FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Cross-Platform Development Approaches

For the Mobile Device Platforms Android and iOS

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Oana-Aurora Moraru
Registration Number 1108261

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer

Vienna, 22.07.2014

(Signature of Author) (Signature of Advisor)

Abstract

This paper provides an introduction to cross-platform programming for mobile devices as well as an overview of different object-oriented programming -based cross-platform development approaches. The presented approaches are described, discussed and evaluated based on their advantages and disadvantages, as well as a practical implementation.
The comparison relies on different aspects, including the preparation of the workspace, necessary programs, installations and plug-ins, their price, the supported platforms, the effort to implement, the ease with which the application programming interfaces can be used and much more.
Thus the investment of time, money and energy, as well as other difficulties, which might occur while following an approach, constitute important factors which contribute to the results of the evaluation. In the end the paper suggests which might be the best and easiest approach, respectively which approaches are more appropriate than others in some certain cases.

Contents

CHAPTER 1

Introduction

1.1 Problem Statement

As the smartphone industry is growing, the number of application developers for mobile platforms has multiplied, as Heitkötter et. al [33] state in their work. Furthermore the authors of the paper define applications, also known as apps, as innovative mobile information systems.
However even though there are already millions of applications on the market, some customers still complain about certain applications not being available on all mobile platforms for instance. The main reason for this problem consists in the difficulty of developing mobile applications that are able to run of various platforms.

Heitkötter et. al [33] state in their paper that the market of mobile operating systems for smartphones is heterogeneous, inconsistent and rapidly changing.
The same source states that Google's Android and Apple's iOS are the market leaders closely followed by Microsoft's Windows Phone.
While these mobile operating systems lead the market, a standardized platform has not yet been established. Furthermore these platforms are all substantially divergent, according to Heitkötter et. al [33].

1.2 Motivation

Allen et. al [6] and Heitkötter et. al [33] state in their work, that there are several aspects which complicate an easy development of mobile applications for more than one platform. This hinders the reaching of a larger potential audience by software developers.
First of all the mobile developer needs to be able to code in various languages, in order to write their applications for each platform individually. Secondly the different platforms often require differential tools or other kind of prerequisite.
Other aspects, which often constitute problems, concern the structure of the program, different

membership requirements as well as fees one has to pay in order to be allowed to run or deploy certain applications on specific devices. In the same work Allen et. al [6] give a very informative overview of the development and distribution on different platforms.

1.3 Solution

However, because of these enumerated problematic aspects, the mobile developers tend to resort to cross-platform development approaches. Heitkötter et. al [33] define cross-platform development as a solution which allows developers to implement applications able to run on a range of different platforms in one single step.
This allows for the avoidance of repetition and an increase of productivity. The difficulty of this task consists in the fact that cross-platforming approaches not only need to provide adequate generality, in order to allow supply of applications for various platforms, but also must enable software developers to take advantage of certain possibilities and advantages of specific smartphones.

Figure 1.1: This figure shows the logograms of the market-leading mobile operating systems: Windows, iOS, Android, HTML5 and BlackBerry (from right to left). The picture can be found on the following source [3].

1.4 Aim and Structure of the Work

This bachelor thesis describes, compares and evaluates a series of cross-platform approaches for mobile devices. Generally, the cross-platform development approaches can be separated into two categories: Object-Oriented Programming -based and Web Technology -based Approaches. However, this paper concentrates on the object-oriented programming -based procedures. These procedures rely on Java, C/C++ and C# as programming languages and are additionally discussed based on a simple practical 3D mobile application.

This is meant to test, compare and evaluate the ease of development and use of typical application programming interfaces, which are normally used in 3D applications and game development.

Furthermore, the first part of the paper presents a short overview of the various approaches, as well as a table illustrating their general and best features. This is meant to provide the reader with a first overall impression.
The second part of the paper consists in a much more in-depth description of each cross-platform development approach, followed by a more extensive comparison, which can be found in the section 'Comparison of the object-oriented programming -based approaches.
Finally the conclusion at the end of the paper will summarize the most important results of the comparison.

CHAPTER 2

Overview

2.1 Object-oriented programming -based cross-platform development approaches

In this section an overview of the object-oriented programming -based mobile cross-platform development approaches libGDX, Murl Engine, Unity, MoSync, Xamarin and Marmalade SDK is given. The basic criteria on which the selection of these frameworks depended was represented by their ability to support the mobile platforms iOS and Android.
Furthermore, the support on 3D topics of the frameworks played an important role in their selection, since the practical application for comparison purposes, which accompanies this bachelor thesis, represents a small 3D demonstration.

libGDX

LibGDX is a cross-platform game development framework from Badlogic Games. The used programming languages are Java, Kotlin and Scala, as well as any other GNOME Display Manager (GDM) language. The deployment to one of the following platforms: Desktop, Windows, Linux, Mac, Android, iOS or BlackBerry, can be realized quite easily at the touch of a button. When using Java, the project can also be deployed as HTML5 web application.
LibGDX offers many advantageous built-in functions, such as Input processing from the touchscreen and model loading, which simplify the coding effort of the developer. More information about libGDX can be found on their official web site [38].

Murl Engine

The Murl Engine represents a free platform-independent tool for developing games, multimedia and mobile applications. The coding language is C++. The Murl Engine embraces a slightly different approach than the other discussed frameworks, which is meant to provide more technical control and flexibility. The developer can maintain a good overview of the elements of the

scene and their location, after having defined these in an XML file. The elements can then be referenced by ids, when implementing the logic of the application.
However the flexibility comes at the price of an increased implementation effort and currently there is not much support on 3D topics, since the developers are still working on the framework and the website. The supported platforms are: Android, iOS, Windows and Mac OS X. Further information about the Murl Engine can be collected from their official web site [95].

Unity

Unity is a cross-platform game-development ecosystem, which contains an integrated development environment, a game engine with an integrated rendering engine, which is meant to simplify the programming effort and comes with a fast and intuitive set of tools and workflows. This enables a fast and easy generation of 2D and 3D content.
The used programming languages are C#, UnityScript and Boo. Unity also disposes of an Asset Store and a big community willing to share their knowledge. The system focuses on game development. Unity supports the following platforms: Windows, OS X, Ubuntu, Android, iOS, Blackberry10, Windows Phone 8, Unity Web Player, Abobe Flash, PlayStation 3, PlayStation 4, Xbox 360, Xbox One, Wii U and Wii. More details can be gathered from the official web site of Unity [124].

MoSync

MoSync SDK is a free, open-source tool for cross-platform mobile application development. It comprises tools for building native applications and an Integrated Development Environment, which is based on Eclipse. The used programming language is C/C++ and the supported platforms are: Android, Blackberry, iOS, Java ME MIDP, Moblin, Symbian, Windows Mobile and Windows Phone.
Unfortunately MoSync is no longer maintained, despite of the fact that it provides various useful tools and supports up to eight platforms. However some support is still provided by the community [66]. More information can be found on the official website of MoSync [55].

Xamarin

Xamarin is a software which enables users to build cross-platform native applications for Android, iOS, Mac and Windows. The download includes Xamarin.iOS, Xamarin.Android, Xamarin.Mac and Xamarin Studio, an Integrated Development Environment. The cross-platform applications built with Xamarin share a C# code basis.
Furthermore, even though Xamarin currently disposes of a quite big community, one does not find much information available for free and the free Xamarin version of the IDE has a restricted compiled code size. However there exists a 30 days free Trial of the Business Edition.

Since Xamarin focuses on productivity and utility applications, it must be used in combination with MonoGame, an extended 3D Engine, to provide support for 3D application and game development, even though model loading is still quite tricky. Xamarin and MonoGame support following platforms: Windows, Windows Phone, Mac OS, iOS, Android, Linux, PlayStation Mobile and OUYA console. More information about Xamarin and MonoGame can be found on their official web sites [71], [74].

Marmalade SDK

Marmalade SDK is a free cross-platform application development tool and game engine. It is a C++ framework enabling the development in Visual Studio or Xcode. The documentation, code samples, tools and library files are all available for free. Marmalade provides all the benefits of C++, including high performance, openness, flexibility and low-level access, when going cross-platform.
Furthermore Marmalade SDK also supports programming in Objective C, LUA and HTML5, by means of Marmalade Juice, Marmalade Quick and Web Marmalade and the deployment to one of the following platforms is possible: Android, iOS, Windows Phone 8, BlackBerry10, Tizen, Mac OS X, Windows Desktop, LG Smart TV, Roku 2 and Roku 3. More information on Marmalade SDK can be found of their official website [47].

2.2 Table of Comparison

The table represented below (Table 2.1) provides an overview of the six object-oriented programming -based cross-platform approaches described in this bachelor thesis. The first column from the left comprises main features of the approaches, which provide an essential basis for the extensive comparison of the approaches.
While the complete comparison as well as its results can be found in the chapters 5.1 and 6 located towards the end of this bachelor thesis, the following table lists the most important features of the approaches in a most compact way, thus enabling a first overall impression.

Main Features	**LibGDX**	**Murl Engine**	**Unity**	**MoSync**	**Xamarin with MonoGame**	**Marmalade SDK**
Developer Team	Badlogic Games	Spraylight GmbH	Unity Technologies	MoSync AB	Xamarin, developers of Mono	Marmalade Technologies Limited
Free Version availability	yes	yes	yes	yes	Xamarin: yes, but with limited app size. MonoGame: yes.	yes
Versions in exchange for a Fee	-	Basic, Standard, Pro	Pro	Basic Pro, Gold Pro	Indie, Business, Enterprise	-
Focus mainly on mobile development	no	no	no	yes	no	yes
Programming Languages	Java, Kotlin, Scala, or any other GDM language	C/C++, LUA	C#, UnityScript, Boo	C/C++, HTML5, JavaScript	C#	C++, Objective C, LUA, HTML5
IDE	Eclipse, IntelliJ, NetBeans	Xcode or favored IDE	Unity IDE	MoSync IDE	Xamarin Studio, Visual Studio	Visual Studio, Xcode
Download and Installation Effort	easy to medium	easy to medium	easy	easy	medium to hard	easy
Registration required	no	no	no	no	no	yes
GUI tool for deployment	yes, the libGDX Project Generator	yes, the Dashboard	no	no	no	yes, the Marmalade Hub

Main Features	LibGDX	Murl Engine	Unity	MoSync	Xamarin with MonoGame	Marmalade SDK
Supported Mobile Platforms	Android, iOS, BlackBerry	Android, iOS	Android, iOS, Blackberry10, Windows Phone 8	Android, iOS, Blackberry, Java ME MIDP, Moblin, Symbian, Windows Mobile, Windows Phone	Android,, iOS, Windows Phone, PlayStation Mobile	Android, iOS, Windows Phone 8, BlackBerry10, Tizen
Other Supported Platforms	Windows, Linux, Mac, HTML5	Windows, Mac OS X	Windows, OS X, Ubuntu, Unity Web Player, Abobe Flash, PlayStation 3, PlayStation 4, Xbox 360, Xbox One, Wii U, Wii	-	Windows, Mac OS, Linux, OUYA console	Windows, Mac OS X, LG Smart TV, Roku 2, Roku 3
Choice of installation path	yes	no	yes	no	yes	yes
Familiariza-tion with the Framework	easy	medium	easy	medium	medium	easy
Tutorials availability	yes	yes, but quite few	yes	no	yes, in exchange for fees	yes
Community	yes	quite small	yes	yes, but framework no longer maintained	yes, in exchange for fees	yes
Deployment to other platforms	easy	easy	easy	easy to medium	quite hard, since it has to be done manually	easy
Running the Desktop Project	easy	easy	easy	-	easy	easy
Running the iOS Project	easy to medium	easy, by means of Xcode	easy, by means of Xcode	easy to medium, by means of Xcode	easy, after project generation	easy to medium, by means of a deployment tool

Main Features	LibGDX	Murl Engine	Unity	MoSync	Xamarin with MonoGame	Marmalade SDK
Running the Android Project	easy	easy, with the GUI	easy	easy	easy, after project generation	easy to medium, by means of a deployment tool
Rendering API support	OpenGL ES 1.0, 1.x, 2.0, 3.0, OpenGL	OpenGL ES 1.1, 2.0, OpenGL, DirectX 9.0, 11.1	OpenGL ES 2.0, OpenGL 2.1, 3.2, DirectX	OpenGL ES 1.1, 2.0	OpenGL ES 2.0, OpenGL 1.x	OpenGL ES 1.x, 2.x
Camera Implementation	easy	easy to medium	very easy	hard, not implemented	hard, not implemented	easy
Camera Controller Implementation	very easy	medium	medium	medium to hard	hard, not implemented	medium, not implemented
Model Loading	easy	medium, conversion to Murl Meshes by means of a Scene Converter tool	very easy	hard, not implemented	hard, conversion necessary due to not implemented Content Pipeline	medium, exportation necessary from a modeling package
Supported Model Formats	.OBJ, .G3DT, .G3D, .MD2, .MD5	model formats supported by Assimp [12]	.FBX, .DAE, .3DS, .DXF, .OBJ	model formats supported by Assimp [12]	.X, .XNB, .FBX	GROUP file
Object Movement	easy	easy	easy, script needed	medium	easy	easy
Lighting Implementation	easy	easy to medium	very easy	medium, not implemented	easy	easy
Touchscreen Interface	very easy	medium	easy to medium	medium to hard	medium	easy
Image Loading	easy	easy	medium	easy to medium	easy	easy
Supported Image Formats	.JPG, .PNG, .BMP	.PNG, .JPG, .WEBP, .DDS, .TGA	.JPEG, .PNG, .GIF, .BMP, .TGA, .IFF, .PICT	.JPEG, .PNG, .SVG	.BMP, .DDS, .DIB, .HDR, .JPG, .PFM, .PNG, .PPM, .TGA	.PNG, .BMP, .GIF, .JPG, .TGA
Sound	very easy	easy	easy	very easy	easy	medium

Main Features	LibGDX	Murl Engine	Unity	MoSync	Xamarin with MonoGame	Marmalade SDK
Supported Sound Formats	.WAV, .MP3, .OGG	.WAV, .OGG	.MP3, .OGG, .AIFF, .WAV	.MP3, .WAV	.WAV, .MP3, .WMA, .XAP	.MP3, .ACC, .QCP (with additional effort: .WAV)

Table 2.1: Comparison of cross-platform development approaches

2.3 Web technology -based cross-platform development approaches

In this section a brief overview of the web technology -based mobile cross-platform development approaches Appcelerator, Widgetpad, RhoMobile, PhoneGap, Sencha and Cocktail is given. The basic criteria on which the selection of these frameworks depended was represented by their ability to support cross-platform mobile programming.
However, these approaches will not be further discussed in this bachelor thesis, since the focus lies on object-oriented programming -based cross-platform development approaches.

Appcelerator

Appcelerator is a private mobile technology company located in Mountain View, California. Its most significant product is Titanium. Titanium is an open-source software development kit which can be used for cross-platform mobile development. The Titanium framework enables the creation of mobile applications for the following platforms: iOS, Android, Windows Phone, BlackBerry OS, and Tizen, using a single JavaScript codebase.
Another important product of Appcelerator is the Appcelerator Platform, which is an enterprise software suite on which mobile applications can be developed, analyzed, tested and deployed. The programming languages which can be used are: HTML5, CSS3, JavaScript, PHP, Python and Ruby. More information on Appcelerator can be collected on their official webpage [9].

Widgetpad

Widgetpad, created by Satoshi Nakajima, represents a collaborative, open source web-based mobile development platform. It enables web developers to implement interactive, stand-alone smartphone applications. The used standard web technologies are HTML 5, CSS 3, and JavaScript. Widgetpad also enables an easy distribution of the applications to all app stores. Using Widgetpad one can create mobile applications for operating systems such as iOS and Android as well as for WebOS.
Widgetpad is available through the web via the Software as a Service (SaaS) model. Furthermore, Widgetpad includes everything which is needed to implement cross-platform smartphone applications, as for example project management, a debugger, source code editing, versioning, collaboration and distribution directly from ones own browser, as stated by the following source [19].

RhoMobile

The RhoMobile Suite, previously known as Rhodes Framework, is an open-source framework for building native cross-platform applications. It is developed by Motorola and currently owned by Motorola Solutions. RhoMobile guarantees the exact same look and feel of the produced application independently of which operating system, device brand or screen size is used.
RhoMobile is released under the MIT license and uses Model-View-Controller pattern. While the programming language used for the views is HTML, controllers can be written in Ruby.

The supported mobile platforms are: iOS, Android, Windows Mobile, BlackBerry and Windows Phone 7. Symbian is no longer supported. More information on RhoMobile can be gathered on their official website [67].

PhoneGap

PhoneGap is a mobile development framework developed by Nitobi. It has been purchased by Adobe Systems in 2011. PhoneGap enables software developers to implement applications for mobile devices. The used programming languages are JavaScript, HTML5, and CSS3, instead of using the platform-specific application programming interfaces. The HTML, CSS and JavaScript code is then wrapped up depending on the platform of the target device.
The features of HTML and JavaScript are being extended in order to work with the specific devices, so that the resulting applications are hybrid. This means that they are neither purely native, since the layout rendering is realized via web views instead of the native UI framework, nor truly web-based, since they are not just web apps, but packaged as applications for distribution with access to the device's native application programming interfaces. In the newest versions it is allowed to freely mix native and hybrid code snippets.
More information on PhoneGap can be collected on their official website [4].

Sencha

Sencha Touch is a user interface JavaScript library and also a framework built specifically for the mobile web. It enables web developers to implement mobile web applications user interfaces, which provide the look and feel of native applications on the supported mobile devices.
Sencha is entirely based on web standards as for example HTML5, CSS3 and JavaScript and aims to enable developers to implement HTML5 based mobile applications fast and easily. The supported mobile platforms are Android, iOS, Windows, Tizen and BlackBerry. Sencha furthermore aims to provide a native-app-like experience inside a browser. More information on Sencha is available on their official website [87].

Cocktail

Cocktail is an open-source rendering engine which implements the W3C HTML and CSS standards. It can be used to create native and web applications and is written as a library for Haxe language. Cocktail is a project supported by Silex Labs released under the MIT license. With Cocktail the developers can implement HTML/CSS application in Haxe and build them for OpenFL and flash/Air.
Since Cocktail uses the standard document object model (DOM) application programming interface, one can also create application to ordinary JavaScript. One can embedded Cocktail inside one's application or game, similarly to a webview, and thus build one's own user interface in HTML/CSS. More information on Cocktail can be collected on the subsequent webpages [88], [30], [37].

CHAPTER 3

Practical Application for Comparison Purposes

The object-oriented programming -based cross-platform development approaches presented in this paper are not only evaluated based on theoretical knowledge and general information, but also on the basis of a practical implementation of a simple game application.
The application is quite elementary, however it comprises image loading, 3D model loading, sound loading, lighting, a free movable camera and a moving object represented by the model.

The scene contains a 2D background image of a galaxy and a 3D model. The model slowly rotates around its vertical axis, while a background music is playing. The camera can be moved freely by touch interaction, thus the model seems to be freely manipulable.
The background image can be found on the following source [129], while an arbitrary model can either be downloaded from the internet or modeled by the developer by means of a modeling packages like Maya, 3DS Max or Blender. The model represented in the image below has been used in the practical implementation of a demo application using libGDX and can be purchased from the tutorial web site, which references the download page [137]. The background music can be downloaded from the following web site [68].

This application has been implemented for each of the presented object-oriented programming -based cross-platform development approaches using the programming languages Java, C/C++ and C#. It serves for the purpose of comparison, since during the practical implementation of the same application by means of different frameworks, one can truly observe the differences between the various approaches. Likewise, the ease of development and use of Application Programming Interfaces can be analyzed and evaluated.

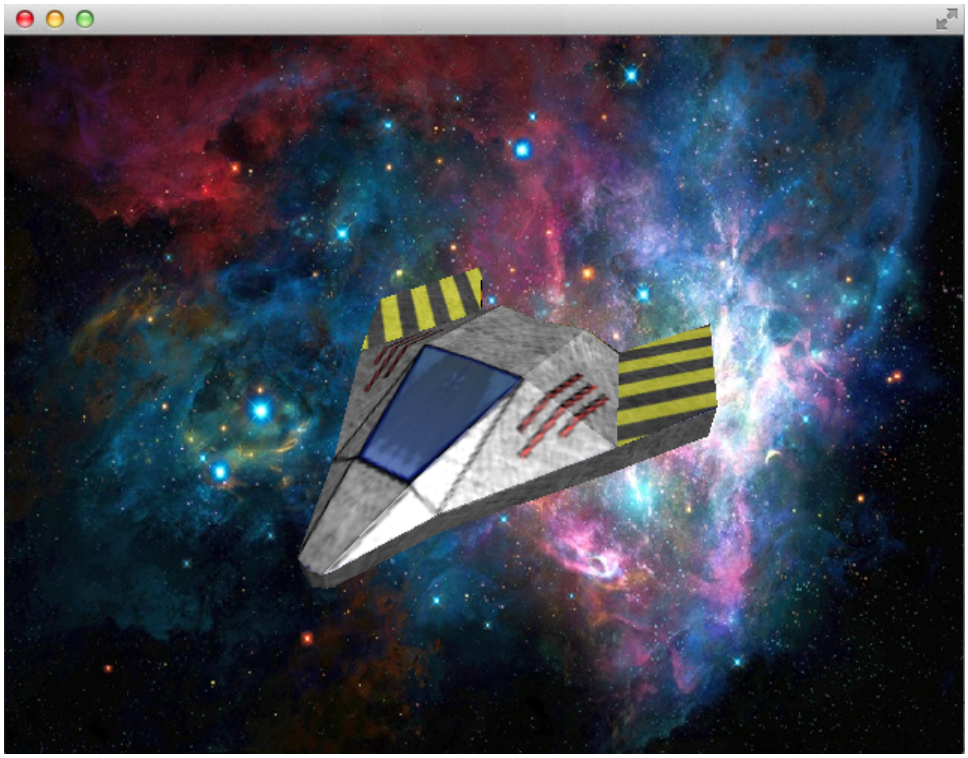


Figure 3.1: This figure shows a Screenshot of the game application using libGDX realized on a Mac computer. It depicts a galactic background image and a 3D model of a space ship.

CHAPTER 4

Object-oriented programming -based cross-platform development approaches

4.1 libGDX

General Information

LibGDX consists of a cross-platform game development framework. It disposes of an extensive documentation, which comprises, among other things, a Javadoc and a Wiki, however it changes frequently due to the numerous contributors, who constantly add new features and improvements.
LibGDX is an extremely convenient and useful framework, since the developer only needs to write the code once and the framework is able to distribute it for various platforms. LibGDX also includes some quite advantageous classes like Screen or ApplicationListener.

Figure 4.1: This figure shows the logograms of libGDX on the left side and Badlogic Games on the right side. The references for the illustrations can be found on the following sources [39], [111].

Developer Team

LibGDX is a project created by Badlogic Games, a small game development shop with their headquarters in Graz, Austria. Mario Zechner is simultaneously the main developer and business leader. Further information on Badlogic Games can be found in the following source [40].

Programming Languages

For implementing a cross-platform application using libGDX, one can choose between the following programming languages: Java, Kotlin, Scala or any other GDM language.

Integrated Development Environments

The libGDX community offers considerable support for three different programming environments: Eclipse, IntelliJ IDE and NetBeans.

Supported Platforms

One can easily deploy their application to the subsequent platforms: Desktop, Windows, Linux, Mac, Android, iOS or BlackBerry.
In case one writes their application in Java, it can also be deployed to the browser as an HTML5 project. LibGDX accomplishes this by transcripting the Java code to Java Script, allowing the developer to run his game directly in his browser.

Gradle

As stated by Muschko [70], Gradle is a JVM-based build tool, which constitutes quite an improvement of already well-know tools like Ant or Maven. According to Muschko, Gradle enhances the better ideas of these tools.
LibGDX uses Gradle as a dependency management tool, therefore Gradle handles the build process and the IDE integration. Gradle even enables programmers from the same team to develop in different IDEs, as long as, if the team uses a distributed revision control and source code management system, they don't commit the IDE specific files to the source control.
However if the team uses Git as such a system, the .gitignore File integrated in all libGDX projects will automatically handle this problem.

Prerequisite

LibGDX is a Java JVM Game Development Framework for multiple platforms, however in order to be able to develop for these platforms certain prerequisites are necessary. This preconditions involve mostly the setup of the development environment.
LibGDX provides extensive support for this in form of Wiki pages, where each step which needs to be fulfilled by the developer in order to be able to work with libGDX is explicitly described.
First of all it is necessary that the developer decides which Integrated Development Environment he will use. According to this decision he will select the respective Wiki page provided by the

libGDX community, where the following steps are specified. In case the developer prefers to use the command line, there also exists a wiki article which specifies the setting up of the environment using the command line.

In the following paragraph the setting up of the Integrated Development Environment (IDE) Eclipse and the installation of some software will be elucidated. This requires approximately an hour, if no unexpected problems emerge.

Installation of Software

- Java JDK - First the Java JDK needs to be installed. This can be downloaded from the official oracle site [73], involves approximately 150MB and is absolutely necessary in order to be able to program in Java.
- Eclipse - An Integrated Development Environment will be needed for the purpose of typing and compiling code. The IDE Eclipse incorporates 150-250MB, depending on the downloaded version and can be downloaded on the official Eclipse homepage [110].
- Android SDK - The next thing one needs to install is the Android Software Development Kit. This represents a necessary installation if the developer intends to export his LibGDX project to Android. This SDK can be downloaded on the official Android Developer web site [7] and involves about 100MB. However it is not recommendable to download the whole Android Development Tools (ADT) Bundle, since the IDE has already been installed in a previous step, so that the button labeled 'GET THE SDK FOR AN EXISTING IDE' should be pressed.
- GWT SDK - Another required piece of software is the Google Web Toolkit SDK. This is needed in case the developer decides to export his LibGDX project also to HTML5 or JavaScript. It can be downloaded on the Google Web Toolkit Project's official web site [32] and incorporates about 100MB.

Installation of Plug-ins

After finalizing all installations mentioned above, the next step consists in the installation of some plug-ins for the Eclipse environment. These plug-ins are absolutely necessary in order to enable a correct functioning of any libGDX project. However all the plug-ins can be installed in a simple and quite similar way.

First of all Eclipse needs to be opened and the location of the workspace needs to be chosen. Then the 'Help' item in the upper toolbar needs to be selected, followed by 'Install New Software'.
This causes a widow to open, which allows the installation of all 3 plug-ins by the same pattern: the link to the desired plug-in must be copied and pasted into the 'Work with' text box. Afterward the 'Add' button is to be pressed and the developer must ensure that the 'Group Items by

Category' option is enabled. After this the developer must select the nodes he wants to install from the presented list.
After having clicked the 'Next' button, the list of software to be installed should be carefully read and the 'Next' button should be pressed once more. The license must be accepted, followed by the pressing of the 'Finish' button. As soon as this happens the plug-in will install itself and Eclipse will be rebooted.

The links to the plug-ins to be installed are listed below:

- Gradle plug-in - the link 'http://dist.springsource.com/release/TOOLS/gradle' [1] represents the latest release of the plug-in. The link has to be copied into the 'Work with' text box. From the presented list of nodes 'Extensions / Gradle Integration' must be selected, followed by 'Gradle IDE'.
- Android ADT-plug-in - the link 'https://dl-ssl.google.com/android/eclipse/' [2] must be copied into the 'Work with' text box. From the list of nodes all of them will be selected.
- GWT plug-in - the link 'https://dl.google.com/eclipse/plugin/4.3' [31] must be pasted into the 'Work with' text box. From the list of nodes all of them will be selected.

Similar steps need to be followed if the developer prefers the IntelliJ IDE or Netbeans instead of Eclipse. Further information can be found on the libGDX Wiki web site.

Installation of the libGDX jar

In order to be able to create cross-platform libGDX projects, one last installation is necessary, namely the LibGDX setup jar. The jar can be downloaded from the official LibGDX web site [21].

Special Prerequisite for iOS

Further things need to be taken into consideration if one intends to program for iOS. This is an important aspect, especially for programmers who develop for iOS for the first time, since there are certain preconditions one has to fulfill before being able to develop for iOS.
The completion of these preconditions might take a few days, thus the possibility of a time delay as well as a monetary investment must be taken into consideration.

1. A Mac - The development for iOS requires a Mac laptop or Mac personal computer. iOS development can not be achieved on Windows or Linux. However the Eclipse IDE can be used normally, as described above.
2. XCode - The latest version of XCode is also necessary, in order to enable the provisioning of Apple devices. To provision an Apple device the setting up of a provisioning profile is necessary. A provisioning profile is represented by a set of digital entities, which enable the connection of devices and developers that have a valid Apple developer account, as stated by the following source [131]. The provisioning is absolutely required, since one can not deploy the project on an Apple device otherwise.

3. RoboVM plug-in - the RoboVM plug-in needs to be added to Eclipse. The installation process consists in the opening of the Eclipse Marketplace, followed by a selection of the RoboVM compiler to install. This plug-in should be updated as often as possible.

Registration as an Apple Developer

Some further information one should be aware of if intending to develop for iOS is that one first needs to be a part of the Apple community by registering as an Apple Developer. This can be achieved for free and simply requires the Apple ID and password of the developer.
After the registration one has access to the Developer Member Center. As a part of the Member Center the developer can join one of the programs Apple offers. The admission to one of the three developer programs: iOS Developer Program, Mac Developer Program or Safari Developer Program, is crucial, since one can not develop for iOS devices without it.

One can gain the membership to one of these programs by paying a fee of 100 USD, respectively 80 EUR, online by credit card. In order to receive the membership one must wait a few working days. Meanwhile a background check will take place and the developer must also send a copy of his identity card in .pdf format to Apple.
After all of this is settled and Apple decides that everything is alright, the developer will be informed by e-mail that he is now a member of the requested developer program. As soon as this happens, the development for iOS can start. More information can be found on the following web site [23].

Addition of the Apple Device to the Member Center

A next step to be fulfilled concerns the addition of the iOS device one would like to run or test their project on to the Member Center.
This can be achieved by opening XCode and then plugging in the respective iOS device to USB on the Mac. After this the device must be informed that it is secure to trust this computer, in case one connects the iOS device to the respective Mac for the first time.
After this the 'Organizer' window will open. The 'Use for Development' button must be clicked, if this has never been done before. In order to connect the device with the Apple Developer Account of the developer, the latter needs to press the icon above 'Add to Member Center'. The Apple Developer ID should then be typed in. Since the developer should already be a member of the Developer Member Center, the developer is able to request a developer certificate for the connected iOS device, as soon as the dialogue stating 'Certificate Not Found' appears.

Normally this automatically triggers the download of a developer profile, however if it fails, one needs to log in to the 'developer.apple.com' [10] Member Center and download the provisioning profile manually.
After this has been done, one needs to return to the Organizer and select 'Provisioning Profile' from the left menu and press the 'Add' icon. The previously downloaded file must then be selected and added to the provisioning profile.

Further details and information concerning this topic are available on the following web site [24]. The device can now be used for developing, running, testing as well as deploying purposes.

Project Creation

After the setting up of the environment is completed, the following step is represented by the creation of the LibGDX project. In order to create a project, the libGDX jar file must be executed. A window should appear, where one should specify the following details related to the new project:

- Name - The name of the project also stands for the name of the game to be developed.
- Package - The package which will contain the project should be named after the subsequent pattern: 'com.yourname.yourgame'.
- Game class - The game class represents the main class of the project. All main functionality of following classes, which the developer might add, should be invoked by this class. According to the libGDX naming convention, the main class should be named the same as the project's name. However, no spaces should separate the words and each word should begin with a capital letter.
- Destination - The destination represents a path to where the project should be created on the developer's computer by the libGDX setup application.
- Android SDK - In case one would like to deploy their game for Android, the path to the installation of the Android SDK on the respective computer must be specified.

Then the libGDX version one would like to use must be selected. It is recommended that one should chose the latest version, since that is the one most likely to contains the last bug fixes. At the moment, the latest version is illustrated by the 'Nightlies' version.
Subsequently, the developer must choose the platforms for which he wants to develop the application. In order to do so, he must mark one or more of the following check boxes: Desktop, Android, iOS, HTML.

Additional Libraries

Additionally the developer also disposes of the possibility to add further libraries to the project. Bullet, Freetype, Tools, Controllers, Box2D and Box2DLight are placed at his disposal.
Bullet is a widely used 3D physics engine. Freetype represents a True Type Font (TTF) library. Tools allow the use of Texture Packer, a BitmapFont creator and a Particle Editor. Controllers allow the use of controllers in order to be used as input for the game. Box2D is a common 2D physics enigne and is also selected by default, while Box2DLight represents a 2D lighting library based on Box2D. These libraries add addiotional functionality to the game.

As soon as the developer completes these tasks, he can press the 'Generate' button. This triggers an automatic download of all additional files which are necessary for the setup of the

Figure 4.2: The window depicted in this figure appears when the libGDX setup jar file is executed. It contains several input fields and check boxes, which are meant to simplify the input of the new libGDX project's details for the user. The picture can be found on the following source [112].

project and libGDX will generate the project. This might take some time if executed for the first time.

Project Folder

The project folder, generated in the path folder specified by the developer, contains among other files a directory for each platform which has been targeted by the developer.
The project folder additionally includes a directory called 'core'. The core project contains all the source code the developer will write.
The sub-project packages named by the targeted platforms are responsible for spawning the code from the core project on the respective platform. So the 'android' directory will include a fully defined Android project, with all the things and files needed for an Android project. For example a Resource directory for icons and images, the Android Manifest as well as the source code for the Android project. This source code merely contains a Launcher, which ensures that the code from the core project can be run on Android devices or simulators.
The same principle is applied to all other platforms, for each platform-specific sub-project contains a Launcher, which facilitates the running of the developer's code from the core project on the particular platform.

Gradle Files Besides the directories for the various platforms, the project folder of the game also includes additional files and folders which are necessary for Gradle. As previously specified, Gradle represents a build and dependency-management system. The build system takes all the source folders and assets in order to package them up for the respective platform.
Furthermore, a dependency-management system allows the specification of the dependencies of a project. For example the involved game has a clear dependency on libGDX. All dependencies are listed in the 'build.gradle' file. In this file one can change the version of already existing dependencies or add new ones.
More information on dependency management with Gradle can be found on the libGDX Wiki web site [25].

Assets The asset files of the game, for examples icons or images, are typically to be located in the 'assets' folder of the Android project. All other sub-projects normally reference that folder. Another 'assets' folder is contained by the desktop project for example, but that folder merely represents a linked folder, that links to the 'assets' folder of the android project. The same principle applies to the HTML and iOS projects. If one however does not want to develop their project for Android, there will be no Android package or sub-project. In this case the 'assets' folder will be located either in the core project or in the desktop project.

Importing the Project

Once the project is successfully created, it can be imported into Eclipse. This is done by opening Eclipse, clicking on the 'Files' item from the upper tool bar and then on 'Import'.
Afterwards 'Gradle', 'Gradle Project' must be selected and the path to the recently created libGDX project must be specified. Then 'Build Model' must be pressed, which causes a list of all the projects for each previously specified platforms to appear after a few seconds.
The developer must ensure that all projects are selected, which is done by simply selecting 'Game Project', and can then press 'Finish'. This causes Eclipse to import the libGDX project

containing all sub-projects for the different platforms. The first time this is done it might take some time, because additional dependencies must be downloaded.
Further information regarding the creation of the libGDX project can be found here [112].
After setting up the libGDX environment and importing the project into Eclipse, the developer can finally start programming.

Running the project

As far as the running of the project is concerned, the game can not be run from the core project. However running the project can be executed from all other sub-projects standing for the various platforms.

Running the Desktop Project

In order to run the game on the Desktop, one first has to right click on the desktop project, then select 'Run As', 'Java Application'. Afterward a desktop starter class must be specified, such as 'DesktopLauncher.java'.

Running the Android Project

To run the game on Android one first has to make sure, that either an Android Simulator is installed, or that an Android device is connected to the computer. Additionally the respective device must be recognized by the Dalvik Debug Monitor Server (DDMS).
After ensuring that the device is available, one must right click on the Android project and then select 'Run As', 'Android Application'.
For more information concerning the DDMS consult the Android Developer Guide [8].

Running the iOS Project

For the running of the application on iOS RoboVM, the developer can choose between two possibilities.
The first option he has is to run the game on an Apple device. However he necessarily has to make sure that the correspondent device is provisioned. Otherwise the running of the project on the device will be precluded.
For more information concerning the provisioning of an Apple device, read the upper chapter entitled 'Special Prerequisite for iOS'.
Once the device is provisioned, one can run the game on it by right clicking on the RoboVM project and then selecting 'Run As', 'iOS Device Application'.

The second option is represented by the developer using an iOS Simulator. The running of the project is realized quite similarly, however 'Run As', 'iOS Simulator Application' must be selected.

Running the HTML Project

Before being able to run the HTML project, one must create a new configuration. In order to do this, the developer has to right click on the HTML project, then select 'Run As', 'External Tools Configuration'. A name should then be given to the configuration, such as HTML for example, and the location field must be set depending on the operating system the developer uses.
If the development takes place on Windows, the location field should be set to 'gradlew.bat'. Otherwise, if the development takes place on Linux or Max, it should be set to 'gradlew'. Both filed can be found in the project folder. Furthermore the working directory should be set to the project's root folder and in the 'Argument' input field the developer should enter 'html:superDev'. Then 'Apply' and 'Run' are to be pressed.

After the code server is ready, which is indicated in the console view, the developer should open the browser an enter the URL 'http://localhost:8080/html'. The application should then run on the browser.
One can change the code or assets without closing the server. After one or more changes, the developer can simply press the 'SuperDev Refresh' button located in the browser, which will recompile the project and reload the page.
This creation of a new configuration represents a bug fix in the Gradle tooling API. However after this is accomplished, the running of the HTML5 can be simplified by using the Gradle integration.

Debugging the Project

Debugging the application is quite similar to running the project, the only difference being that one has to launch their configuration via 'Debug As' instead of 'Run As'. This can be applied to the Desktop and Android projects.
However, one has to take into consideration that debugging is presently not supported by Robo VM, as stated by the following source [28].
The debugging of the HTML build takes place primarily in the browser in the following manner: the developer has to run the HTML application like previously specified. After that he must bring up the developer tools by pressing F12 in Chrome. In the sources tab the Java file he wants to debug can be found and opened. He can then set breakpoints, step in or over and inspect the variables using source maps. As before, the SuperDev Refresh button can be pressed after a change in the code, however the server process must be left running.

Application Packaging

The easiest way to package the application is using the command line. This means that the developer merely has to enter a Gradle command in the command line, which will automatically generate a JAR, APK or IPA file, for example, which can then be run on various devices.
Nevertheless it is also possible to use the Gradle tasks from within Eclipse.
More information concerning packaging libGDX applications for several platforms using Gradle

tasks can be found in the 'Gradle on the Command Line' documentation [26] on the Wiki web site of libGDX.

Implementation

The application described in the section 'Practical Application for Comparison Purposes' can be implemented using libGDX. In order to do so, the required software and plug-ins have to be installed. Since the application should work on both Android and iOS, the acquirement of the Apple Developer Membership as well as the addition of the Apple Device to the Member Center, are necessary. After doing so, the Startup project can be easily created, imported into Eclipse and run on both Andoird and iOS devices.

OpenGL and OpenGL ES

According to Munshi et al. [69], while OpenGL ES is an API for advanced 3D graphics, preponderantly used in handheld or embedded devices, such as cell phones, OpenGL represents a standard cross-platform 3D API mostly used for desktop systems running on various operating systems.

LibGDX disposes of interfaces which enable a direct OpenGL ES 2.0 and 3.0 access. However, if the used LibGDX version is 0.9.9 or earlier, even OpenGL 1.0 and 1.x are available.
OpenGL ES functions are being mapped to the desktop OpenGL functions, resulting in the emulation of OpenGL ES on the desktop. Nevertheless, some platforms, such as iOS, Google Web Toolkit (GWT) or HTML, merely support OpenGL ES 2.0. Further information on the subject are provided by the following web site [27].

Integrated 3D API

LibGDX offers an integrated 3D API, so no further libraries have to be integrated. The main class must implement ApplicationListener, so that the methods create(), render(), dispose(), resume(), resize(int width, int height) and pause() can be overridden. All objects will then be declared and instantiated within the create() method, and drawn within the render() method. The instantiated objects should also be disposed of in the dispose() method. This way no memory leaks will be generated.
The following web page [136] provides a very useful tutorials by Xoppa, which illustrate the 3D basics using libGDX. The basic components of the 3D scene are presented below:

Camera

The first step consists of the addition of the camera. This allows the viewing of the 3D scene from a particular perspective. The camera can be added quite simply, within a few lines of code. For more details consult the following page [136].

```
cam = new PerspectiveCamera(67, Gdx.graphics.getWidth(),
                                Gdx.graphics.getHeight());
cam.position.set(1f, 1f, 1f);
cam.lookAt(0, 0, 0);
cam.near = 1f;
cam.far = 300f;
```

The camera also needs to be updated.

```
cam.update();
```

After doing this, an object should be drawn, respectively a model should be imported, which can then be viewed from the camera's perspective.

Model Loading

The model can be loaded very simply by means of an integrated ModelLoader. However, the model file together with all significant files, such as texture images, have to be included in the 'assets' folder of the android project.

```
ModelLoader loader = new ObjLoader();
model = loader.loadModel(Gdx.files.internal("data/ship.obj"));
```

If one would like to add more than one model to the scene, an AssetManager is recommended. This object takes over the management of the loaded models. More information on loading a model can be collected from the subsequent tutorial [138].
The momentarily supported 3D model formats are Wavefront OBJ (.obj), G3DT (.g3dt), G3D (.g3d), MD2 (.md2) and MD5 (.md5), as stated by the following source [20]. As far as Wavefront OBJ models are concerned, the textures must be assigned by hand.

Moving Object

This can easily be done by manipulating the Instance object, which contains the model, before the model is drawn.

```
ModelInstance instance = new ModelInstance(model);
```

LibGDX offers some built-in transformation functions, such as rotation, translation and scaling. For instance on can use the transform.rotate() method of the instance to be rotated in order to rotate the object around one specific axis. The parameters of the method should comprise the axis, around which the rotation should occur, as well as the multiplication of the rotation speed with the frame rate.

```
instance.transform.rotate(Vector3.Y,
                          5 * Gdx.graphics.getDeltaTime());
```

This multiplication with the delta time, also known as the time between two frames, represents a quite important aspect. That way the model will rotate with the same speed on each device.

Lighting

In order to add Lambert lighting to the scene, the definition of an Environment object is necessary [136]. For this the specification of the color of the Ambient Light is required, as well as the specification of the color and direction of the Directional Light. This can also be realized quite easy and quick.

```
Environment environment = new Environment();
environment.set(new ColorAttribute(ColorAttribute.AmbientLight,
                                   0.4f, 0.4f, 0.4f, 1f));
environment.add(new DirectionalLight().set(0.8f, 0.8f, 0.8f,
                                           -1f, -0.8f, -0.2f));
```

The environment object must then be used as parameter for the render() method of the modelBatch object. The modelBatch object handles the rendering of the model using the correct environment lighting.

```
@Override
  public void create () {
    ModelBatch modelBatch = new ModelBatch();
    ...
        }
@Override
  public void render () {
    ...
    modelBatch.begin(cam);
    modelBatch.render(instance, environment);
    modelBatch.end();
        }
```

Camera Controller

The camera controller allows the interaction with the screen by mouse clicking on the desktop and by Touch on mobile devices.
This interaction triggers the movement of the camera, so that the user can basically control the camera freely and view the object from different perspectives. This is a quite nice feature which can easily be acquired within a few lines of code by using a CameraInputController object [136].

```
CameraInputController camController =
                        new CameraInputController(cam);
Gdx.input.setInputProcessor(camController);
```

The camera controller must also be updated each frame. The update should take place within the render() method.

```
camController.update();
```

Background Image

The loading of an image is quite easy, for a single SpriteBatch object is required. However the setting of the image as a background is a bit tricky, since one also needs a Sprite object in order to do so. The tricky part is represented by the setting of the dimensions of the image, so that it covers the whole background, filling the screen, namely the dimensions of the Sprite must be adjusted, not the dimensions of the SpriteBatch.
Another problematic matter is represented by the rendering of the scene, which takes place in the render() method. This is not as trivial as rendering merely 3D objects or rendering just 2D object, since our scene not contains a 3D model but also a 2D image.

```
SpriteBatch spriteBatch = new SpriteBatch();
img = new Texture("galaxy.jpg");

Sprite sprite = new Sprite(img);
sprite.setSize(Gdx.graphics.getWidth(),
               Gdx.graphics.getHeight());
```

The important aspect here is that the background image has to be rendered first and the 3D model has to be rendered second. The following source specifies that aspect in more detail [141]. That way the image will stay in the background an not be affected by the camera movements, while the 3D object will be rendered over the background and can still be visible.
For the rendering of the image the SpriteBatch object must begin, then the Sprite must be drawn, followed by the ending of the SpriteBatch. Similarly, in order to draw the model, the ModelBatch object must begin, then it must be drawn and then ended.

```
spriteBatch.begin();
  //draw 2D objects
  sprite.draw(spriteBatch);
spriteBatch.end();

modelBatch.begin(cam);
  //draw 3D objects
  ...
modelBatch.end();
```

The supported image formats are .jpg, .png and .bmp.

Sound

The playing of a music file is downright easy within libGDX projects. One simply has to load the piece of music into a Sound object and play it.

```
Sound mp3Sound = Gdx.audio.newSound(Gdx.files.internal("data/
                                                scrapelow.mp3"));
mp3Sound.play(0.5f);
```

For more details consult this source [22]. The momentarily supported formats are WAV, MP3 and OGG.

Potential Problems

The following lists describe various problems which might occur during the installation of the required software, as well as during the implementation.
In case a solution to these problems exists, it is given after the description of the problem.

Installation Problems

Following problems might occur during the installation phase:

- Android SDK might not be up to date. In this case the libGDX application will require an update.
- XCode is not updated to the latest version. This might cause some problems, because the latest version of XCode is explicitly required. However a complication is represented by the fact that one can not purchase the latest XCode version on older Mac operating systems, such as Mac Lion. In this case one would be required to acquire the latest Mac OS first, for example Mavericks at the moment, in order to be able to download the latest XCode version. This plays an important role in the provisioning of the Apple device one would like to run their game on.
- The Android SDK Build Tools must first be installed (from the SDK Manager), or else the Android SDK can not function properly.
- If one can not install the Android Development Tools because of conflicting dependencies, one has to install a new version of Eclipse, install all the plug-ins, such as Android SDK and Gradle, and import the libGDX project again.

Implementation Problems

Following problems might occur during the implementation phase:

- When testing the application on the Android device, the device can be rotated in any direction, however the mode will not switch from the landscape mode to the portrait mode. It seems like the landscape mode is already preassigned.

- The libGDX project does not recognize newly inserted assets into the android/assets folder. The Eclipse project must first be cleaned, and merely afterward the sub-projects will recognize the new files.

- To be able to test the application on Android devices, 'USB Debugging' must be activated. This enables the computer to recognize the device and facilitates the testing of the application on the respective device.

- LibGDX does indeed load models quite easily, however the supported formats are very few and quite unusual, respectively problematic. Since the only accepted formats are Wavefront OBJ, G3DT, G3D, MD2 and MD5, one can export nothing from Blender, except an OBJ file. However there is also the problem, that only the model, without the texture, can be loaded into the libGDX project, for the loading of the .mtl file for materials and textures is not supported until now. It probably will be supported in the future, however until then the textures must be assigned manually.

- Many people complain about the fact that the sound, even if successfully loaded, does not play on Android devices, even though it works fine on the desktop application and on iOS devices. A working solution for this problem does not seem to exist so far. It appears to be a major bug, which has not yet been fixed. Even an official discussion from the Badlogic games forum tackles the Android sound being unreliable [140].

4.2 Murl Engine

General Information

The Murl Engine is a platform-independent developing tool for games, mobile and multimedia applications, available for free.
The developers of the Murl Engine decided that the existing cross-platform developing tools imply certain constraints. This is especially the case when using Black-Box Game-Engines, where most of the functionality is pre-implemented, so that the developer does not have knowledge about the implementation, nor is he capable of altering it. These constraints limit the developer in his/her implementation, according to the web page on Introduction to the Murl Engine [92]. This led them to developing their own framework, which requires a slightly different approach, but allows the users a much wider flexibility during the development without limiting their creativity.

Furthermore, the Murl Engine allows for more technical control over the product, which means that the developer maintains an overview of all aspects of the scene, including the elements' location and the logic of the application, without certain sequences being carried out automatically.
The Murl Engine can be used not merely for the creation of applications and games for Smartphones, but also for Tablets, Notebooks, Personal Computers and Embedded Systems.

Developer Team

The Murl Engine was developed by Spraylight GmbH, a young start-up company founded in 2011, in Graz, Austria. More information on the matter can be found on the following web site [85], as well as on the official web site of the company [104].
Their goal was to create a more flexible, powerful Multimedia Framework with better performance than the ones which already existed on the market.

Figure 4.3: The figure depicts the logogramm of Murl Engine. The picture can be found on the following source [103].

Application Areas

There are numerous fields of application in which the Murl Engine has proven to be quite exercisable. Mobile applications, games, visualization solutions in the field of medicine, teaching and presentations on embedded systems represent merely some of them.

Technology

A very important aspect one needs to know about the Murl Engine is the fact that it represents a time-controlled scene graph framework. More details on the matter will be provided in the 'Implementation' paragraph of the section.
The framework is however divided into a platform-independent centerpiece (the 'Framework Code') and a platform-specific connection (the 'Platform Code').

The framework functions combine three different types of code, as the following list reveals:

- Framework Code - is entirely written in C/C++, platform independent and compiles therefore on every platform.
- User Code - represents the code produced by the developer. The programming languages which can be used for the development are C/C++ and LUA.

- Platform Code - is platform-specific, developed in C/C++, Objective C respectively in Java and can be compiled on the particular platforms.

All three types of code are used to create native applications at the click of a button.
While developing on one platform using a certain programming language as User Code, the code will automatically be translated into platform code during the automatic build, so that the developer need not worry about anything.
The project will be able to run on all supported platforms without further complications.

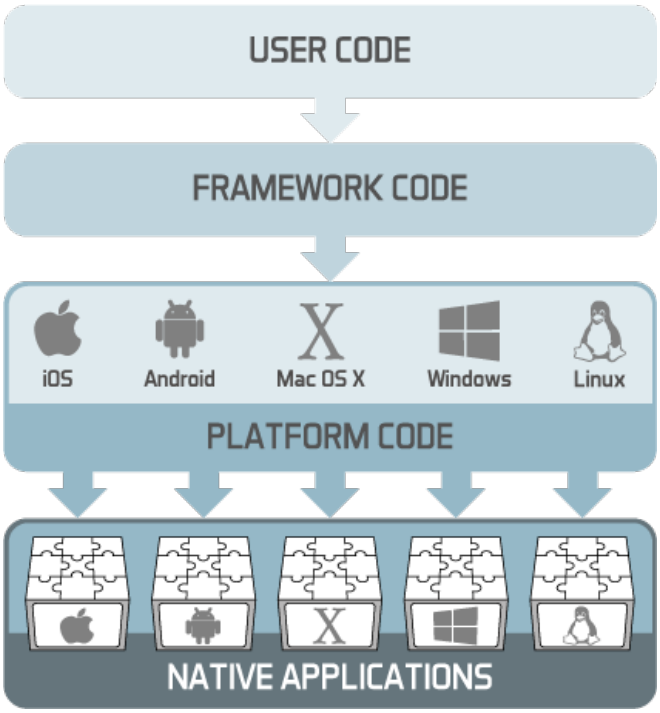


Figure 4.4: The figure illustrates how the various types of code of the Murl Engine, which have been described above, are linked together. The picture can be found on the following source [93].

Platforms

The Murl Engine currently supports the following platforms:

- Android - starting with version 2.0
- iOS - from version 4.3 onwards
- Windows - Win32 API / Win64 API starting with OpenGL version 2.1
- Mac OSX - from version 10.5 x86/x64 onwards

Linux support should be provided in the near future, for the developing team is already working on the matter.

Prerequisite

In order to run the application on an iOS device, one has to fulfill a series of steps, which are all explicitly formulated in the chapter 'Special Prerequisite for iOS' of the 'libGDX' section.
Since this paper rather focuses on the platforms Android and iOS, the most convenient way to develop an application using the Murl Engine is by using a MacBook and XCode to write the code.

Download

Furthermore one needs to download the latest version of the Murl Engine package from the download tab of the official Murl Engine web site [94], after having agreed to the terms of the licence. The package also includes the available tutorials and the Dashboard, which represents a graphic user interface, mainly used to run the project on Android devices, both of which can be of great help.

Installation

First a new directory should be created in which the Murl Engine is to be installed. The name of this new directory should preferably not include spaces, special characters or umlauts. The Murl Engine-archive should then be downloaded and unpacked into this respective directory. After the unpacking the directory should contain the 'Dashboard.app', as well as the directories 'murl' and 'tutorials'.
The following source provides further information on the topic of a Murl Engine Quick Start for Mac [101].

In order for the project to work on Android devices a series of packages must be installed:

- Android SDK - the Android Software Development Kit can be downloaded from the following source [15]. First one needs to click on 'Get The SDK For An Existing IDE', then download the SDK Tools only by clicking on 'Download the stand-alone Android SDK Tools for Mac', followed by accepting the license agreement and finally downloading the 'zip archive'.

- Android NDK - the Android Native Development Kit can be obtained from the following link [16]. In order to do so one merely needs to select either 'Mac OS X 32-bit' or 'Mac OS X 64-bit', depending on the laptop or computer in cause.

- Apache Ant Software Tool - this tool allows for the automation of the software build process and can be downloaded from the subsequent source [108]. To do so one only has to go to 'Current Release Art' and select the 'zip archive'.

As far as the organization of the downloaded packages is concerned, it might be useful to create a directory named 'Android', which should contain three further directories named 'ndk', 'sdk' and 'ant'. After doing so, the downloaded packages should be extracted into the respective

directories.

This is quite useful for the following step, which consists in letting the 'Dashboard.app' know where these exact Android files are located. The user can do this by opening the Dashboard and clicking on the 'Settings' icon in the upper right corner.
This will trigger the opening of a new window, in which the user needs to specify the exact paths to the 'Murl Directory', the Android 'sdk', 'ndk', 'ant' and the 'Java JDK Home'. The finding of these directories should represent no problem, after having organized the downloads as specified.
Once all the paths have been specified, the settings must be saved and the installation and setting up of the Murl Engine are herewith complete.

Project Creation

Tutorials

In order to familiarize oneself with the Murl Engine, one should consider looking into the existing tutorials, for instance the tutorial number zero 'Hello, World!' [109], available on the official web page, in the 'Develop/Tutorials' section [100].
One can open an existing tutorial by starting XCode, selecting 'Open', from the 'File'-menu in the upper toolbar, and opening the project file by selecting 'tutorials', select the first chapter, then the 'hello world' item, followed by 'project', 'osx', 'xcode' and the contained 'hello world' XCode project.

New Project

After having familiarized oneself with the Murl Engine, a new project can be created by simply opening the 'Dashboard.app' and clicking on 'Create Project'. The new window will require the user to select the 'template', which can be either the default C/C++ template or the LUA template, and specify the name of the new project. This will generate a new Murl Engine project into the existing 'Murl Engine' folder.

Open Project

The next step is to open the newly created project. This is done similarly to the opening of the tutorials. Xcode must be started, 'File', 'Open' should be clicked, the project should be selected, followed by the contained 'project' folder and the target platform should be chosen. Then 'xcode' must be pressed, the contained Xcode project must be selected and 'Open' should be pressed. The implementation can thus begin.

Running the Project

Running the Desktop Project

The desktop project can be run after having opened the Xcode project contained by the 'osx' folder of the application project. Then the user has to press on the 'Product'-menu from the

upper toolbar and select 'Run'. This will cause the Scene Viewer App to open and visualize the developer's project.

Running the iOS Project

First of all one needs to ensure that the iOS device is connected to the laptop or computer, so that XCode can acknowledge it's presence.
After this the iOS project can be run quite easy, by merely clicking on the 'Product'-menu of the upper toolbar, and pressing 'Run'.

Running the Android Project

In order to run the project on an Android device, one can make use of the Dashboard.app. The respective Android device needs to be connected to the laptop or computer.
After this is taken care of, the Dashboard needs to be started and the project in cause must be opened, by clicking on the 'Open Project' icon, selecting the project and clicking on the 'Open Project' button. This might take a few seconds to open the project.
After the project is opened in the Dashboard, it can be easily run by clicking on the 'Build Debug' icon, which will trigger the running of the project on the connected Android device.

Implementation

As previously mentioned the Murl Engine is a native, platform independent, time-based scene graph multimedia framework, which allows for a flexible development of games and multimedia applications with good performance.

Time Based

When using the Murl Engine framework the fact that the applications are being executed in a time-based way means that a Render-Thread and a Logic-Thread are being called into action in a parallel way and in a loop, as long as the application is being run.

Logic-Thread The Logic-thread calls the user-code method OnProcessTick() in each loop iteration. This method plays the role of an update() method. Using the GetCurrentTickDuration() method the developer can inquire how much time has passed since the last call of the updating method.
Furthermore all the user input will be intercepted and processed in this method. The resulting information is used to update all the objects of the scene.

Render-Thread After the Logic-thread has intercepted all the necessary information, the Render-thread draws all the visible objects in the scene, a process during which the Logic-thread already begins to evaluate the information from the next loop iteration.

Configuration object The step size, which means the time between two executions of the loop, for the Render-Thread and the Logic-Thread can be set by means of a configuration object, which facilitates fixed as well as variable step sizes. Furthermore, the maximal and minimal time values can be set, as well as the number of steps, which have to be carried out each frame, as specified by the following source [102].

Scene Graph

Besides the logical part, taken care of in the C/C++ classes located in the 'source' folder of the project, there also exist a series of rather important XML-files, located under 'packages/-main.murles'. The most important XML-files are 'package.xml', where all the resources are being declared and linked together, and the 'graph_main.xml' file.
The scene graph represents an object-oriented data structure, corresponding to a directed acyclic graph. Since the scene graph stands for a tree-like structure consisting of nodes, there can exist only one root node with an arbitrary number of child nodes.
The 'graph_main.xml' file contains the scene graph, which is responsible for the description of the objects and their positioning in the scene. Each node contains information regarding the virtual scene, as for instance geometry, transformations, colors, light sources or audio sources. Each entry in the 'graph_main.xml' file must be written manually, which also provides a lot of flexibility.
The 'graph_main.xml' file contains the scene graph, which is responsible for the description of the objects and their positioning in the scene. Each node contains information regarding the virtual scene, as for instance geometry, transformations, colors, light sources or audio sources. Each entry in the 'graph_main.xml' file must be written manually, which also provides a lot of flexibility.

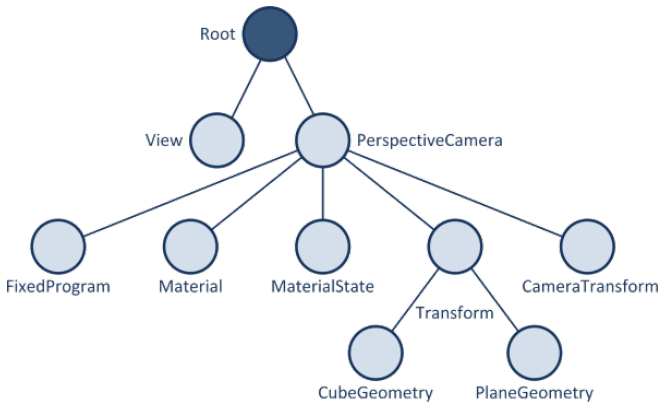


Figure 4.5: The figure represents an example of a typical scene graph, showing the hierarchical structure of the scene's objects. The picture can be found on the following source [102].

OpenGL and OpenGL ES

The rendering application programming interface supports not only OpenGL and OpenGL ES 1.1/2.0, but also by DirectX 9.0/11.1.

Implemented Application

The described application from the section 'Practical Application for Comparison Purposes' has also been implemented using the Murl Engine. The Murl Engine framework has been downloaded and installed and the project has been created as in the description above.

Camera

One can add a camera to the scene by simply inserting an orthographic or perspective camera node in the 'graph_main.xml' file. However besides the 'PerspectiveCamera' node, which contains attributes like the id, the field of view or the near and far plane, two more nodes are necessary, namely 'CameraTransform' and 'CameraState'. The first one specifies the location of the camera, while the second one specifies the id of the camera used to view the following objects.

```
<PerspectiveCamera id="camera"
  viewId="view"
  fieldOfViewX="400"
  nearPlane="400" farPlane="2500"
  clearColorBuffer="1"
>
  <CameraTransform
    cameraId="camera"
    posX="0" posY="0" posZ="400"
  />
  <CameraState
    cameraId="camera"
  />
  ...
</PerspectiveCamera>
```

Another important aspect is represented by the fact that all scene objects which are to be viewed with this camera should become child nodes of the Camera node.
More details concerning the implementation of a 3D perspective camera can be found in the tutorial 'Cube' from the official Murl Engine web site [96].

Model Loading

Loading a model can be achieved by means of the Scene Converter Tool. This can be found in the 'murl/base/binaries/osx/Release' folder of the project and can be used to convert 3D models or scenes into convenient Murl Meshes. The tool can be operated using the command line.

Moving Object

The rotation of the object has been implemented by the use of a transform node. This node has to be declared within the header of the Logic class.

```
Murl::Logic::TransformNode mTransform;
```

Afterward, in the OnInit()-method of the Logic class, the initialization of the transform object takes place liking the transform instance to the id of the scene component to be moved.

```
AddGraphNode(mTransform.GetReference(root, "/myGraph/myCube"));
```

Then the transform object can be manipulated in the OnProcessTick()-method by means of the SetRotation()-method. The angle of the rotation is extracted from the current tick time with the help of the GetCurrentTickTime()-method.

```
Double angle = state->GetCurrentTickTime();
angle = Math::Fmod(angle, Math::TWO_PI);
mTransform->SetRotation(angle, angle, 0);
```

A quite accurate example of moving an object can be found in the same tutorial 'Cube' from the official Murl Engine web page [96].

Lighting

One can add lighting to the scene merely by inserting a 'Light' node in the 'graph_main.xml' file. The specification of an id and the diffuse color of the light is required. Also two further nodes are necessary, namely 'LightTransform', specifying the position of the light source, and 'LightState', specifying which light source should illuminate the contained objects.

```
<Light
  id="light"
  diffuseColor="0.5f, 0.5f, 0.5f, 1f"
/>
<LightTransform
  lightId="light"
  posX="0" posY="0" posZ="800"
/>
<LightState
  lightId="light"
  unit="0"
/>
```

More details regarding lighting can be collected from the official Murl Engine tutorial 'Color Cube' [97].

Camera Controller

To manipulate the camera freely around the scene, the objects of the scene need to be nested within a 'Transform' node. Afterward the whole scene can be rotated in the Logic class according to the user's input. In order to process the user's input a Device Handler is necessary. More information on this can be found in the 'Color Cube' tutorial from the official Murl Engine web page [97].

Background Image

The Murl Engine supports the following image file formats: '.png', '.jpg', '.webp', '.dds' and '.tga'.
The texture represents an image, specified as resource in the 'package.xml' file.

```
<Resource id="galaxy" fileName="images/galaxy.png"/>
```

The background image can then be implemented by defining a new 'FixedProgram' node (similar to a shader), as well as a new 'Material' and a 'FlatTexture' node.

```
<FixedProgram
  id="prg_color_texture"
  textureUnit0Enabled="yes"
/>

<Material
  id="mat_alpha_color_texture"
  programId="prg_color_texture"
  visibleFaces="FRONT_AND_BACK"
  blendMode="ALPHA"
  depthBufferMode="NONE"
/>

<FlatTexture
  id="texture_galaxy"
  imageResourceId="package_main:galaxy"
  pixelFormat="R8_G8_B8_A8"
  useMipMaps="no"
/>
```

After doing so the nodes 'MaterialState' and 'TextureState' should be used to specify which material and texture should be used to render to subsequent geometry. The state will remain the same, until a new state is defined.

```
<MaterialState
  materialId="mat_alpha_color_texture"
  slot="0"
/>
```

```
<TextureState
   textureId="texture_galaxy"
   slot="0"
/>
```

After having specified the texture as a resource and all the other mentioned nodes have been defined, the texture can be assigned to a 'PlaneGeometry'.

```
<PlaneGeometry
   id="plane"
   scaleFactorX="1024" scaleFactorY="1024"
   posX="0" posY="0" posZ="-200"
/>
```

More information concerning images ca be found in the following tutorial from the official Murl Engine web page [99].

Sound

Since the Murl Engine supports merely '.wav' and '.ogg' files, the developer must ensure that the piece of music they would like to use has an accepted format. After doing so, the track must be defined as resource within the 'package.xml' file.

```
<Resource id="scrape"    fileName="sounds/scrape.wav"/>
```

As soon as this is taken care of, an 'AudioSource' node can be inserted into the 'graph_main.xml' file, referencing the audio resource. Furthermore a 'Listener' node must be inserted in order to position the listener into the scene. Additionally a 'Timeline' node is required. The latter specifies when the track will start as well as stop to play and contains a 'AudioSequence' node, which references the audio source, positions it and adjusts the volume.

```
<Listener
   id="listener"
   viewId="view"
/>

<ListenerTransform
   listenerId="listener"
   posX="0" posY="0" posZ="800"
/>

<ListenerState
   listenerId="listener"
/>
```

```
<AudioSource
  id="soundScrape"
  audioResourceId="package_main:scrape"
/>

<Timeline
  id="timeline"
  startTime="0.0" endTime="286.0"
  autoRewind="yes"
>
  <AudioSequence
    id="sequenceScrape"
    audioSourceIds="soundScrape"
    volume="1.0" rolloffFactor="0.0"
    posX="0" posY="0" posZ="800"
  />
</Timeline>
```

After having defined these nodes, one can declare a 'TimelineNode' object in the logic.hpp class, which will then be responsible for playing the music. The 'TimelineNode' must then be initialized in the OnInit() method of the logic.cpp class.

```
AddGraphNode(mMusic.GetReference(root, "/myGraph/timeline"));
```

To play the music, if it is not playing already, one can insert for instance the following lines of code into the OnProcessTick() method of the logic.cpp class.

```
if(!mMusic->IsRunning())
{
  mMusic->Rewind();
  mMusic->Start();
}
```

More information on the Audio component of the Murl Engine can be collected from the official 'Audio' tutorial [98].

Potential Problems

The following list describes some potential problems, which might occur during the installation of the required software, as well as during the implementation. In case a solution to these problems exists, it is given after the description of the problem.

Installation Problems

No problems occurred during the installation phase. Also there was no need for the environment variables to be changed.

Implementation Problems

Following problems might occur during the implementation phase:

- The approach as well as the whole concept of the Murl Engine is rather different from the other ones described in this bachelor thesis, which was actually the intention of its developers, who wanted to revolutionize the cross-platform developing frameworks offering the developers better performance and more flexibility.
  However flexibility can be viewed not only as an advantage but also as a disadvantage, especially for beginners, who are not so familiar with the concept. Thus the problem consisted in the familiarizing with the concept and the framework, given it's originality.

- A second problem consisted in the fact that one can not change the path to a new created project. The default project is located within the Murl Engine folder and can not be changed, for instance for organizational purposes. If the project is not in the default location, the Dashboard.app can not locate it anymore.

- The framework is a rather new one, thus there isn't much help on the internet. There exists no actual community so far, and also the help found in the forum of the official web page is rather humble. Also the official web page is not finished, there are lots of topics which have no thorough documentation and many aspects and topics are announced to be covered soon, but are not available at the moment.
  For example especially for 3D Rendering, there is no aid at all at the moment, thus the model loading and camera controlling aspects have represented quite an issue.
  However the forum administrators as well as the developers of the framework, after having contacted them directly via email, have proven to be quite friendly and helpful.

- Another problem consisted in the fact that the background image and the model placed in front of it could not be rendered at the same time. The reason for this consisted in the fact that the 'depthBufferMode' variable in the respective nodes, was once set to 'READ_AND_WRITE' and once to 'NONE'. Whereas the engine first draws all geometries for which the DepthBuffer is activated, followed by all the other geometries. This caused the model to always be overwritten by the background plane.
  In order to solve this problem one simply needs to set both 'depthBufferMode' variables to the same value, either 'READ_AND_WRITE' or 'NONE'.

4.3 Unity

General Information

Unity represents a cross-platform game development system, which includes not only an integrated development environment (IDE) but also a game engine, for the purpose of simplifying the programming effort. The system is mainly specialized for game development, however it can be utilized also to create other 3D-graphics-applications.
Unity can also be described as a game development ecosystem. This is facilitated by the integrated rendering engine, equipped with a complete and intuitive set of tools and fast workflows.

This enables the users to generate interactive 2D and 3D content quite fast and easily.
The engine furthermore provides an easy way to publish to numerous platforms, an Asset Store as well as a community, whose members are willing to share their knowledge.

Figure 4.6: The figure depicts the logogram of Unity. The picture can be found on the following source [86].

Platforms

The target platforms include not only desktop platforms, but also mobile devices, consoles and web plug-ins. Windows and Mac serve not only as development platforms but also as target platforms.
Since 2005 Unity expanded (more information on this can be found on the following web pages [123], [17]) and now targets over fifteen platforms, which include Windows, OS X, Linux' Ubuntu, Android, iOS, Blackberry 10, Windows Phone 8, Unity Web Player, Abobe Flash, PlayStation 3, PlayStation 4, Xbox 360, Xbox One, Wii U and Wii.
Furthermore Unity constitutes the standard software development kit (SDK) for Nintendo's Wii U video game console platform.

Developer Team

Unity was created by Unity Technologies, located in San Francisco, USA.

Programming Languages

Unity is written in C++ and C#, while the game engine's scripting is based on the open-source implementation of the .NET Framework, Mono. In order to program for Unity, one can use either C#, UnityScript or Boo.
UnityScript is a programming language with syntax inspired by ECMAScript. Even though the software refers to it as JavaScript, the two languages are essentially quite different, as demonstrated by the following source [114].
Boo on the other hand has a syntax inspired by Python.

Integrated Development Environments

One can use the Unity Editor, an included integrated development environment, to implement cross-platform applications.

Fees

While the basic version of Unity can be downloaded free of charge, the Pro-Version provides further functionality in exchange for a fee. If one intends to use Unity Pro, one can either acquire the license ownership or get a monthly subscription. The prices are indicated by the following source [121].
Some examples of supplementary features provided by Unity Pro are:

- Render-to-Texture effects - can be used to implement reflective water surfaces real-time shadows
- Version Control
- Optimization features - such as Level of Detail and Occlusion Culling
- AssetBundles - are files containing assets of the developer's choice that can be exported from Unity. AssetBundles use a compressed format and ensure a better factorization and performance by allowing even whole scenes to be streamed into the application. A more detailed description of AssetBundles can be found in the following source [118].
- Post Processing effects - such as glow, motion blur and edge detection

More information on the advantages provided by the Pro license as well as a comparison between the Pro and the free license can be found in the subsequent sources [117], [126].
So there exist some features which might motivate the users to purchase the Pro license in case of a serious project. These features can provide a more polished look to one's game, are however not absolutely necessary for creating a quality game.

Nevertheless, Unity is not only known for targeting an impressive variety of different platforms, but also for the fact that it lowers the barriers of cost and time invested in the developing of a game, allowing self-employed developers to create particularly beautiful games.

Popularity

A survey of the Game Developer Magazine from 2011 revealed that Unity (together with the Unreal Engine), was elected the most popular game engine, while the Game Development Magazine Gamasutra declared that Unity is the most often used game engine for mobile devices as stated by the following article [113].

Prerequisite

Since this paper focuses on the mobile platforms Android and iOS, the following paragraph will only cover the prerequisite necessary to create a game application for these two platforms.

Prerequisite for Android For the application to be able to run on an Android device, one needs to install the Android SDK. More details concerning this topic can be found in the 'Prerequisite' chapter of the 'libGDX' section.

Prerequisite for iOS In order to run the application on an iOS device, one has to fulfill a series of steps, which are all explicitly formulated in the chapter 'Special Prerequisite for iOS' of the 'libGDX' section.

Downloading Unity

Even though the development of applications which are also suited for iOS devices requires the utilization of a MacBook and the newest Xcode version, the development with Unity does not take place in Xcode, but in the IDE of Unity, which can be downloaded from the official Unity3D web site [125].
Before downloading Unity one has to specify which Unity version is desired, depending on the computer or laptop he or she is developing on. The main reason for this is the availability of different versions of Unity for Mac and Windows. After one has specified the desired version, by toggling between 'Developing on Mac OS X?' and 'Developing on Windows?', one can download the free version of Unity.

Project Creation

The creation of a new project is easily done, by opening Unity and clicking 'File', in the upper toolbar, and then 'New Project'. One will then have to specify the location, the name of the project and whether the application will be a 2D or a 3D one. Finally one must click on 'Create Project', which will initiate a new project at the specified location and the development can thus begin.

Running the Project

Before running the project, one needs to specify the Build Settings. These can be found by clicking 'File', in the upper toolbar, and then selecting 'Build Setting'. A new window will be opened, which allows the user to select the desired platform, on which he or she would like to run the project.
The currently set platform is marked with the unity logogram. So if the desired platform isn't already set, the user must select a new platform and click 'Switch Platform'. This might take a while, because a new project will be created for the respective platform and all the assets need to be copied.

Running the Desktop Project In order to run the project on the desktop, one must click on 'File', 'Build Settings', select 'PC, Mac & Linux Standalone', if not already selected, and press 'Switch Platform'. If an older version of the project has already been created previously, the user can replace the old version with the new project.
After this the developer can press 'Build And Run' and select the desired resolution of the application.

Running the Android Project Running the project on an Android device is triggered by clicking 'File', 'Build Settings', selecting 'Android' and 'Switch Platform'. Then one must make sure that the Android device is connected to the computer or laptop and click 'Build And Run'. After a short period of time the application will run on the device.

Running the iOS Project The running of the application on an iOS device requires similar steps: the clicking of 'File', 'Build Settings', the selection of 'iOS' and the pressing of 'Switch Platfoms'. This course of action will trigger the creation of a new project and the possible replacement of an old one.
After this one must make sure that the iOS device is connected to the laptop or computer and press 'Build And Run'. This will trigger the launching of Xcode, which might take a few seconds. Xcode will then transmit the application to the iOS device and run it.

Implementation

Unity Editor

The implementation takes place in the developing environment Unity Editor. Unity Editor is inspired by established 3D-animation programs, including a main window, which shows the 2D or 3D scene, as well as some menus and forms, which comprise the Hierarchy, the Project and the Inspector, among others. These forms and menus enable a quick and simple manipulation of the objects of the scene.
Another similarity to 3D-animation and modeling programs is represented by the fact that one can select, translate, rotate or scale the components of the scene using the mouse.

The Scene

The scene is engineered as a scene graph, containing all the components of the scene, the so-called 'Game objects'. The attribution of various components to these game objects is possible by simply selecting the game object and then clicking on the 'Add Component' button in the Inspector. The components which can be added include materials, as well as sounds and coding scripts.
After deciding which programming language one would like to write one's script in, the script will be created and opened in MonoDevelop, where the developer can write his or her code down, specifying the behavior of the respective game object.

Game Objects

The game objects can be either simple, unitary objects or more complex components.

Simple Objects Light sources, cameras or other graphic primitives, such as cubes, spheres or planes, are usually considered to be unitary or simple objects. They can be directly generated in the Editor by selecting 'GameObject', in the upper toolbar, then clicking 'Create Other'.

Complex Objects More complex objects, also called 'Assets', on the other hand are created externally, for instance by means of software specialized for 3D-modeling. 3D-models, sounds, animations, or textures are usually referred to as assets and have to be imported into the Project. However, the importing assets can be carried out quite easily by Drag and Drop.
Furthermore, the Assets folder of the Unity project will update itself if modifications of the assets occur during the production of the project.

Game-View

The game-view allows the developer to see his application from the point of view of the main camera, so that he can experience the look of his game without having to actually build and run it.
The framework also provides an export function, which enables the creation of executable applications.

Graphic Support

Unity's graphics engine supports a series of application programming interfaces (APIs) in order to enable graphic support for numerous different target platforms. Some of these APIs are enumerated beneath:

- Direct3D - for Windows and Xbox 360.
- OpenGL 2.1, 3.2- for Windows, Mac and Linux. (Mac does not support OpenGL 4, as mentioned in the Unity Forums [127].)
- OpenGL ES 2.0 - for Android and iOS devices.
- Other proprietary APIs - necessary for video game consoles. ...

Asset Server

Unity furthermore supports Nvidia's physics engine PhysX and an Asset server. The Asset server constitutes a version and asset control system with a graphical user interface, meant to facilitate the members working together. However it is merely provided after the acquisition of the Unity Pro version. Further details on the matter can be found on Unity's official documentation page [119].

Implemented Application

The described application from the section 'Practical Application for Comparison Purposes' has also been implemented with Unity. No further software had to be installed in order to be able to run the application on Android and iOS devices, for all the necessary aspects have been covered by the preparations for the previously discussed frameworks. However the Unity framework has been downloaded and the project has been created as in the description above.

Camera

Since any game needs a camera to view any further assets which might follow, the Main Camera is already present in the scene once a new Unity project is generated. However the location of this camera in the scene can be altered by translation, rotation or scaling, all of which can be executed by means of the mouse.
However further cameras can also be added to the scene, by clicking 'GameObject' in the upper menu and then selecting 'Create Other', 'Camera'.

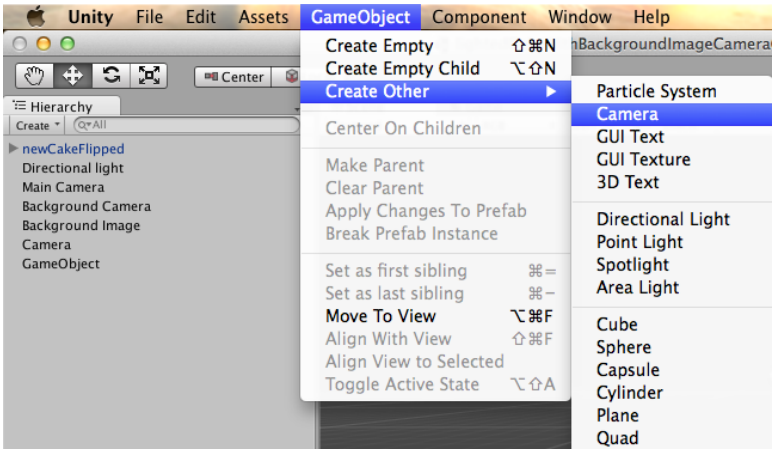


Figure 4.7: The figure shows the ease with which a new camera object can be added to the scene in Unity.

Model Loading

In order to import the assets into the project, one merely has to copy the desired assets into the 'Assets' folder, located in the created Unity project's folder. This will trigger an automatic import of the assets into the project, once Unity has been started and the project is loaded. The

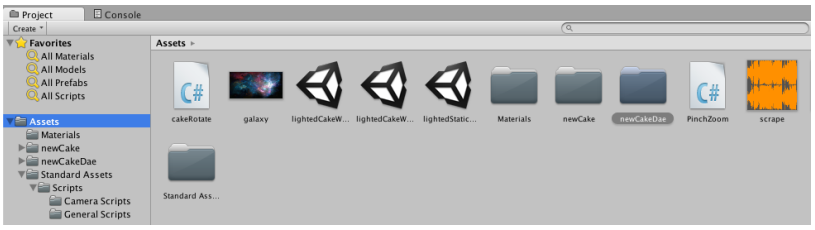


Figure 4.8: The figure shows how the asset folder is being visualized in Unity.

positioning of the models in the scene is done outstandingly easy be dragging and dropping the asset from the 'Assets' folder of the 'Project' hierarchy to the scene. The developer can then alter the location of the asset in the scene either by varying the parameters in the 'Inspector'

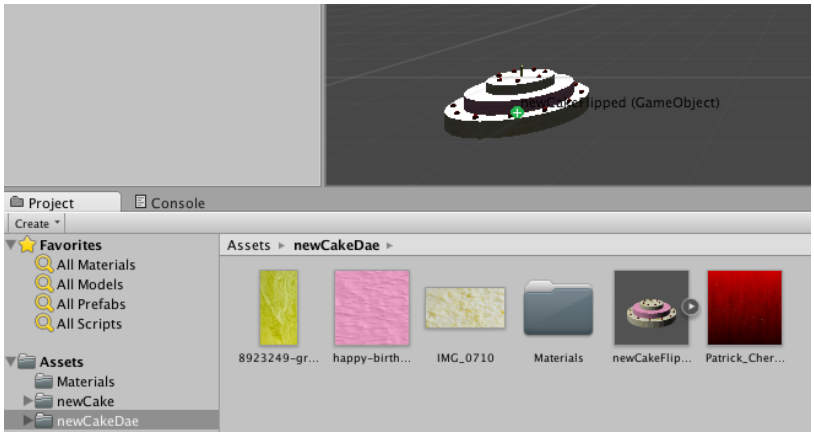


Figure 4.9: The figure shows how assets can be dragged and dropped from the asset folder into the scene in Unity.

or by manipulating the asset by means of the mouse. The user can therefore translate, rotate or scale the object, with the purpose of positioning it, directly in the scene view, without having to write any actual code.

According to the Unity Documentation web page on importing models [120], Unity can read not only .obj (Wavefront) files, but also .FBX, .dae (Collada), .3DS, and .dxf files. Therefore it works quite well with file formats exported from 3D Studio Max or Blender for example.
The model that has been imported into the demonstrative game application is a .dae (Collada) file, representing a cake, which has been modeled by myself in Blender.

Moving Object

In order to be able to put into effect the constant rotation of the cake model around it's own Y axis, a new script had to be created and added to the cake game object. Such a script must be added and written for the purpose of defining the behavior of an object in the game.
The attachment of a new script is done as follows: the developer must decide which object's behavior he wants to define, then the respective object must be selected from the 'Hierarchy'. Then, in the 'Inspector', the 'Add Component' button must be pressed and 'New Script' must be selected from the list which appears. The name of the new script must be entered and the language must be selected. For the demonstrative application I have chosen the language 'CSharp' and named the script 'cakeRotate.cs'.

After the script has been created, one can start coding in MonoDevelop. In order to rotate the cake constantly around it's own Y axis, a single invocation of the transform.Rotate() method was necessary in the Update() function of the script. As parameters of the Rotate() method, one must specify a vector around which the object should rotate and the rotation speed. However, an important fact considering the rotation speed is that it has to be multiplied with the delta time between two frames.

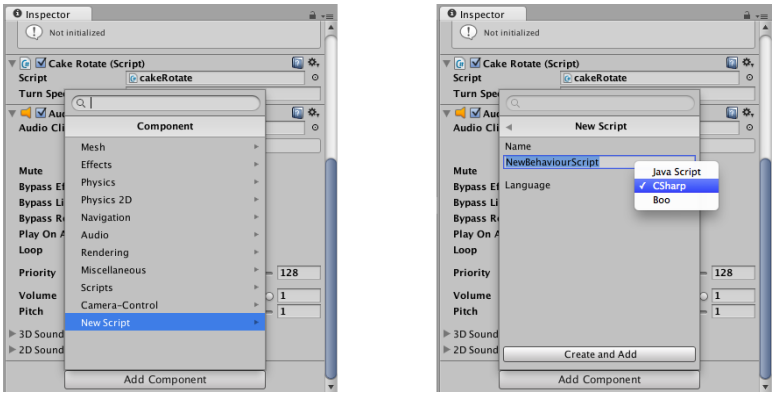


Figure 4.10: These two figures shows how a new script can be created for a selected scene object in Unity.

```
using UnityEngine;
using System.Collections;

public class cakeRotate : MonoBehaviour {
  public float turnSpeed = 50f;

  // Use this for initialization
  void Start () {
        }

        // Update is called once per frame
  void Update () {
    transform.Rotate(Vector3.up, turnSpeed * Time.deltaTime);
  }
}
```

Lighting

Light sources can be added to the scene as easy as cameras, since they both fall into the category of simple, unitary objects, which can be generated directly in Unity. Therefore in order to create the lighting, one simply has to click 'GameObject', followed by 'Create Other' and then chose between different types of light. The offered light types are 'Directional Light', 'Point Light', 'Spotlight' and 'Area Light'. The light I have chosen for my application is a Directional Light. The decision has been made based on the fact that directional light simulates the sun light most accurately.

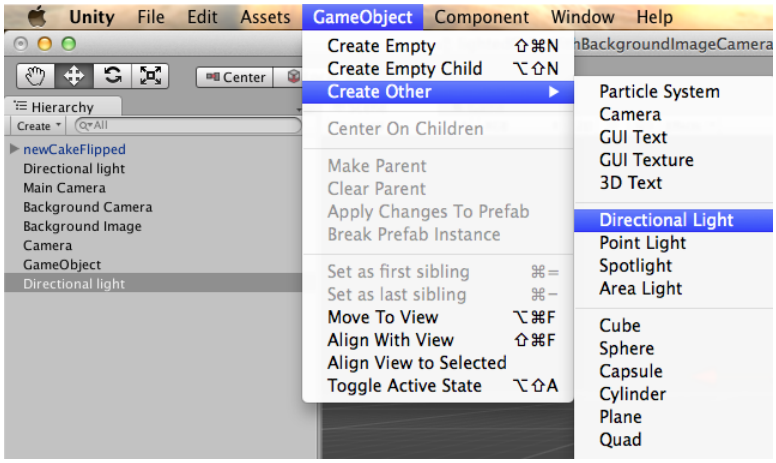


Figure 4.11: The figure shows how new light objects can be added to the scene in Unity.

Camera Controller

Since the camera controller in Unity is not already implemented and encapsulated into an existing object or method as in libGDX, it has to be programmed by the developer using scripts, which have to be added as components to the Main camera, as described below:

Orbit Movement The first script implements the orbit movement of the camera around an object of the scene. The camera's movement is guided by the mouse pointer on the desktop application and by a finger touching the display on mobile devices.
Unity provides some default scripts, which can be imported by clicking 'Assets' in the upper toolbar, 'Import Package', 'Scripts'. This will cause a window to open in the upper left corner of

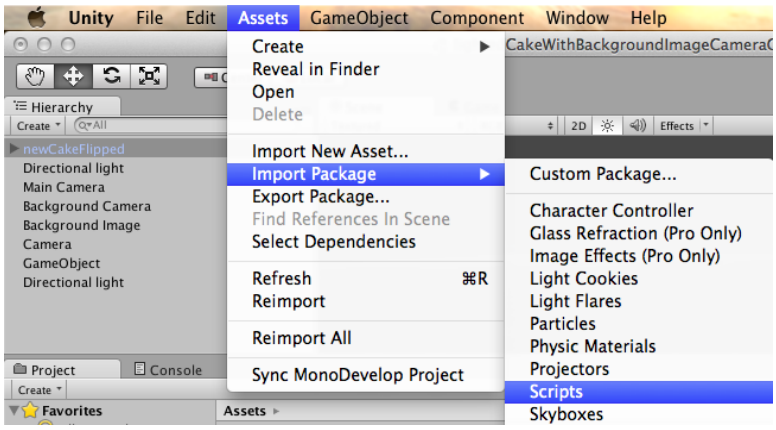


Figure 4.12: The figure shows how to import already existing scripts in Unity.

the framework, presenting a list of items with check boxes. After ensuring that all check boxes

are marked, the developer can press 'Import'. This will load the imported scripts into the 'As-

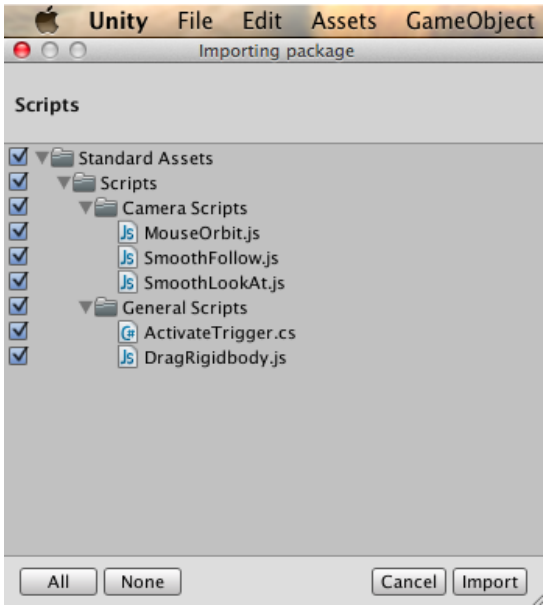


Figure 4.13: The figure shows the list of scripts which can be imported in Unity.

sets' folder, visible in the 'Project' tab. The script needed to orbit the camera around an object is

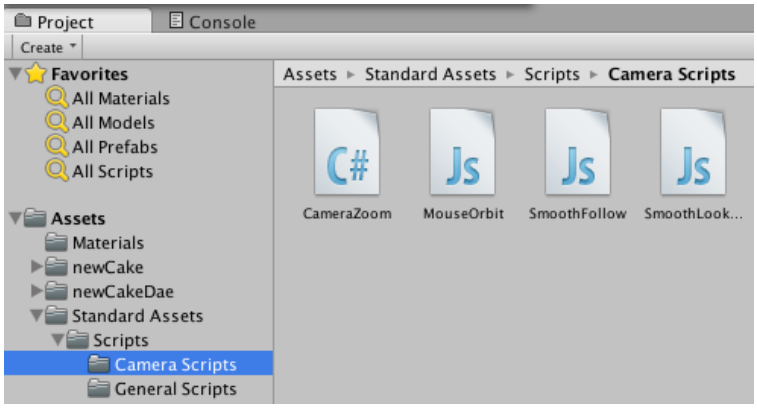


Figure 4.14: The figure shows the list of imported scripts within the Assets tab of Unity.

located under 'Standard Assets', 'Scripts', 'Camera Scripts' and is named 'Mouse Orbit'. In order to add this script to the main camera, the developer must select the camera in the 'Hierarchy' tab and then click and drag the 'MouseOrbit' script from the 'Project' tab into the 'Inspector' tab, where the details of the main camera are visualized. This will automatically add the script as a component to the camera object.

The next step consists in the setting of a target object around which the camera should orbit. In order to do that, the main camera needs to be selected in the 'Hierarchy' tab again, which causes it's details to be shown in the 'Inspector' tab, and then the desired target object needs to be clicked and dragged from the 'Hierarchy' tab to the 'Target' field of the 'Mouse Orbit (Script)' component of the main camera in the 'Inspector' tab. After setting the target, the target object should be highlighted in the 'Hierarchy' view.
When running the project, the camera should now be able to orbit around the target object, following the mouse movement. More information concerning this topic can be found in the tutorial named 'Camera Control in Unity3D' [72].

Scroll Wheel Zoom Even though the camera now orbits around the object, the user can not zoom in on it yet. In order to realize an effective zooming, new scripts have to be written. This paragraph describes the zooming within the desktop application using the scroll wheel of the mouse.
First of all a new script component must be added to the main camera, which can be done as already described in the upper section 'Moving Objects' of the 'Unity' chapter. After doing so and the script is opened in MonoDevelop, the programming can start. First some variables need to be declared and initialized, such as the distance of the zoom, the minimal field of view and the maximal field of view, as well as the sensitivity, which influences the so-called speed of the zooming. After that the scroll wheel input has to be fetched and multiplied with the sensitivity. Then the zooming distance has to be clamped between the minimal and maximal field of view. Finally the camera needs to be repositioned accordingly to the zooming distance.
Now the user should be able to zoom in on the target object within the desktop application by scrolling. Further information upon the matter can be found in the tutorial 'Unity3D Tutorial 58' on Camera Zoom [35].

```
using UnityEngine;
using System.Collections;

public class CameraZoom : MonoBehaviour {
  float distance  = 60;
  float sensitivityDistance = 50;
  float damping = 5;
  float minFOV = 40;
  float maxFOV = 180;

  void Start () {
  }

  void Update () {
    distance -= Input.GetAxis ("Mouse ScrollWheel") *
                                          sensitivityDistance;
    distance = Mathf.Clamp (distance, minFOV, maxFOV);
    camera.fieldOfView = Mathf.Lerp (camera.fieldOfView,
```

```
                                distance, Time.deltaTime * damping);
    }
}
```

Pinching Zoom After managing to implement the zoom within the desktop application, the user will still not be able to zoom on the mobile device. Therefore a new script, for instance 'PinchZoom.cs', has to be added to the camera, as already described in the upper section 'Moving Objects' of the 'Unity' chapter, and implemented in MonoDevelop.
In order to zoom on a mobile device, one needs to fetch two input touches from the display first.

```
Touch touchZero = Input.GetTouch(0);
Touch touchOne = Input.GetTouch(1);
```

Then one should also get the previous locations of the touches by subtracting the 'deltaPosition' of the touches from their current position. After managing that, one can easily calculate the distance between the current touches and the former touches in order to see if the distance has increased or decreased since the previous frame.

```
Vector2 touchZeroPrevPos = touchZero.position -
                                    touchZero.deltaPosition;
Vector2 touchOnePrevPos = touchOne.position -
                                    touchOne.deltaPosition;

float prevTouchDelteMag = (touchZeroPrevPos -
                                    touchOnePrevPos).magnitude;
float touchDeltaMag = (touchZero.position -
                                    touchOne.position).magnitude;

float deltaMagnitudeDif = prevTouchDelteMag - touchDeltaMag;
```

If the distance between the two touches has magnified, the camera should be positioned closer to the object, since the user would like to zoom in. Likewise, if the distance has decreased, the user would like to zoom out and the camera should be positioned further away.

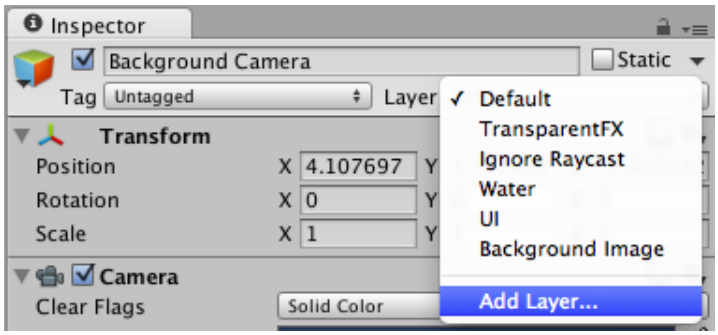
```
if(camera.isOrthoGraphic) {
  camera.orthographicSize += deltaMagnitudeDif *
                                        orthoZoomSpeed;
  //so model will not flip:
  camera.orthographicSize = Mathf.Max(camera.orthographicSize,
                                    0.1f);
}
else //camera is in perspectiveMode
{
  camera.fieldOfView += deltaMagnitudeDif *
                                        perspectiveZoomSpeed;
```

```
    //limit field of view, clamp it between 0 and 180
    camera.fieldOfView = Mathf.Clamp(camera.fieldOfView, 0.1f,
                                     179.9f);
}
```

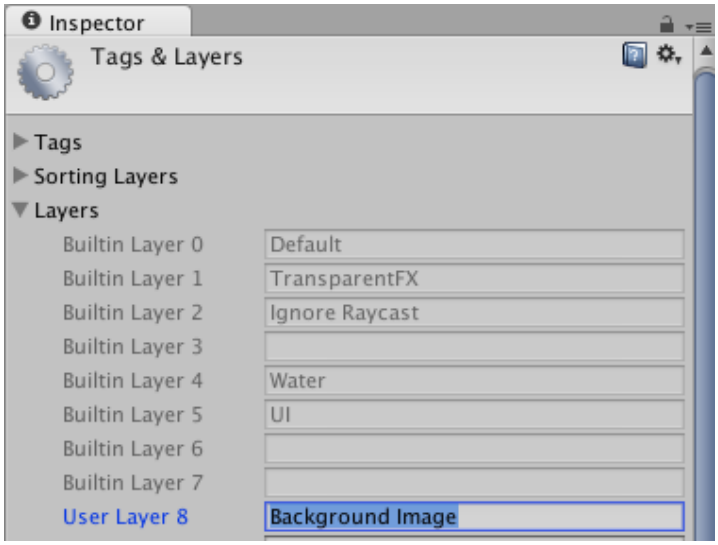
More information on zooming by pinching can be found on the official web site of Unity under 'Tutorials', 'Pinch to Zoom' [122].

Background Image

The setting of the background image is slightly complex. First of all the creation of a new camera is necessary, called 'background camera'. After this is done, a new GUI Texture needs to be created, by clicking 'GameObject', 'Create Other', 'GUI Texture', and named 'background image'. Then a new layer must be added in the 'Inspector' tab, by clicking on the 'Layer' drop down menu of the background image texture and selecting 'Add Layer'.

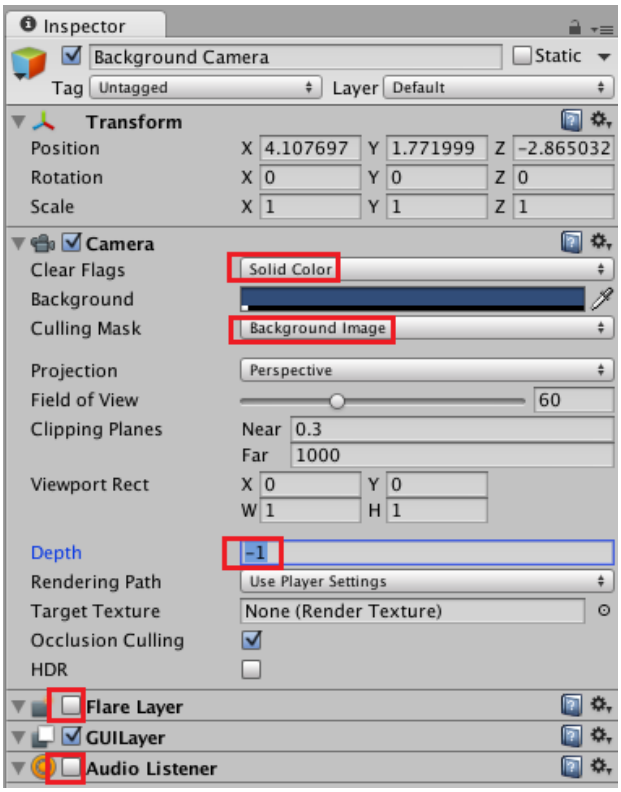


The new layer should be inserted in the next free slot and named 'background image'.

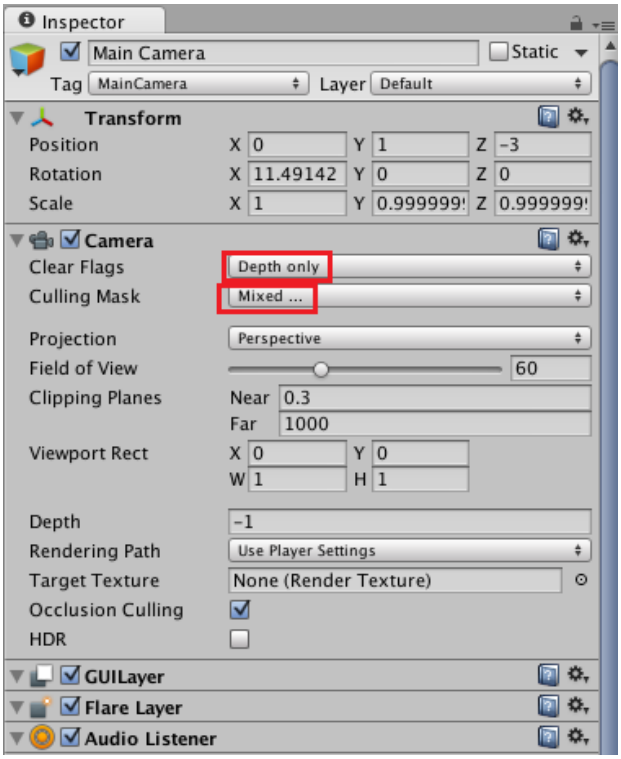


Afterward the background image should be selected from the 'Hierarchy' tab, the desired texture should be chosen and the size should be set to the current screen size, like mentioned on the following web site regarding this subject on Unity Answers [115]. When this is done, the 'layer' drop down menu should be used in order to assign the earlier created 'background image' layer to the texture object.

The next step is represented by the selection of the background camera in the 'Hierarchy' tab and some adjustments in the 'Inspector' tab: 'Flare Layer' and 'Audio Listener' should be unchecked, 'Clear Flags' should be set to 'Solid Color', 'Depth' should be set to -1 while the 'Culling Mask' should first be set to 'Nothing' and then to 'background image'.

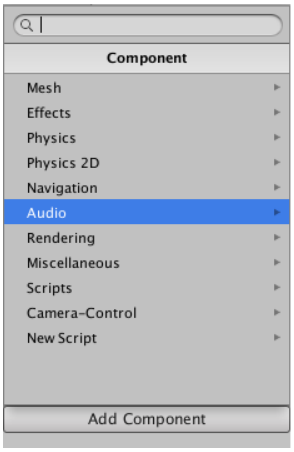


Furthermore the main camera must be selected in the 'Hierarchy' window and then, in the 'Inspector', the 'Clear Flags' must be set to 'Depth Only' and in it's 'Culling Mask' setting, the 'background image' should be unchecked, which results in the word 'Mixed...' to be visualized. Finally, after going though all these steps, the images of the two cameras should be merged in the game view, resulting in the model being displayed in front of the background image. More information on the matter can be found on the following source, presenting a discussion in Unity Answers [116].

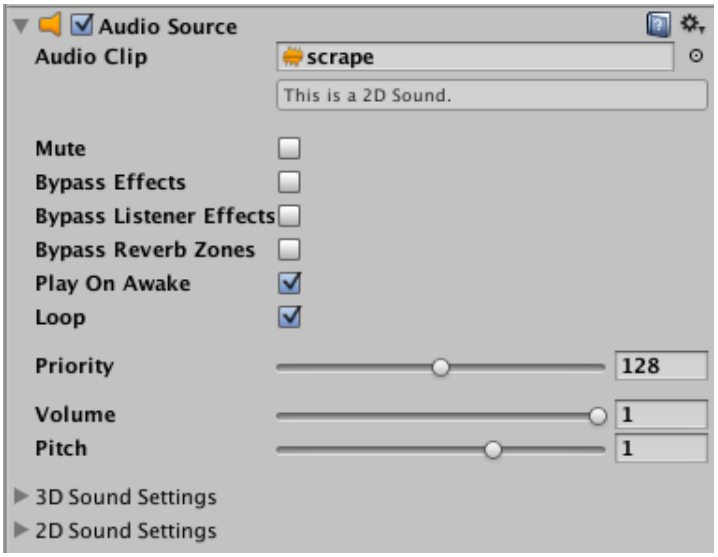


Sound

The playing of a sound is also quite fast and easy with Unity. First, one has to select a game object from which the sound should be coming or which should appear to be generating the sound. The respective object must then be selected in the 'Hierarchy' and 'Add Component' must be pressed in the 'Inspector'. From the list which appears, one must chose 'Audio' followed by 'Audio Source'.



The result of this should then be visualized in the 'Inspector'. In the field 'Audio Clip' one must then enter the desired sound, which should previously have been inserted into the 'Assets' folder.
For the demonstrative application I have also selected the two check boxes 'Play On Awake' and 'Loop'. However this is arbitrary and dependent on the application. After going through these



steps, the audio sounds should play just fine not only within the desktop application and on iOS devices but also on Android devices. Further information considering sound in Unity can be found on the tutorial '11. Unity Tutorial, Sound - Make a Game' [13].

Potential Problems

The following list describes some potential problems which might occur during the installation of the required software, as well as during the implementation. In case a solution to these problems exists, it is given after the description of the problem.

Installation Problems

No problems occurred during the installation phase, on the contrary, the installation took place quite smoothly.

Implementation Problems

The following problems might occur during the implementation phase:

- The Wavefront model of the space ship used in the libGDX demonstrative application could not be loaded by Unity with the correct texturing. The reason for this was that the space ship model was specifically designed to work merely for the libGDX demonstrative tutorials and does not work on other applications, as stated by the following source [139].

In order to solve the problem, I have modeled a new model myself, representing a cake, using Blender and exported it as a Collada file.

- However after importing the new Collada cake model into Unity, only the back faces appeared to be visualized, leading to the conclusion that the normals had somehow been flipped, even though the model looked normal in Blender and visualized with Assimp View. To solve this problem I flipped the normals of the model in Blender.

4.4 MoSync

General Information

MoSync Software Development Kit represents a free, open-source tool for creating cross-platform mobile applications. It incorporates not only tools for building various applications, but also a complete Integrated Development Environment, based on the Eclipse development environment. MoSync enables the development of applications for all considerable mobile platforms by means of one single code base. More information can be found on the official website of MoSync [55]. However, even though MoSync provides numerous useful tools and supports up to eight platforms at the same time, it is no longer maintained. However some support is provided by the still existing community [66].

Figure 4.15: The figure depicts the logogram of MoSync. The picture can be found on the following source [58].

Programming Languages

The MoSync framework uses C/C++ or HTML5 and JavaScript to create native mobile applications for multiple platforms. Furthermore combinations of the two programming languages can

be used to build hybrid applications.

Developer Team

The development of MoSync was realized by a Swedish software company named MoSync AB. MoSync was first released in 2005, providing support for the Java Micro Edition platform. The support has afterward been extended for numerous other platforms.

Target

MoSync's main target group consists of both web developers willing to enter mobile development and ordinary C/C++ desktop developers.

Bankruptcy

The software company MoSync AB has filed for bankruptcy on the 24. June 2013 [57].

Platforms

The supported platforms include Android, Blackberry, iOS, Java ME MIDP, Moblin, Symbian, Windows Mobile and Windows Phone. For further information considering the respective versions consult the following website [64].

Prerequisite

The main prerequisite is represented by the download and installation of the MoSync SDK. For this the framework and IDE must be downloaded from the official website and installed as described in the following source [65].
As far as the iOS support is concerned, the prerequisites from the chapter 'Special Prerequisite for iOS' of the 'libGDX' section should be fulfilled.

Project Creation

When creating a new project, the developer must first choose whether he would like to write the application in C/C++ or HTML5 and then select one of the eight existing templates.
The templates provide some initial help as far as the structure of the mobile application is concerned, and can also include some libraries, which might be of use in specific application types, such as OpenGL ES in 3D applications.

Running the Project

Running the iOS Project

The application must first be built within the MoSync IDE, after 'iOS' and the respective device have been set as active profile. Then the created Xcode project (located in the 'Output' folder of the project, followed by 'Release', 'iOS', 'iPhone', 'package', 'xcode-proj') must be opened in

Xcode and run from there, after having connected the device. Further information can be found on the following page [62].

Running the Android Project

To run the project on Android, one has to specify 'Android' as the active profile. Afterward the target device must be selected, after having connected it, by means of the 'Select Target Device' button in the left side of the upper menu. Finally the 'Send To Target Device' button must be pressed. More information concerning the topic can be collected from the following website [63].

Implementation

Flexibility

Even though MoSync is quite easy-to-use, having in mind that it is based on the well-known IDE Eclipse, and provides quite a lot of flexibility, it lacks ready-to-go functions. This means that it supports almost every feature imaginable, but the programmer has to implement everything by himself.

Moblet

MoSync targets mobile applications using a C++ event manager called Moblet. It plays the role of a base class of the project and controls the main loop of the application.
By implementing the methods keyPressEvent() and keyReleaseEvent(), the Moblet can also receive and handle input from the screen or the keys of the targeted smartphone. More information regarding Moblets can be found of the official documentation page [18].

MAUI versus NativeUI

There exists more than one way to create a user interface with MoSync. For instance one can use either NativeUI or MAUI. MAUI stands for MoSync API for user interfaces and is a C++ library. It is platform-independent and provides the same aspect on every platform. The downside is represented by the fact that it is slower than NativeUI.
NativeUI on the other hand contains libraries written in JavaScript and C++, provides better performance but is not available on all platforms. Furthermore it provides a native look-and-feel in each respective environment.
Further information concerning the various solutions for cross-platform user interface development offered by MoSync can be found on the following source [61].

OpenGL and OpenGL ES

Since MoSync especially targets mobile platforms it supports rather OpenGL ES than OpenGL. The supported OpenGL ES versions are OpenGL ES 1.1, for older devices, and OpenGL ES 2.0.

Implemented Application

The described application from the section 'Practical Application for Comparison Purposes' can also be implemented using the MoSync SDK and IDE. The MoSync framework has been downloaded and installed and the project has been created as in the description above.
However not every feature has been implemented, because of the massive implementation effort that would have been required. Nevertheless the most important aspects and features for the described application will be described in the following sections.

Resources

Resources can be added to the project by creating a new MoSync Resource File. The name of the file must contains the ending '.lst'. In this file the resources must be defined, thus every resource will afterward dispose of an identifier and a path indicating the location of the resource within the project's folder. More information on this topic can be found on the following source [60].

```
.res BACKGROUND_IMAGE
.image "resources/galaxy.png"

.res MUSIC_MP3
.media "audio/mpeg", "resources/scrape.mp3"
```

Background Image

First the background image has to be enlisted in the resource file, as described above, and then an Image object with the dimensions of the device's screen size has to be created, referencing the image resource.

```
Image* img;

int screenSizeWidth;
int screenSizeHeight;

//get scree size
MAExtent screenSize = maGetScrSize();
screenSizeWidth = EXTENT_X(screenSize);
screenSizeHeight = EXTENT_Y(screenSize);

img = new Image(0, 0, screenSizeWidth, screenSizeHeight, NULL,
                                        true, true, BACKGROUND_IMAGE);
```

Then the Image object has to be set as main widget of a newly created Screen object. Furthermore the respective Screen object has to be shown by means of the method show(), otherwise the image will be drawn on a different canvas [56].

```
Screen* newScreen = new Screen();
newScreen->setMain(img);
newScreen->show();
```

Sound

First of all the '.mp3' file has to be enlisted in the Resource file as specified above. Afterward the sound can be played within the application by merely calling the function maSoundPlay(), referencing the identifier of the .mp3 file as first parameter.

```
maSoundPlay(MUSIC_MP3, 0, maGetDataSize(MUSIC_MP3));
```

Camera Controller

The movement of the camera, for example zooming, can be implemented within the pointerPressEvent() and pointerMoveEvent() methods. The factor by which the zooming should occur must be specified by the developer.

Camera

Since MoSync does not offer ready-to-go functions, it does not offer already implemented Camera objects. All code and classes have to be written by the programmer. So in order to have a camera in the scene, one first has to implement two classes, Camera.hpp and Camera.cpp for instance, as well as their methods and specify their positions.

Model Loading

MoSync offers no already implemented way of importing models, however it supports OpenGL ES. This means, that the developer can make use of other libraries, like Assimp [11], and implement their own ModelLoader classes, in order to save the imported models into hierarchical structures.

Moving Object

MoSync doesn't offer any pre-implemented solution for the movement of the imported models. This means that after the ModelLoader has been implemented the model can be moved either by hierarchical or skeletal animation.

Lighting

MoSync does not offer already implemented Light objects. Therefore light can be added to the scene by writing a shader class and defining vectors, which represent the color and intensity of the light.

Potential Problems

Installation Problems

No problems occurred during the installation phase.

Implementation Problems

- One can not select where on the hard disk to install the MoSync SDK. It is always installed in 'Applications', 'MoSync'.
- The project can not be built and can not run on iPhone. In order to fix this problem one first has to quit the MoSync IDE and edit the template file 'project.pbxprojtemplate' found in 'Applications', 'MoSync', 'profiles', 'runtimes', 'iphoneos', '1', 'template' with the OSX command line as follows: all 'GCC VERSION' lines equal with 'com.apple. compilers.llvmgcc42' must be removed and all 'IPHONEOS DEPLOYMENT TARGET' lines equal with '3.0;' must be replaced by them being equal to '5.0;' lines. After this is done the MoSync IDE can be started again. Then the project must be cleaned and rebuild. The solution can be found in the following discussion [59].
- The resource tutorials explaining how to use images are incomplete and do not mention that the namespace 'MAUI' also has to be used and that the MAUI library also has to be integrated into the project. This can be done by clicking 'Project' in the upper menu, followed by 'Properties' and 'Build Settings' and then 'MAUI.lib' can be added in the 'Additional Libraries' input field.
- Even after the MAUI.lib was included in the project correctly, the Resource tutorial does not specify how to show the image on the screen, but merely how to create an Image object using a resource. In order to show the image on the display screen of a mobile device, a Screen object must be generated first, the Image object has to be added to the screen as it's main widget and finally the new screen object can be displayed by means of the method show().
- Generally MoSync supports numerous features, sometimes even importing necessary libraries, like OpenGL ES, directly in the templates for a new project, but provides almost no pre-implemented elements, leaving the whole programming effort to the developer.

4.5 Xamarin

General Information

Xamarin enables users to build native Android, iOS, Mac and Windows applications by means of a shared C# code basis.
The developer team of Xamarin built it wanting to present the users with a better way of designing, developing, integrating and testing cross-platform applications.
Meanwhile Xamarin is being used by over 500.000 developers all around the world. More general information about Xamarin can be found on their official web site [71].

Programming Language

They chose C# as programming language, because in their opinion it is 'the best language for mobile development'. They justify this statement by asserting that everything one can implement in Objective-C or Java, can also be realized with C# [71].

Fees

Xamarin can be downloaded for free, the package including also Xamarin.iOS, Xamarin.Android, Xamarin.Mac, the Xamarin Studio IDE and a Visual Studio integration.
However for most tutorials, code snippets, examples and lessons available online one has to pay a rather expensive fee, so that not much help is available for free on the web, as for instance the tutorials on the following site [75].
Since the free version of Xamarin is so restricted that one can not even implement a relatively small application, the purchase of another version in exchange for a fee is practically necessary. However one can make use of the Xamarin Business edition Trial, which enables the developer to implement his or her project in Xamarin Studio for free for 30 days.

Integrated Development Environments

Xamarin can either be used in combination with Visual Studio or as a stand-alone IDE called Xamarin Studio. It represents a free development environment, which can be used on Mac as well as on Windows. Furthermore, Xamarin Studio enhances the look and feel specific of each platform by means of platform-specific code inserted in diverse places.
Since the MonoDevelop project, enabling cross-platform programming with C#, together with the Mono project are currently maintained by Xamarin, the IDE Xamarin Studio represents actually a renamed version of MonoDevelop 4.0. Further information considering Xamarin Studio can be found on the official website of Xamarin [71].

Developer Team

Xamarin was created in May 2011, San Francisco, California by the same developers who created Mono, MonoTouch and Mono for Android, which all represent cross-platform implementations of Microsoft .NET.
Currently the software company Xamarin disposes of 170 employees. More details concerning the Xamarin Team can be found on the following source [135].

Application Areas

Xamarin specializes on the development of cross-platform mobile applications and the maximization of code-reuse. However the focus lies on productivity and utility applications and not that much on game applications.
In order to receive better support for 3D and game programming, one should use Xamarin in combination with MonoGame. Further information on the topic can be collected on the following site [130].

MonoGame

MonoGame represents the most extended 3D Engine for Xamarin developers [132]. It is a free software, which offers rather good support for cross-platform game development with Xamarin and represents an open source implementation of XNA, as stated by the subsequent source [134]. MonoGame was initially used to make Windows and Windows phone games run also on Mac OS, iOS, Android, Linux, PlayStation Mobile and the OUYA console. More information on MonoGame can be collected from the official web site [74].

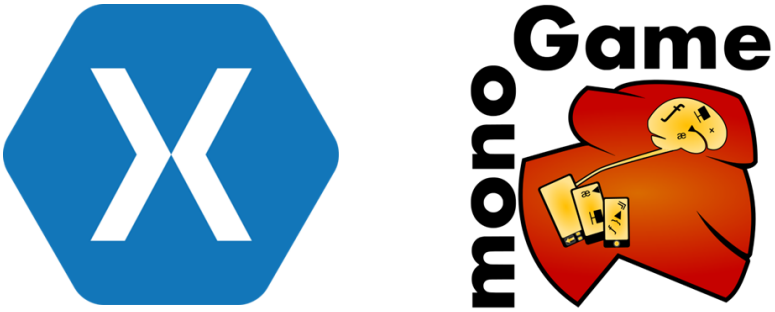


Figure 4.16: The left figure depicts the logogram of Xamarin and Xamarin Studio [5], while the right figure shows the logogram of MonoGame, which can be found on the following page [128].

XNA

Microsoft XNA stands for a tool set which facilitates video game development and management for Microsoft Window, Xbox 360 and Windows Phone.
MonoGame evolved from XNA Touch, as it represents an implementation of the Microsoft XNA 4 application programming interface. Furthermore MonoGame still depends on XNA quite much when it comes to certain topics, as for instance the Content Pipeline.

Prerequisite

The Xamarin executable can be downloaded from the official website [71]. As far as the installation is concerned, one must first accept the license agreement, then specify the location of the Android SDK and install the recommended software:

- Xamarin.iOS - enables the development of iOS applications with Xamarin
- Xamarin.Android - enables the development of Android applications with Xamarin
- Xamarin.Mac - enables the development of Xamarin applications on and for Mac OS
- Xamarin Studio - the IDE can also be downloaded and supports cross-platform development with Xamarin

- Visual Studio integration - makes it possible to use Xamarin in combination with the Visual Studio IDE
- Mono - enables the C# to function on several other platforms besides Windows

Furthermore as far as the iOS support is concerned, the prerequisites from the chapter 'Special Prerequisite for iOS' of the 'libGDX' section should be fulfilled.

Integrating MonoGame

MonoGame can be downloaded from the official MonoGame downloads website [54]. The latest release should be selected, followed by 'MonoGame 3.2 on GitHub', as the actual latest release is 3.2 and this section of the paper discusses the development with Xamarin.
Inside the downloaded package there is an '.mpack' file, which must be added to Xamarin Studio as follows: open Xamarin Studio, click on the application manager called 'Xamarin Studio' in the upper left corner and select 'Add-in Manager'. Afterward the 'Install from file' button of the bottom of the window should be clicked and the MonoGame mpack should be selected and loaded into Xamarin Studio.
This should cause all MonoGame templates to be available for use. The downloading instructions for MonoGame can be found on the following source [36].

Project Creation

New Project

A new Xamarin MonoGame project can be created by clicking on 'File', 'New', 'Solution'. After this the developer much choose between one of the available MonoGame templates. On Mac the available templates contain an Android template, as well as an OUYA, an iOS, a MacOS and a Linux template.

Open Project

An existing Xamarin project can be opened by clicking on 'File', 'Open' to open the folder containing the project and then selecting the '.sln' file, representing the project's solution.

Running the Project

Running the Desktop Project

The desktop project can be easily run after having opened the MacOS project. The developer can built the project by pressing 'Build', 'Build All' in the upper menu, as well as run it by selecting 'Run', 'Start Without Debugging'.

Running the iOS Project

The respective iOS project solution must be opened. The project can be built in the same manner as the desktop project. For running the project, the iOS device must be connected to the laptop

or personal computer. Furthermore, the device must be selected in the second drop down menu at the top of the window, the one specifying the available devices and emulators.
Afterward 'Run' can be selected from the upper menu, followed by 'Run With' and the respective iOS device must be chosen from the contextual drop down menu.

Running the Android Project

To run an Android application the respective project solution must be opened. The project can be built in the same manner as the desktop project. For running the project the Android device must be connected to the laptop or personal computer. Furthermore the device must be selected in the second drop down menu at the top of the window, the one specifying the available devices and emulators.
Afterward, 'Run' can be selected from the upper menu, followed by 'Run With' and the respective Android device must be chosen from the contextual drop down menu.

Implementation

Cross-platform projects: Manual Creation

As far as porting applications from one platform to another is concerned, the only instructions one can find without having to pay fees suggest, that the porting has to be performed manually. The following source [14] shows that Xamarin projects for other platforms can not be created automatically by automatic built, they have to be created manually. Furthermore the source shows how to port a Windows application to an Android project.

From Android to iOS For example one starts creating a Xamarin application for Android but wants to test it also on an iOS device. In order to do so a new project must be created or opened based on the iOS MonoGame template.
The resources must then be inserted into the new project, the 'Build action' of each resource must be set to 'EmbeddedResource' (or to 'AndroidAsset' when porting to an Android project) and 'Copy to output directory' must be set to 'Copy if newer'.
Furthermore the implemented C# classes containing the logic of the game must be imported from the former project and the 'Content.RootDdirectory' object inside the logic class has to be set to 'Assets/Content' instead of simply 'Content'.
Just after fulfilling these steps one can finally test one's application on a different platform. The reason for this is that building the project with Xamarin does not automatically generate projects for the other platforms as the previously described frameworks do.

OpenGL and OpenGL ES

MonoGame draws its graphics performance from OpenGL, as well as OpenGL ES or DirectX. The focus lies however on OpenGL 2 since MonoGame version 3. Before that the earlier releases of MonoGame used mainly OpenGL 1.x versions for the rendering of graphics.

Implemented Application

The described application from the section 'Practical Application for Comparison Purposes' has also been implemented using Xamarin in combination with MonoGame. Xamarin and MonoGame have been downloaded and installed and the project has been created as in the description above.

Camera and Camera Controller

There exists no built-in implementation of a camera in MonoGame, so that the Camera class containing it's whole functionality, such as rotating around objects or pitch zooming, has to be written by the developer. An example of a Camera class implementation can be found on the following source [106].

Model Loading

MonoGame supports merely the following model formats: .x, .fbx, according to the following source [79].
After having acquired such a model or having converted it to one of the respective formats, one can add the model as well as the texture files to the 'Assets/Content' folder of the project. The 'Build action' of the assets must be set either to 'AndroidAsset' on Android, or to 'Resource-Bundle' on iOS. In both cases 'Copy to output directory' should be set to 'Copy if newer'.
To load the file into the Model object one has to use the Content.Load() method within the Load-Content() method. However the type of the object to be loaded must be specified within angle brackets. Furthermore the name of the file, without it's file format, must be specified as parameter.

```
Model model = Content.Load<Model>("Ship");
```

One also needs to specify the model, view and projection matrices. Afterward a new method, DrawModel(), should be implemented, having the three matrices as parameters. This method goes through all the meshes of the model, followed by all the effects of each mesh and sets the three matrices. After all effects have been set the mesh is drawn.

```
private void DrawModel(Model model, Matrix world, Matrix view,
                                                   Matrix proj)
{
  foreach (ModelMesh mesh in model.Meshes)
  {
    foreach (BasicEffect effect in mesh.Effects)
    {
      //effect determine how a model should be displayed on the
      //screen
      effect.World = world;
      effect.View = view;
```

```
            effect.Projection = proj;
        }
        mesh.Draw();
    }
}
```

The DrawModel() method will then be called within the Draw() method of the application.

```
DrawModel(model, world, view, projection);
```

More information considering loading models can be found on the following source [81].

Moving Object

Basic movement of the objects can be achieved by altering the basic matrices. Especially the world matrix, which specifies the location of the model within the scene, should be altered to generate object movement. The methodes 'CreateTranslation()' and 'CreateRotationX, Y and Z(),' will prove to be quite useful. More details on the matter can be found on the following web site [78].

Lighting

One can add lighting to their scene by using the BasicEffect class. This class enables the user to implement a Default Lighting as well as a Custom Lighting.
One can easily add Default Lighting to the scene by inserting the following line of code, 'effect.EnableDefaultLighting();', inside the inner loop of the DrawModel() method, where the world, view and projection matrices are set.
Custom Lighting can be accomplished by adjusting the diffuse color, direction and specular color of one or more directional lights. There are a total of three directional lights built into the BasicEffect class. Additionally one can specify whether an individual light should be turned on or off.
Furthermore the ambient light's color can be altered and the emissive light color can be set. More information considering this matter can be found on the following site [76].

Background Image

MonoGame supports the following image and texture formats: .bmp, .dds, .dib, .hdr, .jpg, .pfm, .png, .ppm, .tga, according to the following source [79].
In order to load an image as a background, one needs to add the image file to the asset folder, while the 'Build action' of the asset must be set either to 'AndroidAsset' on Android, or to 'ResourceBundle' on iOS. In both cases 'Copy to output directory' should be set to 'Copy if newer'.
A Texture2D object is also needed, as a container for the image file. To load the file into the Texture2D object, one has to use the Content.Load() method within the LoadContent() method.

However, the type of the object to be loaded must be specified within angle brackets. Furthermore the name of the file, without it's file format, must be specified as parameter.

```
Texture2D background = Content.Load<Texture2D>("galaxy");
```

Furthermore, a SpriteBatch object needs to be defined and initialized in order to be able draw the texture(s) onto the screen.

```
SpriteBatch spriteBatch = new SpriteBatch (GraphicsDevice);
```

The drawing happens within the Draw() method, where the sprite.Begin() is called, followed by the sprite.Draw(). The parameters of the sprite.Draw() method specify the texture to be drawn as well as the height and width of the image. Finally the sprite must be ended with sprite.End().

```
spriteBatch.Begin ();

spriteBatch.Draw (background, new Rectangle(0, 0,
      GraphicsDevice.PresentationParameters.BackBufferWidth,
      GraphicsDevice.PresentationParameters.BackBufferHeight),
      Color.White);

spriteBatch.End ();
```

More information concerning MonoGame SpriteBatch basics can be found on the following source [80]. Furthermore, the following web site offers a solution to setting the height and width of the texture to the display's resolution [53].

Sound

For starters one must import the 'Microsoft.Xna.Framework.Media' library. This will allow the developer to use a Song object, which will be used to store the music file. MonoGame supports the following music files: .xap (an XACT audio project), .wma, .mp3, .wav, according to the following source [79].
To play a song in the background, the music file has to be added as a resource to the 'Assets/Content' folder first. The 'Build action' of the asset must be set either to 'AndroidAsset' on Android, or to 'ResourceBundle' on iOS. In both cases 'Copy to output directory' should be set to 'Copy if newer'.

```
Song song = Content.Load<Song>("scrape");
```

To load the music file into the Song object one has to use the Content.Load() method within the LoadContent() method. However the type of the object to be loaded must be specified within angle brackets. Furthermore the name of the file, without its file format, must be specified as

parameter.
Right after doing so the song can be played by the command 'MediaPlayer.Play(song)'.

```
MediaPlayer.Play(song);
```

More information on the audio topic can be found on the following web site [82].

Potential Problems

Installation Problems

MonoGame for Xamarin is not that difficult to download, being available on the official web site [74], but is quite difficult to install within Xamarin, since there are very few accurate instruction on how to do so. The detailed instructions can be found in the 'Prerequisite', 'Integrating MonoGame' section of this paper.

Implementation Problems

- Fees - there is a fee for almost all helpful tutorials, lessons and code snippets, which makes the implementation quite difficult, especially for new users of Xamarin.
- No automatic build of projects targeting other platforms - this represented a big problem, since one had to pay a fee to be able to access necessary tutorials and information. The solution used in this Bachelor thesis can be found in the section 4.5 of this paper.
- 'Your app is too large to be developed with Xamarin Starter Edition' - even a rather small application is too large for the Xamarin Starter Edition, so that the developer must either buy the Business edition of Xamarin or start a Trial version of the Business edition. The Trial version can be used up to 30 days, as stated by the following source [133].
- 'entitlements.plist file not found' - this can happen when starting a new iOS project. In order to solve this one needs to Right Click on the project and select 'Options' from the Dropdown menu. This will cause a window to open, where one should select 'iOS Bundle Signing' from the left side of the window and remove 'Entitlements.plist' from 'Custom Entitlements', as suggested by the following source [52].
- The only accepted model file is XNA's XNB - the fact that MonoGame only supports .x and .xnb model files represents quite a problem. This means that in order to be able to use other model file types, these need to be converted to .xnb files using an implementation of XNA's content pipeline. However, MonoGame does not dispose of such a content pipeline at the moment according to [29], which means that the developers need to use XNA's content pipeline to convert their models [77]. Since XNA only works on Windows, a Windows computer or laptop is required for the conversion. Furthermore, Visual Studio must be installed, as well as MonoGame and XNA, which must be integrated within Visual Studio. Then the user must create a new 'MonoGame Content Project' [79], which must

be added to the initial project and will finally be able to convert the model format to the .xnb file format.

- Trouble loading .png files, 'Asset Not Loaded' - PNG files need to be changed to a 24 or 32 bit color format, according to the subsequent source [105]. One can do this by using Photoshop or another graphics editor.

4.6 Marmalade SDK

General Information

Marmalade SDK was formerly known as Airplay SDK. It is defined as a free cross-platform application development tool and game engine. It provides a C++ framework, which enables the development not only in Visual Studio but also in Xcode.
Furthermore Marmalade SDK provides the developers with library files, documentation, useful samples and tools, which enable the development, testing and deployment of mobile applications.
Since Marmalade SDK is a cross-platform development tool it enables the users to write their code only once and then compile or execute it on all supported platforms. This is realized due to a C/C++ based application programming interface, which serves as an abstraction layer for the core API of each platform.
More information on Marmalade SDK can be found of their official website [47].

Developer Team

The Marmalade SDK was developed by Marmalade Technologies Limited, London, United Kingdom. The company was previously known as Ideaworks3D Limited.

Figure 4.17: The figure depicts the logogram of Marmalade SDK. The picture can be found on the following source [107].

Programming Languages

Marmalade SDK supports not only C++ but also Objective C, LUA and HTML5 as programming languages, according to the following source [47].
Namely Marmalade Quick allows developers to use LUA; Web Marmalade provides cross-platform application programming support based on web-technologies, so that the user can use HTML5 in combination with CSS3 and JavaScript; while Marmalade Juice offers Objective C support [50].

Integrated Development Environments

One can use either Visual Studio or Xcode to write their cross-platform application.

Games

Some examples of quite notorious games which have been developed using Marmalade SDK include Call of Duty - World at War: Zombies, Doodle Jump, Lara Croft and the Guardian of Light, Need for Speed: Shift, Plants vs. Zombies, Tetris and Worms.

The Marmalade Hub

The Marmalade Hub is a quite user-friendly graphic user interface tool, which provides an overview of the functionality provided by the SDK. Among other functions the Marmalade Hub enables the developer to create a new Marmalade project, offers templates, code samples, tutorials and documentation.

Architecture

The Marmalade SDK has the C++ SDK at its core. This provides flexibility and performance and enables the developers to directly access the hardware of the device, as well as the middleware and utility modules.
Marmalade Quick, Web Marmalade and Marmalade Juice are also based on the Marmalade C++ SDK, but permit the programmers the choice of using more than one programming language. More details as far as the Marmalade Platform is concerned, can be found at the subsequent source [50].

Technology

The writing of the code only once as well as its compilation and execution on all supported platforms is enabled by the C/C++ based API, which represents an abstraction layer for the core API of every platform.
The Marmalade SDK disposes of two main layers:

- Marmalade System - is a low level C API providing an abstraction layer. It permits the developer to access device functionality, such as file access, memory management, networking, input methods, sound and video output.

- Marmalade Studio - represents a C++ API that providing a higher level functionality. This layer mostly focuses of 2D and 3D graphic rendering, but also comprises a resource management system and HTTP networking.

More information on how Marmalade works and the used technology can be found on the following page [49].

Platforms

When purchasing the license, the developer can select his platforms of interest. The platforms from which one can choose are: Android, iOS, Windows Phone 8, BlackBerry 10, Tizen, Mac OS X, Windows Desktop, LG Smart TV, Roku 2 and Roku 3 [51].

Prerequisite

Download and Installation

Marmalade SDK can be downloaded from the official website [47]. As far as the installation is concerned, an activation key is needed in order to use the SDK. To receive such an activation key, the developer must register to Marmalade, however the registration is for free.
During the installation one must accept the license agreement, as well as specify the location where one wants to have Marmalade installed.
Furthermore, as far as the iOS support is concerned, the prerequisites from the chapter 'Special Prerequisite for iOS' of the 'libGDX' section should be fulfilled.

Project Creation

New Project

A new project can be created by means of the Marmalade Hub GUI. One simply has to click on 'Projects', next to the home icon in the upper left menu bar, and then press 'Create New Project'. Then the desired template can be chosen, as well as the location and the name of the project.

Open Project

In order to open a project, one must click on 'Projects', next to the home icon in the upper left menu bar, then either chose a project from the list and press 'Open Project' or import a different project by clicking on 'Import Project'.

Tutorials

Some quite helpful Marmalade tutorials for basic functionality are also available from the Marmalade Hub. One must simply click on the home icon in the upper left menu bar, then choose 'Marmalade C++' and press 'Starter Tutorial'. This will reveal a list of 7 'Stages', which can help the developer implement different functionality required in a game.

Running the Project

Running the Desktop Project

In order to build and run the project on the desktop, one must first select 'Debug x86' as build type in the upper left corner of the IDE window. After doing so, the project can be easily built by pressing 'Product', in the upper menu bar, 'Build', or run by clicking 'Product', 'Run'. This will run the project on a mobile device simulator.

Running the iOS Project

To build and deploy the project on an iOS device, the device must be connected to the computer and one must select 'Release GCC ARM' as build type in the upper left corner of the IDE window. After doing so one can press 'Product', in the upper menu bar, 'Build' to build the project and 'Product', 'Run' to run it. This will launch the Marmalade Deployment Tool.
'ARM GCC Release' must be selected as Build Type, 'iOS' as platform and finally 'Deploy All' must be clicked. This will create two files in the project's folder, under the 'build' folder ending with the name of the IDE one has is using, 'deployments', 'default', 'iPhone', 'debug'.

The .ipa file located there must then be dragged and dropped onto iTunes. Then the phone must be selected, followed by the 'Apps' tab, the checkbox next to the deployed project's application must be selected and the Sync button should be pressed. This will install the app to the iOS device. When deploying a newer version, the old one should be deleted manually. More information on the matter can be found on the subsequent source [91].
As an alternative solution to running the application on a device the Marmalade Hub could be used, however a configuration of the Hub would be necessary.

Running the Android Project

To run the developed application on an Android device, the device must be connected to the computer and one must select 'Release GCC ARM' as build type in the upper left corner of the IDE window. The Marmalade Hub or the Marmalade Deployment tool can both be configured in order to install and run the application automatically on an Android device.
This can be done by selecting 'Package and Install' or 'Package, Install and Run' in the Deployment section of the Marmalade Deployment tool. This will trigger the generation of a development package, which will be also copied to the device and even run, in case of having selected 'Package, Install and Run'. More information on deploying to Android can be found on the following website [45].

Implementation

OpenGL and OpenGL ES

Both OpenGL ES API versions, 1.x and 2.x, are used for accessing the graphics rendering capabilities of the supported mobile devices. OpenGL is not required since Marmalade SDK specializes on the development of mobile applications.

Implemented Application

The described application from the section 'Practical Application for Comparison Purposes' has also been implemented using the Marmalade SDK. The framework has been downloaded and installed and the project has been created as in the description above.

Camera and Camera Controller

As far as the addition of a camera to the scene is concerned, Marmalade SDK's CIwGameCamera class can be used. Such an object can be attached to a CIwGameScene object and enables the movement of the view around a virtual world. Furthermore CIwGameCamera is a quite simple class and supports the position, rotation and scaling of the view. The developer simply has to create a new CIwGameCamera object and add it to the Scene.
The Camera Controller has to be implemented manually, since there are no pre-implemented methods for pinch zooming or gravitating around an object so far. Further information can be collected from the following website [90].

Model Loading

The first thing which needs to be taken into consideration when intending to load a model with Marmalade SDK is that Marmalade can merely load models from a certain file format, namely from GROUP files. This means that the developer first has to export the model data into a GROUP file using a 3D modeling package.
Marmalade disposes of two exporter plug-ins for the following two modeling packages: Maya and 3DS Max. If one would not like to purchase Maya or 3DS Max, one could download Benoit Mueller's exporter plug-in for Blender, a free 3D modeling package.

After having installed one of the suggested modeling packages as well as the respective exporter plug-in, one can either create a new model or load one into the modeling package and then access the exporter, which allows the exportation of the model into a GROUP file. A GROUP file represents a text file in standard ITX format and defines a single named group, specifying the resources to be loaded into the group.
For more information regarding the exact steps of the exportation, the subsequent source can be consulted [83], and a detailed presentation of the exported file types can be read in the following article [84]. In order to read the articles, one has to log in to SafariBookOnline first, however there is a 10 days free trial available.

The group can then be loaded into the project by means of a resource manager object, which can be fetched using the IwGetResManager() method. Then the LoadGroup() method of the resource manager must be used, specifying the name of the group file as a parameter.
The group can afterward be accessed by means of a pointer to a CIwResGroup object. The pointer gets the value returned by the method GetGroupNamed() of the resource manager, using the name of the group specified in the group file as parameter.
Finally another pointer is needed, namely for the model. The model will be referenced by a

pointer to a CIwModel object, part of the IwGraphics API. The pointer to the model object should get the resource returned by the method GetResNamed(), which can be called from the group pointer object converted to a CIwModel* pointer.

```
// Load the board
IwGetResManager()->LoadGroup("Board.group");
// Get a pointer to the named group
CIwResGroup* pGroup = IwGetResManager()->GetGroupNamed("board");
// Get and store pointer to the model resource
CIwModel* board = (CIwModel*)pGroup->GetResNamed("board",
                                    IW_GRAPHICS_RESTYPE_MODEL);
```

Finally, in order to render the model, the method Render() of the returned CIwModel pointer can be used.

```
board->Render();
```

More details considering the exact code lines to be used for loading a model from a group file can be found on the subsequent source [48].

Moving Object

Before rendering a model, a transformation matrix should be set. This matrix specifies the position, scaling and orientation of the model in world space and can afterward be used to move the object. The transformation matrix can be declared using a CIwFMat object, rotation, scaling and orientation can then be set by using the methods SetRotX(), SetRotY() or SetRotZ(), ScaleRot and SetTrans().

```
// Build object 3D transform
CIwFMat Transform
Transform.SetRotY(Rotation.y);
Transform.ScaleRot(Scale);
Transform.SetTrans(Position);
```

Finally the model can be transformed into world space by setting the actual model matrix by means of the IwGx function IwGxSetModelMatrix(). However IwGx is state based, so that all further models will have the same model matrix until it is changed.

```
// Transform the model into world space
IwGxSetModelMatrix(&Transform);
```

For further information as well as the exact lines of code consult the following page [48].

Lighting

Marmalade SDK uses the Lighting Model to enable scene illumination. The emissive contribution can be activated using IwGxLightingEmissive(), the ambient contribution by IwGxLightingAmbient(), the diffuse contribution by IwGxLightingDiffuse() and the specular contribution by IwGxLightingSpecular(), where the parameter of all these methods can either be true or false, depending on whether one would like to activate or deactivate the respective contribution.
Also emissive, ambient, diffuse and specular lighting can be enabled together by using IwGxLightingOn(), or disabled together using IwGxLightingOff().

```
// Emissive contribution
IwGxLightingEmissive(true or false);

// Ambient contribution
IwGxLightingAmbient(true or false);

// Diffuse contribution
IwGxLightingDiffuse(true or false);

// Specular contribution
IwGxLightingSpecular(true or false);

// Turn emissive, ambient and diffuse lighting on
IwGxLightingOn();

// Turn emissive, ambient and diffuse lighting off
IwGxLightingOff();
```

Further information on the Lighting Model as well as the exact code lines can be looked up one the subsequent sources [46], [41].

Background Image

Using the Iw2D API represents one of the simplest methods to load and render an image with the Marmalade SDK. Even though Iw2D merely permits the rendering of rectangular polygons, these can be flipped and rotated when drawn. To load an image into the project, one first has to generate a new 'textures' folder within the 'data' folder of the project.

After the image file has been added to the 'textures' folder, one can create a pointer of a CIw2DImage object, in which the image can be loaded by means of the Iw2DCreateImage() method. Iw2DCreateImage supports the following image file formats: PNG, BMP, GIF, JPG and TGA.

```
CIw2DImage* image = Iw2DCreateImage("textures/galaxy.png");
```

The method Iw2DCreateImage() needs the name of the file including the file extension. Then the loaded image instance can be rendered to the screen by means of the method Iw2DDrawImage(), which uses the CIw2DImage object pointer, in which the image has been loaded, as well as the coordinates, where the image should be rendered on the screen, as parameters.

```
Iw2DDrawImage(image, CIwFVec2::g_Zero);
```

For more information one can consult the following page [42].

Sound

The audio API s3eAudio and the sound API s3eSound are used by the Marmalade SDK to play compressed audio data. The APIs can play the following file types: MP3, ACC and QCP. Marmalade SDK furthermore provides the sample sound engine IwSound, which supports the playback of compressed audio.
The Stage2 of the Starter Tutorial shows exactly how to play background music. The presented solution wraps up the access to IwSound into a class called 'Audio'. This class is then implemented by the two files 'audio.h' and 'audio.cpp', which are located in the project folder of Stage2.
Stage2 of the presented Starter Tutorial used however an additional class named 'AudioSound'. One can use this class to load and play an individual sound effect. Furthermore the Tutorial contains a change made to the original sample sound engine, so that WAV files are also supported. Further information can be collected from the following sources [44], [43].

Potential Problems

Installation Problems

No problems occurred during the installation phase and there was no need for the environment variables to be changed. However one has to register in order to receive the activation key needed for using the Marmalade SDK.

Implementation Problems

- Creation of the 'textures' folder - the IDE does not find the 'textures' folder, if it has been created from within the IDE's project structure or even if it has been created in the Finder while the project was opened in the IDE. The solution is represented by the closing of the IDE, the creation of a new 'textures' folder within the 'data' folder of the project by means of the Finder and the copying of the image file inside the newly created 'textures' folder. When opening the project in the IDE afterward, the 'textures' folder finally appears.

- Image can not be drawn - when running the application, even if the code was correct, the image could still not be drawn. The reason was represented by the fact that s3e didn't have enough memory size allocated. This can be solved by opening the 'app.icf' file within the

project and setting the 'MEMSIZE' of the s3e group to a greater number, as for instance '[s3e] MemSize $\bar{2}$00000000'. After doing so the image should be rendered onto the screen without further complications. This solution is suggested by the subsequent sources [89], [34].

- Deployment without the Hub - in case the deployment by means of the Marmalade Hub does not work, one can still deploy the application using the Marmalade Deployment tool. How to do this is covered by the 'Running the Project' section .

CHAPTER 5

Comparison

5.1 Comparison of the object-oriented programming -based cross-platform development approaches

As previously specified, this chapter will cover a more extensive comparison of the six cross-platform programming approaches presented in this paper: libGDX, Murl Engine, Unity, MoSync, Xamarin with MonoGame and Marmalade SDK.
The comparison is structured on the basis of the main features of the approaches, which are illustrated in the chapter 2.1 at the beginning of this bachelor thesis (Table 2.1).
For each main feature the best as well as the worst choice among the discussed object-oriented programming -based cross-platform programming approaches will be elucidated. Furthermore, the aspects they have in common and the main differences between the solutions provided by each approach will be emphasized.

Free Version Availability

Each of the six cross-platform programming approaches offers a free version of the respective framework, which is quite practical and convenient. Furthermore each of the approaches allows for them to be used by developers even for money-making purposes.
The free versions of most of the frameworks, namely 6 out of 7 including MonoGame, work quite nicely and without further obstructions, allowing the developers to implement their cross-platform games and applications unimpeded. However, the free version of Xamarin has an app size limitation, which prevents even the implementation of a minimal application, which simply loads an image.
Thus the users are practically forced to buy the Business version right away or use the free 30 days Trial of the Business version first and buy it afterwards.

Versions in Exchange for a Fee

The Murl Engine offers 'Basic', 'Standard' and 'Pro' as further versions one has to pay for; Unity disposes of a 'Pro' version with costs; MoSync also presents their versions 'Basic Pro' and 'God Pro' for a fee and Xamarin provides three further versions, namely 'Indie', 'Business' and 'Enterprise', all in exchange for a fee.
Each of these additional versions provide some extra functionality and liberties, such as render-to-texture effects, version control, optimization features and post processing effects in Unity Pro, however the prices are relatively high.
Meanwhile libGDX, MonoGame and Marmalade SDK each provide a single version of their frameworks, which is available for free and includes all the functionality.

Focus on mobile development

MoSync and Marmalade SDK both focus on mobile development, even though Marmalade targets not exclusively mobile platforms. The remaining approaches target mobile platforms as much as any other platform.
The main advantage, which is provided by the fact that a framework, such as MoSync, exclusively focuses on mobile development, is represented by the fact that the programmed application can be deployed to much more mobile platforms, namely eight, than in the case of Murl Engine or libGDX for example, which only support the deployment to two and respectively three mobile platforms.

Programming Languages

The object-oriented programming language Java has been used once during this bachelor thesis, namely for the libGDX practical implementation section. C# has been used twice, namely with Unity and Xamarin, and C++ represents the most used programming language during this bachelor thesis, being required for all three frameworks Murl Engine, MoSync and Marmalade SDK. The full list of supported programming languages can be found in the chapter 2 (Table 2.1).
One can not state that a certain object-oriented programming language is better than the others, therefore a ranking can not be established. However according to my personal observations, all frameworks based on the same programming language aim for different targets: for instance the Java-based framework libGDX provides more pre-implemented functionality than the others, the C#-based frameworks Unity and Xamarin focus on user-friendliness and fast, intuitive workflows, while the remaining C++ -based frameworks, Murl Engine, MoSync and Marmalade SDK, provide more flexibility and performance.

Integrated Development Environments

While Eclipse was used to implement the libGDX project, Xcode is required for Murl Engine and Marmalade SDK implementations. The remaining three approaches all provide an included IDE, namely the Unity IDE, the MoSync IDE and Xamarin Studio.
The IDEs were equally helpful.

Download and Installation Effort

Prerequisites

LibGDX appeared to require quite a lot of effort during the download and installation part, however this was the case mainly because it represented the first tested framework during this bachelor thesis and because many aspects had to be taken care of only once, such as the provisioning and registration of the iPhone and the installation of the Java SDK and the Android SDK.
Thus after having fulfilled these aspects during the libGDX familiarization time, these steps did not need to be repeated for every framework. In hindsight the download and installation effort of libGDX did not surpass the effort invested in the download and installation of the other frameworks.

Trouble with Xamarin and MonoGame

Almost each of the available frameworks could be installed without further complications, as soon as the developer accepted the respective license agreement.
The only exception was represented by MonoGame, which had to be integrated into Xamarin. The integration of MonoGame was rather costly, since there is almost no free and helpful information on the matter available online.

Registration required

Only one framework requested a registration and a key before enabling it's use, namely the Marmalade SDK. For all other frameworks a registration was not necessary.

Choice of Installation Path

All frameworks except the Murl Engine and MoSync allowed the developers to choose their preferred installation path. The fact that the installation path can not be determined personally when using the Murl Engine and MoSync can lead to certain problems: for instance the installation causing the respective disk to be nearly full, the developer not being able to remember where the framework is installed, or even the impossibility to install the framework at all, when the disk has insufficient free space left.

Familiarization with the Framework

The familiarization periods for libGDX, Unity and the Marmalade SDK have been explicitly shorter than the familiarization time required to understand how the Murl Engine, MoSync and Xamarin with MonoGame are working. This is mainly the case because there are much more tutorials, documentation and generally help found for free available especially for libGDX and Unity. For the Marmalade SDK the tutorials provided by the Marmalade Hub were quite helpful during it's familiarization process.
For the Murl Engine there are very few tutorials availiable on the official web site, and they provide no support at all for 3D applications at the moment. The MoSync tutorials are also not

that many, since it is no longer maintained, so that the significant information had to be collected from forums. For Xamarin there are quite many tutorials, however they are not available for free.

Tutorials

The availability of tutorials facilitates a quicker familiarization with a certain framework, as also mentioned in the section 5.1.
There are quite many tutorials available for Unity, libGDX and even Marmalade SDK. There are also many tutorials available for Xamarin but merely in exchange for a fee. The Murl Engine provides a few tutorials, but they do not offer any help for 3D application programming yet. For MoSync there are almost no tutorials available, so that all necessary information, such as how to set up a project, how to process user input and how image, sound and model loading can be realized, had to be extracted mainly from forum discussions.

Community

Unity posesses the largest community and therefore quite a big amount of help and instructions are available online. Each of the other 5 frameworks also has a community, even if the community of the Murl Engine is still rather small. However one can contact the developers of Murl Engine directly; they proved to be quite helpful and friendly.

Supported Mobile Platforms

All approaches support the mobile platforms Android and iOS, since this represented the main criteria for their selection in order to be presented in this bachelor thesis.
MoSync clearly supports the most mobile platforms of all, namely at least 8, these representing their main target. Some examples of mobile platforms supported merely by MoSync and no other of the described frameworks are: Java ME MIDP, Moblin, Symbian and Windows Mobile. Meanwhile Murl Engine supports the least mobile platforms, namely just Android and iOS.

Other Supported Platforms

All approaches except MoSync support the desktop computer platforms Windows and Mac OS. MoSync does no support any other platforms than the previously mentioned mobile platforms, since the latter ones represent their main target. Unity supports the most other platforms of all, namely at least 11. LibGDX, Unity and Xamarin provide also support for Linux as a target platform.
While Unity provides support for considerably many game consoles, Marmalade SDK supports quite some television devices as platforms. Murl Engine supports the lowest number of other platforms, namely just Windows and Mac OS.

Deployment to other Platforms

The easiest deployment to other platforms is provided by libGDX, since if one makes a change in the core project, all other sub-projects are automatically altered when building.

Xamarin on the other hand does not provide automatic deployment to different platforms, so that the so-called sub-projects have to be created and altered manually.
Murl Engine, Unity, MoSync and Marmalade SDK all provide a relatively smooth deployment, even if Xcode is necessary for the Murl Engine and MoSync in order to deploy the application to iOS devices.

GUI Tool for Deployment

LibGDX, Murl Engine and Marmalade SDK provide the developers with graphical user interface tools, namely the libGDX Project Generator, the Dashboard and the Marmalade Hub. These tools are meant to simplify the deployment on different platforms.
The libGDX Project Generator is the most user-friendly one, the Dashboard provides only help with the deployment to Android and the Marmalade Hub, although it is not that user-friendly, it helped with the deployment to both iOS and Android.

Running the Project

Running the Desktop Project

Running the Desktop Project is enabled quite easily by all frameworks, except for MoSync, which focusses merely on mobile platform development.

Running the iOS Project

Murl Engine, Unity and MoSync all use Xcode in order to run the implemented application on an iOS device, even if the application was not developed in Xcode.
Running the iOS project can be initiated directly from Xamarin Studio when developing with Xamarin, and Eclipse when developing with libGDX.
The running of the iOS project developed with Marmalade SDK can be realized by means of a deployment tool.

Running the Android Project

The libGDX, Unity, MoSync and Xamarin projects can be run on an Android device quite easily, namely directly from the IDE which is used for the development of the application.
In order to run the Murl Engine project on Android, the Dashboard GUI can be used, while running the Marmalade project can be realized by means of a deployment tool.

Rendering API support

All presented cross-platform approaches provide rather extensive OpenGL ES support. Furthermore all of them, except MoSync and Marmalade SDK, provide also OpenGL support. Murl Engine and Unity also support DirectX. The exact versions of the APIs can be seen in the chapter 2 (Table 2.1).

Camera Implementation

The camera can be added most easily to the scene with Unity. However libGDX, Murl Engine and Marmalade SDK also offer pre-implemented camera objects, which can be used after reading the documentation.
The most problematic frameworks concerning the camera implementation are MoSync and Xamarin, where one has to write the entire camera object from scratch.

Camera Controller Implementation

The camera controller can be added very easily when using libGDX, because a lot of helpful functions, such as zooming by scrolling, zooming by pinching and orbitation around an object are already implemented.
Murl Engine and Unity offer some help in this direction at least, even if a complete pre-implementation is not available, in comparison to MoSync and Xamarin, where the developer must implement the Camera Controller from the beginning.

Model Loading

Model Loading is easiest with Unity, since the model importation is realized mainly by dragging and dropping it into the game engine. However, libGDX also offers a quite easy to use pre-implemented Model Loader.
Murl Engine supports model loading, however a conversion of the model to Murl Meshes has to be carried out by means of a Scene Converter tool. Similarily, Marmalade SDK requires that the model is exported from a modeling package in the form of a GROUP file.
The model loading with Xamarin and MonoGame is exceptionally complicated, since the only accepted file formats are .xnb files and since MonoGame does not dispose of a Content Pipeline. Thus the Content Pipeline of another framework, XNA, needs to be used. This entire conversion to .xnb requires further installation effort and quite much time and memory loss.
MoSync does not provide a model loader at all, so that it has to be implemented from scratch by the developer.

Supported Model Formats

LibGDX and Unity support the most model file formats without any conversion being necessary, except the Murl Engine and MoSync, which both accept all model formats supported by Assimp, however a conversion is necessary when using the Murl Engine and MoSync requires the implementation of a model loader from scratch first.
Xamarin with MonoGame and the Marmalade SDK both support just quite few and unusual model formats, so that various conversions are necessary. The complete list of supported model formats can be found in the Chapter 'Overview' (Table 2.1).

Object Movement Implementation

Moving an object or a model can be realized quite easily with libGDX, Murl Engine, Unity, Xamarin and Marmalade SDK, even though each framework has its own approach.
MoSync requires however a full implementation by the developer, since no effort-minimizing methods are available.

Lighting implementation

Lighting can be added to the scene quite easily by means of Unity, libGDX, Xamarin, Marmalade SDK and even Murl Engine. The only framework where the light has to be implemented manually and from the beginning within shaders is MoSync.

Touchscreen Interface

The Touchscreen interface can be accessed extremely easy by means of libGDX, which provides this exact pre-implemented function among others. Unity and Marmalade SDK also provide quite some helpful methods on this topic, however the implementation is not complete. Xamarin and MoSync provide the least support in this direction, so that a higher implementation effort is required.

Image Loading

Image Loading can be realized relatively fast and easy using libGDX, Murl Engine, Xamarin with MonoGame, Marmalade SDK and even MoSync. This time the Unity solution proved to be the most problematic, since a complicated construct using two cameras had to be established.

Supported Image Formats

Xamarin supports the most image formats, followed by Unity. .PNG and .JPG are supported by all described frameworks.
An exact list of the supported image file formats can be viewed in the chapter 'Overview' (Table 2.1).

Sound Implementation

The implementation of the sound is realized very easily with MoSync and libGDX. Murl Engine, Unity and Xamarin also provide good support for sound loading. Marmalade SDK however requires some more implementation, since not all necessary classes are pre-implemented.

Supported Sound Formats

Xamarin and Unity support the most music file formats. .MP3 and .WAV are supported by nearly all frameworks, with the exception of Murl Engine, which does not support .MP3 files and Marmalade SDK, which offers support for .WAV files merely with some additional programming effort.

An exact list of the supported music file formats can be viewed in the chapter 'Overview' (Table 2.1).

CHAPTER 6

Conclusion

This bachelor thesis covers the presentation of six of the currently most popular cross-platform programming approaches. Not only the general information is covered, but also important implementation-related aspects. In the beginning of the thesis an overview of the approaches is presented, followed by the detailed description of each framework. In the end a comparison of the approaches is provided, which is structured based on the main features of the frameworks.

Besides, this bachelor thesis was written with the intention of providing the readers with a basic knowledge of the State of the Art in object-oriented programming -based cross-platform developing approaches, as well as give an overview of their advantages and disadvantages. Furthermore, the thesis was meant to emphasize that even though the approaches are quite different, they still produce similar and comparable results and that each of the approaches aims for different targets. These so-called targets can stand either for certain nonfunctional requirements, such as usability or performance, or for target customers, representing different developers with various implementation styles.

For instance the **libGDX** framework provides more pre-implemented functionality than the other frameworks. libGDX therefore aims for a faster implementation, intending to save the developers the time of developing basic 3D scene components, such as camera objects, light sources, camera controllers and model loaders, which are needed basically in every 3D game application. The developers can thus use this time to focus on the application's logic, the game play and the level design.

Murl Engine aims especially for flexibility and performance, so that it does not provide much pre-implemented functionality, leaving the whole programming effort to the developer. Also Murl Engine embraces a very distinct approach, separating the positioning of the objects in the scene and the logic part very strictly using XML files. This provides a series of advantages, such as a clear view of the scene graph and a good structure of the code.

Unity enables a remarkably fast workflow, by means of its user-friendly, embedded game engine and the impressive ease with which it enables assets management. Unity also provides a relatively quick positioning of the objects in the scene and simultaneously enables the developer to envision how the scene will look like in the end, all of which facilitate the level design. Furthermore Unity supports the most platforms and the most model formats.

MoSync, despite the fact that it is no longer maintained, enables all possibly imaginable features, however their main goal was also flexibility, so that the user, even though more implementation effort was required, could implement all the features to his own taste. In this situation one does not have to adhere to the implementations of built-in functions anymore, in exchange for more implementation effort.

Xamarin also aims for user-friendliness and fast intuitive workflows, however the implementation using Xamarin is relatively expensive and in order to be able to support 3D application implementation it has to be combined with MonoGame. **MonoGame** offers quite good 3D graphics support and also provides much pre-implemented functionality. However model loading still represents a problematic topic due to the missing integrated Content Pipeline.

Last but not least **Marmalade SDK** provides great support for cross-platform mobile development, even though it does not focus on mobile platforms especially. It provides great flexibility and performance due to it's C++ basis and combines at the same time pre-implemented functionality with the liberty of extending an implementation or implementing a functionality from the beginning.

Thus one cannot state that one approach is better than the rest, for every one of them excels in diverging aspects. In the end all presented cross-platform programming approaches are capable of creating amazing applications and games for more than one platform by means of a single code base. Therefore the choice of which one to use is dependent on the context and left to each and every developer.

Bibliography

[1] http://dist.springsource.com/release/TOOLS/gradle. Link to Gradle plug-in. Accessed: 2014-10-14.

[2] https://dl-ssl.google.com/android/eclipse/. Link to Android Eclipse plug-in. Accessed: 2014-10-14.

[3] 360logica.com, Sadik Ali. http://www.360logica.com/blog/2014/05/automated-cross-platform-testing.html. Accessed: 2014-10-12.

[4] Adobe Systems Inc. http://phonegap.com/. PhoneGap, official website. Accessed: 2014-11-17.

[5] Alejandro Ruiz Varela. http://alejandroruizvarela.blogspot.co.at/2014/06/simple-listview-with-xamarinforms.html. Xamarin and Xamarin Studio logogram, source. Accessed: 2014-10-17.

[6] Sarah Allen, Vidal Graupera, and Lee Lundrigan. *Pro smartphone cross-platform development: iPhone, blackberry, windows mobile and android development and distribution*. Apress, 2010.

[7] Android Developers. http://developer.android.com/sdk/index.html. Android SDK download page. Accessed: 2014-10-14.

[8] Android Developers. https://developer.android.com/guide/index.html. Android Developer Guide. Accessed: 2014-10-14.

[9] Appcelerator Inc. http://www.appcelerator.com/. Appcelerator, official webpage. Accessed: 2014-11-17.

[10] Apple Inc. https://developer.apple.com/. Apple Developer web site. Accessed: 2014-10-14.

[11] Assimp Development Team. http://assimp.sourceforge.net/. Assimp, official web page. Accessed: 2014-10-17.

[12] Assimp Development Team. http://assimp.sourceforge.net/main_features_formats.html. Assimp, supported model formats. Accessed: 2014-10-19.

[13] Brackeys. https://www.youtube.com/watch?v=xafs9uljpqE. Unity, Audio Tutorial. Accessed: 2014-10-17.

[14] Dean Ellis. https://www.youtube.com/watch?v=hF0sm5KGPPM. MonoGame, Instructions on how to port a Windows application to Android. Accessed: 2014-10-17.

[15] Developer Android. http://developer.android.com/sdk/index.html. Android SDK, download page. Accessed: 2014-10-15.

[16] Developer Android. http://developer.android.com/tools/sdk/ndk/index.html. Android NDK, download page. Accessed: 2014-10-15.

[17] Dice. http://news.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/. Unity, Game Development Beast. Accessed: 2014-10-17.

[18] doxygen. http://www.mosync.com/files/imports/doxygen/latest/html/class_m_a_util_1_1_moblet.html#details. MoSync Documentation, Moblet. Accessed: 2014-10-17.

[19] Dreamcss. http://blog.dreamcss.com/frameworks/widgetpad-mobile-development-platform/. Widgetpad, information. Accessed: 2014-11-17.

[20] Eric Spitz. https://code.google.com/p/libgdx-users/wiki/ImportingModelsFromBlender. LibGDX, Supported 3D model formats. Accessed: 2014-10-15.

[21] GameFromScratch.com. http://libgdx.badlogicgames.com/download.html. LibGDx setup jar, download page. Accessed: 2014-10-14.

[22] GameFromScratch.com. http://www.gamefromscratch.com/post/2013/11/19/LibGDX-Tutorial-8-Audio.aspx. LibGDX tutorial, audio. Accessed: 2014-10-15.

[23] GameFromScratch.com. http://www.gamefromscratch.com/post/2014/02/20/Registering-for-an-Apple-Developer-ID.aspx. Information on how to register for an Apple Developer ID. Accessed: 2014-10-14.

[24] GameFromScratch.com. http://www.gamefromscratch.com/post/2014/02/23/Deploying-a-LibGDX-RoboVM-project-to-an-actual-iOS-device.aspx. Information on how to deploy a LibGDX RoboVM project to an iOS device. Accessed: 2014-10-14.

[25] GitHub, Inc. https://github.com/libgdx/libgdx/wiki/Dependency-management-with-Gradle. Information on dependency management with Gradle. Accessed: 2014-10-14.

[26] GitHub, Inc. https://github.com/libgdx/libgdx/wiki/Gradle-on-the-Commandline#packaging-for-android. Gradle on the Command Line, documentation. Accessed: 2014-10-14.

[27] GitHub, Inc. https://github.com/libgdx/libgdx/wiki/Opengl-es-support. LibGDX OpenGL ES support. Accessed: 2014-10-15.

[28] GitHub, Inc. https://github.com/libgdx/libgdx/wiki/Project-setup,-running-&-debugging. libGDX, Information on project setup, running and debugging. Accessed: 2014-11-15.

[29] GitHub, Inc. https://github.com/mono/MonoGame/wiki/MonoGame-Content-Processing. MonoGame, Information on MonoGame Content Processing and the Content Pipeline. Accessed: 2014-10-18.

[30] GitHub, Inc. https://github.com/silexlabs/Cocktail. Cocktail, Information on GitHub. Accessed: 2014-11-18.

[31] Google Developers. https://dl.google.com/eclipse/plugin/4.3. Link to GWT plug-in. Accessed: 2014-10-14.

[32] Google Inc. http://www.gwtproject.org/download.html. Google Web Toolkit SDK download page. Accessed: 2014-10-14.

[33] Henning Heitkötter, Sebastian Hanschke, and TimA. Majchrzak. Evaluating cross-platform development approaches for mobile applications. In José Cordeiro and Karl-Heinz Krempels, editors, *Web Information Systems and Technologies*, volume 140 of *Lecture Notes in Business Information Processing*, pages 120–138. Springer Berlin Heidelberg, 2013.

[34] Ideaworks3D Limited. https://legacy.madewithmarmalade.com/ru/node/51039. Marmalade, Information on fixing Memory Allocation Issues. Accessed: 2014-10-18.

[35] JesseEtzler0. https://www.youtube.com/watch?v=wi0iI8LgWSQ. Unity, Camera Zoom by Scrolling Tutorial. Accessed: 2014-10-17.

[36] Justin Aquadro. http://jaquadro.com/2013/09/monogame-hello-world-on-mac-os-x-and-xamarin-studio/. MonoGame, Install instructions. Accessed: 2014-10-17.

[37] LinkedIn Corporation. http://de.slideshare.net/raphaelharmel/ny-coders-crossplatform-development-with-haxe-openfl-and-cocktail. Cocktail, Information on slideshare. Accessed: 2014-11-18.

[38] Mario Zechner. http://libgdx.badlogicgames.com/index.html. libGDX official website. Accessed: 2014-10-12.

[39] Mario Zechner. http://www.badlogicgames.com/wordpress/?attachment_id=1884. Logogram of libGDX, source. Accessed: 2014-10-13.

[40] Mario Zechner. http://www.badlogicgames.com/wordpress/?page_id=2. About Badlgic Games. Accessed: 2014-10-13.

[41] Marmalade Technologies Ltd. http://developer.roolez.com/marmaladesdk/api/iwgxapidocumentation/iwgxapioverview/lighting/lightingscenelights.html. Marmalade, Information on Lighting: Scene Lights. Accessed: 2014-10-18.

[42] Marmalade Technologies Ltd. http://docs.madewithmarmalade.com/display/MD/Displaying+images#Displayingimages-LoadingandrenderingimagesusingIwGLandOpenGLES. Marmalade Documentation, Displaying images. Accessed: 2014-10-18.

[43] Marmalade Technologies Ltd. http://docs.madewithmarmalade.com/display/MD/Input+and+Audio. Marmalade Documentation, Stage 2 - Input and Audio. Accessed: 2014-10-18.

[44] Marmalade Technologies Ltd. http://docs.madewithmarmalade.com/display/MD/S3E+Audio+overview. Marmalade Documentation, S3E Audio overview. Accessed: 2014-10-18.

[45] Marmalade Technologies Ltd. http://docs.madewithmarmalade.com/display/MD/Testing+and+debugging+on+Android+devices. Marmalade Documentation, Information on Testing and debugging on Android devices. Accessed: 2014-10-18.

[46] Marmalade Technologies Ltd. http://docs.madewithmarmalade.com/display/MD/The+Lighting+Model. Marmalade Documentation, The Lighting Model. Accessed: 2014-10-18.

[47] Marmalade Technologies Ltd. https://www.madewithmarmalade.com/. Marmelade official site. Accessed: 2014-10-12.

[48] Marmalade Technologies Ltd. https://www.madewithmarmalade.com/blog/easy-3d-lights-camera-action. Marmalade, Easy 3D – Lights, camera, action! Tutorial. Accessed: 2014-10-18.

[49] Marmalade Technologies Ltd. https://www.madewithmarmalade.com/marmalade/how-marmalade-works. Marmalade Technology. Accessed: 2014-10-18.

[50] Marmalade Technologies Ltd. https://www.madewithmarmalade.com/marmalade/marmalade-platform. Marmalade Platform, Information on Marmalade Quick, Web Marmalade and Marmalade Juice. Accessed: 2014-10-18.

[51] Marmalade Technologies Ltd. https://www.madewithmarmalade.com/marmalade/supported-platforms. Marmalade, Supported Platforms. Accessed: 2014-10-18.

[52] Microsoft. http://community.monogame.net/t/entitlements-plist-not-created-by-xamarin-ios-plugin/1014. Xamarin, Solution to 'entitlements.plist file not found'. Accessed: 2014-10-18.

[53] Microsoft. https://monogame.codeplex.com/discussions/406917. MonoGame, Information on fullscreen images. Accessed: 2014-10-18.

[54] MonoGame Team. http://www.monogame.net/downloads/. MonoGame, Download page. Accessed: 2014-10-17.

[55] MoSync AB. http://www.mosync.com/. MoSync official website. Accessed: 2014-10-12.

[56] MoSync AB. http://www.mosync.com/content/how-embed-image. MoSync, Information on How to embed an image. Accessed: 2014-10-17.

[57] MoSync AB. http://www.mosync.com/content/mosync-future-bankruptcy. MoSync, information of bankruptcy. Accessed: 2014-10-17.

[58] MoSync AB. http://www.mosync.com/content/mosync-logo-colour-vertical-light. MoSync logogram, source. Accessed: 2014-10-17.

[59] MoSync AB. http://www.mosync.com/content/unsupported-compiler-architecture-armv7. MoSync, solution to project not being able to run on iPhone. Accessed: 2014-10-17.

[60] MoSync AB. http://www.mosync.com/docs/sdk/cpp/guides/adding-resources-mosync-project/index.html. MoSync, Information on Resource Management. Accessed: 2014-10-17.

[61] MoSync AB. http://www.mosync.com/docs/sdk/cpp/guides/ui/creating-user-interfaces-mosync/index.html. MoSync, Information on Creating User Interfaces With MoSync, MAUI vs NativeUI. Accessed: 2014-10-17.

[62] MoSync AB. http://www.mosync.com/docs/sdk/tools/guides/ide/developing-iphone-applications/index.html. MoSync, Run on iPhone. Accessed: 2014-10-17.

[63] MoSync AB. http://www.mosync.com/docs/sdk/tools/guides/ide/sending-to-device/index.html. MoSync, Run on Android. Accessed: 2014-10-17.

[64] MoSync AB. http://www.mosync.com/docs/sdk/tools/references/feature-platform-support/index.html. MoSync, supported platforms. Accessed: 2014-10-17.

[65] MoSync AB. http://www.mosync.com/download. MoSync, Download. Accessed: 2014-10-17.

[66] MoSync AB. http://www.mosync.com/forum/. MoSync Mobile Development Forum, Community. Accessed: 2014-10-12.

[67] Motorola Solutions, Inc. http://rhomobile.com/. RhoMobile, official website. Accessed: 2014-11-17.

[68] mp3olimp. http://www.mp3olimp.net/blue-stahli-scrape-instrumental/. Reference for the game's background music: Blue Stahli - Scrape (Instrumental), free download page. Accessed: 2014-10-12.

[69] Aaftab Munshi, Dan Ginsburg, and Dave Shreiner. *OpenGL ES 2.0 programming guide*. Pearson Education, 2008.

[70] Benjamin Muschko. *Gradle in Action*. Manning, 2014.

[71] MXamarin Inc. http://xamarin.com/. Xamarin official web site. Accessed: 2014-10-12.

[72] Nesis. https://www.youtube.com/watch?v=6eUt1yHvH_U. Unity, Camera Controll Tutorial. Accessed: 2014-10-17.

[73] Oracle Corporation. http://www.oracle.com/technetwork/java/javase/downloads/index.html. Java SDK download page. Accessed: 2014-10-14.

[74] Philip Stolpe, MonoGame Team. http://www.monogame.net/. MonoGame official site. Accessed: 2014-10-12.

[75] pluralsight. http://pluralsight.com/training/courses/TableOfContents?courseName=monogame. Xamarin, Tutorials on Cross-Platform Game Development with MonoGame in exchange for fee. Accessed: 2014-10-17.

[76] RB Whitaker. http://rbwhitaker.wikidot.com/basic-effect-lighting. MonoGame, BasicEffect Lighting Tutorial. Accessed: 2014-10-18.

[77] RB Whitaker. http://rbwhitaker.wikidot.com/monogame-accessing-the-xna-content-pipeline. MonoGame, Information on Accessing the XNA Content Pipeline. Accessed: 2014-10-18.

[78] RB Whitaker. http://rbwhitaker.wikidot.com/monogame-basic-matrices. MonoGame, Basic Matrices Tutorial, Information on model movement. Accessed: 2014-10-17.

[79] RB Whitaker. http://rbwhitaker.wikidot.com/monogame-managing-content. MonoGame, Supported Content Formats. Accessed: 2014-10-17.

[80] RB Whitaker. http://rbwhitaker.wikidot.com/monogame-spritebatch-basics. MonoGame, Monogame Spritebatch Basics, Image Loading Tutorial. Accessed: 2014-10-18.

[81] RB Whitaker. http://rbwhitaker.wikidot.com/monogame-using-3d-models. MonoGame, Model Loading Tutorial. Accessed: 2014-10-17.

[82] RB Whitaker. http://rbwhitaker.wikidot.com/playing-background-music. MonoGame, Playing Background Music Tutorial. Accessed: 2014-10-18.

[83] Safari. https://www.safaribooksonline.com/library/view/marmalade-sdk-mobile/9781849693363/ch04s03.html. Marmalade, Information on Using a 3D modeling package to create and export model data. Accessed: 2014-10-18.

[84] Safari. https://www.safaribooksonline.com/library/view/marmalade-sdk-mobile/9781849693363/ch04s04.html. Marmalade Information on The Marmalade 3D model datafile formats, loading and rendering models. Accessed: 2014-10-18.

[85] Science Park Graz GmbH. http://sciencepark.at/aktuelles/news/458/murl-engine-geht-in-die-open-beta. Information on the Murl Developers. Accessed: 2014-10-15.

[86] Sebastien Dubois. http://blog.gfx47.com/2011/03/08/unity3d-httphttps-proxy-problem-solved-o/. Unity Logo, source. Accessed: 2014-10-17.

[87] Sencha Inc. http://www.sencha.com/. Sencha, official website. Accessed: 2014-11-17.

[88] Sencha Inc. http://www.silexlabs.org/haxe/cocktail/. Cocktail, Information on Silexlabs. Accessed: 2014-11-18.

[89] Shlomi Noach. http://www.drmop.com/index.php/2011/09/27/marmalade-sdk-bitesize-tutorial-running-out-of-memory-and-how-to-fix-it/. Marmalade, Bitesize Tutorial – Running out of memory and how to fix it. Accessed: 2014-10-18.

[90] Shlomi Noach. http://www.drmop.com/index.php/2011/10/28/marmalade-sdk-tutorial-actors-scenes-and-cameras-make-the-world-go-round/. Marmalade - Actors, Scenes and Cameras Make the World Go Round Tutorial. Accessed: 2014-10-18.

[91] Shlomi Noach. http://www.drmop.com/index.php/2011/12/10/marmalade-sdk-tutorial-apple-iphone-and-ipad-deployment-and-submissions/. Marmalade, Apple iPhone and iPad Deployment and Submissions Tutorial. Accessed: 2014-10-18.

[92] Spraylight GmbH. http://murlengine.com/?murlpage=about&murllang=de. Introduction to the Murl Engine. Accessed: 2014-10-15.

[93] Spraylight GmbH. http://murlengine.com/?murlpage=about&murllang=de. Introduction to the Murl Engine, source of code types diagram. Accessed: 2014-10-15.

[94] Spraylight GmbH. http://murlengine.com/?murlpage=download&murllang=de. Murl Engine, download page. Accessed: 2014-10-15.

[95] Spraylight GmbH. http://murlengine.com/?murlpage=murl&murllang=de. Murl Engine official website. Accessed: 2014-10-12.

[96] Spraylight GmbH. http://murlengine.com/tutorials/de/tut0101_cube.php. Murl Engine, Cube tutorial, information regarding 3D camera and moving objects. Accessed: 2014-10-15.

[97] Spraylight GmbH. http://murlengine.com/tutorials/de/tut0102_color_cube.php. Murl Engine, Color Cube tutorial, information regarding lighting and user input. Accessed: 2014-10-15.

[98] Spraylight GmbH. http://murlengine.com/tutorials/de/tut0104_audio.php. Murl Engine, Audio tutorial. Accessed: 2014-10-15.

[99] Spraylight GmbH. http://murlengine.com/tutorials/de/tut0107_images.php. Murl Engine, Images tutorial. Accessed: 2014-10-15.

[100] Spraylight GmbH. http://murlengine.com/tutorials/de/tutorials.php. Murl Engine, Tutorials. Accessed: 2014-10-15.

[101] Spraylight GmbH. http://murlengine.com/usersguide/de/target_installation_guide_mac.php. Murl Engine, information of installation. Accessed: 2014-10-15.

[102] Spraylight GmbH. http://murlengine.com/usersguide/de/user_guide_short_introduction.php. Murl Engine, Introduction to the development with the framework. Accessed: 2014-10-15.

[103] Spraylight GmbH. http://spraylight.at/?spllang=de&splpage=murl. Murl Engine logogram, source. Accessed: 2014-10-15.

[104] Spraylight GmbH. www.spraylight.at. Spralight official site. Accessed: 2014-10-15.

[105] stack exchange inc. http://stackoverflow.com/questions/14450449/monogame-loading-a-png-file. MonoGame, Information on solving problem when loading png files. Accessed: 2014-10-18.

[106] Superbest. http://pastebin.com/Wt4F321m. MonoGame, XNA camera implementation example code. Accessed: 2014-10-17.

[107] Thalo LLC. http://news.doddleme.com/news-room/rim-lures-app-makers-with-marmalade-software-2/attachment/blackberry-rim-marmalade-logo/. Marmalade logogram, source. Accessed: 2014-10-18.

[108] The Apache Software Foundation. http://ant.apache.org/bindownload.cgi. Apache Ant Software Tool, download page. Accessed: 2014-10-15.

[109] The Apache Software Foundation. http://murlengine.com/tutorials/de/tut0100_hello_world.php. Murl Engine, Hello World Tutorial. Accessed: 2014-10-15.

[110] The Eclipse Foundation. http://www.eclipse.org/downloads/. Eclipse IDE download page. Accessed: 2014-10-13.

[111] ToDroid. http://www.todroid.com/android-gdx-game-creation-part-i-setting-up-up-android-studio-for-creating-games/. Logogram of Badlgic Games, source. Accessed: 2014-10-13.

[112] ToxSick Productions. http://www.toxsickproductions.com/libgdx/how-to-setup-your-libgdx-project/. Information on how to set up a libGDX project. Accessed: 2014-10-14.

[113] UBM Tech. http://www.gamasutra.com/view/news/169846/Mobile_game_developer_survey_leans_heavily_toward_iOS_Unity.php#.UFze0I1lTAE. Article on Unity being elected best mobile device game engine. Accessed: 2014-10-17.

[114] Unify Community. http://wiki.unity3d.com/index.php?title=UnityScript_versus_JavaScript. Unity, Difference between UnityScript and JavaScript. Accessed: 2014-10-17.

[115] Unity Answers. http://answers.unity3d.com/questions/292620/scale-gui-texture-to-current-screen-size.html. Unity, Information on Scaling GUI Texture to current screen size. Accessed: 2014-10-17.

[116] Unity Answers. http://answers.unity3d.com/questions/9729/how-can-i-display-a-flat-background-2d-image-not-a.html. Unity, Information on how to display a 2D background behind all models of the scene. Accessed: 2014-10-17.

[117] Unity Community. http://answers.unity3d.com/questions/12909/pro-versus-free.html. Unity Answers, Pro versus Free. Accessed: 2014-11-17.

[118] Unity Technologies. http://docs.unity3d.com/Manual/AssetBundlesIntro.html. Unity, Information on AssetBundle. Accessed: 2014-11-17.

[119] Unity Technologies. http://docs.unity3d.com/Manual/AssetServer.html. Unity Documentation, Asset Server. Accessed: 2014-10-17.

[120] Unity Technologies. http://docs.unity3d.com/Manual/HOWTO-importObject.html. Unity Documentation, on importing models. Accessed: 2014-10-17.

[121] Unity Technologies. https://store.unity3d.com/products/pricing. Unity, License pricing. Accessed: 2014-11-17.

[122] Unity Technologies. http://unity3d.com/learn/tutorials/modules/beginner/platform-specific/pinch-zoom. Unity, Camera Pinch Zoom Tutorial. Accessed: 2014-10-17.

[123] Unity Technologies. http://unity3d.com/public-relations. Unity, democratizing development. Accessed: 2014-10-17.

[124] Unity Technologies. http://unity3d.com/unity. Unity official website. Accessed: 2014-10-12.

[125] Unity Technologies. http://unity3d.com/unity/download. Unity, Download page. Accessed: 2014-10-17.

[126] Unity Technologies. http://unity3d.com/unity/licenses. Unity, Licenses comparison. Accessed: 2014-11-17.

[127] Unity3D Forum. http://forum.unity3d.com/threads/opengl-4.152867/. Unity, Information on Mac not supporting OpenGl 4. Accessed: 2014-10-18.

[128] Wikimedia Commons. http://commons.wikimedia.org/wiki/File:MonoGame_Logo.svg. MonoGame logogram, source. Accessed: 2014-10-17.

[129] Wonderful Engineering. http://wonderfulengineering.com/35-hd-galaxy-wallpapers-for-free-download/. Reference for the game's background picture, galaxy. Accessed: 2014-10-12.

[130] Xamarin Inc. http://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_0_-_overview/. Xamarin, Building Cross Platform Applications Overview, information on Xamarin game apps better with MonoGame. Accessed: 2014-10-17.

[131] Xamarin Inc. http://developer.xamarin.com/guides/ios/getting_started/installation/device_provisioning/. Xamarin, Information on the Provisioning of iOS devices. Accessed: 2014-11-15.

[132] Xamarin Inc. http://forums.xamarin.com/discussion/11129/going-3d. Xamarin, Information on MonoGame and Going 3D with Xamarin. Accessed: 2014-10-17.

[133] Xamarin Inc. http://forums.xamarin.com/discussion/2367/your-app-is-too-large-to-be-developer-with-xamarin-starter-edition-for-employeedirectory. Xamarin, Information on Xamarin Business Edition 30 days Trial. Accessed: 2014-10-18.

[134] Xamarin Inc. http://rbwhitaker.wikidot.com/monogame-accessing-the-xna-content-pipeline. Xamarin, MonoGame Content Pipeline and MonoGame Definition. Accessed: 2014-10-17.

[135] Xamarin Inc. http://xamarin.com/about. Xamarin, Information on the Xamarin Team. Accessed: 2014-10-17.

[136] Xoppa. http://blog.xoppa.com/basic-3d-using-libgdx-2/. Xoppa tutorial, basic 3d tutorials using LibGDX. Accessed: 2014-10-15.

[137] Xoppa. http://blog.xoppa.com/loading-models-using-libgdx/. Loading models using LibGDX Tutorial by Xoppa. Reference for the 3D model of a ship. Accessed: 2014-10-12.

[138] Xoppa. http://blog.xoppa.com/loading-models-using-libgdx/. Xoppa tutorial, Loading models using LibGDX. Accessed: 2014-10-15.

[139] Xoppa. http://blog.xoppa.com/loading-models-using-libgdx/. libGDX, Model Loading Tutorial - warning considering space ship model not working on other applications. Accessed: 2014-10-17.

[140] Zoom dinosaurs. http://badlogicgames.com/forum/viewtopic.php?f=11&t=11464&p=51607&#p51550. Badlogic games forum - Discussion on Android sound being unreliable. Accessed: 2014-10-15.

[141] Zoom dinosaurs. http://badlogicgames.com/forum/viewtopic.php?f=11&t=11786. Information on the use of SpriteBatch in mixed 2D and 3D rendering, libGDX. Accessed: 2014-10-15.