

PRZEDSIĘBIORCZOŚĆ INNOWACYJNA – GRAFIKA KOMPUTEROWA

3D OBJECTS VISUALIZATION ON MOBILE DEVICES

WIZUALIZACJA OBIEKTÓW 3D NA URZĄDZENIACH MOBILNYCH

Dominika Cis

Opiekun pracy: dr inż. Aleksander Krzyś

Wrocław 2014

Table of Contents

Abstract.....	5
1.Theoretical Background.....	7
1.1.World of Computers and 3D.....	7
1.1.1.Representation of 3D object.....	7
1.1.2.Projections and camera.....	10
1.1.3.Shadows, lights and textures.....	11
1.2.Characteristic of mobile devices.....	13
2.Implementation.....	14
2.1.Selected mobile platform: Android.....	14
2.2.Selected 3D engine: Unity.....	14
2.3.Project overview.....	15
2.4.Setting the scene.....	15
2.5.Programming the scene.....	18
2.5.1.User Interface.....	18
2.5.2.Accelerometer rotation and touch.....	19
2.6.Results.....	21
3.Conclusion.....	23
3.1.Cognitive conclusions.....	23
3.2.Further usage.....	23
Bibliography.....	24
Figures.....	25
Attachments.....	26

Abstract

The goal of this thesis is to investigate the capabilities of mobile technologies for displaying three dimensional objects.

Paper consists of three chapters.

First section covers mathematical representation of 3D scenes. It also consists of information about mobile devices. Furthermore, it presents tools selected for research.

Second chapter is a practical part of the thesis. It shows the application of a selected tools for creating 3D objects visualization on selected mobile platform. It consists of step by step description of creation process and implementation, as well as testing.

Conclusion sums up the whole thesis.

Wstęp

Celem tej pracy jest zbadanie możliwości technologii mobilnych do tworzenia trójwymiarowych wizualizacji obiektów.

Praca składa się z trzech części.

Część pierwsza jest o matematycznej reprezentacji scen w trójwymiarze. Rozdział ten zawiera również informacje o urządzeniach mobilnych. Ponadto zaprezentowane są tu narzędzia użyte do stworzenia pracy.

Drugi rozdział jest częścią praktyczną tej pracy. Pokazuje on użycie wybranych narzędzi do stworzenia wizualizacji trójwymiarowych na wybranej platformie mobilnej. Zawiera opis jak aplikacja została stworzona i zaimplementowana, jak również wyniki testów.

Zakończenie podsumowuje całą pracę.

1. Theoretical Background

1.1. World of Computers and 3D

Development of geometrical techniques enabled a 3D scene to be **rendered** on a 2D tableau. Rendering is forming a visible image of the 3D scene that we have modeled. The primary purpose of three-dimensional computer graphics is to produce a two-dimensional image of a scene or an object from a description or model of the object.

There is no unambiguous definition what visualization is. We can assume, that a **visualization** is a technique used to present an information, usually in graphical form.

In 3D computer graphics the essential part are **3D models**. 3D model is a “*mathematical representation of three-dimensional object (real or imagined) in a 3D software environment*” [7]. Models can be viewed by and any angle, they can be scaled, rotated, translated, mirrored and it is possible to apply other transformations. Model consists of points, which are placed in 3D coordinate system and connected with each other. To apply transformations, every point has to be recalculated. Then it is necessary to recalculate the view – again by recalculating the position of each point projected onto 2D scene. Three-dimensional graphics require great number of calculations.

1.1.1. Representation of 3D object

To place some points or object in space it is necessary to specify their location relative to some reference point. Usually used reference system is a **Cartesian plane**. In two dimensions, it represents surface by vertical and horizontal lines called axes. Horizontal line represents *x-axis* and vertical line is for *y-axis*. Point where lines intersect is called *origin* and its coordinates are $x=0$ and $y=0$.

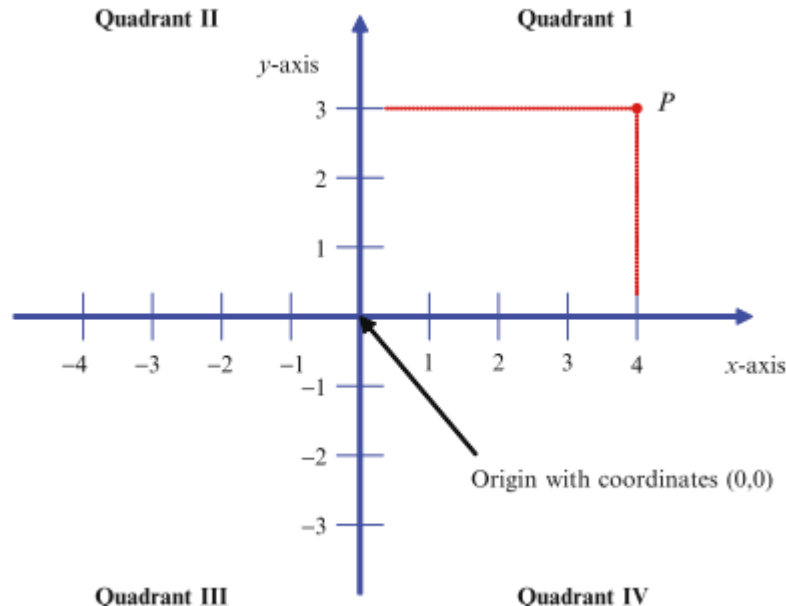


Figure 1.1: Point $P=(4,3)$ presented in Cartesian plane divided into four quadrants

Source: [1], page 69

3D is 2D with one extra dimension, usually called *z*. It means that the representation of points in a 3D space is achieved through the inclusion of a third *z-axis* which is orthogonal to the *x-axis* and *y-axis*. While points in 2D space were represented by two coordinates, each point in 3D space is represented by three coordinates (x, y, z).

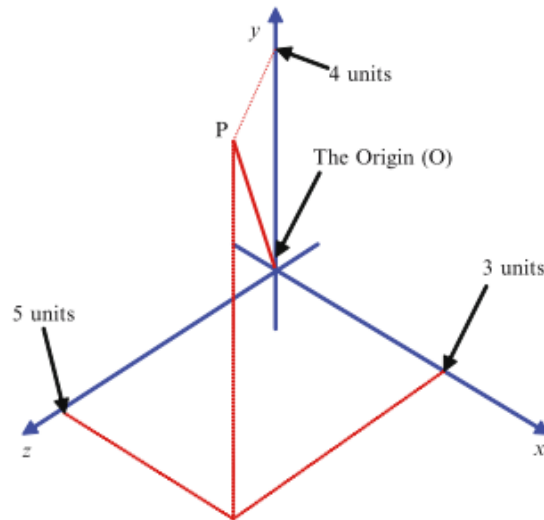


Figure 1.2: Point $P=(3,4,5)$ representation in 3D space
Source: [1], page 259

The most common way of representing 3D models is the *Polygonal Model*. It is a classic representation of 3D forms. Model is represented by a **mesh** of polygonal **faces**. Models are built on primitives, which are the basic geometric shapes like cubes, spheres or cylinders. Primitives are modeled, shaped and manipulated in other ways to create desired object. To mathematically present some shape it is necessary to have the list of **vertices** of a model. It is essential to know which points are connected by an **edge** and which are not. In the simplest case a polygon mesh is a structure that consists of polygons represented by a list of linked (x, y, z) coordinates that are the polygon vertices.

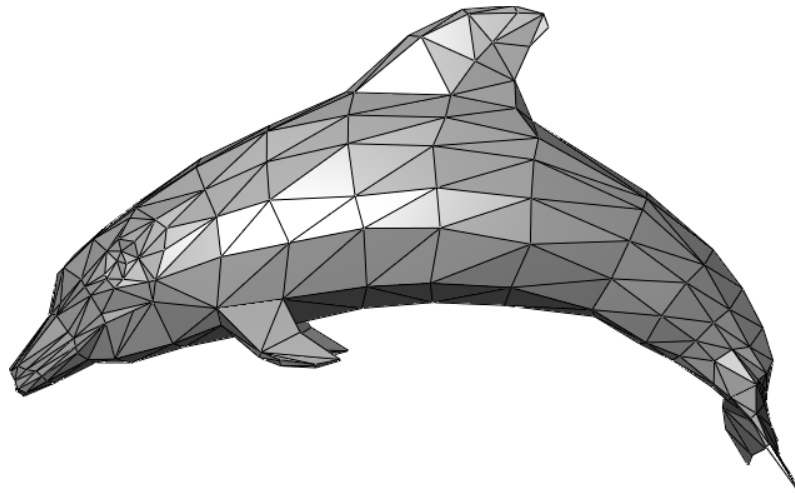


Figure 1.3: Mesh of 3D dolphin model
Source: http://en.wikipedia.org/wiki/File:Dolphin_triangle_mesh.png

3D polygonal models consist of the following parts:

- vertices – a points in 3D space
- edge – a line that connects two vertices
- faces – are flat surfaces limited by edges
- polygons – groups of faces
- surfaces – groups of polygons

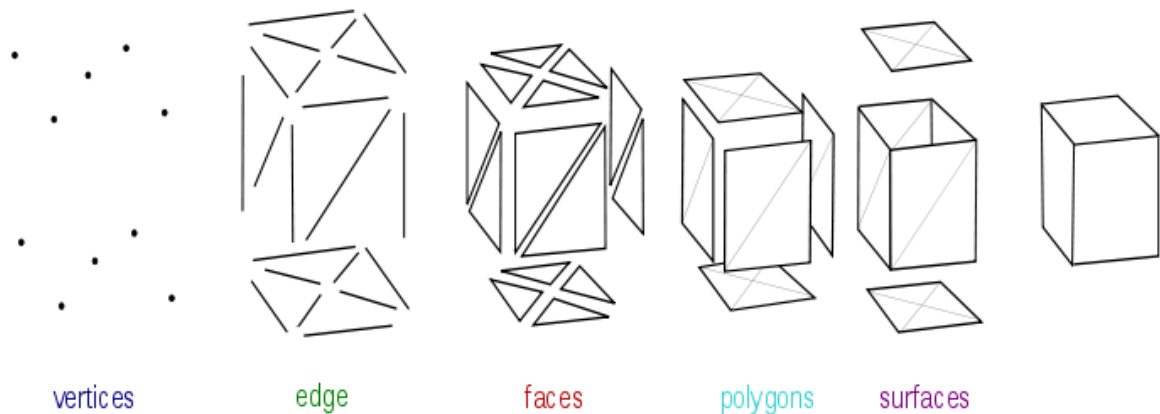


Figure 1.4: Elements of polygonal model

Source: http://en.wikipedia.org/wiki/File:Mesh_overview.svg

The number of polygons in a mesh is very important. The more polygons the model has, the more detailed it becomes. But with the growth of granularity, the computation power needed to calculate the model is also growing. To describe the granularity of a model *low poly* or *high poly* names are used. *High poly* means that model consist of many polygons, while *low poly* indicates meshes with smaller number of polygons.

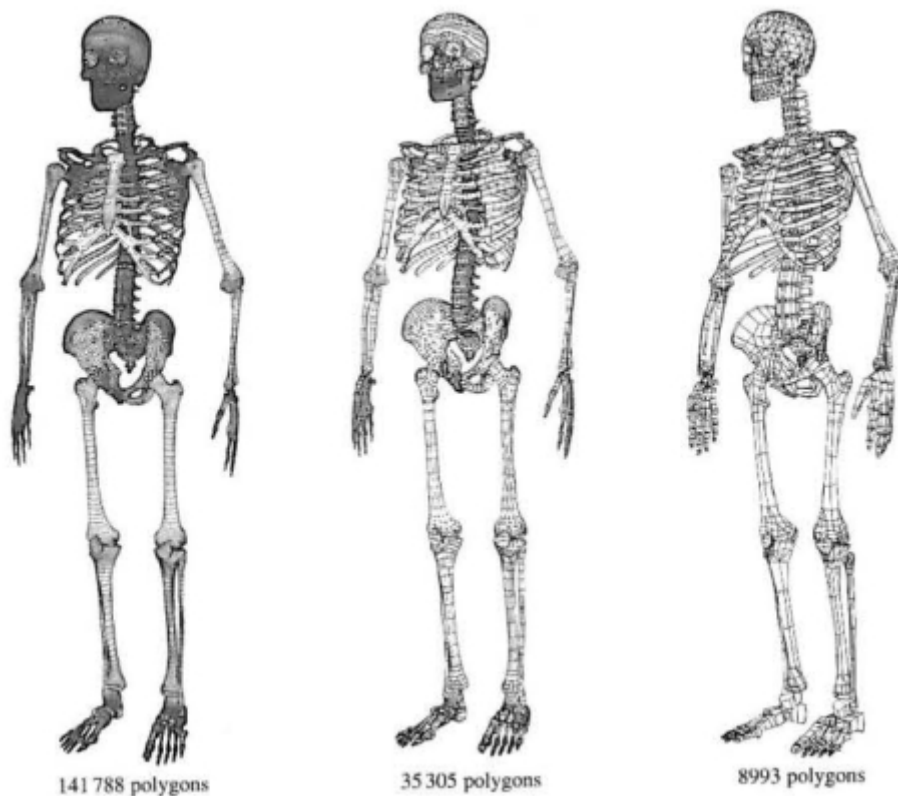


Figure 1.5: Difference between the number of polygons and the granularity of a model

Source: [9], page 30

Instead of polygons, often used shapes for faces are triangles, since triangles are the simplest shapes to calculate.

1.1.2. Projections and camera

To represent 3D scene on 2D screen, it is necessary to use **projections** – mappings 3D objects on 2D plane, which are in other words simply associations between points on 3D object and points on a plane. Projections can be described as geometric transformations. “*For the representation of a three-dimensional scene, the viewer’s position and the projection plane need to be defined. The viewer looks in the direction of the projection plane, which can be interpreted as a kind of window behind which the virtual world lies. The projection of an object onto this plane is obtained by connecting the points of the object with the centre of projection and computing the intersection points of these lines, called projectors, with the projection plane*” [7]. This description is about *perspective* projection. There are two types of projections which are usually used: *perspective* and *parallel*. *Parallel* projection is when center of projection is infinitely far from the projection plane that the projectors become parallel.

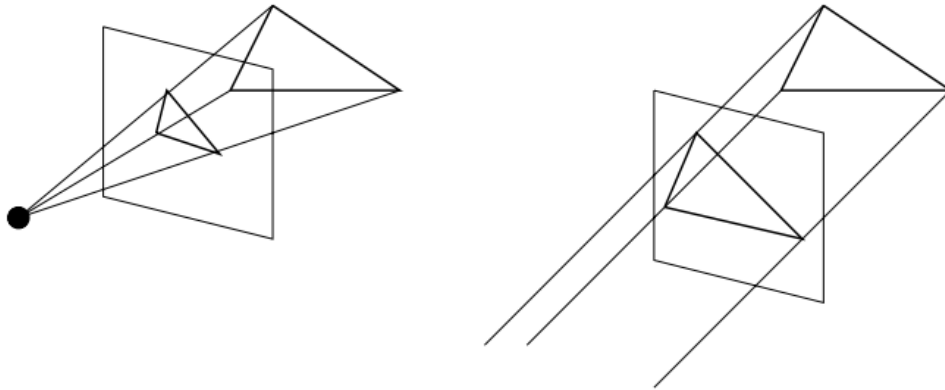


Figure 1.6: Perspective and parallel projections

Source: [7], page 140

Commonly used projection is perspective projection. It simulates the real world perspective, thus it produces more realistic outcomes than parallel projection.

To display a three-dimensional scene, the visible objects must be determined. There are two aspects of determining visibility. Firstly, **clipping** will remove all objects that are not in the range that the viewer can oversee. Rendering is only necessary for the objects inside the clipping region. Second aspect is that an object may be in the viewer’s range, but it is invisible because it is behind other object.

To properly render 3D scene, several elements are needed: the coordinates of the point where the viewer stands, direction where the viewer looks, the projection plane (usually rectangular) and an angle of view. This angle determines how far the field of view extends to the left and the right from the viewer, which indicates the width of the **clipping rectangle** on the projection plane (height can be chosen proportional to window on which the scene is displayed). The front clipping plane specifies the shortest distance in which objects can be seen and the back clipping plane defines the largest distance in which objects are still in focus. The clipping volume is the space which can be seen and has the shape of a pyramid with cut off tip. All object outside the clipping volume are not visible.

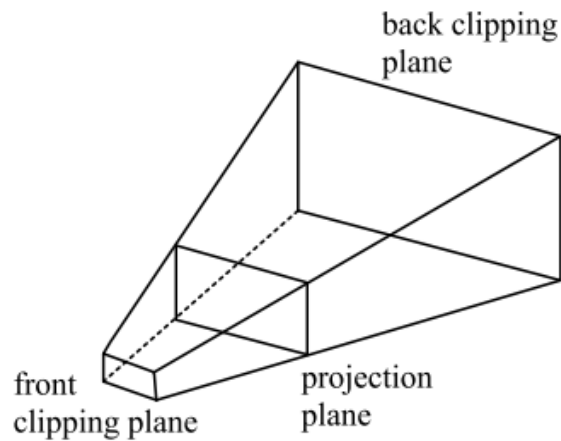


Figure 1.7: Clipping volume of perspective projection
Source: [7], page 181

The problem of determining which objects are visible and which ones are not is referred to as *hidden line* and *hidden surface elimination* or *visible line* and *visible surface determination*. There are several algorithms which can solve this problem, for example *back-face culling*, *spatial partitioning* or *z-buffer algorithms*. The aim of this algorithms is to determine which parts of an object are visible and which are not. This topic is wide and is not covered in this thesis.

1.1.3. Shadows, lights and textures

Proper representation of an object requires to introduce light, shadows and textures. This section contains a very general information about given problems.

Lights

It is important for the scene to be illuminated. There are several forms of light. The first is an *ambient light*. Such light is a light that do not come from the specific light source and has no direction, its intensity is the same everywhere. Because of its complicated computation, ambient light is rarely used in real-time computer graphics. The next type of light is a *directional light*. As the name indicates, such light has one source and specified direction. Such light is often used to model light coming from almost infinite distance, for instance sun. Third form is a *point light*. It has position and rays are spread in all directions. The intensity of a light weakens with the distance. The next type is *spotlight*, which has direction, source and its rays spread in a form of a cone. The intensity of spotlight is becoming smaller with the increase of distance and on borders of a cone, but the lighten surface grows with the distance.

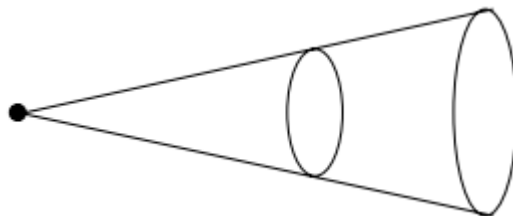


Figure 1.8: Cone of light from spotlight
Source: [7], page 204

Illumination and shading

To achieve illumination and shading effects, it necessary to specify for all surfaces of objects in the scene how they reflect light to synthesize the interaction between objects and

light. Lets assume that non-mate object is illuminated. There are two main types of light reflections: *specular* and *diffuse*. *Specular reflection* occurs when light from single direction is reflected into single outgoing direction. Such reflection is best visible on glossy surfaces. The next type is *diffuse reflection*. In contrary to the *specular*, *diffuse reflection* reflects the incoming ray into many directions.

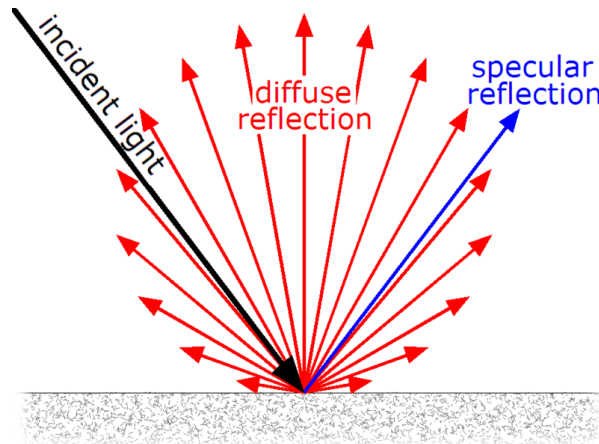
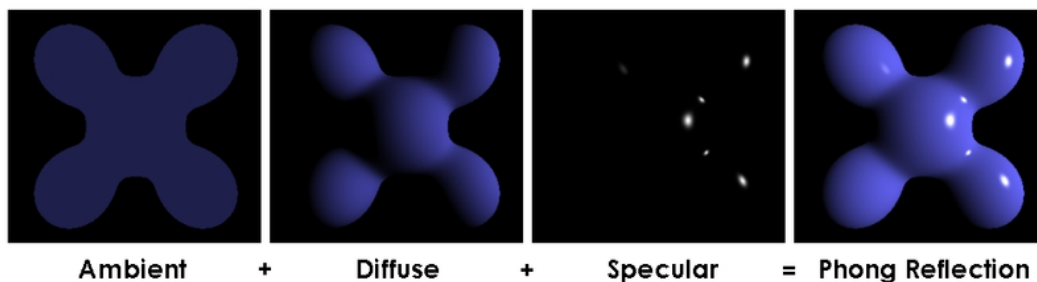


Figure 1.9: Specular and diffuse reflection
Source: <http://en.wikipedia.org/wiki/File:Lambert2.gif>

Different reflections produces different effects. The next important factor is the type of the light. There are many algorithms for calculating reflections of the materials. One of the most popular is *Phong reflection*.



Ambient + Diffuse + Specular = Phong Reflection

Figure 1.10: Phong Reflection with ambient lighting

Source: http://en.wikipedia.org/wiki/File:Phong_components_version_4.png

Inevitable part of illumination are *shadows*, which present depth by different levels of darkness. The simplest form of shading is *flat shading*. It is relatively easy method and do not require many computational powers, but it does not present realistic effect.

The better results brings the usage of different algorithm based on interpolation. The most popular are *Phong shading* and *Gouraud shading*. *Phong* method is more precise and produces better outcome, but it requires more calculations, thus more time is needed, than in case of *Gouraud shading*.

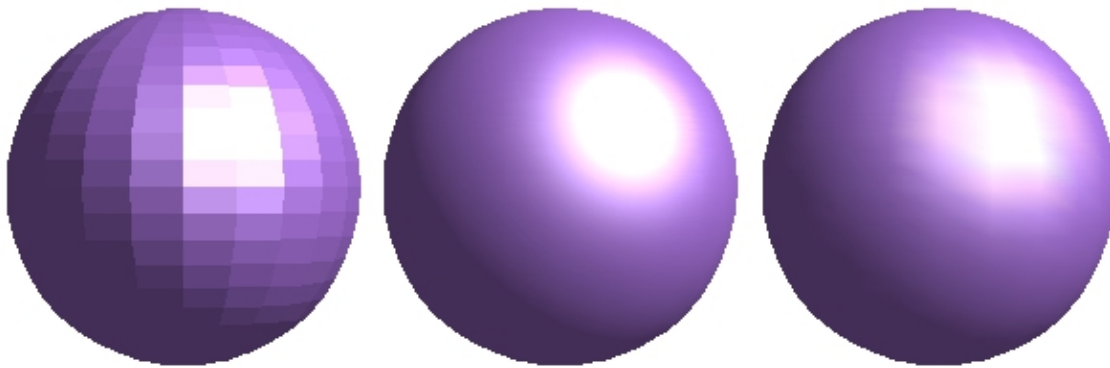


Figure 1.11: From left: Flat, Phong and Gouraud shading
 Source: <https://wiki.openoffice.org/wiki/File:DG3Ch7F25.png>

Textures

A texture is an image that is *wrapped* around the object. Such wrapping is called *mapping*. In addition to two-dimensional properties like color and brightness, a texture is also encoded with three-dimensional properties like transparency and reflectivity. Textures are used in many fields of computer graphics. Texture can be defined as a background, without the need to be assigned to any surface. They may be also used for more complex illumination techniques, which help to render more realistic images.

1.2. Characteristic of mobile devices

Mobile device is a small, hand-held computing device which allows to process, receive or send the data. It typically has a touch screen, sometimes mini-keyboard and is light weighted. Such device usually has operating system, is capable of running dedicated applications and connect to the Internet. It also can be equipped with Wi-Fi, Bluetooth, GPS.

There are several types of mobile devices: smartphones, tablets, PDAs, calculators, digital cameras, portable media players (like mp3 or mp4 players, CD players, iPods), portable consoles (Gameboy, Nintendo 3DS, PlayStation Vita), head-mounted displays and many more.

The main focus of this thesis are **smartphones**. Smartphone is a mobile phone with a mobile operating system. Smartphones allow not only to make telephone calls or sending text messages, but they became something like personal assistants or even personal computers.

Modern smartphones have several hardware elements like accelerometer, touch screen, camera, wifi, bluetooth, etc. They have similar capabilities like computers from few years ago. But in case of 3D rendering there is a problem of resources. In most computers along with *Central Processing Unit (CPU)* which is responsible for performing many operations, including arithmetical and logical operations, there is also *Graphics Processing Unit (GPU)*, which is responsible for calculations concerning displaying graphics. In older smartphones *GPU* is absent, so the all calculations are performed in device's processor, although newest smartphones have *GPU*.

In mobile devices with touch screen it is possible to navigate with **gestures**. Gestures in case of mobile touch screen are the specific hand movements on the surface of the screen. Smartphone recognizes the gesture and performs action related to it. There are some predefined actions, but it is possible to create own responses to the gesture.

2. Implementation

2.1. Selected mobile platform: Android



Figure 2.1: Android logo

Platform selected to implement the application corresponding to the thesis topic is *Android*. The selection of this platform is motivated by several factors: it is free, open, has easy accessible documentation and huge community, devices with android are relatively cheap and to create applications there is no need to have special and expensive hardware or software. What is more, it is the most popular platform now, so it is worth to learn.

Android is an operating system based on Linux kernel. Initially it was released by *Android Inc.*, but later was bought by *Google* in 2005. In 2007 the project was shown to the public, but first device with *Android (HTC Dream)* was released at the end of 2008. It is designed primarily for touchscreen mobile devices. There are several versions of this OS. The third version of the system, but first released to public was Android 1.5. *Cupcake*. The latest is called *Lollipop* and the version number is 4.5 or 5 (still not precised).

Android architecture was designed to produce less errors. Every application works inside its own virtual machine, which is called *DALVIK*. *DALVIK* was optimized for the needs of mobile devices. It is based on *Java* virtual machine. Such solution lessens the probability of permanent device failure caused by invalid application. Lately *Google* started to introduce new system for running Android applications – *ART*, but this solution is not yet commonly used.

Basic system services are handled by *Linux* kernel, which works as a hardware abstraction layer, which mediates between hardware and *Android* stack. System kernel is responsible for many functions including memory management, processes and threads management, providing access to drivers for display, buttons, camera, *Wi-Fi*, memory cards, sound system.

In Android it is possible to use well knows libraries including libraries for: databases (*SQLite*), web browsing (*Ssl*, *WebKit*), computer graphics (*SGL*, *OpenGL ES*), audio and video (*MPEG*, *MP3* and others), hardware and location-based services (*Wi-Fi*, *Bluetooth*).

2.2. Selected 3D engine: Unity

To create application for this thesis, the *Unity* engine will be used. It is very powerful engine with visual editor, which is very intuitive and quite clear even for novice. It is possible to write scripts in *JavaScript* and *C#*. I am programming in *C#*, so selection of this platform was strongly connected with this fact. Furthermore, there is huge community and clear documentation for the engine. Cross-platform in case of *Unity* concerns both the possibility to run engine on many platforms, as well as create applications for many platforms.

Unity is a complete 3D environment with built-in *IDE* called *MonoDevelop*. The user interface of the editor is well organized and panels can be fully customized.

Engine provides many facilities which makes developing 3D games easier. Documentation of the engine is detailed and for most functions are provided examples. What is more, *Unity* has its own community, where it is easy to find solution for a problem.

It is possible to publish project to several platforms. It is not necessary to rewrite whole code to change the platform, but some functionalities are handled differently in different platforms. Nevertheless, to handle iOS and Android devices, mostly the same code is used, so it is easy to switch between platforms.

Unity is nice engine for 3D applications. It is easy to use and to learn, free for educational purposes and small businesses.

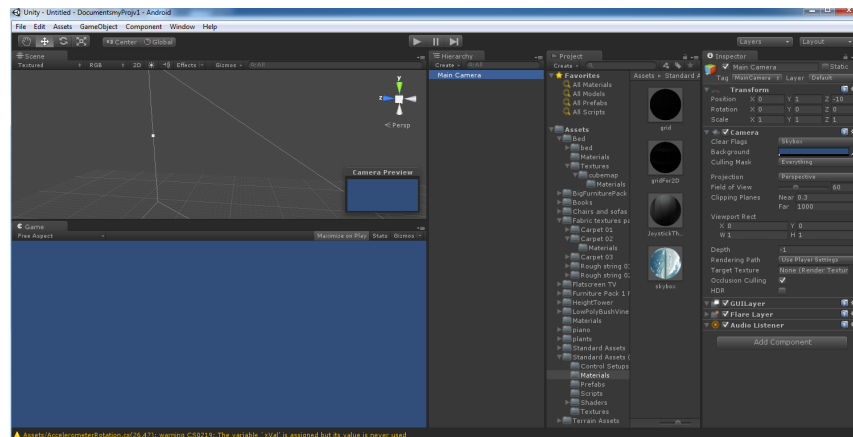


Figure 2.2: Unity interface

2.3. Project overview

The goal is to create application which will visualize a bedroom. Also the main menu will be provided. Application will use several features of mobile devices: accelerometer and touch screen. Application will allow to interactively play with the scene: walk through room and rotate around.

During the *Kreator Innowacyjności* course I learned a lot about 3D world thanks to OpenGL classes, as well as 3D modeling workshops, so it was easier for me to start working in new environment.

3D rendering requires large amount of complex computations, so it is very resources-consuming. Since application will be built for smartphones or tablets, the amount of details like shadows, complicated lights and shades should be reduced to provide possibility of interaction. Mobile devices usually have less capabilities than standard computers, do not have graphic cards and because of that effectiveness of rendering a 3D scenes is not very high compared to average PC. Nevertheless, newest smart phones has more efficient processors and include graphical processing unit, so the capabilities for such devices are high.

2.4. Setting the scene

In *Unity* there is an editor, where it is possible to create a scene. At first there are only grid and coordinate system provided. It is possible to place some basic shapes using menu *GameObject->Create Other* and then selecting desired object. Among possible choices there are also some engine-specific object, but for now it is not important.

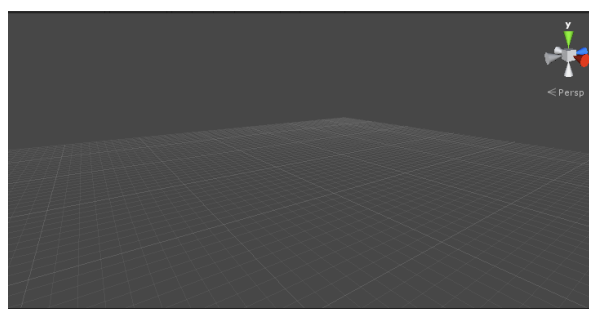


Figure 2.3: Empty scene

The first step is to create a *plane*. It is a surface where the “world” will be placed. It can be created in the menu mentioned above. Any object in the editor can be transformed using menu in the left top corner. It allows to (from left) move the scene view, move (translate), rotate and scale currently edited object. Object can be selected by clicking on it in the editor or by selecting the object name in *Hierarchy* window.



Figure 2.4: Transformations toolbar

Unity has an *Asset Store*. It is a place where it is possible to get various assets for game, either free or paid. *Asset Store* is very convenient, because models are ready to be used in the editor, without necessity to import external models. After downloading the object, it is necessary to import the asset into project. The asset appears in browser of *Project* section. Along with the model, all textures and materials are provided. The only difficulty is to apply proper material if model was not a ready prefab with all parts. In that case it is enough to find proper material, drag it and drop on the object. Of course there is a possibility to import external models and use them in project.

Firstly I created plane, which has gray color. To change it it was necessary to import new texture. Texture was taken from one of the imported assets.

The next step was to build a room. For this purpose simple cubes were used. Cubes then were transformed in such way, that they created walls of a room.

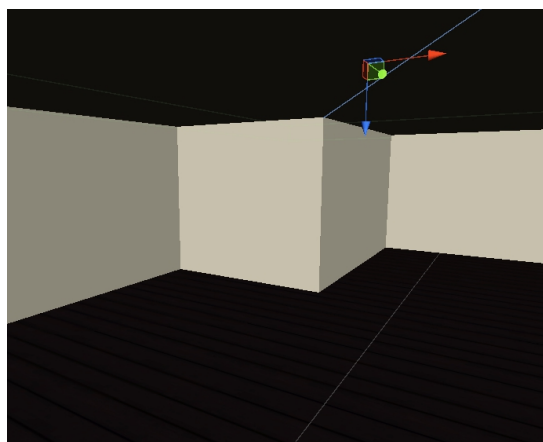


Figure 2.5: Walls

Using imported assets, I created a room. I also downloaded several models from the internet. Here is list of sources of used models and textures both from asset store and internet:

- Bookcase 1 – source: <http://tf3dm.com/3d-model/bookcase-71688.html>
- Bookcase 2 – source: <http://tf3dm.com/3d-model/bookcase-77347.html>
- Drawer – source: <http://tf3dm.com/3d-model/drawer-33074.html>
- Free Furniture Pack 1 – source: <https://www.assetstore.unity3d.com/#/content/11859>
- Props for classroom – source: <https://www.assetstore.unity3d.com/en/#!/content/5977>
- Free Furniture Props – source: <https://www.assetstore.unity3d.com/#/content/8822>
- Sleeping bed – source: <https://www.assetstore.unity3d.com/#/content/8329>
- Coffee table – source: <http://tf3dm.com/3d-model/low-poly-stylish-modern-desk-table-6006.html>
- Small Plants – source: <https://www.assetstore.unity3d.com/#/content/6930>

- Chairs and sofas pack – source: <https://www.assetstore.unity3d.com/#/content/6411>
- Fabric textures pack – source: <https://www.assetstore.unity3d.com/#/content/13002>
- Books texture – source: <http://www.morguefile.com/archive/display/87396>
- Clothes texture – source: <http://www.morguefile.com/archive/display/903671>
- Pinboard – source: <http://tf3dm.com/download-page.php?url=pinboard-45710>
- Glass – source: <http://tf3dm.com/3d-model/glass-91748.html>
- Fruits basket – source: <http://tf3dm.com/3d-model/fruits-27139.html>
- Door – source: <http://tf3dm.com/3d-model/door-with-small-windows-99132.html>
- Lamp – source: <http://www.3dmodelfree.com/models/33494-0.htm>
- Bulb – source: <http://tf3dm.com/3d-model/bulb-lamp-8543.html>
- Chocolates – source: <https://www.assetstore.unity3d.com/en/#!/content/14187>

It was long process to place and to fit all objects into desired scene. After placing the furniture, there was time to add some lights.

Lights are very important thing. Without them the whole scene does not look good or even is not visible at all. In Unity lights are used to simulate the sun, light bulbs, flashlights and other sources of light.

I added three spot lights (from both windows and lamp on the ceiling) to light everything. It helped to take a better look at the scene and to fix remaining mistakes. I added also point lights where lamps were placed, but lately I decided to turn them off.

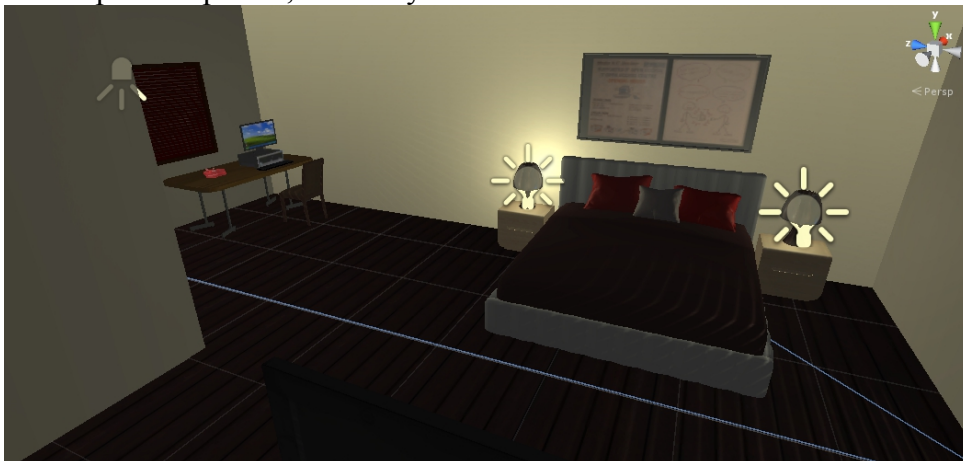


Figure 2.6: Furniture added to a scene

To imitate the sunlight shining through the blinds in windows, I used *cookie*. *Cookie* is a texture used to give a shape to the light. I created simple stripped pattern and added to the light as a *cookie*. It is visible both on bottom image (Figure 2.5).

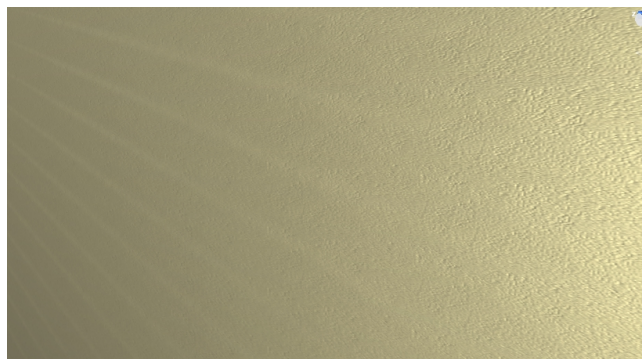


Figure 2.7: Cookie on wall

There are several places, where lights are illuminated by the surfaces. It is done thanks to surface shaders. There are several shaders available through components, but there is also a possibility to create own shader and attach it to the object. There are variety of shaders and each of them produces different effect. I tested many shaders for each object to produce as good effect as I could.

After setting the lights and shaders I started to add more detail to the scene.

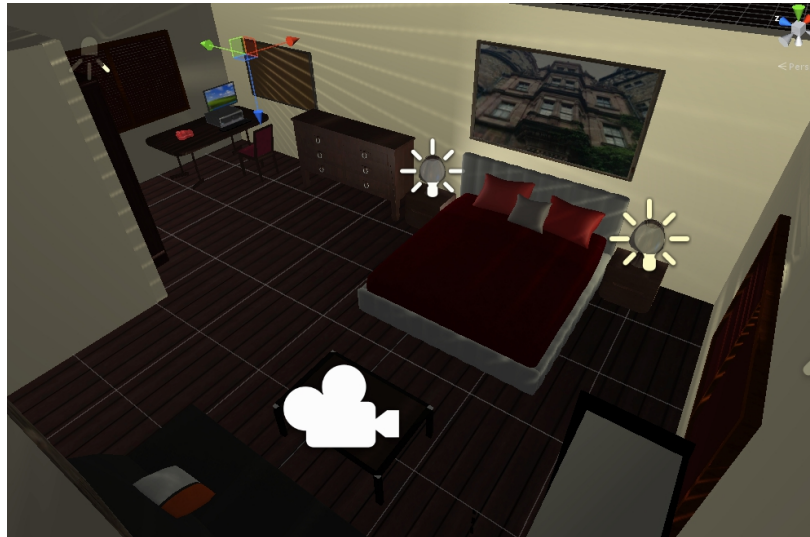


Figure 2.8: Adding more details

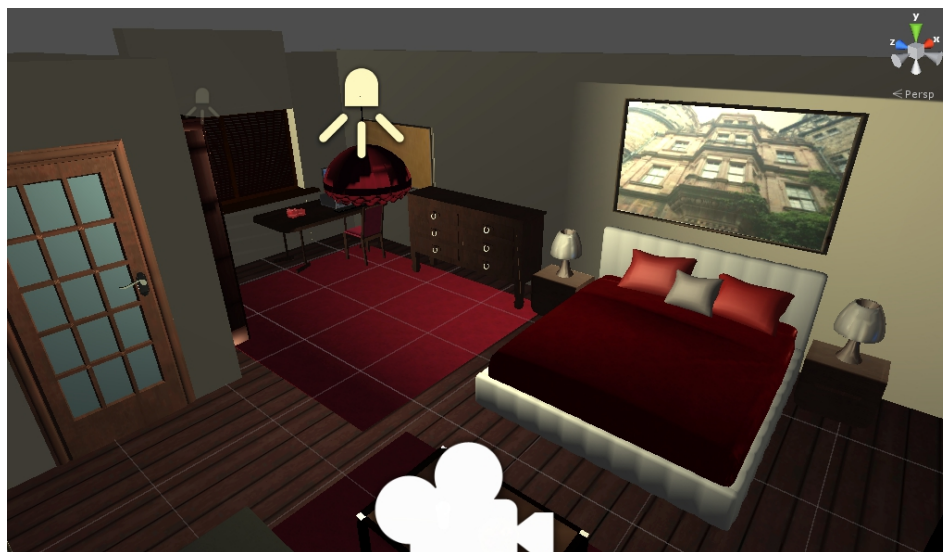


Figure 2.9: Final scene

I placed camera in the middle of the room to give it a good view to the scene. Later camera will be programmed to move around the room.

2.5. Programming the scene

2.5.1. User Interface

After setting the scenes, it was time for making user interface. Firstly, I made a screen of my scene. Then in *Photoshop* I made simple background with buttons to navigate and launcher icon.

On main screen user should have the possibility to go to the scene, view about note or exit application. While viewing the scene, there will be possibility to return to main menu. To achieve this goal, simple HUD (*Head-Up Display*) will be provided.

To create the HUD in Unity it is necessary to use *UnityGUI*. It requires to write several lines of code to create buttons or other graphical user interface elements and attach the script to the camera.



Figure 2.10: Simple interface for application

```

57
58     if (GUI.Button (new Rect (Screen.width*0.5f - 270,Screen.height*0.5f,539,138), btnAbout, style)) {
59         currentScreen = GuiScreens.AboutScreen;
60     }
61
62     if (GUI.Button (new Rect (Screen.width*0.5f - 270,Screen.height*0.7f,539,138), btnExit, style)) {
63         Application.Quit();
64     }
65 }
66
67 else if(currentScreen == GuiScreens.AboutScreen){
68     GUI.DrawTexture(new Rect(0,0,1920,1080), backgroundTxt);
69
70     GUI.DrawTexture(new Rect(Screen.width*0.5f - 315,Screen.height*0.5f - 330,630,714), backgroundAboutTxt);
71
72     if (GUI.Button (new Rect (90, Screen.height * 0.5f - 90, 181,181), btnBack, style)) {
73         currentScreen = GuiScreens.FirstScreen;
74     }
75 }

```

Figure 2.11: Code snippet for programming GUI

2.5.2. Accelerometer rotation and touch

Modern mobile device have built-in accelerometers. Accelerometer is a device that measures acceleration. They are used for detecting rotations and vibrations. In this application accelerometer will be used to rotate the camera around to look around the room in the scene.



Figure 2.12: Accelerometer
Source: <http://www.reactable.com/mobile/manual/accelerometer.html>

In *Unity* there is interface called *Input*, which is responsible for tracking users actions. In case of *Android*, *Input* is capable of detect multiple fingers touching the screen simultaneously. It is capable also to detect the changes made by accelerometer. By using proper variables and functions of *Input*, it is possible to program scene interactively. Accelerometer is also handled by *Input*.

There are several types of touch phases:

- began – finger touches the screen
- moved – finger was moved on the screen
- stationary – finger touches the screen, but does not move
- ended – finger was lifted from the screen
- canceled – system stopped tracking the touch

To walk back and forth is necessary to put finger on the screen. To walk continuously, the finger has to touch the screen all the time, thus it has to be detected all the time. That fact indicates that *stationary* is the most suitable phase for such gesture.

To move forward it is necessary to put finger in the upper part of the screen, to move backward – in lower part. The idea is simple, but the tricky part was to determine the direction of movement, because back and forward directions are dependent on the direction of the camera. It is possible to determine the direction of the camera by properties of camera itself. To move forward with known camera direction it is enough to add the current object's position and camera heading.

Input calculates the acceleration value for the all axes. Then it is necessary to apply rotation of the camera by the acceleration. The rotation has to be performed in horizontal axis, so this is enough only to modify the part for *y-axis* in the scene. But there was a difference between axes orientation used in device and in scene: to produce proper rotation, the *y-axis* in scene has to be rotated by the negative acceleration of *x-axis* of the device. Application is in left-landscape orientation.

```
float yVal = Input.touches[0].position.y;
if(Screen.height*0.6f>yVal)
{
    Vector3 position = this.transform.position;
    position = position-speed*cam.transform.forward*Time.deltaTime;
    this.transform.position = position;
}

if (Screen.height*0.7f<yVal)
{
    Vector3 position = this.transform.position;
    position=position+speed*cam.transform.forward*Time.deltaTime;
    this.transform.position = position;
}
```

Figure 2.13: Code snippet for moving the camera

2.6. Results

Tested on device *LG Google Nexus 5* with *Android* version 4.4 *Kitkat*, 2048MB RAM, Quad-core 2.3 GhzKrait 400 processor and Adreno 330 GPU with FULL HD IPS display.

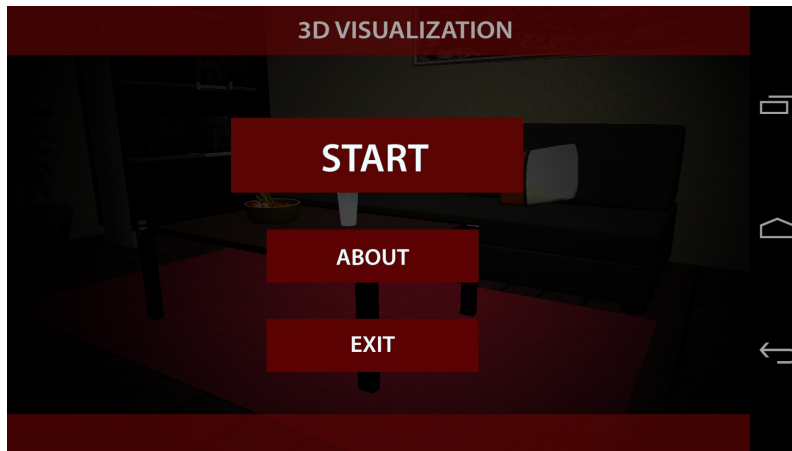


Figure 2.14: Main menu



Figure 2.15: Fragment of a scene (1)

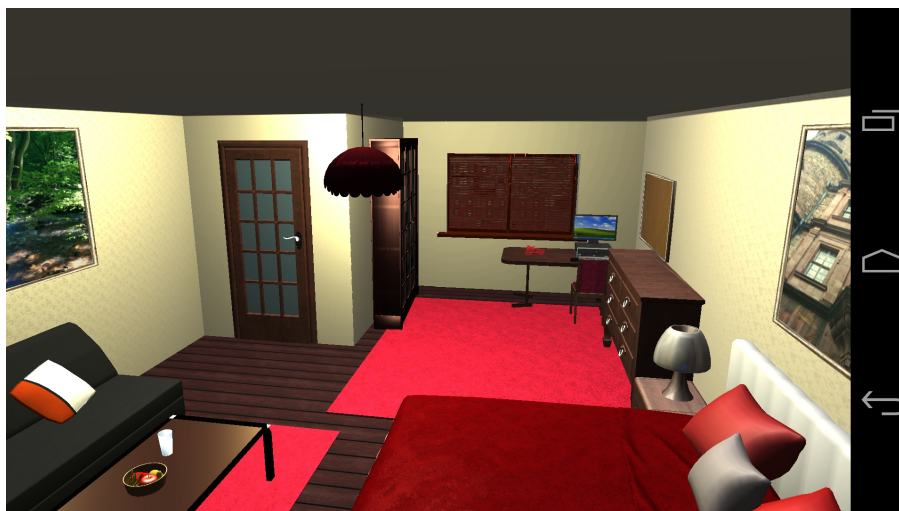


Figure 2.16: Fragment of a scene (2)

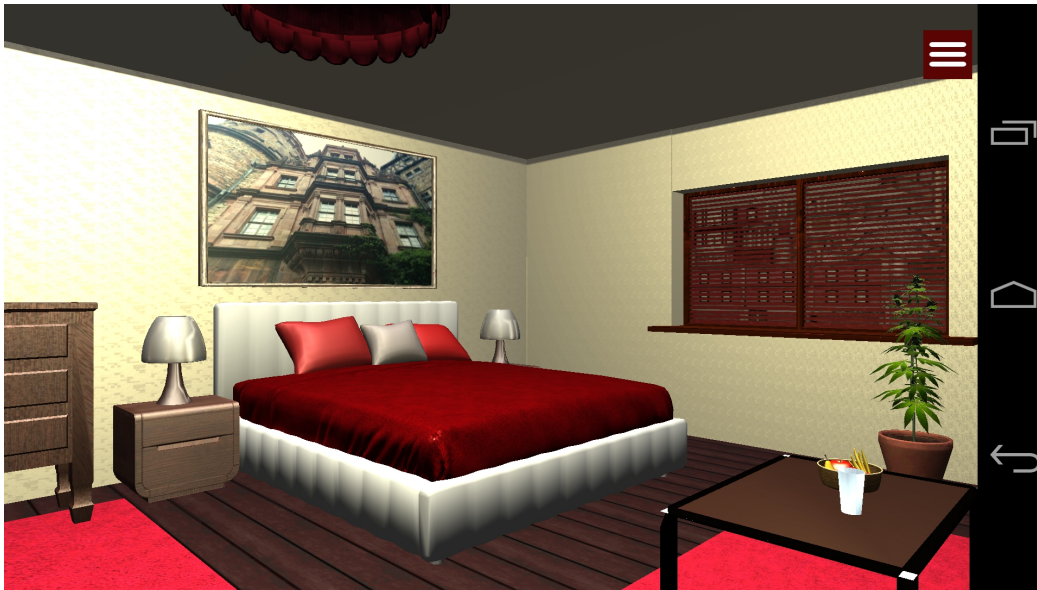


Figure 2.17: Fragment of a scene (3)



Figure 2.18: Fragment of a scene (4)



Figure 2.19: About page

3. Conclusion

In this thesis I tried to give a try to develop application to visualize 3D objects on mobile devices, precisely on smartphones. Firstly I described the mathematical base for displaying scenes in three dimensions on 2D display to understand the way computer graphics is rendered. I also provided a short description what are the mobile devices. In the last chapter implementation was presented, as well as screens from test of the application.

3.1. Cognitive conclusions

Application on tested device worked fluently and scene was rendered in different quality than it was visible on the computer screen, which may be caused by different management of 3D by graphical cards of both devices.

Produced result was a trade-off between the number of elements and their complexity (thus visual effect), and limits of a mobile device, which was used for testing. It was not easy to do, since graphics in three dimensions is very hard in calculations. Its complexity is visible in math of simple points transformations, projections or the overall concepts lying behind rendering 3D scene on 2D display. Fortunately, with the development of the technology, computers become capable of handling those calculations. What is more, libraries, applications and engines for 3D rendering are available, making it relatively easy to create complex 3D models or to create scenes and 3D games.

Some time ago I had a possibility to test similar application on two devices – one of them was one of the flagship phones of known manufacturer and the second one was a cheaper, budget model. The differences in quality, speed and performance were tremendous. While testing application from this thesis on flagship form end of last year, the differences also were significant. It is clear that smartphones are becoming more efficient every year.

3.2. Further usage

3D visualizations are used to present various data. From simulations of machines, to building houses. Applications to visualize interiors are commonly used for commercial purposes. In case of designing a house or the decorations for the office, customers like to see how ordered product look like before it will be built in reality. The possibility to walk through the visualized place is causing that application is more attractive and makes better impression than simple image render. The fact that it is possible to create visualizations for mobile devices, causes such applications to be available for viewing in every place and every time. Smartphones have integrated sensors like accelerometer, which enrich the interaction with application.

Such project can be used for example by a company which produce furniture to attract customers and present products in realistic way. It can also be a tool to present to a customer exemplary arrangements of furniture offered by a company without necessity to arrange room in reality. Thank to such application company can show their offer not only to people in cities nearby their showroom, but also to people from whole county or even whole world.

Those reasons allow me to conclude that 3D interactive object visualization is a promising and very interesting concept, worth further development. Unfortunately, mobile devices are still weaker than PC when it comes to the computational resources, but it is a matter of time.

Bibliography

1. Blundell Barry G., *An Introduction to Computer Graphics and Creative 3-D Environments*, Springer-Verlag, London, 2008, pages: 11-28, 67-103, 255-321, 335-355.
2. Bura J., *Pro Android Web Game Apps Using HTML5, CSS3 and JavaScript*, Apress, 2012, pages: 333-356.
3. Cassavoy L., *What Makes Smartphone Smart?*, [online], [access: 16 August 2014], Available on: http://cellphones.about.com/od/smartphonebasics/a/what_is_smart.htm
4. Chu P., *Learn Unity 4 for iOS Game Development*, Apress, 2013
5. Conder S., Darcey L., *Android: programowanie aplikacji na urządzenia przenośne*, Edition II, Helion S.A., Gliwice, 2011, pages: 35-60, 441-473.
6. Copeland J., *Chapter One: Introduction*, 2010, [online], [access: 14 August 2014], Available on: <http://www.colossus-computer.com/sample.htm>
7. Klawonn M., *Introduction to Computer Graphics Using Java 2D and 3D*, Springer-Verlag, London, 2008, pages: 139-145, 179-182, 201-229
8. Slick J., *Anatomy of a 3D Model*, [online], [access: 18 August 2014], Available on: <http://3d.about.com/od/3d-101-The-Basics/a/Anatomy-Of-A-3d-Model.htm>
9. Watt A., *3D Computer Graphics*, Edition III, Addison-Wesley, Harlow, 2000, pages: 27-37.
10. *Mobile device*, [online], [access: 20 August 2014], Available on: http://en.wikipedia.org/wiki/Mobile_device
11. *Mobile operating system*, [online], [access: 20 August 2014], Available on: http://en.wikipedia.org/wiki/Mobile_operating_system

Figures

Figure 1.1: Point $P=(4,3)$ presented in Cartesian plane divided into four quadrants Source: [1], page 69.....	
Figure 1.2: Point $P=(3,4,5)$ representation in 3D space Source: [1], page 259.....	
Figure 1.3: Mesh of 3D dolphin model Source: http://en.wikipedia.org/wiki/File:Dolphin_triangle_mesh.png	
Figure 1.4: Elements of polygonal model Source: http://en.wikipedia.org/wiki/File:Mesh_overview.svg	
Figure 1.5: Difference between the number of polygons and the granularity of a model Source: [9], page 30.....	
Figure 1.6: Perspective and parallel projections Source: [7], page 140.....	10
Figure 1.7: Clipping volume of perspective projection Source: [7], page 181.....	11
Figure 1.8: Cone of light from spotlight Source: [7], page 204.....	11
Figure 1.9: Specular and diffuse reflection Source: http://en.wikipedia.org/wiki/File:Lambert2.gif	12
Figure 1.10: Phong Reflection with ambient lighting Source: http://en.wikipedia.org/wiki/File:Phong_components_version_4.png	12
Figure 1.11: From left: Flat, Phong and Gouraud shading Source: https://wiki.openoffice.org/wiki/File:DG3Ch7F25.png	13
Figure 2.1: Android logo.....	14
Figure 2.2: Unity interface.....	15
Figure 2.3: Empty scene.....	15
Figure 2.4: Transformations toolbar.....	16
Figure 2.5: Walls.....	16
Figure 2.6: Furniture added to a scene.....	17
Figure 2.7: Cookie on wall.....	17
Figure 2.8: Adding more details.....	18
Figure 2.9: Final scene.....	18
Figure 2.10: Simple interface for application.....	19
Figure 2.11: Code snippet for programming GUI.....	19
Figure 2.12: Accelerometer Source: http://www.reactable.com/mobile/manual/accelerometer.html	19
Figure 2.13: Code snippet for moving the camera.....	20
Figure 2.14: Main menu.....	21
Figure 2.15: Fragment of a scene (1).....	21
Figure 2.16: Fragment of a scene (2).....	21
Figure 2.17: Fragment of a scene (3).....	22
Figure 2.18: Fragment of a scene (4).....	22
Figure 2.19: About page.....	22

Attachments

CD with source files for the project (*Unity* project, folder: *ProjektDominikaCis*), movie with application demonstration (file: *vis3dDominikaCis.mp4*) and digital version of this paper.