Diploma in Software Development

DSE 730 Mobile Application Development

Assessment: Project Report

Total marks: 100

Course Weighting: 100%

Due Date: Monday, 3rd September, 2018

Student Name(s): Pavel Sobolev

EXECUTIVE SUMMARY

During last decade development of mobile applications became a significant filed in modern industry of software engineering. Mobile devices have reached rather high technical and computational abilities almost in any aspect of possible areas of their practical usage. This fact made possible implementation of high-demanding calculations concerned with computer graphics. Modern mobile devices capable to build complicated and animated 3D-scenes which can include thousands of inner shaders, vertices, colours, textures and son on.

Necessity of implementation of sophisticated graphical drawings on the screens of mobile devices entailed emerging of special "mobile" versions of popular graphical libraries. There are two prevailing technologies of accelerated computer graphics which directly utilize graphical equipment abilities and demonstrate high performance: Open graphical library (OpenGL) (KHRONOS GROUP, 2018) and DirectX (Microsoft Corporation, 2018).

DirectX is the proprietary technology for creation of high performance multimedia applications which is primarily targeted for using on Microsoft operating systems and devices.

Vice versa, OpenGL is an open source project and cross platform set of technologies which can be used by wide range of operating systems and hardware platforms which is continuously developing (created in 1992 by "Silicon graphics"). OpenGL has several versions for using in different environments of different types of computing devices. Concerning the topic of this paper it is important to notice that Khorons group provides version for mobile and embedded devices which is called OpenGL ES (OpenGL for embedded systems) (KHRONOS GROUP, 2018).

Figure 1: OpenGL and OpenGL ES logos *(KHRONOS GROUP, 2018)*

The principal aim of the project is to create mobile application for Android operating system (Google LLC, 2018) which uses basic possibilities of "OpenGL | ES" API for implementation of graphical animations of 2D and 3D types. Also, application which is planned for development, should use data being provided by the sensors of mobile device in order to realize live interactivity.

Introduction

Aim of the application

The name of application is "K-OGL" (which stands for Kotlin-OpenGL ES). The aim of the application is to show a user set of 2D and 3D animated graphical scenes and to provide him with ability of controlling these scenes and interact with them by using standard graphical interface of Android operating system and built-in sensors of used mobile device. The application is created using native Android API and Kotlin programming language. Table 1 shows list of requirements.

Another important set of goals lies in the area of personal interests of the author. Namely the following personal goals can be listed:

1. to learn how to implement principal ideas of software architecture patterns;
2. to apply in practice the ideas of SDLC;
3. to study how to implement ongoing testing activities for creating of quality software;
4. to learn creating of mobile applications for Android operating system;
5. to study Kotlin programming language and to use this language in practice;
6. to study "OpenGL ES" API and use it for practical application in creation of 2D and 3D interactive graphical scenes on the screen of mobile device;
7. to study how to program interaction with user of mobile device with the usage of the device's sensors.

Table 1: application requirements

Requirement code	Description
GUI requirements (GUIR) according to Google guidline *(Google LLC, 2018)*	
GUIR-1	Material design conventions should be implemented (includes color schemes, appearance of icons and controls).
GUIR-2	The software should contain controls (standard main menu with buttons) for providing a user with ability to control behavior and appearance of the graphical scenes.
GUIR-3	The software should follow global look and feel of operating system GUI which is set by the user.
GUIR-4	The software supports only portrait orientation of the screen of a device.
GUIR-5	The software should have appropriate icon for representation on the screen of used system launcher.
Functional requirements (FR)	
FR-1	The software should render two types of graphical scenes (flat scene and three-dimensional scene) on the base of utilization of abilities OpenGL ES API.
FR-2	Flat scene should render animation which consists of three moving objects which are: a square, a triangle, and a spiral.

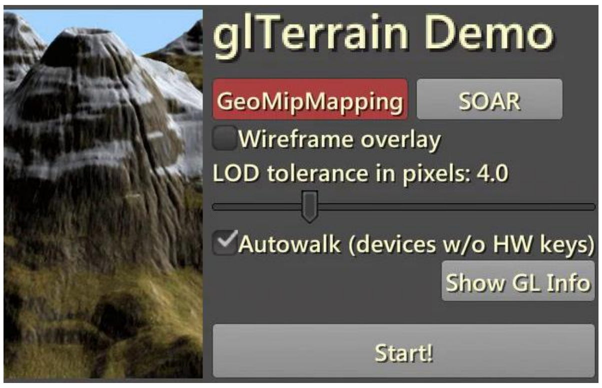
FR-3	Each object of flat scene should have its own way of coloring of vertexes (changing within runtime) and its own rotation angle.
FR-4	The square and the triangle should initially move along the line of spiral in the opposite directions. After tilting of device in the "tilt mode" direction can be changed arbitrarily.
FR-5	After reaching of extreme points of the spiral the objects should change direction of their moving to the opposite.
FR-6	Three-dimensional scene should consist of five textured objects which are: 2 regular hexahedrons (cubes), 2 regular tetrahedrons (pyramids), one combined polyhedron (union of a cube and a pyramid).
FR-7	Images of textures are loaded from files of PNG and JPEG formats from inner resource of the application.
FR-8	Each body should rotate around its central point.
FR-9	Three-dimensional scene should provide movable light source for producing of realistic view.
FR-10	The software should provide a user with ability to choose type of output graphical scene (2D or 3D) by tapping of appropriate button in the main menu.
FR-11	The software should provide a user with ability to start or stop animation of 3D scene by tapping of appropriate button in the main menu.
FR-12	The software should provide a user with ability to start or stop processing of changing of device tilt in the space by tapping of appropriate button in the main menu.
FR-13	The software should provide a user with ability to change the way of processing of screen touches by tapping of appropriate button in the main menu. Two possible modes of rotation of the entire scene should be provided: around vertical axis or around horizontal axis.
FR-14	The software should provide a user with ability to change the characteristics of 3D scene (distance to 3D-objects) by tapping of appropriate button in the main menu and using of special customization window.
FR-15	The software should provide a user with ability to change the appearances of the objects of 3D scene (surface textures) by tapping of appropriate button in the main menu.
FR-16	The software should provide a user with ability to change the direction of moving of objects of 2D-scene by tilting the device.
FR-17	The software should provide a user with ability to change the direction of rotation of objects of 3D-scene by tilting the device.
Non-functional requirements (NFR)	
NFR-1	Graphical scenes should look similarly on the screens of mobile devices with different versions of Android operating system.
NFR-2	Application should provide graphical scenes which adapt themselves to different screens resolutions and sizes.

Technologies used

Software “K-OGL” was crated for functioning int the executive environment of Android operating system and has standard graphical user interface of this operating system. The application forms interactive graphical scenes of two types: flat scene (2D) and virtual volume scene (3D). User can control scenes by using standard Android’s graphical interface controls, screen touch gestures and by tilting the device in arbitrary direction.

Availability of the same or similar applications on other platforms

Analogical applications can be found in the standard application repository of Android OS which is called "Google Play Store". For example, all the Android interactive 3D-games are created with "OpenGL ES", but cannot be called analogical because of their significant complexity (this level of complexity cannot not be reached in this project because of time limitations). Used criteria of similarity is that analogical applications should have demonstrative type and form illustrative graphical scenes with the aid of OpenGL ES technology. At least three analogical applications can be found in the Google play store (actually many more than three): "glTerrainDemo" (created by Marek Mauder) and "OpenGL experiment" (created by Enthusiast Studio), "OpenGL ES 1.0 Demo" (created by RT Software Studio). Table 2 shows result of comparison on the basis of contrasting of implemented functionality.



Wireframe overlay

LOD tolerance in pixels: 4.0

✓ Autowalk (devices w/o HW keys)

Figure 2: glTerrainDemo settings window

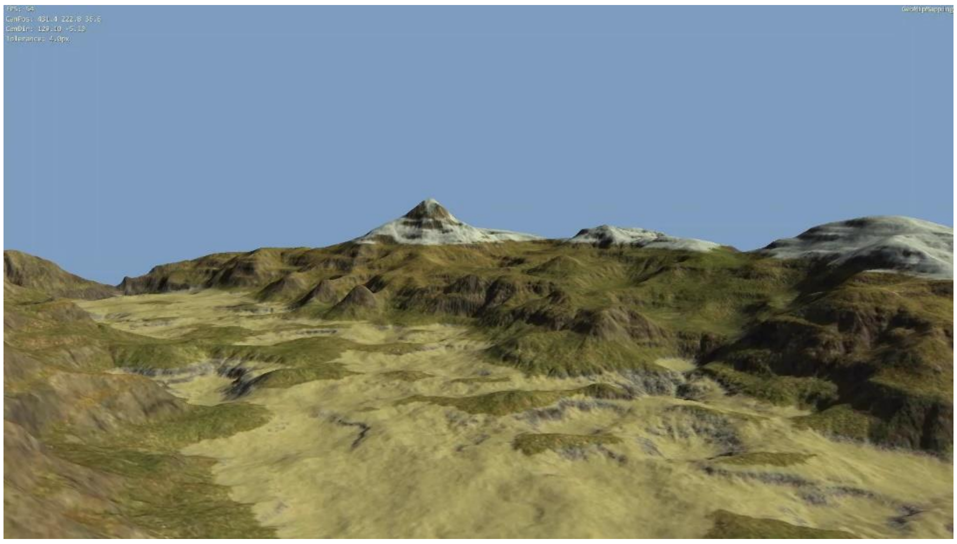


Figure 3: "glTerrainDemo" main activity with calculated virtual terrain

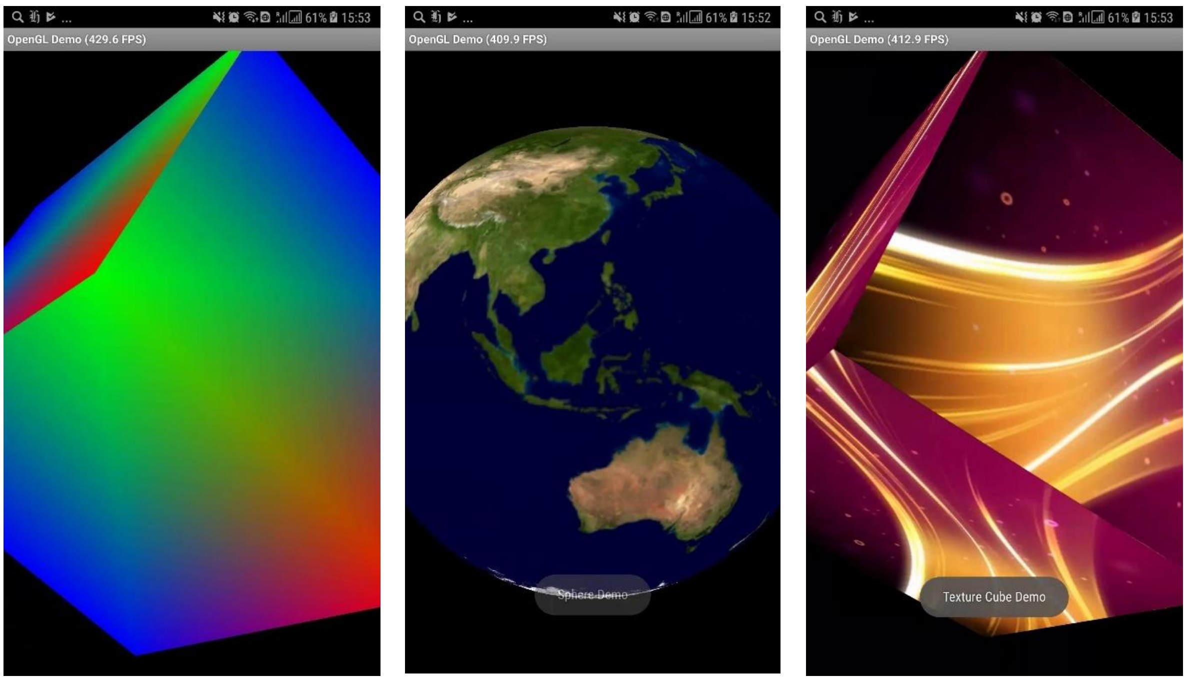


Figure 4: examples of scenes generated by "OpenGL ES 1.0 Demo" application

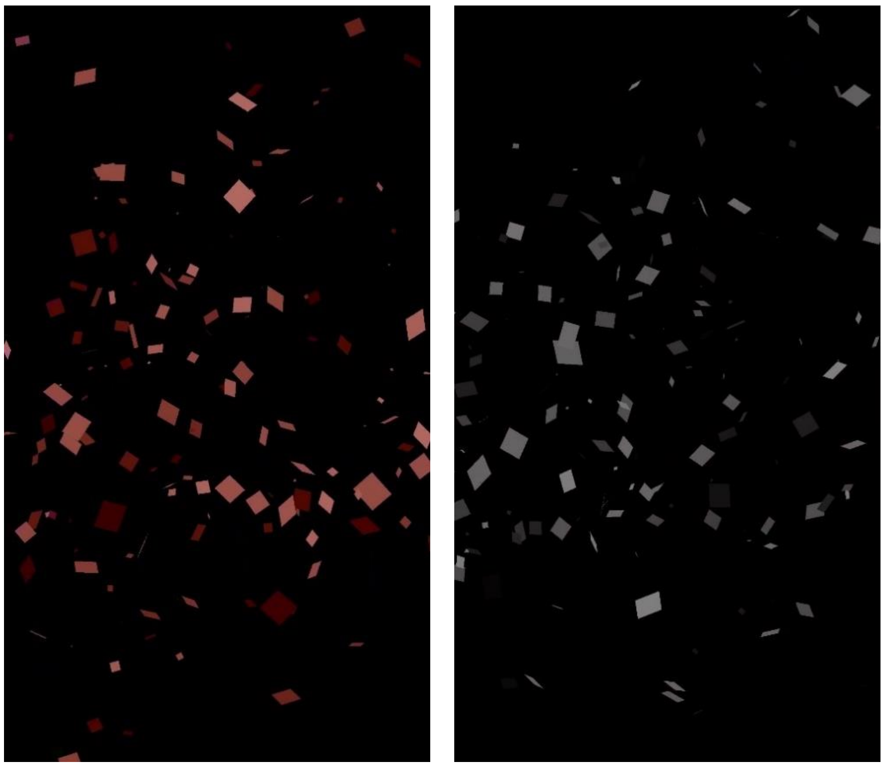


Figure 5: examples of scenes generated by "OpenGL experiment" application

Table 2: comparison of analogical applications

Implemented features	K-OGL	glTerrainDemo	OpenGL ES 1.0 Demo	OpenGL experiment

OpenGL key features are demonstrated	Yes	Yes	Yes	Yes
Standard Android material design and interface	Yes	No	No	No
Controlling of scenes by setting window	Yes	Yes	No	No
Using of touches and gestures for controlling the scenes	Yes	Yes	Partially	No
Using of the device's accelerometer	Yes	No	No	No
Using of custom textures	Yes	No	Yes	No
Using of OpenGL coordinate system's transformations	Yes	Yes	Yes	Yes
Demonstration of 2D scene	Yes	No	Yes	No
Demonstration of 3D scene	Yes	Yes	Yes	Yes
Using of OpenGL lighting	Yes	Yes	No	Yes
Using of OpenGL color interpolation	Yes	No	Yes	Yes
Calculation of OpenGL normal vectors	Yes	Yes	No	Yes

Project Planning and Execution

Project Plan and Gantt Chart

Table 3: project plan in the form of Gant chart

#	Ativities within the project / Dates	Mon, 30, Jul	Tue, 31, Jul	Wed, 1, Aug	Thu, 2, Aug	Fri, 3, Aug	Sat, 4, Aug	Sun, 5, Aug	Mon, 6, Aug	Tue, 7, Aug	Wed, 8, Aug	Thu, 9, Aug	Fri, 10, Aug	Sat, 11, Aug	Sun, 12, Aug	Mon, 13, Aug	Tue, 14, Aug	Wed, 15, Aug	Thu, 16, Aug	Fri, 17, Aug	Sat, 18, Aug	Sun, 19, Aug	Mon, 20, Aug	Tue, 21, Aug	Wed, 22, Aug	Thu, 23, Aug	Fri, 24, Aug	Sat, 25, Aug	Sun, 26, Aug
1	Task analysis, eliciting of requirements	■	■	■	■	■																							
2	Risk analysis and description				■	■			■	■																			
3	Choice of SDLC model	■	■																										
4	Planning of software architecture			■	■	■																							
5	Choice of technologies	■	■																										
6	Learning of OpenGL ES			■	■	■			■	■	■	■	■																
7	Learning of Kotlin programming language			■	■	■			■	■	■	■	■																
8	Design and planning of OpenGL animation		■	■	■	■																							
9	Implementation of software								■	■	■	■	■			■	■	■	■	■			■	■					
11	Testing activities				■	■			■	■	■	■	■			■	■	■	■	■			■	■					
12	Working on report																						■	■	■	■	■		

Risk Management

Risk management is a methodology which allows developers to elicit possible risks that can happen during the process of development and, thus, be prepared in the case of their emerging. Three parts of risk management can be called: assessment of possible risks, mitigation of risks which happened or precautions for prevention or mitigation, and risk costs evaluation (Stoneburner, Goguen, & Feringa, 2002).

On the base of analysing of sources (Arnuphaptrairong, 2011; Wallace & Keil, 2004) all software project risks can be categorised by so called risk dimension, which defines the perspective of view for determining source of possible emerging risks. According to mentioned authors 6 risk dimensions can be elicited: risks coming from users or customers; risks connected with software requirements; risks coming from project complexity; risks from project planning and control; risks coming from the project team; risks coming from organizational environment; risks coming from the used equipment and software. Result of the analysis is presented on the following list.

1. Users/customers risks:
    a. Lack of cooperation;
    b. Resistance to changes;
    c. Negative attitude toward the project.
2. Requirements risks:
    a. Continual changings;
    b. Not enough adequate identification;
    c. Unclearness and haphazardness;
    d. Incorrect formulating.
3. Project complexity:
    a. New or immature technologies are used;
    b. High level of technical complexity,

    c. Technologies which was not used before are introducing within currently evolving project.
4. Control and planning risks:
    a. Not sufficiently effective management technologies;
    b. Not sufficiently effective assessment of planned resources;
    c. Not quality project planning;
    d. Milestones are not thoroughly defined;
    e. Not sufficiently effective communication.
5. Risks from development team:
    a. Members are unexperienced;
    b. Members are inadequately trained;
    c. Members can experience lack of specialized skills.
6. Environment risks:
    a. Organizational management has changed;
    b. Negative effect from unsuccessful corporate politics;
    c. Environment is unstable;
    d. Restructuring of organization while ongoing project.
7. Software and hardware risks:
    a. Hardware failures and errors;
    b. Operating systems issues and bugs;
    c. Using of not enough tested technologies, frameworks, programming languages and so on.

Based on studied sources devoted to risk management (Arnuphaptrairong, 2011; Wallace & Keil, 2004; Stoneburner, Goguen, & Feringa, 2002), 7 possible development risks for "K-OGL" software were detected (table 4).

Table 4: "K-OGL" project risks

#	Possible risk	Possible effect	Precautions for prevention
1	Hardware failure (first of all, data storage system failure).	Possibility to lose project files, failure of projects due dates.	Using of software for control versioning; usage of cloud storages.
2	Operating system failure (due to computer viruses, bugs).		
3	Mobile device GPU issues.	Malfunction in the software while rendering of graphical 3D scenes (OpenGL errors can emerge).	Handling of errors of OpenGL finite state machine and exceptions.
4	Destruction or damaging of data in persistent internal storage of Android.	The application can crash during the process of starting of execution.	Checking existence of data before reading them from persistent data storage. Setting the default values for characteristic of the scene in the case of data loss.

5	Lack of experience of developing using OpenGL ES version.	Failure of due date. Creation of graphical scenes with inefficient implementation or not meeting the requirements.	Thorough reviewing of abilities and peculiarities of new technologies from information sources and official documentation.
6	Lack of experience of developing using Kotlin programming language.	Creation of not sufficiently effective implementation in the sense of inner code structure.	Thorough reviewing of abilities and peculiarities of new technologies from information sources and official documentation.
7	Not correct time management during the project implementation.	Creation of non-quality software which doesn't meet the requirement. Failure of the project's due dates.	Following the project plan (see previous section).

Software Development Lifecycle

Software development lifecycle is a process which controls both overall and concrete activities within the process of creation of software product. According to studied sources (Langer, 2016; Pressman, 2001) taking of decision about model of SDLC to be used in some project is not easy decision and it should take in consideration a lot of different aspects. But some concrete considerations can be listed according to the mentioned above sources and existing definitions and descriptions of modern models of SDLS being used in the software engineering. Concerning "K-OGL" project set of questions were compiled and on the base of the prevailing answers final SDLC model was chosen.

Table 5: comparison of SLDC models for "K-OGL" project

Characteristic of the project / Does model correspond this characteristic?	Waterfall	V-model	Spiral
All requirements are well known and easily detectable.	✓		
All the requirements were defined preliminary.	✓		
Requirements are not be changeable.	✓		
There is no need to implement some requirements on early stages of the project despite quality of their implementation.	✓		
Mobile development is not a new topic for the developer of the project.	✓		
Tools of development relatedly new to the developer (Kotlin programming language and OpenGL ES computer graphics technology).		✓	✓

There is not team working on the project. Changing of roles is not applicable.	✓	✓	
It is possible to get some additional training for the project developer during the process of development.	✓	✓	✓
Users are not going to be involved in the process of development.	✓		
Possible users can evaluate current condition of the project.		✓	✓
Users do not participate in each phase of project SDLC.	✓	✓	
Users are not observing the process of development.	✓		
This project isn't middle or large scaled project.	✓		
The project doesn't extend existing projects.	✓	✓	
This project is not expected to be exploited for a long period of time.	✓		
Ready program is not expected to be changed after finishing of the project.	✓	✓	
The project has strict time limitation.	✓	✓	
Created components of the projects are not planned for reusability.	✓		
There are enough resources for the development of the project (time, development tools).	✓	✓	

So, according to undertaken analysis waterfall model of SDLC is the most appropriate for this project.

Architecture and Design

Alternative architectural designs considered

Architectural design (Mallawaarachchi, 2018) is a crucial thing for medium and large scaled software projects, but even in learning projects it is very important to follow effective and widely used practices concerning the structure and inner logic of application being created. Preliminarily planned usage of software architectural patterns and software design patterns provide the application with quality attributes such as reusability, scalability, maintainability and so on (Liu, 2009).

1 "Singleton" design pattern

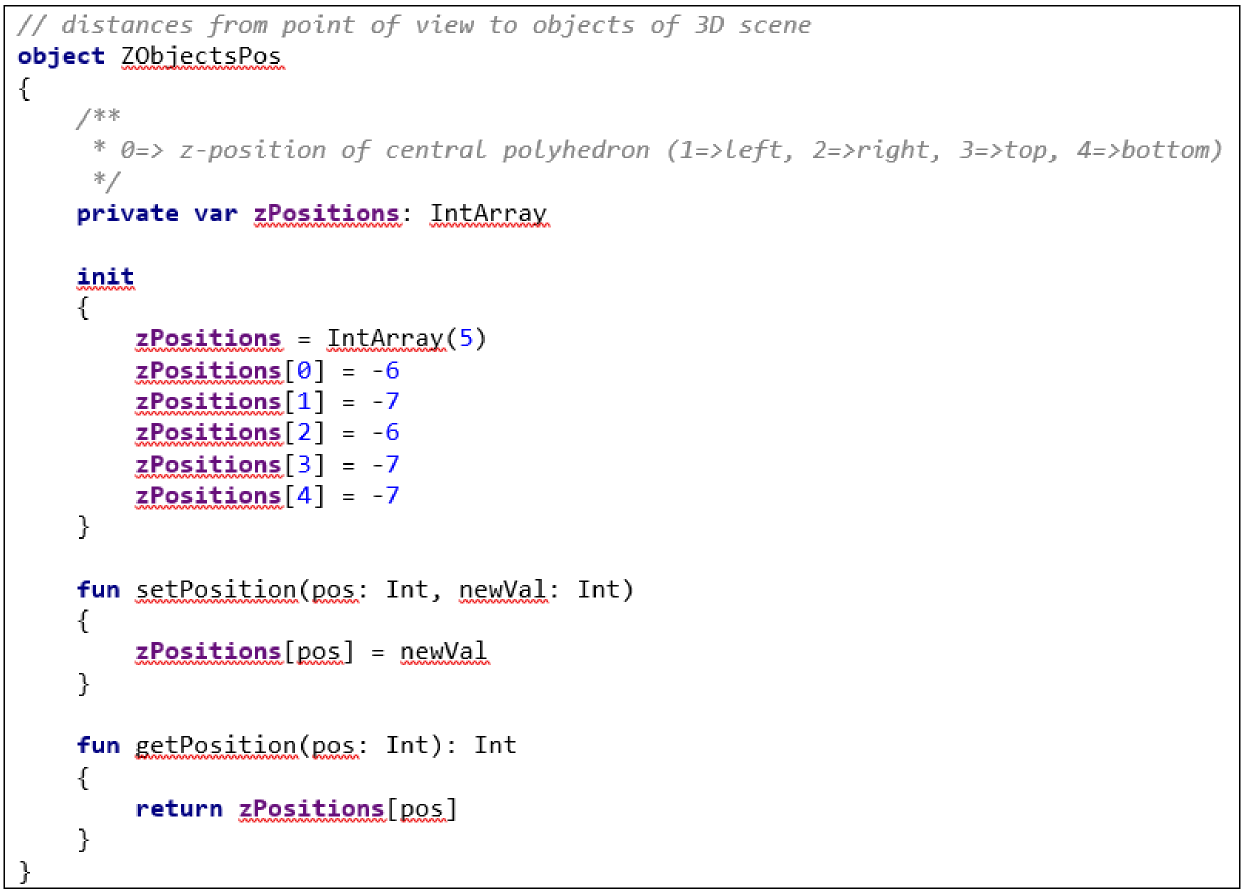
While planning the logical structure some design patterns were utilized. For the goals of centralization of usage of important data singleton pattern was implemented. Singleton (Booch, 1991) is a design pattern of object-oriented programming which allows the application to create and use only one instance of some class (Table 6).

Table 6: singleton classes used in the project "K-OGL"

Name	Stereotype	Description
TiltData	object/singleton	TiltData provides client classes with information which represents data given by device's accelerometer. This information is provided in the "consumable" form (in the sense of 2D and 3D graphical OpenGL scenes).
ZObjectsPos	object/singleton	ZObjectsPos provides consumers with data which is necessary for rendering of objects of 3D scene (distance from each object to the point of view of virtual observer (or virtual camera).
ShaderSource	object/singleton	ShaderSource provides consumer classes of OpenGL subsystem with source code of shader programs. Shader program is independent executable code which is executed by GPU to calculate each vertex in the graphical scene (colour, 3D-position, texturing, lighting, culling and so on). Source codes stored in raw text files in the project resources and loaded dynamically from this class by demand. After that these codes are delivered to OpenGL API which compile and link them during the program execution.
VertexSource	object/singleton	VertexSource provides consumer classes of OpenGL subsystem with initial coordinates of 3D-scene's objects (cubes, pyramids and their combinations).

Kotlin programming language (Moskala & Wojda, 2017) provides programmers with built-in implementation of this pattern which is called “object”. Example of declaration is shown on listing 1 (extraction from “K-OGL” source code). Object allows to declare usual class which is used as singleton and can be used everywhere in the project by referencing its name. Operation of instantiation of “object” class is prohibited. Inheritance for this kind of class is not supported.

Listing 1: example of singleton object declaration in Kotlin



```
// distances from point of view to objects of 3D scene
object ZObjectsPos
{
    /**
     * 0=> z-position of central polyhedron (1=>left, 2=>right, 3=>top, 4=>bottom)
     */
    private var zPositions: IntArray

    init
    {
        zPositions = IntArray(5)
        zPositions[0] = -6
        zPositions[1] = -7
        zPositions[2] = -6
        zPositions[3] = -7
        zPositions[4] = -7
    }

    fun setPosition(pos: Int, newVal: Int)
    {
        zPositions[pos] = newVal
    }

    fun getPosition(pos: Int): Int
    {
        return zPositions[pos]
    }
}
```

Usage of singleton pattern provides application being created with the global centralized source of immutable objects which are used from other classes throughout the entire project. This approach gives some benefits. First of all, it allows to avoid re-creation of objects which are designated for solving of one task but are needed in different classes. This significantly simplifies the code and support loose coupling of application modules and contribute reducing of memory consumption. Possible slight drawback of singleton usage (Emmatty, 2018) is that such classes are not suitable for inheritance. But in the case of this project this cannot be considered as disadvantage because inheritance for project’s singletons is not needed.

Advantage of using singletons in the “K-OGL” project is that it gives ability to implement “client-server” architectural pattern. This type of architecture allows sharing common data between different classes of application. Singletons can be considered as servers and other interested classes as their clients. This architecture allows to concentrate particular type of responsibilities in one class and to reduce the level of coupling between modules (Flowler, 2013).

2 “Observer” design pattern

During the process of project development some problems had emerged, and they related to way of passing data between “activities” while processing events of accelerometer of mobile device. The accelerometer was used to provide user with ability to change the direction of rotation of bodies placed within virtual 3D-scene. Each change of accelerometer data (which signal that a device was given some tilt) implies passing of the following numbers:

1. value of shift along each of three axes (x-shift, y-shift, z-shift),
2. new x-coordinate of rotation axis,
3. new y-coordinate of rotation axis,
4. new z-coordinate of rotation axis,
5. new rotation angle calculated on the base of new and old data of tilt.

Originally designed direction of data flow between objects of the application is shown on figure 13 (arrows show direction of dataflow). This architecture entails from Android application structure demands and OpenGL API interoperability implementation. Instance of SpaceGLRenderer class is responsible for final drawing of the surface of the active view.

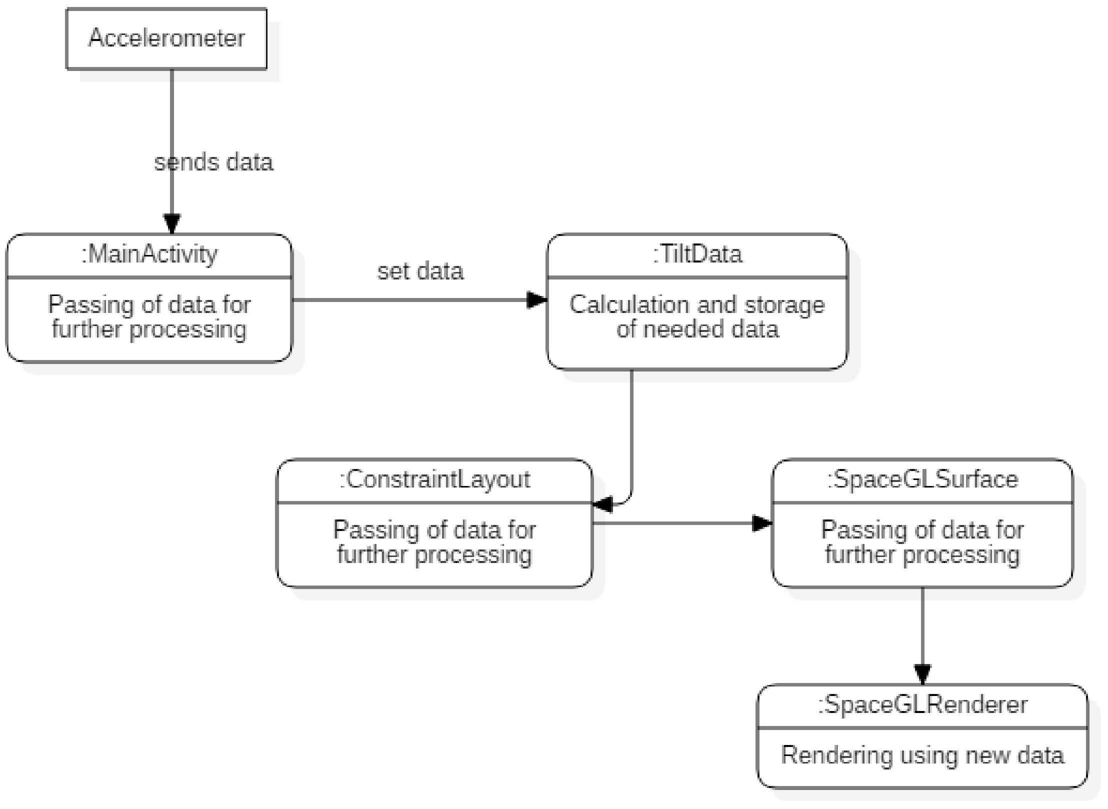

Accelerometer
sends data
:MainActivity
Passing of data for further processing
set data
:TiltData
Calculation and storage of needed data
:ConstraintLayout
Passing of data for further processing
:SpaceGLSurface
Passing of data for further processing
:SpaceGLRenderer
Rendering using new data

Figure 6: dataflow diagram in original application architecture

Implementation of such a long journey of data leads to the useless consumption of memory and creation of code which do same thing many times and involves creation of several Intent classes. Also, it increases volume of code, makes code cumbersome because we have to pass the same data to many classes. At the same time this type of realization has some benefits because makes code easier to understand and easer to maintain.

After some analysis of the problem changes of architecture were undertaken. Namely, “observer” design pattern was utilized. “Observer” is an object-oriented design pattern which allows to create objects which can notify other interested objects of the application about changes which happened within the runtime (Rizwan, 2014).

Technically introducing of “observer” pattern was done by using built-in abilities of Java class library (see figure 7). TiltData class was implemented as descendant of “Observable” class. Interested classes (SpaceGLSurface and MyGLSurface) were rewritten as implementations of Observer interface. Also, TiltData was reimplemented as singleton class because this class the only source of information about tilt which was got by the device. Thus, classes of renderers can use actual data in TiltData without getting them from Intents and arguments of methods. Also, important to mention that this approach contribute loose coupling between modules of the application (Booch, 1991).

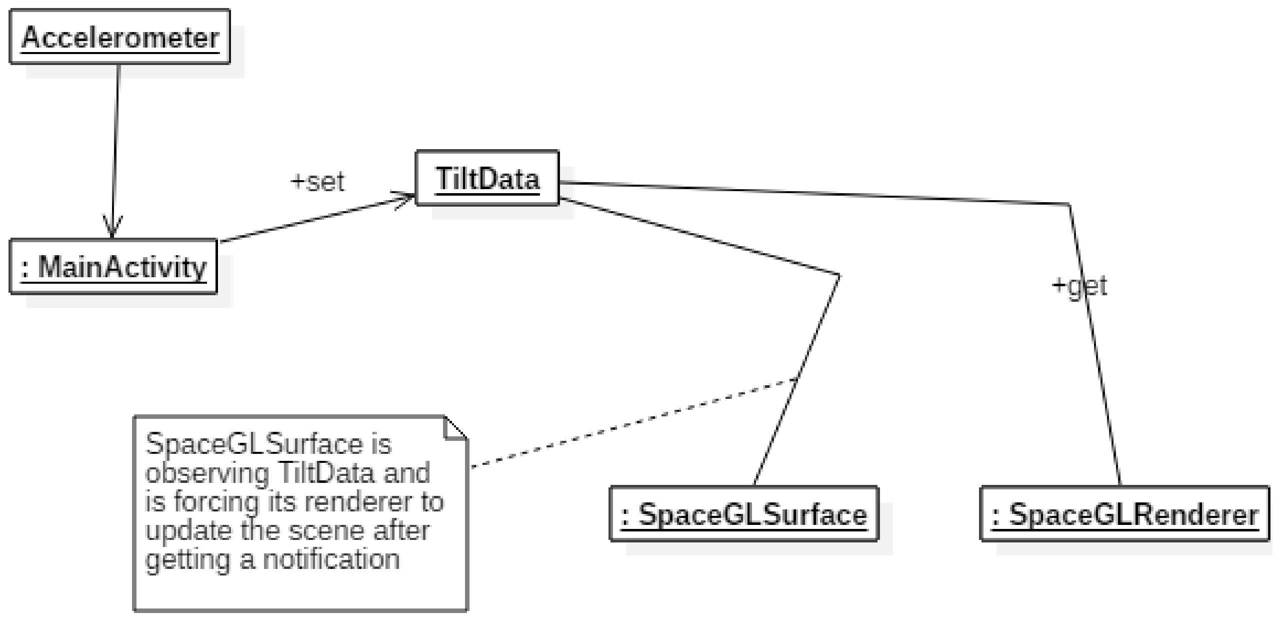

Accelerometer
+set
TiltData
: MainActivity
+get
SpaceGLSurface is observing TiltData and is forcing its renderer to update the scene after getting a notification
: SpaceGLSurface
: SpaceGLRenderer

Figure 7: architecture of application after introducing of “observer” pattern

This approach also contributes of implementing of so called “event-bus” architectural pattern, which provides the application with event-driven custom mechanism which consists of the source of event (the observable class in this project), event listener (methods which are written in the observer classes), channel and event bus (which are provided by executable mechanisms of Java Virtual Machine of Android operating system).

So, usage of observer pattern allows to implement efficient way of data interchange (because data is stored in centralized storage) in appropriate moments of time. Disadvantage of this pattern is that it can creates nonobvious relationships between objects and introduce some difficulties in code understanding. Thus, it is important to thoroughly document the project and to describe relations between classes for its future maintenance (Buschmann, 2013).

Reasons for your choice

According to previous discussions and studied sources (Rizwan, 2014; Booch, 1991; Mallawaarachchi, 2018) following table was compiled (table 4). It shows what reasons was taken in consideration while choosing this or that way of software constructing. The main idea was to determine how concrete design or architectural pattern positively influences future quality attributes of application being created.

Table 7: reasons to choice of architectural and design patterns described above

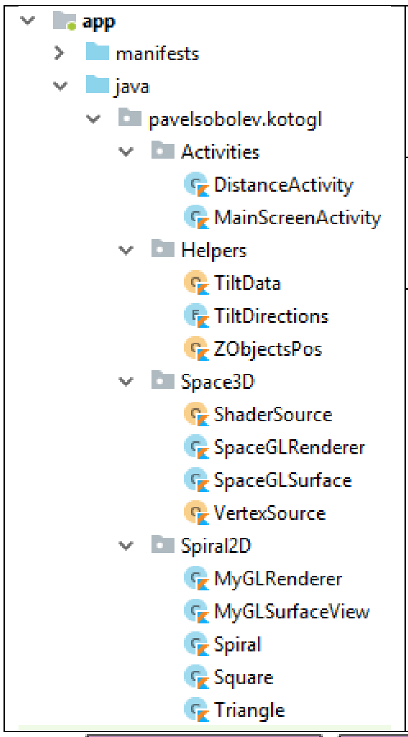
Chosen architectural decision / Contributes quality attributes	Loose coupling	Cohesion	Maintainability	Reusability
Usage of Kotlin modules		✓	✓	✓
Usage of Observer pattern	✓	✓	✓	✓
Usage of singletons (Kotlin objects)	✓	✓	✓	✓

Overall architecture of the final design. i.e., activities, services, intents

Final architecture of application is presented on the figure 6 and in the table 5. Final architecture contains packages, classes, objects (in the terms of Kotlin programming language object is immutable static class which is corresponds singleton pattern).

Table 8: Description of the application structure

	Package name	Package description
	Activities	The package contains classes which represent activities of application.



```
app
  manifests
  java
    pavelsobolev.kotogl
      Activities
        DistanceActivity
        MainScreenActivity
      Helpers
        TiltData
        TiltDirections
        ZObjectsPos
      Space3D
        ShaderSource
        SpaceGLRenderer
        SpaceGLSurface
        VertexSource
      Spiral2D
        MyGLRenderer
        MyGLSurfaceView
        Spiral
        Square
        Triangle
```

Helpers	The package contains classes for implementing service functionality which is used by all other classes of application.
Space3D	The package contains classes which implement virtual 3D scene.
Spiral2D	The package contains classes which implement flat scene.

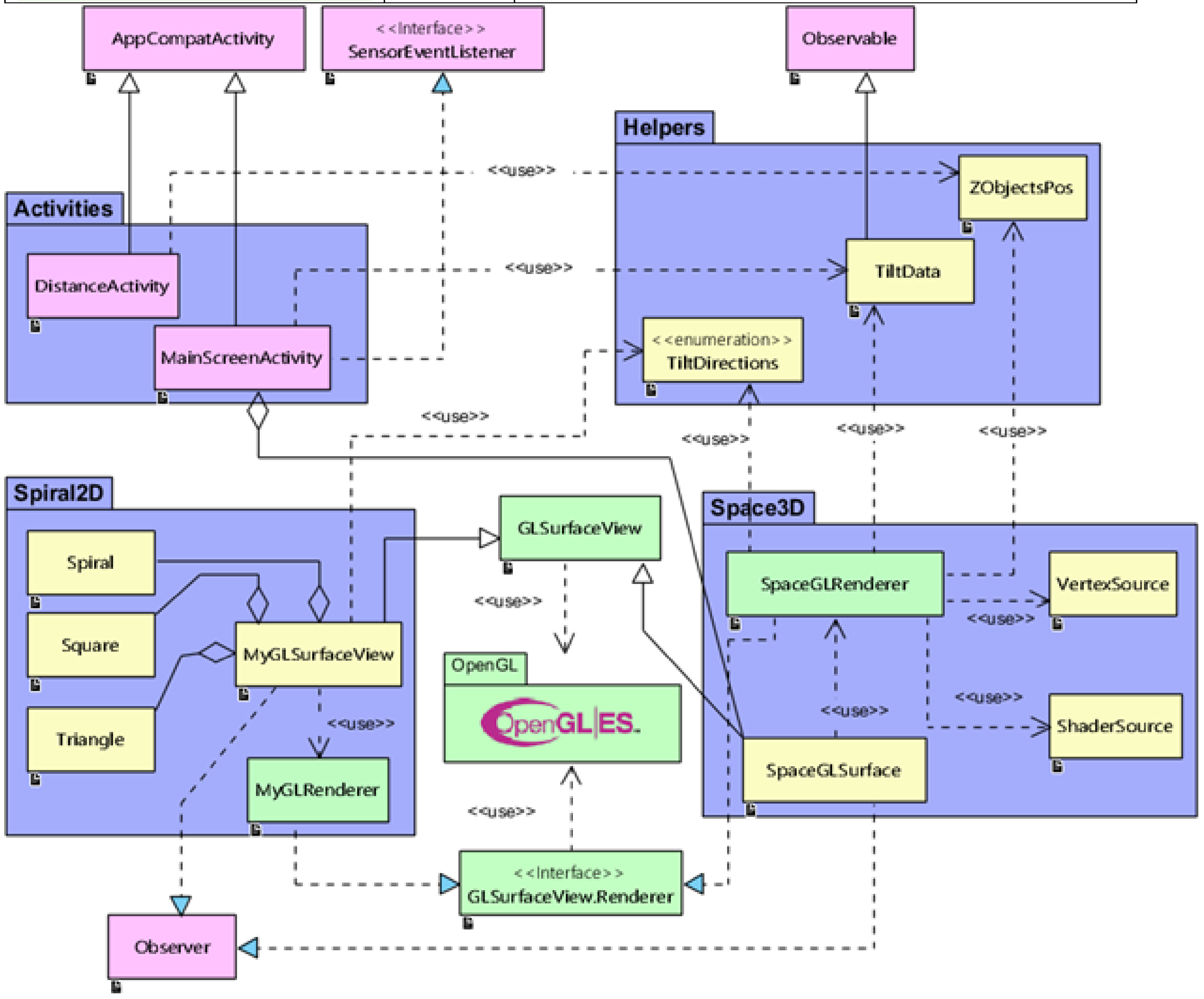


Figure 8: class diagram

Table 9: Description of classes

Name	Stereotype	Inherits/implements	Description
DistanceActivity	class	AppCompatActivity/-	Dialog window (activity) for setting distances to the objects of 3D-scene.
MainScreenActivity	class	AppCompatActivity and SensorEventListener/-	Main activity of application with main tool bar.
TiltDirections	enum	-	Enumeration for usage by other classes to determine direction of a tilt got by device from a user.
TiltData	object/singleton	Observable/-	This class encapsulates methods and data for getting and calculating specific characteristics of graphical scenes dependent from the accelerometer data stream.
ZObjectsPos	object/singleton	-	This class contains data about distances of objects of 3D scene from the virtual viewer (located in the point (0,0,0)).
ShaderSource	object/singleton	-	This class provides consumer classes of OpenGL subsystem with source code of shader programs (see description in table3).
SpaceGLRenderer	class	GLSurfaceView.Renderer/-	The renderer class implements functionality for rendering 3D-scene on the connected surface (SpaceGLSurface).
SpaceGLSurface	class	GLSurfaceView/Observer	This class encapsulates functionality for creating a standard Android view capable to accept OpenGL output.
VertexSource	object/singleton	-	This class provides consumer classes of OpenGL subsystem with

			initial coordinates of 3D-scene's objects.
MyGLRenderer	class	GLSurfaceView.Renderer/-	The renderer class implements functionality for rendering 3D-scene on the connected surface (MyGLSurfaceView).
MyGLSurfaceView	class	GLSurfaceView/Observer	This class encapsulates functionality for creating a standard Android view capable to accept OpenGL output.
Spiral	class	-	This class encapsulation of appearance and behavior of spiral of 2D-scene.
Square	class	-	This class encapsulation of appearance and behavior of square of 2D-scene.
Triangle	class	-	This class encapsulation of appearance and behavior of Triangle of 2D-scene.

Discussion of technologies used

The application "K-OGL" implements 2D and 3D geometrical drawings by utilizing powerful graphical API OpenGL (Brothaler, 2013). OpenGL is evolving since 1992 and has wide support of different types of GPUs. Also, OpenGL is called commonly accepted industry standard in the area of computer graphics which has implementations for all major operating systems and hardware platforms. The main strong feature of OpenGL API is that it is capable to implement highly efficient low-level interaction between applications and graphical equipment. This enables to create applications which can draw complicated animated graphical scenes with high level of responsiveness. It is important to notice that OpenGL has special version for embedded and mobile systems which is called OpenGL ES (Ginsburg & Purnomo) and is supported by mobile devices of major vendors of smartphones, such as Google and Apple. OpenGL ES has some consecutive versions and the last version has number 3. The mostly used version of OpenGL ES is 2.0 and this API specification is supported by Android 2.2 (API level 8) and higher. This means that OpenGL ES 2.0 calls is supported by all the smartphones running Android operating system.

But OpenGL ES is not the only option to create quick and effective computer graphics on the screens of mobile devices. Nowadays it is possible to name at least one competitive technology which have the same abilities with OpenGL ES. Advanced Microdevices (AMD) and Chronos group (which also possesses trademark of OpenGL and OpenGL ES) announced Vulkan 3D API in 2015 (Sellers, 2017). This API is to be highly effective real-time library which can be used in the same area of application as OpenGL. First version of Vulkan API was released in 2016 and has support of almost all modern operating systems, such as Linux, Windows, macOS, iOS, Android, Nintendo Switch and Tizen. Vulkan is supported by Google from Android API level 24 or higher. This means that at the moment only about 8% of devices with Android onboard can execute applications which invoke Vulkan API commands.

On the base of analysis of the sources (Brothaler, 2013; Sellers, 2017; KHRONOS GROUP, 2018) comparison of OpenGL ES and Vulkan was undertaken. The goal of the comparison of two APIs is to take the final decision about the choice of one of them.

Table 10: comparison of OpenGL and Vulkan APIs

Feature	OpenGL ES	Vulkan
Finite state machine characteristic	Usage of one global state machine for the entire application (this means that any global characteristic of scene such background color is preserved until it is changed by direct command).	Usage of multiple sets of states for the entire application (it is possible to have multiple states of the same characteristics and apply them when needed).

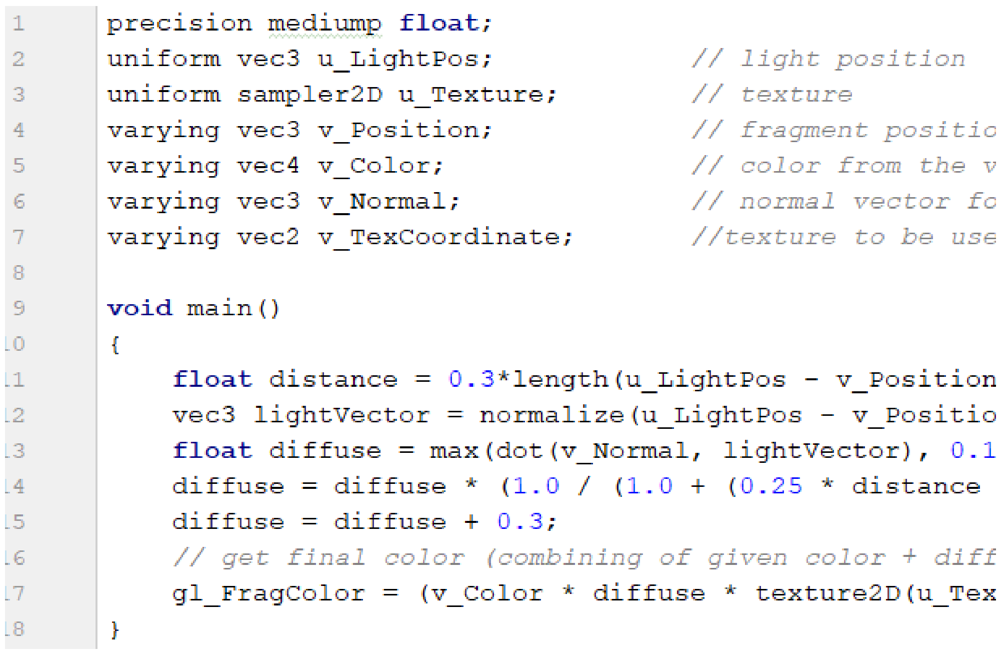
Support of simultaneous execution	Only sequential execution is supported.	Parallelism is supported (multithreading is built-in since the first version).
Way of controlling the interaction with hardware	Implicit way is used.	Explicit way of controlling memory operations and other low-level operations.
Usage of graphical drivers	Installed graphical drivers are used to implement all the operations (no guarantee of predictable efficiency).	Layer of drivers is not used. API directly controls all the GPU operations.

So, obvious *benefits* of Vulkan API are multithreading, support of multiple states and even more low-level (and hence, more effective) mechanisms of scenes rendering. It is said that Vulkan has very "close relationship" with hardware, that characterizes this API as extremely low-level library. The main *disadvantage* is that this library is not so widely supported on all versions of Android operating system. Another drawback is that technology is young and not mature, so created applications can probably experience of unstable or unpredicted functioning. So, my final choice is OpenGL ES API, which has wider support in all versions Android operating system and guaranteed stability. But in the future Vulkan API is probably is to replace OpenGL ES in the area of graphical applications development for mobile and embedded platforms, because this technology was specifically goaled to this area of usage (Sellers, 2017).

Discussion of data organisation

The "K-OGL" project uses input data are texture files, shader program texts, data stream from the device's accelerometer and data about screen touches, made by the user.

Texture is an image which covers the surface of an object of 2D or 3D scenes (Figure *18*: main activity with 2D (a) and 3D (b) scenes). Texture can be retrieved from a graphical file (of some standard format, such as JPEG, PNG, BMP and some others) or can be generated programmatically. "K-OGL" application creates textures from the files which are stored in the resources of the project in specially designated directory (../res/drawable).



```
precision mediump float;
uniform vec3 u_LightPos;              // light position
uniform sampler2D u_Texture;          // texture
varying vec3 v_Position;              // fragment positio
varying vec4 v_Color;                 // color from the v
varying vec3 v_Normal;                // normal vector fo
varying vec2 v_TexCoordinate;         //texture to be use

void main()
{
    float distance = 0.3*length(u_LightPos - v_Position
    vec3 lightVector = normalize(u_LightPos - v_Positio
    float diffuse = max(dot(v_Normal, lightVector), 0.1
    diffuse = diffuse * (1.0 / (1.0 + (0.25 * distance
    diffuse = diffuse + 0.3;
    // get final color (combining of given color + diff
    gl_FragColor = (v_Color * diffuse * texture2D(u_Tex
}
```

Figure 9: example of shader program code

Each modern OpenGL application should use shaders (Figure 9: example of shader program code). Shaders can be described as the instrument for direct interaction between software being created and OpenGL rendering pipeline. Shaders are used to calculate characteristics of any pixel before drawing it on the screen. Shader is created from shader program inside the software during runtime by loading its text into memory. After that compiling and linking are executed inside the application being run. Shader programs are usually stored inside the software as string data consisting of codes written in OpenGL ES Shading programming language. When shader program is long it is expedient to store its text in the separate text file and load in memory for compilation and linking when the software needs to create some shader program. "K-OGL" application uses text files for storing of input shader programs. Each shader program resides in separate file in the specially designated directory of project resources (../res/raw).

Reflection

Development of the "K-OGL" project gave possibility to deepen and to widen my knowledge and practical skills in the area of software architecture. I paid attention to some details which were known but sometimes they were neglected because of not enough diligent planning.

Namely it is important to consider and to plan the usage of some widely known architectural patterns (such as "client-server" or "event-bus" patterns) in order to reduce future necessity to rewrite and reorganize the code of the application. The created software utilizes "observer" design pattern and "singleton" design pattern which contribute overall cohesion and loose coupling within the project.

One of the most interesting things was to learn how to implement chosen architectural ideas in Kotlin programming language which is new for me. Practical learning of such implementation contributes not only to upscaling of knowledge of language. This practice helps to clarify the understanding of the ideas being implemented.

UI Design

Initial screen designs showing layout and user interaction

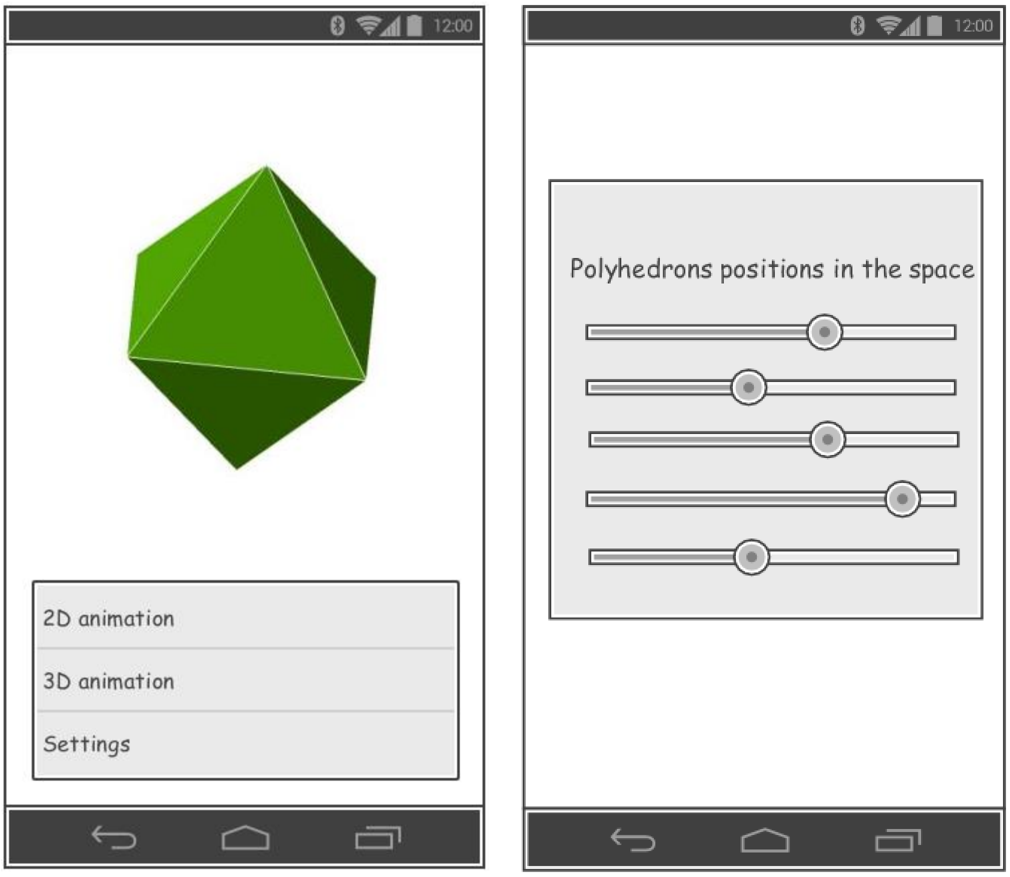


Figure 10: initial sketch of the application activities

Final designs

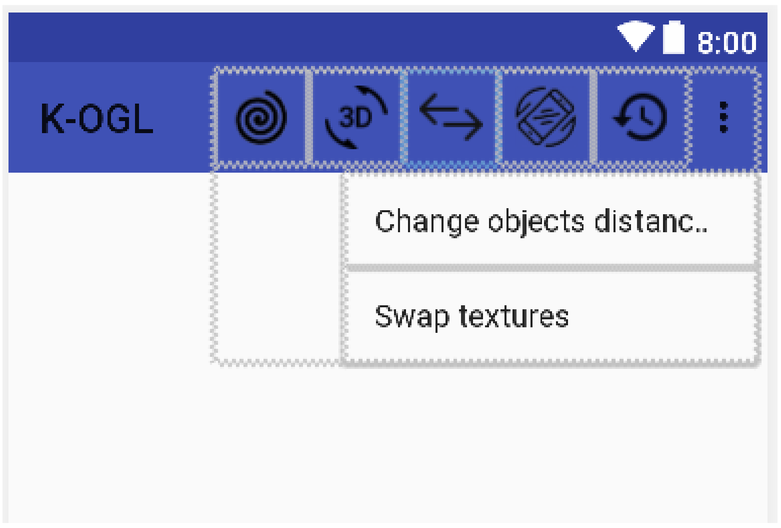
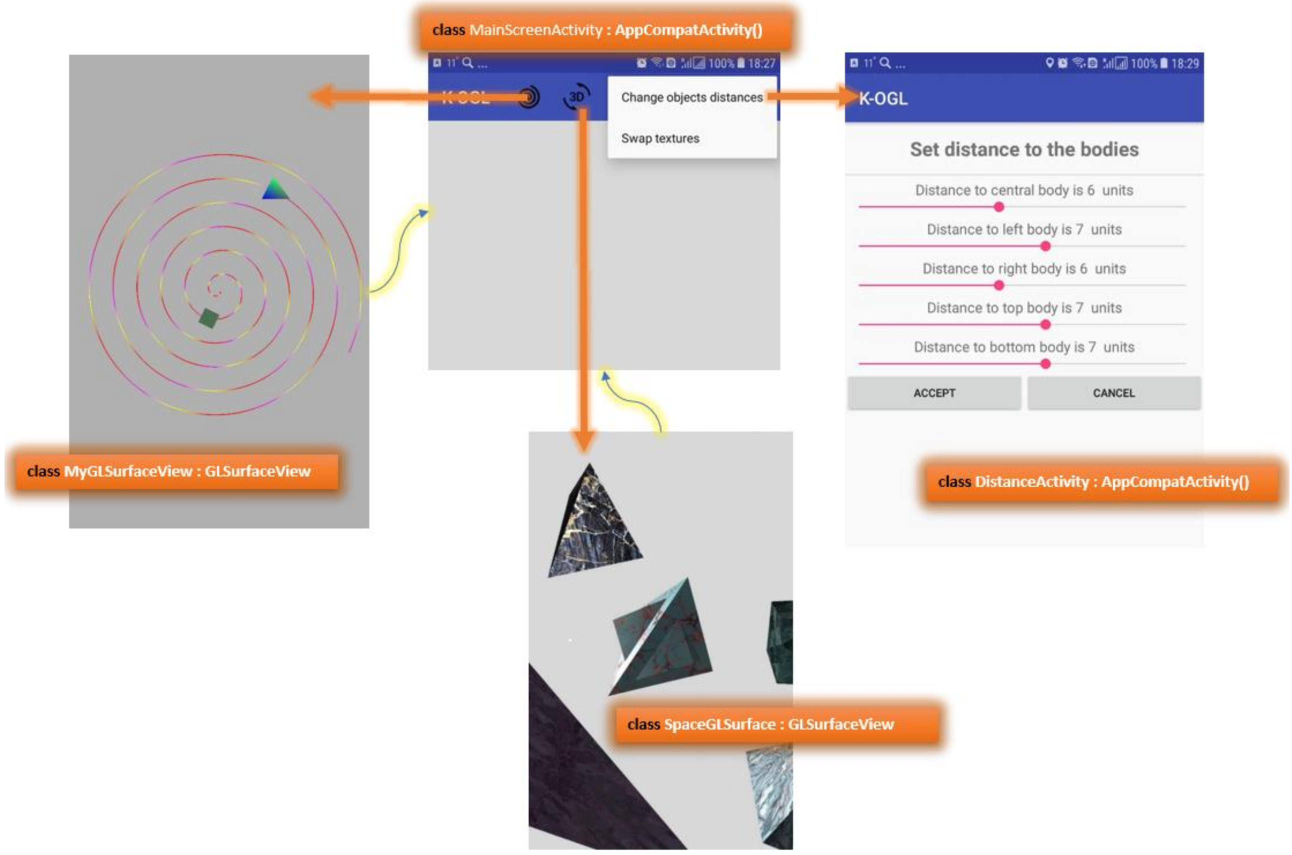


Figure 11: design of the main menu (file “menu_main_screen.xml”)



class MainScreenActivity : AppCompatActivity()
K-OGL
Change objects distances
Swap textures
K-OGL
Set distance to the bodies
Distance to central body is 6 units
Distance to left body is 7 units
Distance to right body is 6 units
Distance to top body is 7 units
Distance to bottom body is 7 units
ACCEPT
CANCEL
class DistanceActivity : AppCompatActivity()
class MyGLSurfaceView : GLSurfaceView
class SpaceGLSurface : GLSurfaceView

Figure 12: diagram of interactions between activities of the application

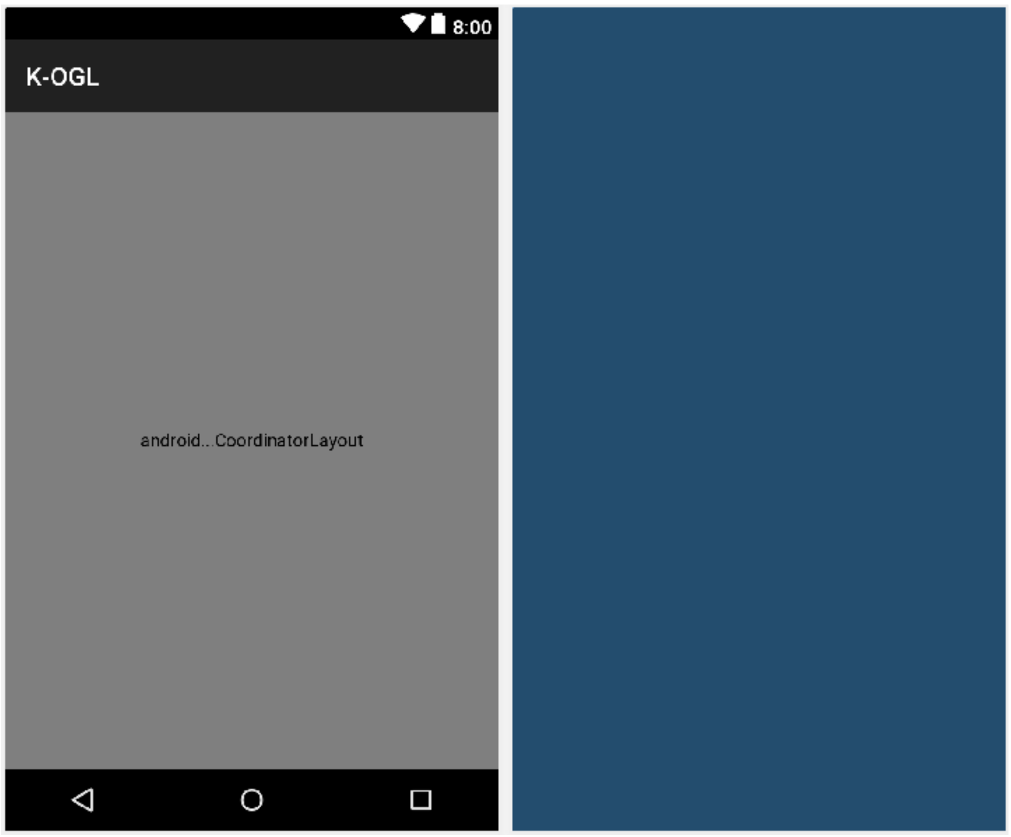


Figure 13: design of main activity (file "activity_main_screen.xml")

Main activity contains ConstrainLayout instance for placing and showing of OpenGL rendering scenes.

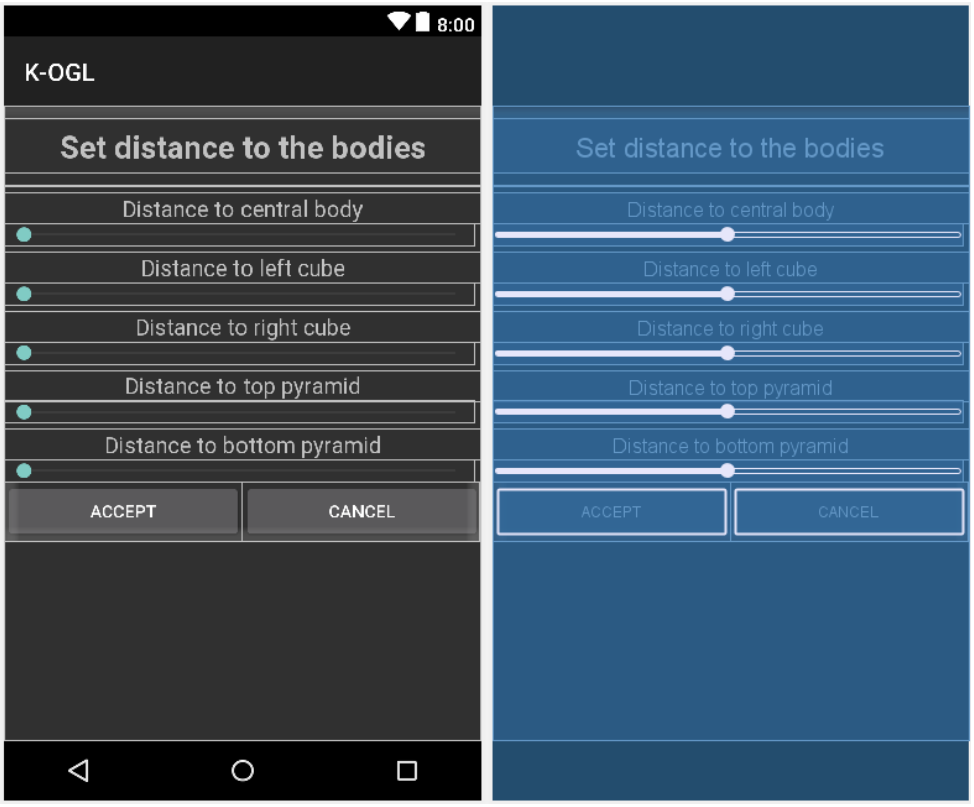


Figure 14: distance settings activity (file "activity_distance.xml")

Reasons for your final choices

Final design of the application being developed includes two activities. One of them is main, and another is used to set new values for distance of objects in 3D scene. Main activity possesses one layout and can show two different graphical scenes depending on the user choice.

Possible alternative decision for design being developed is to create different layouts for different graphical scenes – one for each scene. In this case design of application would consists of two additional layouts with their own menus and independent modules. Such design was not chosen because of possible increase of complexity in common data sending between these additional activities.

The final design allows to control both 2D and 3D scenes in universal way from one central layout using its main menu. Main benefit of this approach is that it makes structure of the application is clearer and simpler which can contribute better understandability and maintainability of the program.

Reflection

Android applications have general structure which is similar to the structure of application for any other operating system with graphical user interface. Approach which is used in Android Studio for designing of its windows (called layouts) presented by XML-code is not original and earlier was well developed in Microsoft Presentation Foundation technology (XAML) and JavaFX (FXML). At the same time design of GUI for Android has some peculiarities. One of them is ability to include mark-up code of some layout from an external file inside the code of another layout. This can contribute scalability of the application being developed. Such feature was used in my project for using of constraint layout in the window of main activity.

Implementation

Database schema

Database was not used in this project because there was no necessity to store complicatedly structured data between each execution of the software.

Android technologies

The "K-OGL" application uses the following Android technologies for implementing its own tasks:

1. Kotlin programming language;
2. Standard Android internal data storage of preferences was used to keep characteristics of graphical scenes when the application is not active or suspended;
3. Android SDK classes and interfaces for interoperability with OpenGL;
4. Android SDK classes and interfaces for interoperability with device's accelerometer;
5. Android SDK classes and interfaces for interoperability with touch screen.

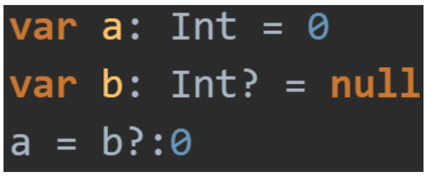
Kotlin

Kotlin is an object-oriented general-purpose technology (Samuel & Bocutiu, 2017). Kotlin includes programming language and set of underlying libraries and instruments that can provide a programmer with some interesting possibilities. First of all, it should be mentioned that Kotlin allows to build executables files of several types. These types cover formidable area of possible necessities of software developers. Kotlin has compilers for creation of applications for JVM (in this case compiler creates JVM-compatible byte code) and for creation of applications in native binary executable formats for major operating systems (Windows, Linux, MacOS). Kotlin is officially supported language for Android development since 2017, meanwhile the language was created by JetBrains company in 2011.

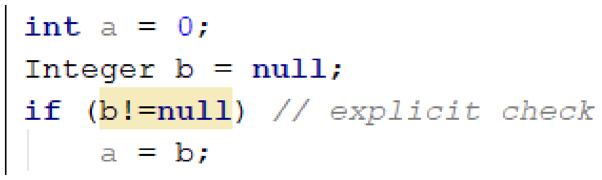
According to studied sources, Kotlin has some advantages over Java in some major aspects (Moskala & Wojda, 2017; Samuel & Bocutiu, 2017). Kotlin provides some improvements of drawbacks of Java. As widely known Java still doesn't have effective way of detecting of problem of null references which is very often resulted in runtime issues. Null-reference issues demands attention and meticulousness of a programmer and cannot be attributed to Java itself. But we can only welcome when the programming language helps a developer to eschew mistakes and Kotlin does it. Kotlin developers introduced (after C# and Swift languages) conception of nullable and non-nullable types which can ease the problems of nullability.

Table 11: comparison of ways of handling of null-reference in Kotlin and Java programming languages

Kotlin implementation	Java implementation



```
var a: Int = 0
var b: Int? = null
a = b?:0
```



```
int a = 0;
Integer b = null;
if (b!=null) // explicit check
    a = b;
```

Problem of Java implementation is that if a programmer simply forgets to write explicit checking statement, then previously showed snippet causes runtime error and fatal crash of application. Concerning Kotlin's snippet it is needed to say that the following assignment "a=b" is impossible syntactically. In this case Kotlin demands to use !! operator of safe nullable value cast. In this case the assignment "a=b?:0" just puts zero to "a" variable. Other strong features of Kotlin are safety (problems of nullability and immutability were taken in thorough consideration), conciseness (if to compare with Java programming language), wide interoperability with existing libraries (it is possible to use C and C++ programming language standard libraries for native application development or for programming with Android NDK). Also, it should be mentioned that Kotlin doesn't have limitations concerning old versions of Android operating system (it is possible to create applications for versions that even precede 4th version of Android operating system). So, Kotlin is capable technology for creation of mobile applications.

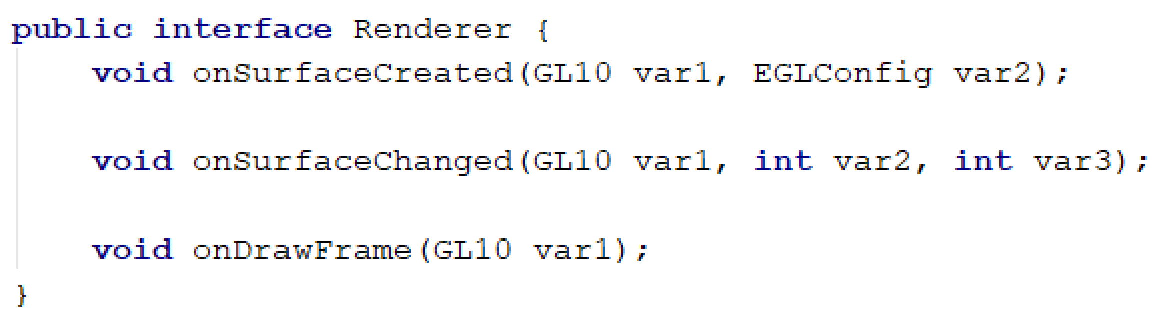
Another reason for choosing of this technology and language is personal interest of the author of the project. Kotlin is a modern and booming technology and it is important for personal professional portfolio to know and be able to use Kotlin.

Persistent storage for application's characteristics

Standard internal data storage provided by Android was used in order to ensure predictable appearance and behavior of the application in the cases of resuming after it was stopped or suspended by a user or operating system. To implement such functionality SharedPreferences API was used. This technology allows to store raw set of simple data in specially designated persistent memory on the base of dictionary principle (key-value). Usage of this kind of storage is expedient here because the application doesn't demand complicated structure of data and describes the scenes by the set of real numbers (of float data type). In this relatively simple case of data structure this approach has advantage over usage of relational database (such as SQLite or any other). SharedPreferences API has essential *limitations* in the case of usage of complicated data structure when we need to store data about several entities with inter-dependencies between them.

Interoperability with OpenGL

Android API provides developers with simple model for interoperability with OpenGL ES technology. This interoperability can be established by using two main things. They are GLSurfaceView class and GLSurfaceView.Renderer interface. The application should create surface for construction of OpenGL scene by inheriting of GLSurfaceView class. This provides a programmer with standard Android view-class which is capable to get fast output from OpenGL executing system. Created descendant should be provided with a renderer which is a instance of a class which implements GLSurfaceView.Renderer interface (Figure 15).

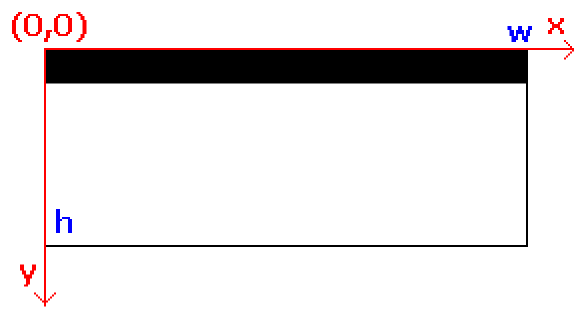
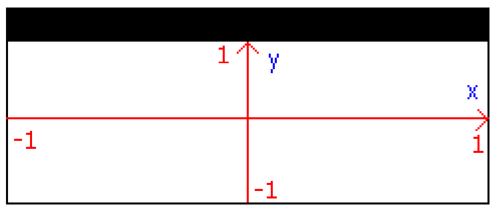


```
public interface Renderer {
    void onSurfaceCreated(GL10 var1, EGLConfig var2);

    void onSurfaceChanged(GL10 var1, int var2, int var3);

    void onDrawFrame(GL10 var1);
}
```

Figure 15: definition of "GLSurfaceView.Renderer" interface

All presented methods of the interface have self-descriptive names. "onSurfaceCreated" is normally used to set up global parameters of finite state machine of OpenGL (like colour for cleaning of colour buffer (background scene colour); usage of multi-buffered mode; enabling the calculations of "depth buffer" (for realistic 3D drawing) and so on). "onSurfaceChanged" method is usually used to set up a viewport (area of the physical screen which will be used by OpenGL for its rendering), setting up the coordinate system (which normally has the origin in the centre of the viewport and has visible points' coordinate range from -1 to 1 along three axes (Table 12)). Also, it can be mentioned that OpenGL coordinate system uses real numbers unlike standard window coordinate system which uses integers.

Table 12: difference between window and OpenGL coordinate systems

Coordinate system of Android window	Coordinate system of OpenGL scene
	

Also, this method is responsible for setting up the so-called "view volume" (which has "cutting planes" and defines what part of a virtual space will be considered as visible for the user) and the way of projection of 3D scene to the plane of the screen (orthogonal projection and perspective projection can be used). In the case of orthogonal projection view volume is a bounding parallelepiped, and in the case of usage of perspective projection view volume is constructed as a frustum pyramid (Brothaler, 2013).

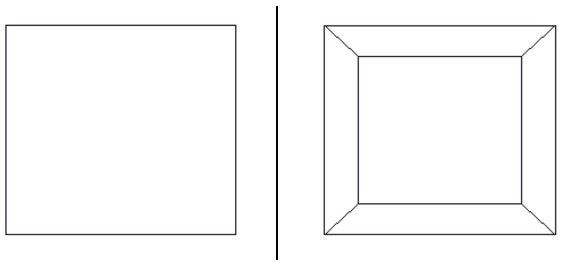


Figure 16: comparison of two types of projection

Concerning the "K-OGL" application, perspective projection was chosen in order to deliver more realistic view to scene, when more distant objects look smaller and closer objects look larger. In the case of orthogonal projection there is a significant drawback because it is not possible to express the relation between scene's objects on the base of the distance between them.

Concerning the usage of OpenGL there is possibility that some old or cheap (and thus "weak") devices don't have capability to execute OpenGL ES commands. For that case the program should explicitly specify the usage of OpenGL ES and describe the version is being used. This specification is placed in the manifest file and looks like that:

```
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true" />
```

If the goaled device is not capable to render OpenGL ES scenes, then the application won't be shown in its Google Play Market list.

Interoperability with device's accelerometer

Accelerometer is a sensor of mobile device which is used to determine the tilt of the device along the three basic axes. By default, the accelerometer continuously sending this data to operating system which is dispatching it to the interested applications and services. The "K-OGL" project uses data stream from accelerometer to change the direction of rotation of polyhedrons in the 3D scene and to change the direction of motion of elements of 2D scene.

Data from accelerometer can be caught by the instance of SensorManager class. This class incapsulates all the functionality for interaction with sensors of a device. In the case of accelerometer, we need to use the following command to instantiate the object which is planned to be used for receiving of its stream of data (Kotlin programming language):

```
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
```

Class which is intended to catch events from the sensor should implement SensorEventListener interface. As this done that class must have method "onSensorChanged(sensorEvent: SensorEvent)". Argument of type SensorEvent contains data from the desired sensor.

Interoperability with device's touch screen

All modern smartphones are equipped with touch screens. "K-OGL" application uses touch screen in order to provide a user with ability to rotate the 3D-scene around the central body in vertical or horizontal direction by arbitrary angle. Handling of touch event does not have any peculiarities in comparison with the handling of mouse clicks events in desktop applications with graphical user interface.

Screen shots

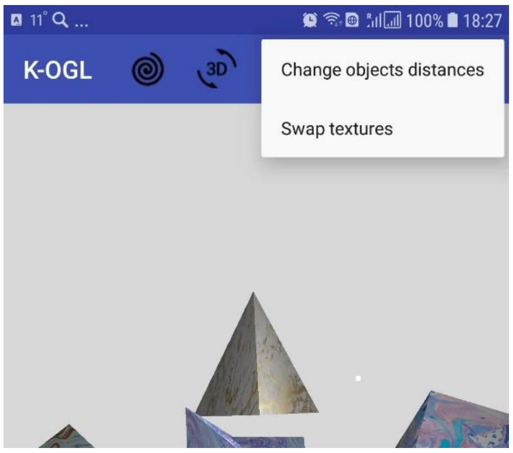


Figure 17: main activity with opened main menu

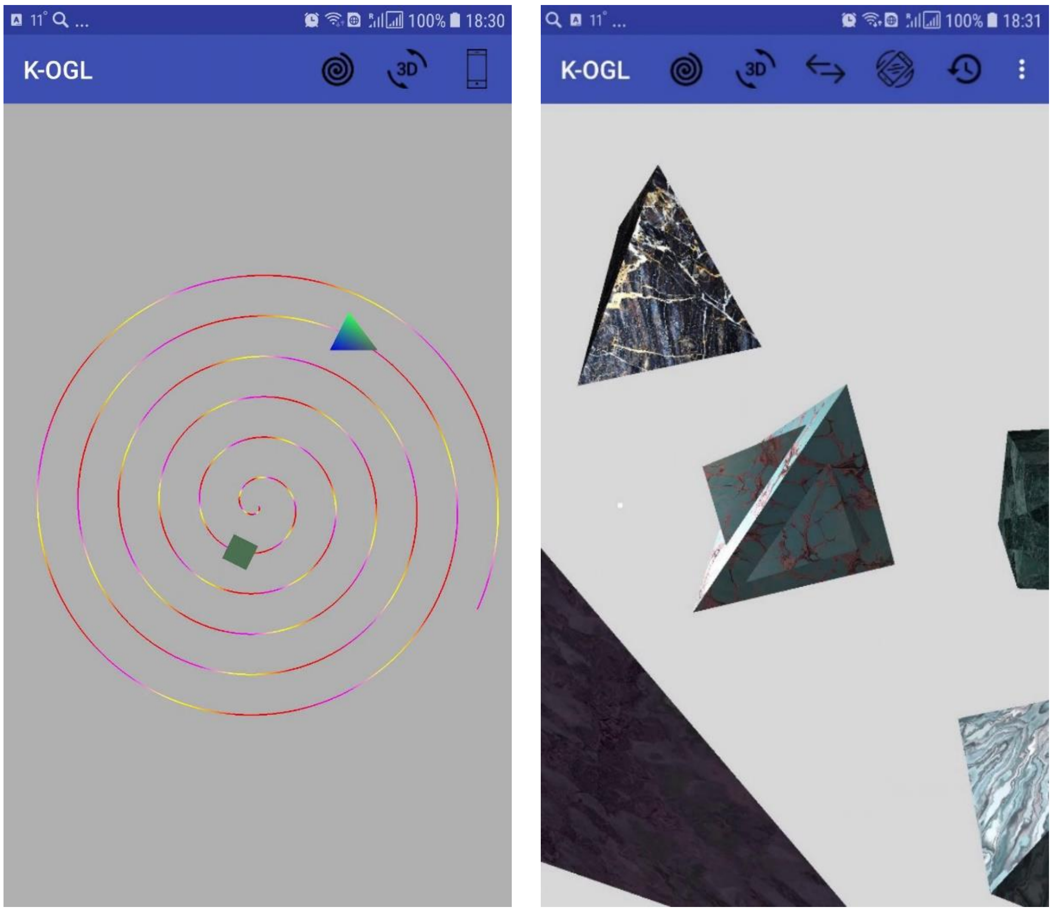
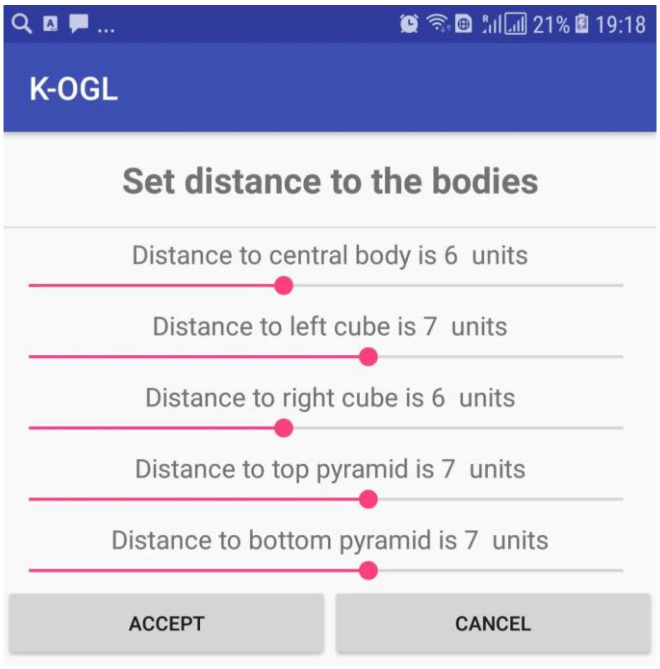


Figure 18: main activity with 2D (a) and 3D (b) scenes



Set distance to the bodies

Distance to central body is 6 units

Distance to left cube is 7 units

Distance to right cube is 6 units

Distance to top pyramid is 7 units

Distance to bottom pyramid is 7 units

ACCEPT CANCEL

Figure 19: activity for setting of distances to the objects

Reflection

During the process of implementation new and valuable knowledge was obtained. I learned knew programming language and technology Kotlin which is capable to create all the types of modern mobile applications.

Also, I've gained valuable experience in development with usage of OpenGL ES technology which is capable to quickly render complicated realistic 3D graphical scenes. In comparison with previous desktop version of OpenGL, this new version is more complicated and forces a developer to deepen inside low levels of graphical programming (for example, desktop version had simple helper functions, like gluCylinder or gluShere for drawing solid bodies; or glFog function for introducing visual effect of foggy space; but in the mobile version all of them were removed for the reason of optimisation). Now programmer of mobile application is responsible for doing such things. But there some help coming from third party developers who create additional helper add-ons to reduce complexity of graphical software development with OpenGL ES. For example, GLKit library was introduced, but it is not applicable for my project because it was developed for Apple iOS operating system. In general, software OpenGL ES is worthwhile to study because it produces efficient and responsive graphical scenes.

Handling of sensors' data in Android operating system is worth to understand and implement in mobile applications. Aside that, this the interesting topic by itself, such handling allows to enrich user's experience and make it more comfortable.

Testing

Test planning

Table 13: Testing plan for "K-OGL" project

Code	Name of testing	Expecting result
T-IN	Testing of GUI of the mobile applications	The program's GUI elements should have appearance and behavior which comply standard requirements (table 1, GUIR).
T-U	Unit testing	Code of the program must function correctly according to the requirements.
T-FN	Functional testing	The program demonstrates predictable behavior which complies the requirements (table 1, FR) and doesn't have hangings and runtime errors.
T-N	Non-functional testing	The program demonstrates compliance to the requirements (table 1, NFR, SR).

Table 14: graphical user interface testing plan

Test case code	Tested requirement	Expected result
T-IN-1	GUIR-1	Material design conventions should be implemented (includes color schemes, appearance of icons and controls).
T-IN-2	GUIR-2	The software should contain controls (standard main menu with buttons) for providing a user with ability to control behavior and appearance of the graphical scenes.
T-IN-3	GUIR-3	The software should follow global look and feel of operating system GUI which is set by the user.
T-IN-4	GUIR-4	The software supports only portrait orientation of the screen of a device.
T-IN-5	GUIR-5	The software should have appropriate icon for representation on the screen of used system launcher.

Table 15: unit testing plan

Test case code	Tested module/method	Expected result
T-U-1	TiltData	Data form device's accelerometer is usefully passed to rendering object to draw the scene with the respect of updated data of rotational angles.
T-U-2	SpaceGLSurface	Touch sensor data are passed to the renderer to draw the scene with the respect of updated data of rotational angles.

T-U-3	ShaderSource	Shader OpenGL programs' texts are loaded to memory to form images of solid bodies of the 3D scenes.

Table 16: functional testing plan

Test case code	Tested requirement	Expected result
T-FN-1	FR-1	The software should render two types of graphical scenes (flat scene and three-dimensional scene) on the base of utilization of abilities OpenGL ES API.
T-FN-2	FR-2	Flat scene should render animation which consists of three moving objects which are: a square, a triangle, and a spiral.
T-FN-3	FR-3	Each object of flat scene should have its own way of coloring of vertexes (changing within runtime) and its own rotation angle (changing within runtime).
T-FN-4	FR-4	The square and the triangle should move along the line of spiral in the opposite directions.
T-FN-5	FR-5	After reaching of extreme points of the spiral the objects should change direction of their moving to the opposite.
T-FN-6	FR-6	Three-dimensional scene should consist of five textured objects which are: 2 regular hexahedrons (cubes), 2 regular tetrahedrons (pyramids), one combined polyhedron (union of a cube and a pyramid).
T-FN-7	FR-7	Images of textures are loaded from files of PNG and JPEG formats from inner resource of the application.
T-FN-8	FR-8	Each body should rotate around its central point.
T-FN-9	FR-9	Three-dimensional scene should provide movable light source for producing of realistic view.
T-FN-10	FR-10	The software should provide a user with ability to choose type of output graphical scene (2D or 3D) by tapping of appropriate button in the main menu.
T-FN-11	FR-11	The software should provide a user with ability to start or stop animation of 3D scene by tapping of appropriate button in the main menu.
T-FN-12	FR-12	The software should provide a user with ability to start or stop processing of changing of device position in the space by tapping of appropriate button in the main menu.
T-FN-13	FR-13	The software should provide a user with ability to change the way of processing of screen touches by tapping of appropriate button in the main menu.
T-FN-14	FR-14	The software should provide a user with ability to change the characteristics of 3D scene (distance to 3D-objects) by tapping of appropriate button in the main menu and using of special customization window.

T-FN-15	FR-15	The software should provide a user with ability to change the appearances of the objects of 3D scene (surface textures) by tapping of appropriate button in the main menu.
T-FN-16	FR-16	The software should provide a user with ability to change the direction of moving of objects of 2D-scene by tilting the device.
T-FN-17	FR-17	The software should provide a user with ability to change the direction of rotation of objects of 3D-scene by tilting the device.
T-FN-18	FR-18	Application should provide a user with ability to rotate the whole 3D-scene (around vertical axis or around horizontal axis) by touching the screen of the device.

Table 17: non-functional testing plan

Test case code	Tested requirement	Expected result
T-N-1	NFR-1	Graphical scenes should look similarly on the screens of mobile devices with different versions of Android operating system.
T-N-2	NFR-2	Application should provide graphical scenes which adapt themselves to different screens resolutions and sizes.

Testing

Results of GUI testing

Table 18: graphical user interface testing results

Test case code	Tested requirement	Actual result
T-IN-1	GUIR-1	Passed
T-IN-2	GUIR-2	Passed
T-IN-3	GUIR-3	Passed
T-IN-4	GUIR-4	Passed
T-IN-5	GUIR-5	Passed

Results of unit testing

Table 19: result of unit test T-U-1

#	Class name	Method name	Method description
T-U-1	TiltData	setData	Method should set values of the tilts of vertical and horizontal axes and notify interested objects about these changes.

Package name	Helpers
Method signature	fun setData (x:Float, y:Float)

Input data	x is a float number which describes tilt along the OX-axis; y is a float number which describes tilt along the OY-axis; x and y are got from the listener of accelerometer's events normally x and y are numbers from the range (-10,10)			
Returns/result	void			
Test description	checking of in-time notification about each change of tilt got by device			
Pre-condition	mode of accelerometer data handling should be turned on			
Step	Test Data	Expected result	Actual result	Status
1	x=11, y=11	Data is declined, no action is undertaken.	Data is declined, no action is undertaken.	Passed
2	x=-9, y=9	New values of tilt are set, notification about the changes is sent. The renderer object redraws the entire scene with new values of rotational angles.	New values of tilt are set, notification about the changes is sent. The renderer object redraws the entire scene with new values of rotational angles.	Passed
3	x=-9, y=-9	New values of tilt are set, notification about the changes is sent. The renderer object redraws the entire scene with new values of rotational angles.	New values of tilt are set, notification about the changes is sent. The renderer object redraws the entire scene with new values of rotational angles.	Passed
4	x=9, y=-9	New values of tilt are set, notification about the changes is sent. The renderer object redraws the entire scene with new values of rotational angles.	New values of tilt are set, notification about the changes is sent. The renderer object redraws the entire scene with new values of rotational angles.	Passed
5	x=9, y=9	New values of tilt are set, notification	New values of tilt are set, notification	Passed

		about the changes is sent. The renderer object redraws the entire scene with new values of rotational angles.	about the changes is sent. The renderer object redraws the entire scene with new values of rotational angles.	

Table 20: result of unit test T-U-2

#	Class name	Method name	Method description	
T-U-2	SpaceGLSurface	onTouchEvent	Method passes data to 3D-scene for rendering of the objects when a user touches the screen and moves the finger.	
Package name	Space3D			
Method signature	**override fun onTouchEvent(e: MotionEvent): Boolean**			
Input data	Motion event form operating system			
Returns/result	True if result of handling was successful			
Test description	Test ensures that data is correctly passed to the renderer object			
Pre-condition	Mode of touch event listening should be enabled			
Step	Test Data	Expected result	Actual result	Status
1	e==null	return value is false; data is not passed to mSpaceGlRenderer object	return value is false; data is not passed to mSpaceGlRenderer object	Passed
2	e!=null	return value is false; data about touch event is passed to mSpaceGlRenderer object	return value is false; data about touch event is passed to mSpaceGlRenderer object	Passed

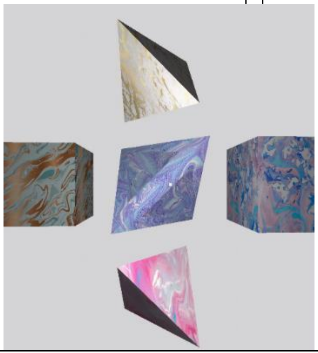
Table 21: result of unit test T-U-3

#	Class name	Method name	Method description
T-U-3	ShaderSource	getTextFromResourceFile	Method is used to retrieve OpenGL shader programs.
Package name	Space3D		
Method signature	fun getTextFromResourceFile(appContext: Context, appResourceId: Int): String?		
Input data	Context of main activity, integer ID of shader program being loaded.		

Returns/result	String with the text of the shader program, or null if the programs was not loaded.			
Test description	Test checks behavior of the program in different conditions concerning presence of shader programs' flies being loaded.			
Pre-condition	1. Text files with shader programs' texts exist in the catalog of the project and contain syntactically correct source code. 2. Text files with shader programs' texts does not exist in the catalog of the project and contain syntactically correct source code.			
Step	Test Data	Expected result	Actual result	Status
1	Method arguments generated by OS and OpenGL automatically. Precondition 1 takes place.	Shader programs is loaded into memory, object in the 3d-scene are successfully created.	Shader programs is loaded into memory, object in the 3d-scene are successfully created.	Passed
2	Method arguments generated by OS and OpenGL automatically. Precondition 2 takes place.	Program stopes its functioning because of inability to create graphical scenes.	Program stopes its functioning because of inability to create graphical scenes.	Passed

Results of functional testing

Table 22: Test case T-FN-1 results

#	Application functionality to be tested	Possible issues	
T-FN-1	FR-1 (table 1)	Application failed to switch between 2D and 3D scenes.	
Step	Action	Expected result	Status
1	Launch the application "K-OGL"	The main window appears as shown below. 	Passed
2	Touch the button "show 2D" Icon: ◎	2D scene appears as shown below.	Passed

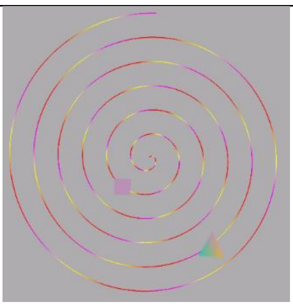
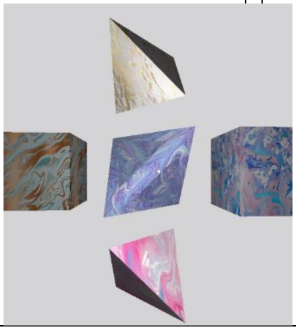
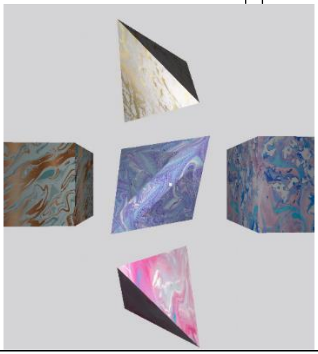
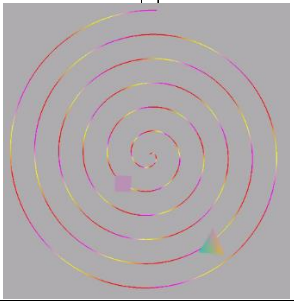
		 In this case only three buttons are stay visible in the main menu. It is possible to switch the program back to 3D-scene, stop or start the processing of accelerometers events.	
3	Touch the button "show 2D" Icon: ⟳	The main window appears as shown below. 	Passed

Table 23: Test case T-FN-2 results

#	Application functionality to be tested	Possible issues	
T-FN-2	FR-2 (table 1)	Application failed to render 3 animated figures.	
Step	Action	Expected result	Status
1	Launch the application "K-OGL"	The main window appears as shown below. 	Passed
2	Touch the button "show 2D" Icon: ◎	2D scene appears as shown below. 	Passed

		The spiral should rotate around central its point. Square should be moving from the center of the spiral to its end. The triangle should be moving from the end point of the spiral to its central point.	

Table 24: Test case T-FN-3 results

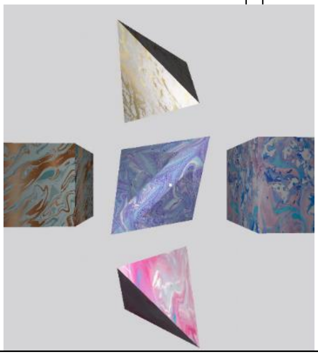
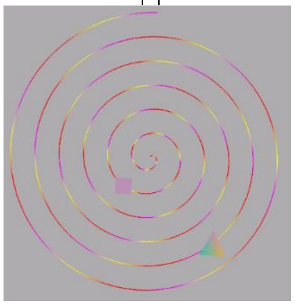
#	Application functionality to be tested		Possible issues
T-FN-3	FR-3 (table 1)		The figures of the flat scene don't have appropriate changes of their vertexes' colors.
Step	Action	Expected result	Status
1	Launch the application "K-OGL"	The main window appears as shown below. 	Passed
2	Touch the button "show 2D" Icon: ◎	2D scene appears as shown below.  The square is constantly changing color of its vertices from red to red (through intermediate colors). Each vertex of triangle is constantly gradually alternating its colors.	Passed

Table 25: Test case T-FN-4 results

#	Application functionality to be tested		Possible issues
T-FN-4	FR-4 (table 1)		The program fails to provide required functionality.
Step	Action	Expected result	Status

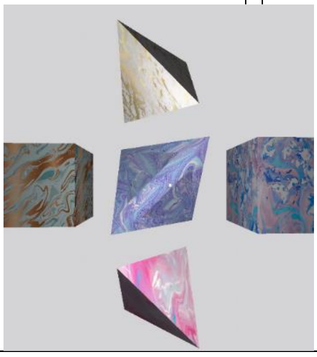
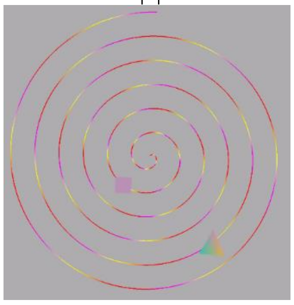
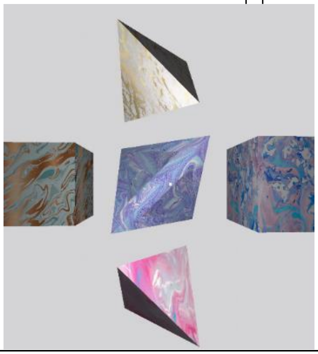
1	Launch the application "K-OGL"	The main window appears as shown below. 	Passed
2	Touch the button "show 2D" Icon: 🌀	2D scene appears as shown below.  In the scene which appeared the square and the triangle are moving to the opposite directions.	Passed

Table 26: Test case T-FN-5 results

#	Application functionality to be tested		Possible issues
T-FN-5	FR-5 (table 1)		The figures of the 2D scene fail to change direction of their movement when reaching the extreme point of the spiral.
Step	Action	Expected result	Status
1	Launch the application "K-OGL"	The main window appears as shown below. 	Passed
2	Touch the button "show 2D" Icon: 🌀	2D scene appears as shown below.	Passed

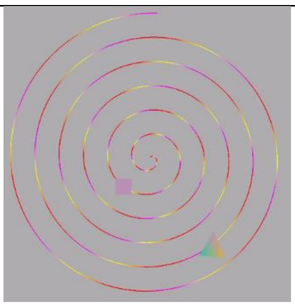
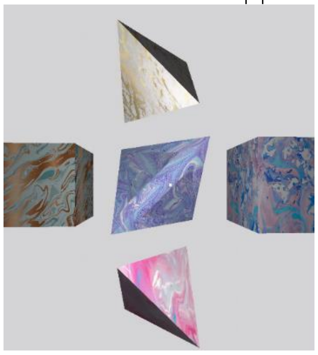
		 In the scene which appeared the square and the triangle are moving to the opposite directions.	
3	Wait until the square reaches the end point of the spiral.	On arriving to the end point of the spiral the square starts moving to the center of the spiral.	Passed
4	Wait until the square reaches the central point of the spiral.	On arriving to the central point of the spiral the square starts moving to the margin of the spiral.	Passed
5	Wait until the triangle reaches the end point of the spiral.	On arriving to the end point of the spiral the triangle starts moving to the center of the spiral.	Passed
6	Wait until the triangle reaches the central point of the spiral.	On arriving to the central point of the spiral the triangle starts moving to the margin of the spiral.	Passed

Table 27: Test case T-FN-6 results

#	Application functionality to be tested		Possible issues
T-FN-6	FR-6 (table 1)		3D scene fails to render 5 solid bodies (2 cubes, 2 pyramids, 1 combination of cube and pyramid).
Step	Action	Expected result	Status
1	Launch the application "K-OGL"	The main window appears as shown below.  Current 3D scene consists of 5 textured objects (2 cubes, 2 pyramids, 1 combination of cube and pyramid).	Passed
2	Touch the button "show 2D" Icon:	3D scene disappears and 2D scene appears as shown below.	Passed

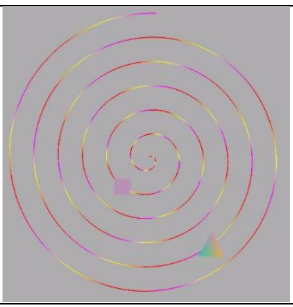
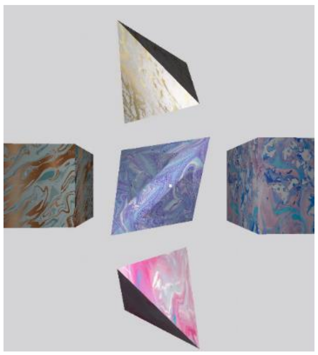
	◎		
3	Touch the button "show 3D" Icon: 3D	2D scene disappears and 3D scene appears as shown below.  Current 3D scene consists of 5 textured objects (2 cubes, 2 pyramids, 1 combination of cube and pyramid).	Passed

Table 28: Test case T-FN-7 results

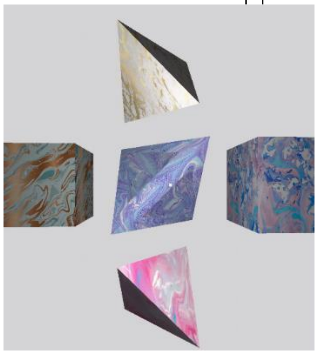
#	Application functionality to be tested		Possible issues
T-FN-7	FR-7 (table 1)		The program fails to load files of textures from its inner resource.
Step	Action	Expected result	Status
1	Launch the application "K-OGL"	The main window appears as shown below.  Each object is textured by an image from inner resource of application.	Passed

Table 29: Test case T-FN-8 results

#	Application functionality to be tested	Possible issues

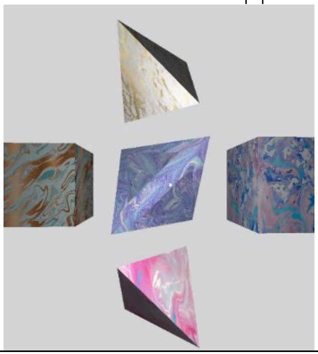
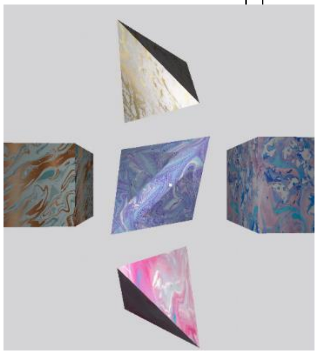
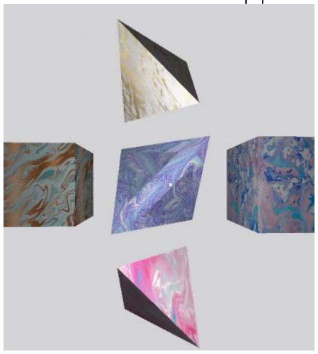
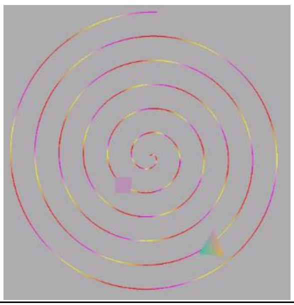
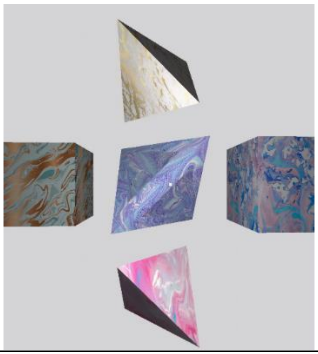
T-FN-8	FR-8 (table 1)		The program fails to show animated rotation of the 3D-scene's bodies.
Step	Action	Expected result	Status
1	Launch the application "K-OGL"	The main window appears as shown below. 	Passed
2	Touch the button "show 2D" Icon:	Each object of the current 3D-scene starts rotating independently around its central point.	Passed
3	Touch the button "show 2D" Icon:	All objects of the current 3D-scene stop rotating.	Passed

Table 30: Test case T-FN-9 results

#	Application functionality to be tested		Possible issues
T-FN-9	FR-9 (table 1)		The program can't draw source of light for delivering a realistic 3D-view in the 3D-scene.
Step	Action	Expected result	Status
1	Launch the application "K-OGL"	The main window appears as shown below.  There is a white point in the center of the screen in front of central body. This standalone vertex emits white light along all the directions. Frontal parts of the bodies are brighter than their non-frontal parts.	Passed
2	Touch the button "show 2D" Icon:	Each object of the current 3D-scene starts rotating independently around its central point. The source of light start rotating	Passed

	⟲	independently around central body. While its moving it is possible to notice that different parts of the bodies get more or less of the source's light depending on their own rotation and the position of the light source.	

Table 31: Test case T-FN-10 results

#	Application functionality to be tested		Possible issues
T-FN-10	FR-10 (table 1)		The program fails to provide corresponding functionality by using of touch event.
Step	**Action**	**Expected result**	**Status**
1	Launch the application "K-OGL"	The main window appears as shown below.  Current 3D scene consists of 5 textured objects (2 cubes, 2 pyramids, 1 combination of cube and pyramid).	Passed
2	Touch the button "show 2D" Icon: ◎	3D scene disappears and 2D scene appears as shown below. 	Passed
3	Touch the button "show 3D" Icon: 3D	2D scene disappears and 3D scene appears as shown below. 	Passed

		Current 3D scene consists of 5 textured objects (2 cubes, 2 pyramids, 1 combination of cube and pyramid).	

Table 32: Test case T-FN-11 results

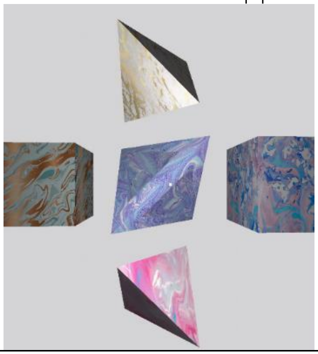
#	Application functionality to be tested		Possible issues
T-FN-11	FR-11 (table 1)		The program fails to show animated rotation of the 3D-scene's bodies.
Step	Action	Expected result	Status
1	Launch the application "K-OGL"	The main window appears as shown below. 	Passed
2	Touch the button "show 2D" Icon:	Each object of the current 3D-scene starts rotating independently around its central point.	Passed
3	Touch the button "show 2D" Icon:	All objects of the current 3D-scene stop rotating.	Passed

Table 33: Test case T-FN-12 results

#	Application functionality to be tested		Possible issues
T-FN-12	FR-12 (table 1)		The program fails to process accelerometer events in order to change the direction of rotation of the scene's objects.
Step	Action	Expected result	Status
1	Launch the application "K-OGL"	The main window appears as shown below.	Passed

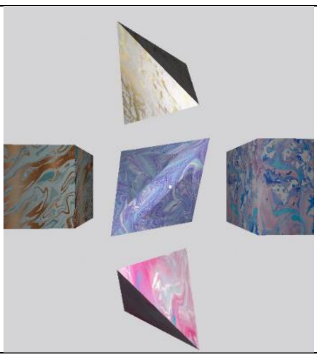
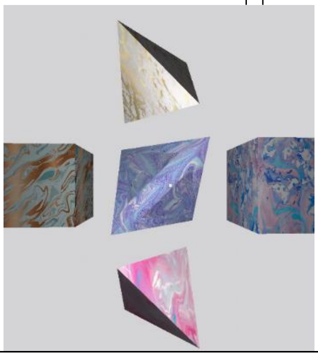
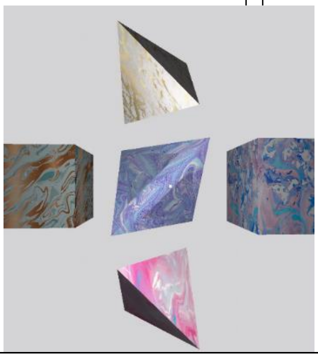
			
2	Touch the button "show 2D" Icon: 🕓	Each object of the current 3D-scene starts rotating independently around its central point.	Passed
3	Touch button "Accelerometer": 🔄	Icon the button will be changed to the following: 📱 Direction of rotation of the objects can be alternated depending on the current tilt given to the smartphone.	Passed
4	Touch button "Accelerometer": 📱	Icon the button will be changed to the following: 🔄 Direction of rotation of the objects cannot be alternated depending on the current tilt given to the smartphone.	Passed

Table 34: Test case T-FN-13 results

#	Application functionality to be tested	Possible issues	
T-FN-13	FR-13 (table 1)	The program fails to stop and start processing of accelerometer event for controlling of	
Step	**Action**	**Expected result**	**Status**
1	Launch the application "K-OGL"	The main window appears as shown below. 	Passed

		It is possible to rotate the entire scene around the horizontal axis by moving a finger along the surface of the device's screen.	
2	Touch the button "Choose rotation direction". Icon: ⇅	After that the icon of the button is changed to the following: ⇆ In this mode rotation of the entire scene can be made around vertical axis the same way as was described above.	Passed

Table 35: Test case T-FN-14 results

#	Application functionality to be tested		Possible issues
T-FN-14	FR-14 (table 1)		The program fails to change distances of objects from the frontal plane of the screen or changes them incorrectly.
Step	**Action**	**Expected result**	**Status**
1	Launch the application "K-OGL"	The main window appears as shown below. 	Passed
2	Activate main sub-menu by touching "triple dot" icon and touch item "Change objects distances".	Distance setup window appears.	Passed
3	Press "Cancel" button.	Distance setup window disappears, no any changes applied to the objects of the scene.	Passed.
4	Activate main sub-menu by touching "triple dot" icon and touch item "Change objects distances".	Distance setup window appears.	Passed
5	Change distance of the central body by moving first slider to the minimum position (3 units) and press "Accept" button.	Distance setup window appears and show current distances. After accepting of new values this window closes, then the central body moves to the most possible close position to the plane of the screen.	Passed
6	Change distance of the central body by moving first slider to the maximum	Distance setup window appears and show current distances. After accepting of new values this window closes, then the central	Passed

	position (10 units) and press "Accept" button.	body moves to the most possible distant position from the plane of the screen.	
7	Change distance of the left cube by moving 2nd slider to the minimum position (3 units) and press "Accept" button.	Distance setup window appears and show current distances. After accepting of new values this window closes, then the left cube moves to the most possible close position to the plane of the screen.	Passed
8	Change distance of the left cube by moving 2nd slider to the maximum position (10 units) and press "Accept" button.	Distance setup window appears and show current distances. After accepting of new values this window closes, then the left cube moves to the most possible distant position from the plane of the screen.	Passed
9	Change distance of the right cube by moving 3rd slider to the minimum position (3 units) and press "Accept" button.	Distance setup window appears and show current distances. After accepting of new values this window closes, then the right cube moves to the most possible close position to the plane of the screen.	Passed
10	Change distance of the right cube by moving 3rd slider to the maximum position (10 units) and press "Accept" button.	Distance setup window appears and show current distances. After accepting of new values this window closes, then the right cube moves to the most possible distant position from the plane of the screen.	Passed
11	Change distance of the top pyramid by moving 4th slider to the minimum position (3 units) and press "Accept" button.	Distance setup window appears and show current distances. After accepting of new values this window closes, then the top pyramid moves to the most possible close position to the plane of the screen.	Passed
12	Change distance of the top pyramid by moving 4th slider to the maximum position (10 units) and press "Accept" button.	Distance setup window appears and show current distances. After accepting of new values this window closes, then the top pyramid moves to the most possible distant position from the plane of the screen.	Passed
13	Change distance of the bottom pyramid by moving 5th slider to the minimum position (3 units) and press "Accept" button.	Distance setup window appears and show current distances. After accepting of new values this window closes, then the bottom pyramid moves to the most possible close position to the plane of the screen.	Passed
14	Change distance of the bottom pyramid by moving 5th slider to the maximum position (10 units) and press "Accept" button.	Distance setup window appears and show current distances. After accepting of new values this window closes, then the bottom pyramid moves to the most possible distant position from the plane of the screen.	Passed

Table 36: Test case T-FN-15 results

#	Application functionality to be tested	Possible issues

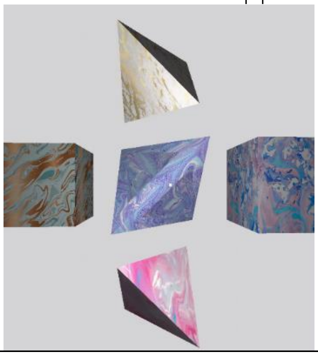
T-FN-15	FR-15 (table 1)	The program fails to apply changes of textures of the objects.	
Step	Action	Expected result	Status
1	Launch the application "K-OGL"	The main window appears as shown below. 	Passed
2	Activate main sub-menu by touching "triple dot" icon and touch item "Swap textures".	The program cyclically changes the scene's objects' textures (overall 14 different textures are provided which are consecutively applied to each object of the scene).	Passed

Table 37: Test case T-FN-16 results

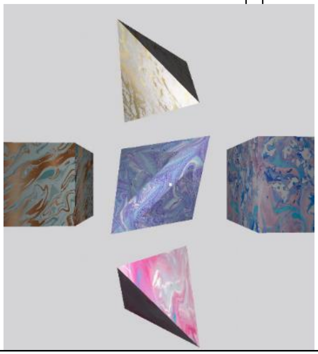
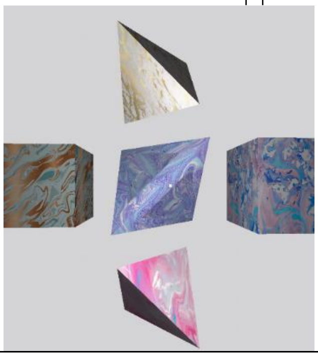
#	Application functionality to be tested	Possible issues	
T-FN-16	FR-16 (table 1)	The program fails to provide required functionality.	
Step	Action	Expected result	Status
1	Launch the application "K-OGL"	The main window appears as shown below. 	Passed
2	Touch the button "show 2D" Icon:	In the scene which appeared the square and the triangle are moving to the opposite directions.	Passed
3	Touch button "Accelerometer":	Depending on the current device's tilt: • Left/right tilt – the triangle alternates direction of its movement to the opposite. • Up/down tilt – the square alternates direction of its movement to the opposite.	Passed

Table 38: Test case T-FN-17 results

#	Application functionality to be tested	Possible issues
T-FN-17	FR-17 (table 1)	The program fails to process accelerometer events in order to change the direction of rotation of the scene's objects.

Step	Action	Expected result	Status
1	Launch the application "K-OGL"	The main window appears as shown below. 	Passed
2	Touch the button "show 2D" Icon:	Each object of the current 3D-scene starts rotating independently around its central point.	Passed
3	Touch button "Accelerometer":	Direction of rotation of the objects can be alternated depending on the current tilt given to the smartphone.	Passed
4	Place the device parallel to the horizontal plane and give it significant tilt to the right (after that you can return original position).	All objects of the 3D-scene (except central) changes rotational direction to the right (counter-clockwise).	Passed
5	Place the device parallel to the horizontal plane and give it significant tilt to the left (after that you can return original position).	All objects of the 3D-scene (except central) changes rotational direction to the left (clockwise).	Passed
6	Place the device parallel to the horizontal plane and give it significant tilt from yourself keeping its portrait orientation (after that you can return original position).	All objects of the 3D-scene (except central) changes rotational direction and start rotating around horizontal axis (clockwise).	Passed
7	Place the device parallel to the horizontal plane and give it significant tilt to yourself keeping its portrait orientation (after that you can return original position).	All objects of the 3D-scene (except central) changes rotational direction and start rotating around horizontal X-axis (counter-clockwise).	Passed

Results of non-functional testing

Table 39: results of test T-N-1

Test ID	Requirement ID	Description of undertaker activity
T-N-1	NFR-1	The program was executed under different version of Android API which are defined by the concrete version of this operating system (in Android Studio emulator). In the series of tests, the principal goal was to preserve similarity of appearance and behavior of the graphical scenes. All object should be visible to the user and preserve their position in any type of possible screen resolution. All scenes were compared visually to sample scenes which is generated by the device of the project's developer (Samsung J7 2017, Android version is 7, API version is 28, screen resolution was not taken in consideration).

Tested API version	Expected result	Test result
24	Graphical scenes have similar view and behavior in different versions of Android operating system.	Passed
25		Passed
26		Passed
27		Passed
28		Passed

Table 40: results of test T-N-2

Test ID	Requirement ID	Description of undertaker activity
T-N-2	NFR-2	The program was executed under different screen resolutions in Android Studio emulator. In the series of tests, the principal goal was to preserve similarity of appearance of the graphical scenes. All object should be visible to the user and preserve their position in any type of possible screen resolution. All scenes were compared visually to sample scenes which is generated by the device of the project's developer (Samsung J7 2017, screen resolution 1920×1080, version of API is 28).

Tested screen resolution and size (inch)	Expected result	Test result
720×1280 (4.7)	Graphical scenes were proportionally scaled according the actual size of the screen. Formed scenes are similar to sample view.	Passed
768×1280 (4.7)		Passed
1080×1920 (5)		Passed
1080×1920 (6)		Passed
1440×2560 (5.5)		Passed
1440×2560 (6)		Passed
1200×1920 (7)		Passed
2048×1536 (8.9)		Passed

2560×1800 (9.9)		Passed
2560×1600 (10.1)		Passed
480×800 (4)		Passed

Remaining defects

Despite undertaken testing activities there is possibility that some bugs are still remaining in the software. I believe that lack of resources on the device can cause unpredictable behavior and even crash of the application, but this kind of situation is difficult to catch on the emulator.

Reflection

Executing of tests for this project delivered new experience concerning the ways of testing of mobile applications. Android operating system has different versions, supports different devices with different screen resolutions and different levels of efficiency of its hardware. All of these obstacles can create substantial difficulties during the testing. This is especially actual for project which involve technologies containing computer graphics. My project uses OpenGL ES technology which luckily is supported by major part of mobile devices and from old versions of Android operating system. This testing process demonstrated that it is not easy to provide a mobile application with real portability, scalability and responsiveness in common case.

Conclusion

Reflection

This project gave me possibility to widen and to deepen my knowledge in the area of developing, programming, debugging and testing of mobile applications for Android operating system. Mobile development has some peculiarities but in general is gradually approaching the way of usual desktop applications' development. Effective and successful development of software of any kind (for desktop or mobile, or for microcontrollers) is impossible without understanding and following of some basic principles, which describe overall demands and recommendations which can significantly influence quality attributes. Here architectural and design patterns can be named. If to say about concrete things which I've studied within the time of the project implementation, following can be mentioned:

1. I've learned how to implement principal ideas of software architecture patterns and design patterns ("client-server", "event-bus", "observer", "singleton");
2. I've studied ways of creating of mobile applications of common types for Android operating system (with single activity, with multiple activities, graphical);
3. I've learned and practically used Kotlin programming language;
4. The most intrigue thing to learn was API of OpenGL ES. This technology is familiar to me as desktop version. But mobile version was significantly changed (but at the same time this version remained the same architecturally). All the redundant and high-level features were cut off. Now the programmer is responsible for implementing of almost everything. And coming new API named Vulkan will only aggravate this situation but will bring even more efficient computer graphics to mobile platforms. Another new thing which I had to understand, and implement was the idea of shader and shader program. Shader is a program inside the program which controls rendering of each pixel of the graphical scene. Shaders are controlled by their own C-lie programming language. Also, it was fascinating to realize how Android API interacts with OpenGL and how to pass data from devices' hardware (accelerometer, touch screen) to the OpenGL renderer which has its own 3D coordinate system (unlike 2D coordinate system of the screen).
5. I've studied how to program interaction with user of mobile device with the usage of the device's sensors.
6. I've studied how to implement testing activities for mobile applications.

If to say about possible improvements I would to mention that OpenGL ES technology nowadays is gradually becoming weaker than its young contemporary which is called Vulkan. But Vulkan API is not so widely supported and implemented. Google announced and implemented interoperability with this technology about 2 years ago. So it was not expedient to create the application using Vulkan API now, but in the future usage of this technology can be considered as a real alternative to OpenGL API.

Summary

Result of this project is the mobile application which implements graphical 2D and 3D scenes. These scenes are fully formed with the use of OpenGL ES (open graphical library for embedded systems) application programming interface. Created application meets formulated requirements (table 1) what was demonstrated during undertaken testing activities.

References

Arnuphaptrairong, T. (2011). Top Ten Lists of Software Project Risks:Evidence from the Literature Survey. *Proceedings of the International MultiConference of Engineers and Computer Scientists*. Hong Kong: IMECS 2011.

Booch, G. (1991). *Object oriented design with applications*. Redwood: The Benjamin/Cummings Publishing Company, Inc.

Brothaler, K. (2013). *OpenGL ES 2 for Android*. Dallas: The Programatic Bookshelf.

Buschmann, F. (2013). *Pattern-Oriented Software Architecture*. New York: Wiley.

Emmatty, J. (2018, August 21). *Singleton Pattern – Positive and Negative Aspects*. Retrieved from Code Project: https://www.codeproject.com/Articles/307233/Singleton-Pattern-Positive-and-Negative-Aspects-2

Flowler, M. (2013). *Patterns of Enterprise Software Architecture*. Addison-Wesley.

Garlan, D., Ivers, J., & Little, R. (2011). *Documenting software architecture*. Pearson Education.

Ginsburg, D., & Purnomo, B. (n.d.). *OpenGL ES 3.0 Programming Guide*. 2014: Addison-Wesley.

Google LLC. (2018, August 20). *Android*. Retrieved from Android: https://www.android.com/

Google LLC. (2018, August 20). *Android Developers*. Retrieved from Build for Android: https://developer.android.com/design/

Hunt, A., & Tomas, D. (2000). *The pragmatic programmer*. Boston: Addison-Wesley.

Khronos Group. (2016). *OpenGL ES Shading Language*. Khronos Group.

KHRONOS GROUP. (2018, August 18). *OpenGL ES Overview - The Khronos Group Inc*. Retrieved from The Khronos Group Inc: https://www.khronos.org/opengles/

KHRONOS GROUP. (2018, August 18). *OpenGL Overview - The Khronos Group Inc*. Retrieved from The Khronos Group Inc: https://www.khronos.org/opengl/

Langer, A. (2016). *Guide to Software Development. Designing and Managing the Life Cycle*. New York: Springer.

Liu, H. (2009). *Software Performance and Scalability*. Hoboken: Wiley&Sons.

Mallawaarachchi, V. (2018, August 21). *What is an Architectural Pattern?* Retrieved from Medium – a place to read and write big ideas and important stories: https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

Microsoft Corporation. (2018, August 20). *DirectX Graphics and Gaming | Microsoft Docs*. Retrieved from Microsoft - Official Home Page: https://docs.microsoft.com/en-us/windows/desktop/directx

Moskala, M., & Wojda, I. (2017). *Android Development with Kotlin.* Birmingham-Mumbai: Packt Publishing.

Patton, R. (2005). *Software testing.* New York: Sam publishing.

Portnov, M. (2018). *Online courses of software testers*. Retrieved from Portnov computer school: https://www.portnov.com/2018

Pressman, R. (2001). *Software engineering.* New York: McGraw-Hill.

Rizwan, M. (2014). Impact of Design Patterns on Software Maintainability. *International Journal of Intelligent Systems and Applications*, 41-46.

Samuel, S., & Bocutiu, S. (2017). *Programming Kotlin.* Birmingham: Packt Publishing.

Sellers, G. (2017). *Vulkan Programming Guide.* Pearson education.

Stoneburner, G., Goguen, A., & Feringa, A. (2002). *Risk Management Guide for Information Technology Systems.* Gaithersburg: National Institute of Standards and Technology.

Wallace, L., & Keil, M. (2004). Software Project Risk and their Effect on Outcomes. *Communication fo the ACM*, 68-73.