newcomer, E. (2002). *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley. ISBN: 9780201750812 (cited on: p. 170).
- Niehaves, B. (2007). On Epistemological Diversity in Design Science: New Vistas for a Design-Oriented IS Research? *ICIS*. URL: <http://citesseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.231.2753&rep=rep1&type=pdf> (cited on: p. 13).
- Nistér, D. (2003). An Efficient Solution to the Five-point Relative Pose Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, pp. 756–770 (cited on pp. 58, 117).
- Nister, D. and Stewenius, H. (2006). Scalable Recognition with a Vocabulary Tree. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. New York, NY, USA, pp. 2161–2168. DOI: 10.1109/CVPR.2006.264 (cited on: p. 111).
- Norman, D.A. (1988). *The Psychology of Everyday Things*. Basic Books. ISBN: 9780385267748 (cited on: p. 161).
- Nourani-Vatani, N., Borges, P. V. K. and Roberts, J. M. (2012). A Study of Feature Extraction Algorithms for Optical Flow Tracking. *2012 Australasian Conference on Robotics and Automation*. Wellington, New Zealand. URL: <https://pdfs.semanticscholar.org/d25e/903d8511a07e31935a410e645e6f12248923.pdf> (cited on: p. 115).
- Nunamaker, J., Chen, M. and Purdin, T. (1990). Systems Development in Information Systems Research. *Journal of Management Information Systems*, vol. 7 no. 3, pp. 89–106. DOI: 10.1080/07421222.1990.11517898 (cited on pp. 22, 187).
- OGRE3D Team (2018). *OGRE3D Rendering Engine*. URL: <https://github.com/OGRECave/ogre/blob/master/OgreMain/include/OgreVector3.h> (visited on 17/06/2018) (cited on: p. 203).
- Ohta, Y. and Tamura, H. (2014). *Mixed Reality: Merging Real and Virtual Worlds*. 1st ed. Springer Publishing Company, Incorporated. ISBN: 9783642875144 (cited on: p. 2).
- Olson, C. F. (1997). Efficient Pose Clustering Using a Randomized Algorithm. *International Journal of Computer Vision*, vol. 23 no. 2, pp. 131–147 (cited on: p. 51).
- OpenCV (2017). *OpenCV Fisheye Camera Model*. URL: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#fisheye (cited on: p. 98).
- Ortega-Arjona, J. L. (2003). The Shared Resource Pattern. An Activity Parallelism Architectural Pattern for Parallel Programming. *Proceedings of the 10th Conference on Pattern Languages of Programming, PLoP* (cited on: p. 90).
- Ortega-Arjona, J. L. and Roberts, G. (1998). Architectural Patterns for Parallel Programming. *Proceedings of EuroPLoP'98*, (cited on: p. 84).
- Owen, C.L. (1998). Design Research: Building the Knowledge Base. *Design Studies*, vol. 19 no. 1, pp. 9–20. DOI: 10.1016/S0142-694X(97)00030-6. URL: http://design.osu.edu/carlson/id785/Owen_desstud97.pdf (cited on: p. 12).
- Panda, D., Rahman, R. and Lane, D. (2014). *EJB 3 in Action*. 2nd ed. Manning Publications Company. ISBN: 9781935182993 (cited on: p. 179).
- Peddie, J. (2017). *Augmented Reality: Where We Will All Live*. Springer International Publishing AG. ISBN: 978-3-319-54502-8 (cited on: p. 45).
- Penate-Sánchez, A., Andrade-Cetto, J. and Moreno-Noguer, F. (2013). Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35 no. 10, pp. 2387–2400. DOI: 10.1109/TPAMI.2013.36 (cited on: p. 60).

BIBLIOGRAPHY

- Pessoa, S., Moura, G., Lima, J., Teichrieb, V. and Kelner, J. (2010). Photorealistic rendering for Augmented Reality: A global illumination and BRDF solution. *Virtual Reality Conference (VR), 2010 IEEE*, pp. 3–10. DOI: 10.1109/VR.2010.5444836 (cited on: p. 68).
- Pflugfelder, R. and Bischof, H. (2005). Online Auto-Calibration in Man-Made Worlds. *Digital Image Computing: Techniques and Applications (DICTA'05)* (cited on: p. 94).
- Pfrommer, B., Sanket, N., Daniilidis, K. and Cleveland, J. (2017). PennCOSYVIO: A Challenging Visual Inertial Odometry Benchmark. *2017 IEEE International Conference on Robotics and Automation, ICRA*. Singapore, Singapore, pp. 3847–3854. DOI: 10.1109/ICRA.2017.7989443. URL: <https://daniilidis-group.github.io/penncosyvio> (cited on: p. 210).
- Phong, B. T. (1975). Illumination for Computer Generated Pictures. *Communications of the ACM*, vol. 18 no. 6, pp. 311–317. DOI: 10.1145/360825.360839 (cited on: p. 143).
- Pizarro, O., Eustice, R. M. and Singh, H. (2003). Relative Pose Estimation for Instrumented, Calibrated Imaging Platforms. *Proceedings of the Seventh International Conference on Digital Image Computing: Techniques and Applications, DICTA 2003*. Sydney, Australia. URL: <http://robots.engin.umich.edu/publications/opizarro-2003a.pdf> (cited on: p. 117).
- Poupyrev, I., Billinghurst, M., Weghorst, S. and Ichikawa, T. (1996). The Go-go Interaction Technique: Non-linear Mapping for Direct Manipulation in VR. *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*. UIST '96. Seattle, Washington, USA, pp. 79–80. ISBN: 0-89791-798-7. DOI: 10.1145/237091.237102 (cited on pp. 64, 160).
- Poupyrev, I., Ichikawa, T., Weghorst, S. and Billinghurst, M. (1998). Egocentric Object Manipulation in Virtual Environments: Empirical Evaluation of Interaction Techniques. *Computer Graphics Forum*, vol. 17 no. 3, pp. 41–52. DOI: 10.1111/1467-8659.00252 (cited on: p. 63).
- Pryss, R., Geiger, P., Schickler, M., Schobel, J. and Reichert, M. (2016). Advanced Algorithms for Location-based Smart Mobile Augmented Reality Applications. *Procedia Computer Science*, pp. 97–104 (cited on: p. 162).
- Purao, S. (2002). Design Research in the Technology of Information Systems: Truth or Dare. *GSU Department of CIS Working Paper*, URL: <http://purao.ist.psu.edu/working-papers/dare-purao.pdf> (cited on: p. 17).
- Purao, S. (2013). Truth or Dare: The Ontology Question in Design Science Research. *Journal of Database Management*, pp. 51–66 (cited on pp. 14, 20).
- Raguram, R., Frahm, J. and Pollefeys, M. (2008). A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus. *Proceedings of the 10th European Conference on Computer Vision: Part II*. ECCV '08. Marseille, France: Springer-Verlag, pp. 500–513. ISBN: 978-3-540-88685-3. DOI: 10.1007/978-3-540-88688-4_37. URL: <https://inf.ethz.ch/personal/pomarc/pubs/RaguramECCV08.pdf> (cited on: p. 112).
- Rathore, R. and Prinja, R. (2007). An Overview of Mobile Database Caching. *CiteSeerX*, vol. 10. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.9481&rep=rep1&type=pdf> (cited on: p. 181).
- Reagen, B., Adolf, R. and Whatmough, P. (2017). *Deep Learning for Computer Architects*. Morgan & Claypool Publishers. ISBN: 9781627057288 (cited on: p. 51).
- Redmon, J., Divvala, S. K., Girshick, R. B. and Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788 (cited on: p. 52).
- Reicher, T. (2004). ‘A Framework for Dynamically Adaptable Augmented Reality Systems’. Dissertation. Technische Universität München (cited on pp. 79, 80).
- Renaudin, V. and Combettes, C. (2014). Magnetic, Acceleration Fields and Gyroscope Quaternion (MAGYQ)-based Attitude Estimation with Smartphone Sensors for Indoor Pedestrian Navigation.

BIBLIOGRAPHY

- Sensors*, vol. 14 no. 12. DOI: 10.3390/s141222864. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4299043/pdf/sensors-14-22864.pdf> (cited on: p. 102).
- Reynolds, V., Hausenblas, M., Polleres, A., Hauswirth, M. and Hegde, V. (2010). Exploiting Linked Open Data for Mobile Augmented Reality. *W3C Workshop: Augmented Reality on the Web*. Vol. 1 (cited on: p. 167).
- Richardson, I. E. (2010). *The H.264 Advanced Video Compression Standard*. 2nd. Wiley Publishing. ISBN: 9780470516928 (cited on: p. 124).
- Rosten, E. and Drummond, T. (2006). Machine Learning for High-speed Corner Detection. *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*. ECCV'06. Graz, Austria: Springer-Verlag, pp. 430–443. ISBN: 978-3-540-33832-1. URL: <http://citeseerx.ist.psu.edu/view/doc/download?doi=10.1.1.60.3991&rep=rep1&type=pdf> (cited on: p. 49).
- Rosten, E., Porter, R. and Drummond, T. (2010). Faster and Better: A Machine Learning Approach to Corner Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 1, pp. 105–119. DOI: 10.1109/TPAMI.2008.275. URL: <http://arxiv.org/pdf/0810.2434.pdf> (cited on: p. 49).
- Rosten, E., Reitmayr, G. and Drummond, T. (2005). Real-time Video Annotations for Augmented Reality. *International Symposium on Visual Computing*, pp. 294–302 (cited on pp. 69, 162).
- Rublee, E., Rabaud, V., Konolige, K. and Bradski, G. (2011). ORB: An Efficient Alternative to SIFT or SURF. *IEEE International Conference on Computer Vision (ICCV)*. Barcelona, Spain, pp. 2564–2571. URL: http://www.vision.cs.chubu.ac.jp/CV-R/pdf/Rublee_iccv2011.pdf (cited on pp. 49, 108).
- Saatsi, J. (2017). *The Routledge Handbook of Scientific Realism*. Routledge Handbooks in Philosophy. Taylor & Francis. ISBN: 9781351362900 (cited on: p. 10).
- Saho, K. (2018). Kalman Filter for Moving Object Tracking: Performance Analysis and Filter Design. *Kalman Filters - Theory for Advanced Applications*. ISBN: 978-953-51-3827-3. DOI: 10.5772/intechopen.71731. URL: <https://cdn.intechopen.com/pdfs/57673.pdf> (cited on: p. 114).
- Samet, H. (2006). *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Elsevier Science. ISBN: 9780123694461 (cited on pp. 111, 160).
- Saunders, M., Lewis, P. and Thornhill, A. (2007). *Research Methods for Business Students*. 4th ed. Pearson Education (cited on pp. 20, 21).
- Scali's OpenBlog (2012-08). *A Brief Overview on the History of 3D Graphics Hardware*. URL: <https://scalibq.wordpress.com/2012/08/25/a-brief-overview-on-the-history-of-3d-graphics-hardware/> (cited on: p. 140).
- Schall, G., Mendez, E., Kruijff, E., Veas, E., Junghanns, S., Reitinger, B. and Schmalstieg, D. (2009). Handheld Augmented Reality for Underground Infrastructure Visualization. *Personal Ubiquitous Computing*, vol. 13 no. 4, pp. 281–291. DOI: 10.1007/s00779-008-0204-5 (cited on: p. 164).
- Schlick, C. (1993). A Customizable Reflectance Model for Everyday Rendering. In *Fourth Eurographics Workshop on Rendering*, pp. 73–83 (cited on: p. 68).
- Schmandt-Besserat, D. (2014). *The Evolution of Writing*. URL: <https://sites.utexas.edu/dsb/tokens/the-evolution-of-writing/> (cited on: p. 2).
- Schmidt, D., Stal, M., Rohnert, H. and Buschmann, F. (2000). *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. 2nd. Wiley Software Patterns Series. John Wiley & Sons, Inc. ISBN: 9780471606956 (cited on pp. 171, 174, 180, 181, 188).
- Schmidt, J. and Niemann, H. (2001). Using Quaternions for Parametrizing 3-D Rotations in Unconstrained Nonlinear Optimization. *Vision, Modeling, and Visualization*, vol. 1, pp. 399–406 (cited on: p. 205).

BIBLIOGRAPHY

- Sellers, G. and Kessenich, J. (2016). *Vulkan Programming Guide: The Official Guide to Learning Vulkan*. New York, NY, USA: Pearson Education. ISBN: 9780134464688 (cited on pp. 141, 142).
- Shantaiya, S., Verma, Kesari and Mehta, K. (2015). Multiple Object Tracking Using Kalman Filter and Optical Flow. *European Journal of Advances in Engineering and Technology*, vol. 2, pp. 34–39. URL: <http://www.ejaet.com/PDF/2-2/EJAET-2-2-34-39.pdf> (cited on: p. 115).
- Sharma, A., Kumar, M. and Agarwal, S. (2015). A Complete Survey on Software Architectural Styles and Patterns. *Procedia Computer Science*, vol. 70. Proceedings of the 4th International Conference on Eco-friendly Computing and Communication Systems, pp. 16–28. DOI: <https://doi.org/10.1016/j.procs.2015.10.019> (cited on: p. 78).
- Sherman, W. R. and Craig, A. B. (2002). *Understanding Virtual Reality: Interface, Application, and Design*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 1558603530 (cited on: p. 45).
- Shi, J. and Tomasi, C. (1993). *Good Features to Track*. Tech. rep. Ithaca, NY, USA (cited on: p. 115).
- Simon, H. (1981). *The Sciences of the Artificial*. 2nd ed. MIT Press (cited on pp. 11, 13).
- Smith, J. M. (2010). The Pattern Instance Notation: A Simple Hierarchical Visual Notation for the Dynamic Visualization and Comprehension of Software Patterns. *Journal of Visual Languages and Computing*, vol. 22, pp. 355–374 (cited on: p. 32).
- Smith, J. M. (2012). *Elemental Design Patterns*. 1st. Addison-Wesley Professional. ISBN: 9780321711922 (cited on: p. 36).
- Sommerlad, P. and Rüedi, M. (1998). Do-it-yourself Reflection. *EuroPLoP*, pp. 337–366 (cited on: p. 178).
- Soulier, K. E., Selzer, M. N. and Larrea, M. L. (2017). Estimating Illumination Direction for Augmented Reality in Real-Time by using Low-Cost Sensors. *Journal of Computer Science & Technology*, vol. 17 (cited on: p. 155).
- State, A., Livingston, M. A., Garrett, W. F., Hirota, G., Whitton, M. C., Pisano, E. D. and Fuchs, H. (1996). Technologies for Augmented Reality Systems: Realizing Ultrasound-guided Needle Biopsies. *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New Orleans, USA: ACM, pp. 439–446. ISBN: 0-89791-746-4. DOI: [10.1145/237170.237283](https://doi.org/10.1145/237170.237283) (cited on: p. 164).
- Statler, S. (2016). *Beacon Technologies: The Hitchhiker's Guide to the Beacosystem*. APress. ISBN: 978-1-4842-1888-4 (cited on pp. 96, 167).
- Stewénius, H., Engels, C. and Nistér, D. (2006). Recent Developments on Direct Relative Orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 60 no. 4, pp. 284–294 (cited on pp. 58, 117).
- Stumberg, L. von, Usenko, V. and Cremers, D. (2018). Direct Sparse Visual-Inertial Odometry using Dynamic Marginalization. *icra 2018 International Conference on Robotics and Automation*. Brisbane, Australia. URL: https://vision.in.tum.de/_media/spezial/bib/stumberg18vidso.pdf (cited on: p. 63).
- Suya, Y., Neumann, U. and Azuma, R. (1999). Hybrid Inertial and Vision Tracking for Augmented Reality Registration. *Proceedings IEEE Virtual Reality*, pp. 260–267. DOI: [10.1109/VR.1999.756960](https://doi.org/10.1109/VR.1999.756960) (cited on: p. 102).
- Sweeney, C. (2015). *Theia Multiview Geometry Library: Tutorial & Reference* (cited on pp. 112, 117, 118, 120).
- Sweeney, C., Flynn, J., Nuernberger, B., Turk, M. and Höllerer, T. (2015). Efficient Computation of Absolute Pose for Gravity-Aware Augmented Reality. *Proceedings of the 2015 IEEE International Symposium on Mixed and Augmented Reality*. Washington, DC, USA: IEEE Computer Society, pp. 19–24. ISBN: 978-1-4673-7660-0. DOI: [10.1109/ISMAR.2015.20](https://doi.org/10.1109/ISMAR.2015.20) (cited on pp. 61, 120).

BIBLIOGRAPHY

- Sweeney, C., Flynn, J. and Turk, M. (2014). Solving for Relative Pose with a Partially Known Rotation is a Quadratic Eigenvalue Problem. *Proceedings of the 2014 International Conference on 3D Vision*. Vol. 1. Washington, DC, USA: IEEE Computer Society, pp. 483–490. ISBN: 978-1-4799-7000-1 (cited on pp. 61, 118, 120).
- Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. 1st. Berlin, Heidelberg: Springer-Verlag. ISBN: 9781848829343 (cited on: p. 48).
- Tahchiev, P, Leme, F, Massol, V. and Gregory, G. (2011). *JUnit in Action*. 2nd. Manning Publications Co. ISBN: 9781935182023 (cited on: p. 180).
- Tan, L., Wang, Y., Yu, H. and Zhu, J. (2017). Automatic Camera Calibration Using Active Displays of a Virtual Pattern. *Sensors*, vol. 17 (cited on: p. 57).
- Tanenbaum, A. S. and Van Steen, M. (2007). *Distributed Systems: Principles and Paradigms*. 2nd ed. Prentice-Hall. ISBN: 0-13-239227-5 (cited on: p. 168).
- Tapp, K. (2016). *Matrix Groups for Undergraduates*. 2nd ed. Student Mathematical Library. American Mathematical Society. ISBN: 9781470427221 (cited on: p. 201).
- Tareen, S. A. K. and Saleem, Z. (2018). A Comparative Analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. Sukkur, Pakistan. DOI: 10.1109/ICOMET.2018.8346440 (cited on pp. 50, 113).
- Thai, T.L. (1999). *Learning DCOM*. O'Reilly Media. ISBN: 9781449308872 (cited on: p. 170).
- The MathWorks Inc (Aerospace Toolbox) (2019-01-01). MATLAB Aerospace Toolbox, vol. R2019. URL: <https://www.mathworks.com/products/aerospace-toolbox.html> (cited on: p. 99).
- Thompson, W., Fleming, R., Creem-Regehr, S. and Stefanucci, J. K. (2011). *Visual Perception from a Computer Graphics Perspective*. 1st ed. Natick, MA, USA: A. K. Peters, Ltd. ISBN: 9781568814650 (cited on: p. 68).
- Tian, M., Zhang, W. and Liu, F. (2007). On-line Ensemble SVM for Robust Object Tracking. *Proceedings of the 8th Asian Conference on Computer Vision - Volume Part I*. ACCV'07. Tokyo, Japan: Springer-Verlag, pp. 355–364. ISBN: 9783540763857 (cited on: p. 111).
- Tisseur, F. and Meerbergen, K. (2001). The Quadratic Eigenvalue Problem. *SIAM Review*, vol. 43 no. 2. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.6.6670&rep=rep1&type=pdf> (cited on: p. 118).
- Tommasini, T., Fusello, A., Trucco, E. and Roberto, V. (1998). Making Good Features Track Better. *Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 178–183. DOI: 10.1109/CVPR.1998.698606. URL: <https://pdfs.semanticscholar.org/41bf/68c8527802082a46112a803ac379d89752b0.pdf> (cited on: p. 115).
- Treiber, M.A. (2013). *Optimization for Computer Vision: An Introduction to Core Concepts and Methods*. Advances in Computer Vision and Pattern Recognition. Springer London. ISBN: 9781447152835 (cited on pp. 60, 201).
- Triggs, B., McLauchlan, P. F., Hartley, R. I. and Fitzgibbon, A. W. (2000). Bundle Adjustment - A Modern Synthesis. *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*. ICCV '99. Springer-Verlag, pp. 298–372. ISBN: 3-540-67973-1 (cited on: p. 62).
- Tsumura, N., Dang, Mi, Makino, T. and Miyake, Y. (2003). Estimating the Directions to Light Sources Using Images of Eye for Reconstructing 3D Human Face. *11th Color and Imaging Conference Final Program and Proceedings*, pp. 77–81 (cited on: p. 155).
- Urban, S., Leitloff, J. and Hinz, S. (2016). MLPnP - A Real-Time Maximum Likelihood Solution to the Perspective-n-Point Problem. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*. Vol. 3, pp. 131–138. URL: https://github.com/urbste/MLPnP_matlab (cited on: p. 60).

BIBLIOGRAPHY

- Vaishnavi, V. K. and Kuechler, W. (2007). *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*. Boston, MA, USA: Auerbach Publications. ISBN: 9781420059328 (cited on: p. 13).
- van der Linden, F. J., Schmid, K. and Rommes, E. (2007). *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Berlin, Heidelberg: Springer-Verlag. ISBN: 3540714367 (cited on: p. 28).
- Van Heesch, U., Hezavehi, S. M. and Avgeriou, P. (2011). Combining Architectural Patterns and Software Technologies in One Design Language. *Proceedings of the 16th European Pattern Languages of Programming (EuroPLoP 2011)*, (cited on: p. 84).
- Van Sickel, J. (2017). *Basic GIS Coordinates*. 3rd. Boca Raton, FL, USA: CRC Press, Inc. ISBN: 9781498774628 (cited on: p. 94).
- Venable, J., Pries-Heje, J. and Baskerville, R. (2012). A Comprehensive Framework for Evaluation in Design Science Research. *Proceedings of the 7th International Conference on Design Science Research in Information Systems: Advances in Theory and Practice*. Ed. by K. Peffers, M. Rothenberger and B. Kuechler. DESRIST '12. Las Vegas, NV: Springer-Verlag, pp. 423–438. ISBN: 978-3-642-29862-2. DOI: 10.1007/978-3-642-29863-9_31 (cited on: p. 16).
- Vince, J. (2011). *Quaternions for Computer Graphics*. Springer-Verlag London. ISBN: 978-0-85729-759-4 (cited on: p. 199).
- Viola, P. and Jones, M. (2001). Rapid Object Detection Using a Boosted Cascade of Simple Features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1. Kauai, HI, USA. DOI: 10.1109/CVPR.2001.990517 (cited on: p. 164).
- Voelter, M., Kircher, M. and Zdun, U. (2004). *Remoting Patterns - Foundations of Enterprise, Internet and Real-time Distributed Object Middleware*. Wiley Series in Software Design Patterns. Hoboken, NJ, USA: J. Wiley & Sons (cited on pp. 37, 169, 171, 191).
- Voss, M., Asenjo, R. and Reinders, J. (2019). *Pro TBB: C++ Parallel Programming with Threading Building Blocks*. 1st. USA: Apress. ISBN: 1484243978. URL: <https://rd.springer.com/book/10.1007/978-1-4842-4398-5> (cited on: p. 85).
- W3C Multimodal Interaction Working Group (2017-07). *Multimodal Architecture and Interfaces (MMI)*. URL: <https://www.w3.org/2002/mmi/> (cited on: p. 148).
- Walls, J. G., Widmeyer, G. R. and El Sawy, O. A. (1992). Building an Information System Design Theory for Vigilant EIS. *Info. Sys. Research*, vol. 3 no. 1, pp. 36–59. DOI: 10.1287/isre.3.1.36 (cited on pp. 16, 17).
- Wang, J. and Olson, E. (2016). AprilTag 2: Efficient and Robust Fiducial Detection. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Daejeon, Korea (cited on: p. 210).
- Want, R., Wang, W. and Chesnutt, S. (2018). Accurate Indoor Location for the IoT. *Computer*, vol. 51 no. 8, pp. 66–70. DOI: 10.1109/MC.2018.3191259 (cited on: p. 167).
- Welicki, L., Yoder, J. W. and Wirfs-Brock, R. (2008). The Dynamic Factory Pattern. *Proceedings of the 15th Conference on Pattern Languages of Programs*. ACM. ISBN: 978-1-60558-151-4 (cited on: p. 178).
- Weng, S., Kuo, C. and Tu, S. (2006). Video Object Tracking using Adaptive Kalman Filter. *Journal of Visual Communication and Image Representation*, vol. 17 no. 6, pp. 1190–1208. DOI: <https://doi.org/10.1016/j.jvcir.2006.03.004> (cited on: p. 114).
- Wieringa, R.J. (2009). Design Science As Nested Problem Solving. *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*. Ed. by V. Vaishnavi.

BIBLIOGRAPHY

- DESRIST '09. Philadelphia, Pennsylvania: ACM. ISBN: 978-1-60558-408-9. DOI: 10.1145/1555619.1555630 (cited on: p. 14).
- Winer, D. (2003-01). *XML-RPC specification, 1999*. URL: <http://www.xmlrpc.com/spec> (cited on: p. 168).
- Wither, J., DiVerdi, S. and Höllerer, T. (2009). Annotation in Outdoor Augmented Reality. *Computers & Graphics*, vol. 33 no. 6, pp. 679–689. DOI: 10.1016/j.cag.2009.06.001. URL: <http://www.cs.ucsb.edu/~holl/pubs/Wither-2009-CandG.pdf> (cited on: p. 69).
- Wither, J., Yun-Ta, T. and Azuma, R. (2011). Indirect Augmented Reality. *Computers & Graphics*, vol. 35 no. 4, pp. 810–822. DOI: 10.1016/j.cag.2011.04.010. URL: http://cgit.nutn.edu.tw:8080/cgit/PaperDL/CUC_111022011402.PDF (cited on: p. 72).
- Wu, C., Yang, Z. and Liu, Y. (2018). *Wireless Indoor Localization: A Crowdsourcing Approach*. Springer. DOI: 10.1007/978-981-13-0356-2 (cited on: p. 96).
- Yang, J., Li, H., Campbell, D. and Jia, Y. (2016). Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38 no. 11, pp. 2241–2254. DOI: 10.1109/TPAMI.2015.2513405. URL: http://jlyang.org/tpami16_go-icp_preprint.pdf (cited on: p. 121).
- Yang, N., Wang, R., Gao, X. and Cremers, D. (2018). Challenges in Monocular Visual Odometry: Photometric Calibration, Motion Bias and Rolling Shutter Effect. *IEEE Robotics and Automation Letters*, vol. 3. DOI: 10.1109/LRA.2018.2846813. URL: <https://arxiv.org/pdf/1705.04300.pdf> (cited on: p. 63).
- Zhang, E. and Mayo, M. (2010). Improving Bag-of-Words Model with Spatial Information. *International Conference Image and Vision Computing New Zealand*, DOI: 10.1109/IVCNZ.2010.6148795 (cited on: p. 111).
- Zhang, Y. and Yang, Y. (2001). Multiple Illuminant Direction Detection with Application to Image Synthesis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23 no. 8, pp. 915–920. DOI: 10.1109/34.946995 (cited on: p. 155).
- Zhang, Y., Zhou, L., Liu, H. and Shang, Y. (2016). A Flexible Online Camera Calibration using Line Segments. *Journal of Sensors*, vol. 2016 (cited on: p. 94).
- Zheng, Y., Kuang, Y., Sugimoto, S., Åström, K. and Okutomi, M. (2013). Revisiting the PnP Problem: A Fast, General and Optimal Solution. *2013 IEEE International Conference on Computer Vision*, pp. 2344–2351. DOI: 10.1109/ICCV.2013.291 (cited on: p. 60).