



TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

YU GUO
3D GRAPHICS PLATFORMS AND TOOLS FOR MOBILE APPLI-
CATIONS
Master of Science Thesis

Examiners: Prof. Irek Defee, Dr. Heikki Huttunen
Examiners and Topic approved in the
Faculty Council Meeting on
5.5.2014

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY

Master's Degree Programme in Information Technology

YU GUO: 3D GRAPHICS PLATFORMS AND TOOLS FOR MOBILE APPLICATIONS

Master of Science Thesis, 63 pages.

May 2014

Major: Multimedia

Examiner: Prof. Irek Defee, Dr. Heikki Huttunen

Keywords: mobile application, 3D graphics, OpenGL, OpenGL ES, graphics platforms, mobile GPU, mobile platforms, 3D modelling software, 3D game engine, 3D interface, Maya, Unity, 3D animation.

The objective of the thesis is the investigation of mobile 3D graphics platforms and tools. This is important topic since 3D graphics is increasingly used in mobile devices. In the thesis platforms and tools specific for 3D graphics are analysed. Platforms are device- and operating system independent foundations for dealing with 3D graphics. This includes platforms based on 3D graphic languages: Open GL, Open GL ES, Open CL, Web GL and Web CL. Based on the platforms, there are specific tools which facilitate applications development. In the thesis the Maya software environment for the modelling and Unity 3D game engine for animation are described. Workflow for using these tools is demonstrated on an example of application relying on 3D graphics. This application is fitting of clothes for Web fashion shop. Buying clothes from Web shops is complicated since it is impossible to check how the cloth will actually fit. This problem can be attempted to solve by using 3D model of the user and cloth items. The difficulty is dynamic visualization of model and cloth interaction. Example of this process is shown in the thesis. The conclusion is that new mobile devices will soon have graphics capabilities approaching requirements for sophisticated 3D graphics allowing to develop new types of applications. More work in the mobile 3D graphics area is needed still for creating more real and less cartoon-like models.

PREFACE

I would like to thank my supervisor Irek Defee for giving me inspirations and instruction throughout the process of writing the whole thesis. We had discussions about the work and he was keeping me updated with the latest technology. In the final process of writing the thesis, he helped me to enlarge the scale of the thesis to make it more valuable in content. I would also like to delicate my thanks to my parents who give me high moral values throughout my life; this has helped me in my every life endeavour and has taught me the wonders of life and support me in my study financially. I would also thank my fellow friends who had discuss with me whenever I have problems, Yanan Mi and Yuanyuan Zhang, with whom I start the idea and we proposed it to a competition at the early stage of development, Ruibin Ye who gave me the idea of using Unity and helped me with brainstorming. Also for my friends, Qianting Liu, Yichong Yun, Yanan Ren, Peng Zhang, Chufan Shuai and Siyuan Deng, who kept me optimistic and arranged rich activities during my stay in Finland. The thesis could not have been done without the help out all of you.

Yu Guo

May 2014

CONTENTS

ABBREVIATIONS	2
LIST OF TABLES AND FIGURES.....	3
1. INTRODUCTION	4
2. BACKGROUND	6
2.1 Overview of Mobile GPU	6
2.2 3D Graphics Creation Pipeline.....	7
2.3 Hardware Support for 3D Graphics	9
3. 3D GRAPHICS PLATFORMS	15
3.1. OpenGL.....	16
3.2. OpenGL ES	22
3.3. WebGL.....	24
3.4. OpenCL	27
3.5. WebCL	29
4. 3D GRAPHICS DEVELOPMENT TOOLS.....	32
4.1. Web shop with mobile 3D graphics application	32
4.2. 3D Modelling Software Tools.....	33
4.3. Game Engines for Mobile Applications.....	35
4.4. 3D Graphics Application Generation.....	37
4.4.1. Modelling in Maya.....	38
4.4.2. Animation in Unity	40
4.4.3. Mesh Rendering	43
4.4.4. User Interface	46
4.4.5. Finger Gesture and Mouse Interaction.....	49
4.5. Exporting project.....	51
5. RESULTS AND DISCUSSION	52
5.1. Results	52
5.2. Discussion	54
5.2.1. Colliders in Animation.....	54
5.2.2. Material on Meshes	55
5.3. Designing of Overall System	56
6. CONCLUSIONS.....	59
REFERENCES.....	61

ABBREVIATIONS

CPU	Central Processing Unit
RAM	Random-access Memory
GPU	Graphics Processing Unit
GPGPU	General-Purpose Computing on Graphics Processing Units
OpenGL	Open Graphics Library
OpenGL ES	Open Graphics Library for Embedded Systems
WebGL	Web Graphics Library
OpenCL	Open Computing Language
WebCL	Web Computing Language
API	Application Programming Interface
OS	Operating System
APK	Android application package file
IOS	iPhone OS
FLOPS	Floating-point Operations per Second
HTML5	HTML5 is a mark-up language used for structuring and presenting content for the World Wide Web and a core technology of the Internet.
FBX	FBX (Filmbox) is a proprietary file format (.fbx) developed by Kaydara and now owned by Autodesk.
NGUI	Next-Gen UI kit, Plug-in of Unity for creating user interface
NURBS	Non Uniform Rational B-spline

LIST OF TABLES AND FIGURES

<i>Table 2.1. Comparisons of different GPU series</i>	10
<i>Table 3.1. Ten Open GL primitive types</i>	17
<i>Table 3.2. Performance comparison of JavaScript vs. WebGL</i>	30
<i>Table 4.1. Most popular 3D developing tools/engines for mobile devices</i>	35
<i>Table 4.2. Interactive Cloth Properties in Unity</i>	41
<i>Table 4.3. Functions for Mouse and User Interaction</i>	47
<i>Table 4.4. Variables and Functions in Unity for Interaction</i>	49
<i>Figure 2.1. 3D Graphics Pipeline</i>	7
<i>Figure 3.1. OpenGL Related Ecosystem</i>	16
<i>Figure 3.2. OpenGL Pipeline Architecture</i>	16
<i>Figure 3.3. Output of OpenGL demo</i>	18
<i>Figure 3.4. Camera in OpenGL</i>	21
<i>Figure 3.5. Pipeline 2.0</i>	23
<i>Figure 3.6. WebGL Rendering Pipeline</i>	24
<i>Figure 3.7. Output of WebGL</i>	26
<i>Figure 3.8. Architecture of OpenCL</i>	27
<i>Figure 3.9. OpenCL Memory Model</i>	28
<i>Figure 3.10. Executing OpenCL Programs</i>	29
<i>Figure 3.11. OpenGL, OpenCL, WebGL and WebGL relations</i>	30
<i>Figure 4.1. Snapshot of Body Skeleton Construction with</i>	39
<i>Figure 4.2. Snapshot of Whole Body with Skin(Left) and Skeleton(Right)</i>	40
<i>Figure 4.3. Model with colliders and output in the engine</i>	43
<i>Figure 4.4. Material Inspector and Corresponding UI Layout</i>	44
<i>Figure 4.5. Logical Process of Building UI and its output</i>	47
<i>Figure 4.6. UIButton Message Inspector</i>	48
<i>Figure 4.7. Distorted cloth when rotating model itself</i>	50
<i>Figure 4.8. Export Setting and Platform Options</i>	51
<i>Figure 5.1. Scene Organization in 3D engine</i>	52
<i>Figure 5.2 Application output UI snapshot</i>	53
<i>Figure 5.3. Material Setting Example</i>	55
<i>Figure 5.4. The Whole Application System Structure</i>	56

1. INTRODUCTION

Mobile devices enjoyed rapid evolution from telephones to advanced smartphones which are universal information devices. At the same time the number of users hugely increased. This was possible due to the progress in hardware and software on smartphone and has allowed very broad range of applications helping and entertaining people in every imaginable way.

In some ways, smartphones versatility and processing power is similar to PC but it also differs in many ways requiring special attention to developing applications [1]:

- Mobile device is personal, very portable, always connected which allows location-based services and navigation. Everybody is or will be a user of a smartphone
- The physical display size of smartphone is fixed and relatively small. Current devices are on the limit of size, some seen even too big. There is emerging another class of devices, small tablets but they are not as portable
- Displays of smartphones are now at high-definition level that is the same as PC, processing power is on the level of PC's from several years ago, multiprocessing is widely used, 64-bit mobile processors are beginning to appear
- 3D graphics has been limited in mobile devices. Recently, however quite powerful graphics processors for mobile started to appear. Their power roughly corresponds to the processing power of average PC graphics cards from a couple of years ago but it is very quickly improving
- Power is the ultimate bottleneck. Usually mobile is not plugged to the wall while using, so the duration of using solely depends on battery. But the battery only improves 5~10% per year generally while the processor and graphics performance demands a great deal of power consumption. In addition, thermal management must be considered.

The topic of the thesis is focusing on the 3D mobile graphics platforms and tools. The technology advanced to the point in which high-quality, high-resolution 3D graphics is possible on mobile. Graphics technology is now one of the main driving forces in the progress of mobile devices. This is related to the popularity of mobile gaming but one can expect there will be increasing trend for all other applications and user interfaces using 3D graphics. That will be significant change how the applications are developed since it will require using specific 3D platforms and application development tools.

There are mainly three mobile ecosystems in use today: Android, iOS, and Windows Phone. Android is by far dominant in the number of users, others are rather marginalized but still may have their segments of popularity. Fortunately, in the case of 3D graphics one does not have to differentiate strongly between these ecosystems since the 3D graphics platforms considered in this thesis are universal and supported by each of them.

In the thesis we provide some background about mobile graphics and 3D graphics technology in Chapter 2. In Chapter 3 there is an overview of graphics platforms which are languages and API's for defining and describing 3D objects and scenes. These platforms are strictly standardized and include traditional OpenGL, its mobile version OpenGL ES, and the formulation of OpenGL adopted to the Web called WebGL. In additions there are two specialized additions called OpenCL and WebCL which are dedicated for optimizing parallel computing methods that accelerate the graphics processing speed. In Chapter 4, the overview of the 3D modelling software, 3D game engines and two software tools for 3D graphics development based on the 3D platforms are described. One is Maya which is mostly used for modelling of objects and Unity which is 3D graphics engine used for animation. Then, the specific application and workflow for its development are elaborated. The aim here is to show opportunities which advanced 3D graphics brings to the mobile user experience. The concept is using 3D models for Web fashion shop. The problem with buying fashion over the Web is looking at item in space and fitting it to body. This could be solved or at least greatly improved over the current static 2D display of fashion items if there is available 3D body model of user and 3D flexible fashion items models. We show how applications using 3D body and item models can be developed using the graphics tools to run in mobile devices. The thesis ends with the conclusions summarizing the developments and results.

2. BACKGROUND

In this chapter, we provide background information about 3D graphics in mobile devices. The main problem with 3D graphics is computational power required for the generation, processing and displaying 3D information. This problem exists even in most powerful graphics workstations but it is especially difficult in mobile devices where there are very severe constraints on the processing power, size of the device and heat dissipation. Due to these reasons mobile graphics started with very basic systems and only recently it started developing into full 3D as known from the PC. In this chapter we provide some background information about these developments and the current state-of-the-art.

2.1 Overview of Mobile GPU

In 2001, experts in Nokia began to work on mobile 3D graphics. They used concepts borrowed from the PC/workstation area adapting application programming interface OpenGL to mobile requirements. In 2002 Nokia introduced first rudimentary graphics game “Snake” in the Nokia 6610 device. This was a breakthrough in mobile graphics and games, over 400 million devices have been shipped since then. Though the 6610 had a very simple display and interface, and the gaming strategy was simple, it attracted millions of users [2]. Mobile gaming started but it was clear that it has major limitations due to the processing power of the device.

Then it comes to the concept of Graphics Processing Unit GPU. Graphics processing requires significant computational power which exceeds typical processor CPU capability. Specialized processor for graphics called GPU is needed. In personal computers traditionally the GPU is installed as a separate cards called video cards or graphics cards. The cards take over the graphics processing job of the CPU and since the GPU is highly specialized for graphics computations there is very significant performance improvement. However intensive graphics processing computations in GPU consume more power than the computer itself. Thus, graphics capabilities of mobile devices are limited by the power consumption. Only recently due to optimization and reduced size of the GPU chips advanced graphics in mobile devices became possible.

In mobile devices the GPU is located on the mainboard with the CPU and memory. The working principle and the tasks of mobile GPU are similar to PC, releasing the computational load of CPU on a device and taking over the display and image processing jobs.

There is a big difference between GPU on PC and on smartphones. In most mobile GPUs, there is no independent temporary writable memory, which is also known as frame buffer on GPU to store and process rendering data and pixels.

As present, the mobile graphics industry enjoys a boom in the graphics area. GPU became standard part of every device and processing power is quickly increasing. The mobile graphics GPUs not only process 3D contents but also are responsible for video playing, recording and helping with photographing on the smartphone. This “liberates” CPU not only from graphics processing but also from other complex image and video tasks. Users can enjoy fantastic graphics and visual applications on their devices.

2.2 3D Graphics Creation Pipeline

Pipelines of creating 3D graphics and scenes on a screen require several main tasks which are shown in Figure 2.1 and briefly described below [3].

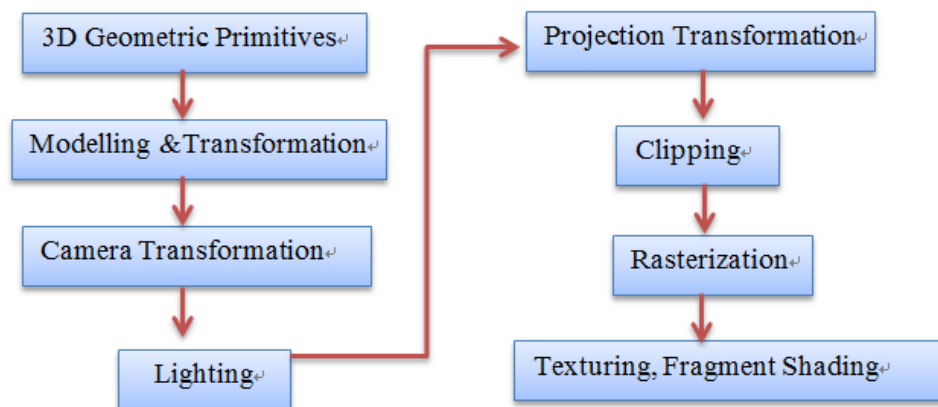


Figure 2.1. 3D Graphics Pipeline

- **3D geometric Primitives**
First, there should be created a model of a 3D object in the scene, which is the main character in the creation. Creation a 3D object model is made by its construction of geometric primitives such as triangles and polygons. This will approximate the object with a wireframe mesh, the more primitives the better approximation. For example, with the proper position of vertices of the primitives, one can construct simulated model of complete human body mesh.
- **Modelling and Transformation**
Transform from the local coordinate system to the 3d world coordinate system. A model in abstract is placed in the coordinate system of the 3d world for display and interaction with the virtual physical world.

- **Camera Transformation**

Transform the 3d world coordinate system into the 3d camera coordinate system, with the camera as the origin. Camera is for scene recording and rendering in display. The camera position in the scene can be constantly changing and the graphics system must be able to provide correct rendering of a view of the scene from any position.
- **Lighting**

Lighting is actually the illumination according to lighting and reflectance. In this step the effect of lighting and reflections are calculated. Without lights, nothing can be seen in the scene. Lights type and position must be selected and switched on. Lighting in the 3D scene is made by simulating light interaction with the textures on the surface objects. Depending on the lighting and type of the materials of the texture, there will be different reflections and diffusion of light which can be calculated based on physics. For realistic light effects this requires a lot of computations and puts high demands of graphics hardware.
- **Projection transformation**

Projection transform is mapping the 3d world coordinates into the 2d view of the camera. This is achieved by dividing the X and Y coordinates of each vertex of each primitive by its Z coordinate (which represents its distance from the camera). In an orthographic projection, objects retain their original size regardless of distance from the camera.
- **Clipping**

Geometric primitives that now fall completely outside of the viewing frustum will not be visible and are discarded at this stage.
- **Rasterization**

Rasterization is the process by which the 2D image space representation of the scene is converted into raster format and the correct resulting pixel values are determined. From now on, operations will be carried out on each single pixel.
- **Texturing, fragment shading**

At this stage of the pipeline individual fragments (or pre-pixels) are assigned a color based on values interpolated from the vertices during rasterization, from a texture in memory, or from a shader program. Only mesh objects with wireframes are obviously not enough. There should be texture put on the mesh to simulate surface properties like color. This is done by texturing which is a way to map images of texture patterns onto the object wireframe.

- Other Stuff

Simulation of a 3D scene must also include the simulation of physical effects of motion and interactions of objects and in particular animation. This requires correct calculations of scene dynamics. Real-time scene dynamics requires very high computational power of graphics hardware.

All these aspects above can be integrated and realized by programming on graphics platforms with the support of the advanced mobile GPU. The development of mobile GPU brings possibilities for the creation of 3D scenes with stunning effects known from high-end PC graphics and movie industry. This will expand the range of applications which is illustrated in this thesis.

2.3 Hardware Support for 3D Graphics

GPUs are the chips that support graphics transforms and lighting operations in hardware. These are most demanding aspects of 3D graphics processing requiring computing polygon positions and dynamic lighting effects. The main several benchmarks for measuring the performance of GPUs are the capability of generating polygons (in most cases triangles) called triangle throughput, rendering pixels per second called Fill Rate and Floating-point Operations per Second called FLOPS.

- *Triangle/Polygon throughput*: It measures the number of triangles per second that a GPU can process. Every displayed frame consists of huge number of polygons and the scene rendering is based on these polygons. So the higher capability of outputting polygons, the better GPU performance and the more exquisite the 3D images and scenes.
- *Fill rate*: It is the number of pixels a GPU can render and write to video memory in a second. [4] The mobile display is composed of pixels and current high-end devices have full HD resolution of 1920x1080 pixels. This requires the same fill rate in mobile devices as in desktop PC. GPU renders pixels determining the color, location and other attributes of the pixel. Only the rendered pixels can be seen on screen,. The higher the pixel fill rate, the better the GPU performance. There is another measuring standard - texturing fill-rate, which is defined as the number of texels (textured pixels) used for rendering during one second. This measurement takes the rendering capability into account in GPU performance.
- *FLOPS*: FLOPS (for Floating-point Operations per Second) is a measure of computer performance or processors, useful in fields of scientific calculations that make heavy use of floating-point calculations. For such cases it is a more accurate measure than the generic instructions per second.

Design of advanced mobile graphics chips is very complex and few companies are specializing in this area. These companies are Imagination Technologies, Qualcomm, NVIDIA and ARM. Each of them is producing own chips which differ in some details of processing. Below there is a comparison list for the GPU from these companies and short descriptions of current mobile GPU chips. The entries marked in yellow are the latest chips with the best performance which there are records. Some new GPU have released but there are no detailed data for them, and these chips are not listed here. The table next is the comparison of the current GPU from the four mentioned companies and the ones marked in yellow are the best among their own. Data are got from GFXBench which is a unified 3D graphics performance benchmark. [5]

Table 2.1. Comparisons of different GPU series

GPU	Fill Rate	GFLOPS	Example Devices
PowerVR			
SGX 554MP4	2377 MTexels/s	76.8	Apple iPad 4
SGX 544MP3	2161 MTexels/s	51.1	Samsung Galaxy S4
SGX 543MP4	1311 MTexels/s	32.0	Apple iPad 3
SGX 544MP2	1095 MTexels/s	34.1	Asus Fonepad 7 ME372CG
Adreno			
Adreno 330	4243 MTexels/s	166.5	Samsung Galaxy S5
Adreno 320	2940 MTexels/s	97.2	KDDI HTC One J HTL22
Adreno 225	834 MTexels/s	25.6	HTC One XL
Adreno 305	821 MTexels/s	21.6	Samsung Galaxy Tab 3 7.0
Adreno 220	532 MTexels/s	17	HTC EVO 3D
NVIDIA			
Tegra K1	3849 MTexels/s	326	Lenovo ThinkVision 28
Tegra 4	2568 MTexels/s	96.8	Xiaomi MI 3
Tegra 3	781 MTexels/s	12.5	HTC One X
ARM			
Mali-T760 MP16	1390MTriangle/s 11200Mpixel/s.	326.4	N/A
Mali-450 MP6	4631 MTexels/s	53.8	Tronsmart Vega S89
Mali-400 MP4	2251 MTexels/s	19.2	Infotouch iTab M9 Pro
Mali-T628 MP6	2199 MTexels/s	102.4	Samsung Galaxy Note 10.1
Mali-T604 MP4	1446 MTexels/s	72.5	Google Nexus 10

Below are the overview of the four main GPU providers and their products in short descriptions. [6]

1. PowerVR SGX chips from Imagination Technologies

Imagination Technologies from UK released first PowerVR SGX chips in 2005. Till now, it is the most widely used GPU series in mobile devices ranging from low-end to high-end smartphones. Here we will describe the main product series be PowerVR SGX 5 and PowerVR SGX 5XT.

PowerVR SGX 5 uses USSE (Universal Scalable Shader Engine) and supports OpenGL ES 2.0/1.1. Moreover, SGX 535/545 support also Microsoft DirectX API version DX9 and SGX 545 supports version DX10.1. SGX 530 chips are used in low-end phones. Their polygon throughput is 14 million triangles per second (Mtri/s) and the fill rate is 4.8 Gigapixels per second (Gpix/s). SGX 535/540 is upgraded version of SGX 530 with the polygon throughput of 24 Mtri/s and 4.8 Gpix/, they were used in the iPhone4 and iPad.

PowerVR SGX 5XT is an enhanced version of SGX 5 and includes Power SGX 543, SGX 54 and SGX 554 chips. This version doubles the capability of peaking FLOPS (Floating-point Operations per Second) compared with USSE and strengthens the integration of multiple cores to reach the maximum 16 cores. The latest Apple A5, A5X, A6, A6X are using SGX 543MP2, SGX 543MP4, SGX 543MP3 and SGX 554MP4. The best among the products are SGX 544MP3 and SGX 554MP4 with the GFLOPS at shipping frequency of 51.1GFLOPS and 76.8 GFLOPS, which means the computational capability of the GPU is very outstanding. SGX 544MP3 is used in Samsung Galaxy S4 and SGX 554MP4 is used in Apple A6X Chipset in iPad 4.

Imagination announced the next generation upcoming GPU PowerVR 6 (Rogue) in 2012, aiming at improving the FLOPS to TERA level which is even very competitive with desktop GPU.

2. Adreno chips from Qualcomm

Officially announced by Qualcomm that available Adreno products are Adreno 330, Adreno 320, Adreno 225, Adreno 220, Adreon 205, Adreon 200, Adreon 130. Adreno series chips are famous for their capability of polygon computations throughout while Imagination chips are focusing more on pixel rendering.

Adreno 220 is integrated in the Snapdragon™ S3 processors and supports OpenGL ES 2.0/1.1, EGL 1.3, OpenVG 1.1 and DX9. Adreno 320 is integrated in Snapdragon™ 600 processors and delivers over 300% increase in graphics processing performance supporting for advanced graphic and compute API, including OpenGL ES 3.0, DirectX, OpenCL, render-script compute and flex-render and providing a superior user experience for HTML5 Web browsing, 3D games, 3D user interfaces, and other graphics applications.

Right now the Adreno 330 is one of world's fastest GPUs for smartphones with fill rate of 4243 MTexels/s and FLOPS of 166.5 GFLOPS. It is built in Snapdragon™ 800 series processors and used inside the Nexus 5, Amazon Kindle HDX series tablets, Nokia Lumia 2520 tablet, Nokia Lumia 1520, Nokia Lumia ICON, Nokia Lumia 930, Samsung Galaxy S5, Sony Xperia Z1, Sony Xperia Z2, Sony Xperia Z Ultra, and LG G2 smartphones.

Qualcomm has announced that the upcoming Snapdragon™ 808 and 810 processors with Adreno 418 and 430 GPU inside will be released in 2015. In terms of graphics performance, the Adreno 418 is apparently 20% faster than the Adreno 330, and the Adreno 430 is 30% faster than the Adreno 420 (100% faster in GPGPU performance).

3. Tegra chips from NVIDIA

NVIDIA is company with very strong technical background based on the GPU in PC. It entered the mobile GPU area later and has 4 generations of products - Tegra 2, Tegra 3, Tegra 4 family and Tegra K1 listed in the order of release.

Tegra 2 is the world's earliest Cortex-A9 dual-core processor, Tegra 3 is the earliest Cortex-A9 quad-core processor and Tegra 4 is the earliest Cortex-A15 quad-core processor. Tegra 2 is using GeForce ULP architecture and has 8 cores (4 vertex units and 4 pixel units). Tegra 3 for Android also uses ULP GeForce but the 3D performance has improved by 300% with Tegra 2. The numbers of cores are increasing to 12 (4 vertex units and 8 pixel units). Tegra 4 is a breakthrough that allowing 72 custom GPU cores (24 vertex units and 48 pixel units) that enjoys unique mobile device innovations in photography, media, gaming, and web—including High Dynamic Range (HDR) imaging, WebGL, and HTML5. All the three products support OpenGL ES 2.0 fully and Tegra 4 supports OpenGL ES 3.0 except some functions. The key and core technology for Tegra is the design of separating the vertex units and pixel units instead of using unified rendering. Regarding to polygon throughput and fill rate, even Tegra 2 reaches 90Mtri/s and 12Gpix/s and other upgrade versions' performance is much better with Tegra 3 reaching 781 MTexels/s and Tegra 4 reaching 2568 MTexels/s in textured pixels fill rate.

The latest innovative new Tegra K1 processor features the same high-performance, power-efficient NVIDIA Kepler™ -based GPU that drives the world's most powerful supercomputers and PC gaming systems. This means users can now count on even more unbelievable graphics performance, powerful computing, and truly unique features in every Tegra K1-powered mobile device. NVIDIA Kepler architecture GPU has 192 NVIDIA CUDA cores inside and it is the first GPU architecture to span from supercomputers to PC to mobile devices with new rendering and simulation techniques such as tessellation, compute-based deferred rendering, advanced anti-aliasing and post-processing algorithms, physics and simulations. Kepler supports the full spectrum of OpenGL – including the just-announced OpenGL 4.4 full-featured graphics specification and the OpenGL ES 3.0 embedded standard. It also supports DirectX 11, Microsoft's latest graphics API. [7] Additionally, NVIDIA has made some optimizations especially for mobile, such as adding a new low-power inter-unit so that Kepler uses less than one-third the power of GPUs in leading tablets, such as the iPad4. The fill rate of Tegra K1 has reached 3849 MTexels/s and 326 GFLOPS which is very competitive with desktop GPU.

4. Mali chips from ARM

Mali is the GPU solution that ARM provides for smart devices like smart TV and smartphones. Products of Mali series can be categorized into two levels - Mali-300, Mali-400 and Mali-450 supporting OpenGL ES 2.0 with the code “Utgard” and Mali-T604, Mali-T624, Mali-T628 and Mali-T678 supporting OpenGL ES 3.0 with the code “Midgard”.

In the first series of products, Mali-400MP is the most commonly used one and is used in Samsung Galaxy S3 smartphone with 4 GPU cores that makes the polygon throughput reaching 120 Mtri/s and fill rate 1.1Gpix/s. However, Mali is using unusual rendering methods and supports less graphics formats. So, not so many companies are using Mali in their mobile devices. When it comes to the second series of chips, their performance is much better but popularity is lower than chips from other manufacturers.

Mali-T760 MP16 is the latest product of ARM that boosts the Midgard architecture into a new era of energy efficiency and it is 400% the energy efficiency of the ARM Mali-T604 GPU and it has 16 cores inside. With full support for current and next generation graphics and compute API, it boasts stunning graphics and guarantees the excellent execution of compute-intensive tasks such as computational photography, gesture recognition and image stabilization by supporting OpenGL ES 3.0/2.0/1.1 and Microsoft Windows compliant for Direct3D 11.1. The triangle throughput is

1390Mtri/s and fill rate is 11.2Gpix/s and the FLOPS is the same with NVIDIA Kepler reaching 326 GFLOPS.

The trend in mobile GPU is moving towards capabilities of PC GPUs. As the PC GPUs are constantly evolving towards higher performance and also their power consumption grows, the mobile GPUs are following but restrictions on the power consumption mean that chip complexity must be limited. Roughly speaking, this means 10:1 performance ratio between the high-end PC GPUs and mobile GPUs. However, high-end mobile GPUs are comparable with low-end PC GPUs consuming only a fraction of their power which is amazing achievement.

3. 3D GRAPHICS PLATFORMS

As described before, in 3D graphics, objects are constructed of polygons, vertices of polygons are located in the 3D space using x, y, z coordinates. Each polygon has a material property which is concerning color, texture and reflective features and since the vertices are positioned in 3D coordinates, the matrix mathematics can be used to realize transformations of the vertices like rotation, translation and zoom. Basic 3D graphics processing pipeline includes:

- Projection of polygons onto the screen by determining which pixels are affected
- Smooth shading that calculates light factors at each vertex, computes the color interaction with lights and surface properties and interpolate colors between the vertices
- Texture mapping the surface of the object by computing image coordinates to paste
- Environment mapping by pasting reflection of image of environment at each pixel.

These operations are quite complicated and require deep knowledge of matrix mathematics and physics for simulation. They would be very difficult to implement for non-specialists. To facilitate graphics applications there was early on an effort to provide specialized tools which would allow users to skip the complexity. The resulting basic system is called OpenGL and it is universally used as a foundation for development of tools and applications for 3D graphics. The OpenGL Application Programming Interface (API) is supported in all operating systems and in GPUs. OpenGL is maintained by the Khronos Group made be major companies in the 3D graphics area like AMD, Intel, ARM, NVIDIA, Broadcom, Apple, Google, Microsoft, and game companies like Epic, EA and Unity. Some modifications of OpenGL were developed by Khronos for more specialized usage. This includes OpenGL ES which focuses on 3D graphics on mobile and embedded devices by restricting certain OpenGL functionalities. OpenCL (Open Computing Language) is focused on parallel graphic computing facilitating programming that executes on heterogeneous platforms with multiple processors including CPUs, GPUs and other processors. WebGL, enables creation of 3D content in Web systems, e.g. directly in browsers with no need for plug-ins. WebCL adapts OpenCL for Web applications. One can thus now talk about standardized ecosystem built around OpenGL for the development of 3D applications as shown in Figure. 3.1. [8].



Figure 3.1. OpenGL Related Ecosystem

In this chapter, the components of the OpenGL will be described in details and short examples will be given for illustrating the basic concepts.

3.1. OpenGL

OpenGL is a widely-used low-level graphics application programming interface for 2D and 3D interactive graphics, which is independent from operating systems, hardware and windowing system. From the programmers' perspective, it specifies the geometric objects, describes object properties, defines the views of objects and move camera or objects around for animation. Generally speaking, OpenGL states variables like vertex color, line width, current viewing position, material properties and so on by applying to drawing commands, i.e. the input is the description of geometric objects and the output is pixels sent to display onto the screen. This takes form of a pipeline shown in Figure 3.2.

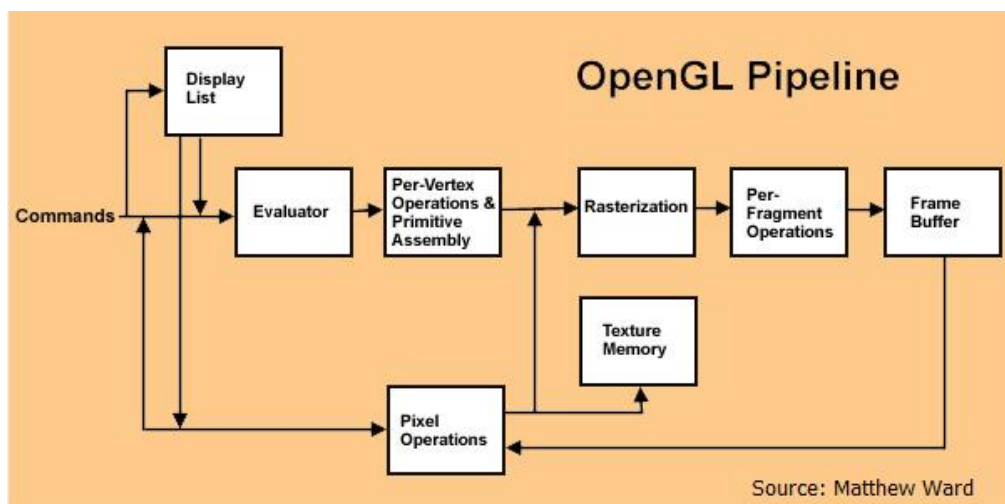
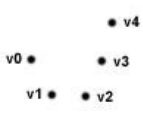
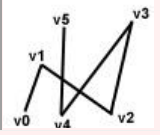
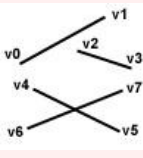
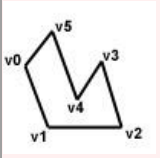
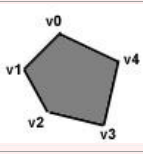
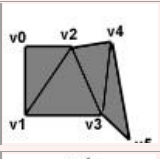
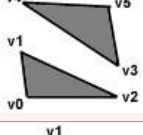
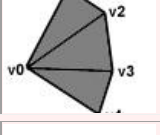
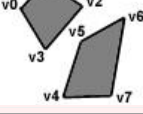
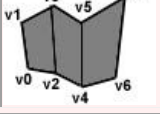


Figure 3.2. OpenGL Pipeline Architecture

As seen in Figure. 3.2, “**Display List**” is OpenGL drawing commands which are pre-compiled for efficiency. “**Evaluator**” is for vertex pre-processing which takes polynomial evaluator commands and convert them into corresponding vertex attributes commands effectively. “**Per-Vertex Operations & Primitive Assembly**” applies geometric transformations which contain the information of vertex data and optionally normal, texture coordinates, material properties and colors to each vertex and groups primitives together to form triangles, polygons and so on. The Table 3.1 shows primitive types in OpenGL. Once primitives are assembled, they are clipped to fit in a 3 dimensional region which is called projection or view frustum. “**Rasterization**” converts viewport-mapped primitives into pixels in fragments which consist of pixel location in frame-buffer, color, texture coordinates and depth. “**Per-fragment Operations**” is the stage before being put into frame-buffer and they are tests to determine fragment visibility such as depth test and stencil test. “**Pixel Operations and Texture Memory**” means that pixels which are ready to be placed in frame-buffer can be copied, texture can be mapped and saved for reuse so there is no need to recreate it. “**Frame-buffer**” is a rectangular array of n bitplanes for fragments produced by rasterization for display. Frame-buffer is organized bitplanes in logic of color, depth, stencil and accumulation that are containing corresponding information [9].

Table 3.1. Ten Open GL primitive types

Points	individual points GL_POINTS		Line Strip	series of connected line segments GL_LINE_STRIP	
Lines	pairs of vertices interpreted as individual line segments GL_LINES		Line Loop	same as above, with a segment added between last and first vertices GL_LINE_LOOP	
Polygon	boundary of a simple, convex polygon GL_POLYGON		Triangle Strip	linked strip of triangles GL_TRIANGLE_STRIP	
Triangles	triples of vertices interpreted as triangles GL_TRIANGLES		Triangle Fan	linked fan of triangles GL_TRIANGLE_FAN	
Quads	quadruples of vertices interpreted as four-sided polygons GL_QUADS		Quad Strip	linked strip of quadrilaterals GL_QUAD_STRIP	

Drawing of 3D objects is based on the assembling the primitives of polygons, and with describing attributes of the vertices [10]. Below is an example of rendering a color pyr-

amid on a screen starting with `glBegin(GL_ObjectsType)` and end with `glEnd()`. In between, there is drawing of four triangles with vertices connected between each other and defining the colors of vertices, the colors in between will be the gradient.

```
glBegin(GL_TRIANGLES); // Begin drawing the pyramid with 4 triangles
// Front
glColor3f(1.0f, 0.0f, 0.0f); // Red
glVertex3f( 0.0f, 1.0f, 0.0f);
glColor3f(0.0f, 1.0f, 0.0f); // Green
glVertex3f(-1.0f, -1.0f, 1.0f);
glColor3f(0.0f, 0.0f, 1.0f); // Blue
glVertex3f(1.0f, -1.0f, 1.0f);

// Right
glColor3f(1.0f, 0.0f, 0.0f); // Red
glVertex3f(0.0f, 1.0f, 0.0f);
glColor3f(0.0f, 0.0f, 1.0f); // Blue
glVertex3f(1.0f, -1.0f, 1.0f);
glColor3f(0.0f, 1.0f, 0.0f); // Green
glVertex3f(1.0f, -1.0f, -1.0f);

// Back
glColor3f(1.0f, 0.0f, 0.0f); // Red
glVertex3f(0.0f, 1.0f, 0.0f);
glColor3f(0.0f, 1.0f, 0.0f); // Green
glVertex3f(1.0f, -1.0f, -1.0f);
glColor3f(0.0f, 0.0f, 1.0f); // Blue
glVertex3f(-1.0f, -1.0f, -1.0f);

// Left
glColor3f(1.0f,0.0f,0.0f); // Red
glVertex3f( 0.0f, 1.0f, 0.0f);
glColor3f(0.0f,0.0f,1.0f); // Blue
glVertex3f(-1.0f,-1.0f,-1.0f);
glColor3f(0.0f,1.0f,0.0f); // Green
glVertex3f(-1.0f,-1.0f, 1.0f);
glEnd(); // Done drawing the pyramid
```

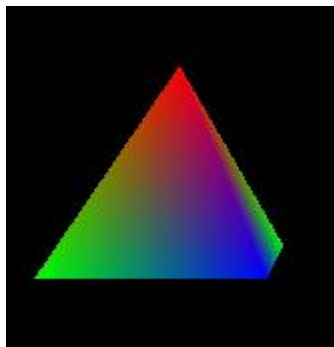


Figure 3.3. Output of OpenGL demo

The pyramid produced is shown in Figure. 3.3, and as can be seen from the OpenGL usage all complexity of graphics calculations is hidden from the user. This brings enormous simplifications in programming. Here is a brief list of what OpenGL can do [11]:

- **Drawing Commands:**

Functions: begin with `gl` (example: `glBegin` – draw an object);

Constants: begin with `GL_` (example: `GL_POLYGON` – a polygon);

Types: begin with `GL` (example: `GLfloat` – single-precision float);

All objects in OpenGL are constructed from convex polygons represented by their vertex coordinates. The argument type is specified by the suffix to the OpenGL function name. `<func_name><dim><type>` (argument list) is the function format. For example, `glVertex3f (200.3f, -150f, 40.75f)` draws a vertex with the coordinate $x=200.3$, $y=-150$, $z=40.75$ in 3 dimension space. Based on the vertices and the constants defined in the beginning, shapes like polygons, triangles, strips and loops form 3D objects.

- **Drawing Attributes**

Attributes affect the manner objects are drawn. These are placed between each `glBegin` and `glEnd` pair and once set, they affect following subsequent objects, until they are changed again. Points attributes like point size: `glPointSize (2.0)` and point color: `glColor3f (0.0, 0.0, 1.0)` set in red, green and blue order, lines attributes like line width: `glLineWidth (2.0)` and line color: `glColor3f (0.0, 0.0, 1.0)` and polygons attributes like face color: `glColor3f (0.0, 0.0, 1.0)` and lighting material properties: `glMaterialf ()` can modify the objects in appearance.

- **Color**

All subsequent primitives will be this color `glColor3f(r,g,b)` and colors are not attached to objects. Users can change the color statement and red, green & blue components are ranging from 0-1. [12]

- **Lighting & Reflective effects**

Additionally, the color of each vertex is determined based on the object's material properties and the relationship to light sources. Surface interactions based on lights are ambient lighting with a background glow that illuminates all objects, irrespective of light source location, diffusion with a uniform scattering of light, characterized by matte (non-shiny) objects, like cloth or foam rubber, specular shiny (metallic-like) reflection, purely reflection with no light scattering and transparent/translucent with light passing through material. Here are some example commands for lighting such as `GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR`, `GL_EMISSION`, `GL_AMBIENT_AND_DIFFUSE`, `GL_SHININESS` and so on. Properties like location, color and intensities of the lights are controlled with func-

tions and the light is enabled and disabled by `glEnable (GL_LIGHTING)` and `glDisable (GL_LIGHTING)` [13].

- **Texture and Fog**

The job of texturing is to map from texture to object space. Create a new texture with an unused ID with function `glGenTextures (GLsizei n , GLuint* textureIDs)` and bind texture to object with function `glBindTexture (GLenum target , GLuint textureID)`, where `target` is `GL_TEXTURE_1D`, `GL_TEXTURE_2D`, or `GL_TEXTURE_3D`. When enabled, a fragment's texture coordinates are used to index into a texture image, generating a texel. The texel modifies the fragment's color based on the current texture environment, which may involve blending with the existing color. After texturing, a fog function may be applied to the fragments. This blends a fog color based on the distance of the viewer from the fragment. Seen from another angle, texture is the surface details other than lighting to add some realism to the objects such as natural surfaces like stone, wood, gravel, grass, printing and painting like printed labels, billboards, newspapers and clothing and fabric like woven and printed patterns, upholstery.

- **Transformation Operations**

Below are the common standard transformation in OpenGL and “GLtype” is either “GLfloat” or “GLdouble”. `glTranslate{fd} (GLtype x, GLtype y, GLtype z)` post-multiply the active matrix by a translation matrix that translates by (x, y, z). `glRotate{fd} (GLtype angle, GLtype x, GLtype y, GLtype z)` post-multiply the active matrix by a rotation matrix that rotates CCW by angle degrees about the vector (x y z). `glScale{fd} (GLtype sx, GLtype sy, GLtype sz)` post-multiply the active matrix by a scale matrix that scales x by sx, y by sy and z by sz.

- **Camera**

Camera views (Figure. 3.4) which determine the display angle of the 3D world. In physical world, the camera parameters include positions (x, y, z), orientation (yaw, roll, pitch) and lens (field of view). In OpenGL, there are two projection ways, orthographic `glOrtho (left, right, bottom, top, near, far)`. and perspective `gluPerspective (fovy, aspect, near, far)`. In general, the transformation of camera is realized by `gluLookAt (eye_x , eye_y , eye_z , at_x , at_y , at_z , up_x , up_y , up_z)` corresponded with location of the eye, the point the viewer is looking at and the direction relative to the camera, which results in the transformations on the objects relatively. The data type is GLdouble.

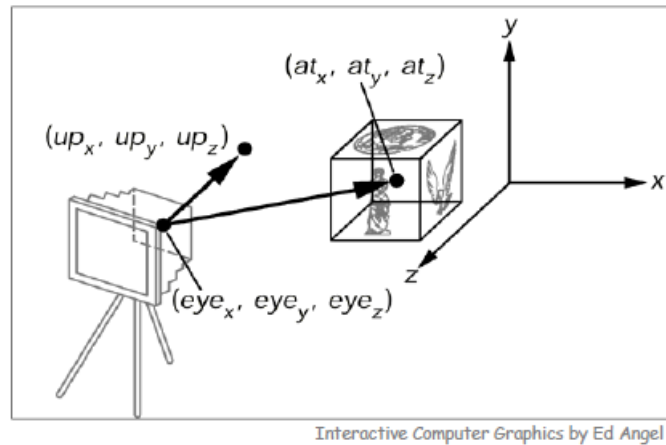


Figure 3.4. Camera in OpenGL

- **Event-Driven Computing**

Typical process for non-interactive program to compute event are reading data, processing data and output results. From system's perspective, event-driven computing is to check whether an event has occurred, if yes, then call function eventHandler and repeat the routine. From programmer's perspective, developers first register "event-handler" pairs, and for each "event" call a function called a callback that performs "handles" this event and returns. Then, pass control to the operating system. Operating System or Window Management System copies all handled events to an Event Queue with the first-in first-out order.

- **Pixel Buffers and Operations**

OpenGL maintains from one to many pixel buffers. These buffers store different types of information and different functions. Pixel is known as the element of pictures. Bitmap is a 2D array of single-bit pixels (0/1 or black/white) and Pixmap is stack of bitmaps. The number of bits per pixel is called its depth. OpenGL has color buffer to stores image color information (RGB or RGBA, Alpha-channel used for blending operations, such as transparency), depth buffer to stores distance to object pixel and is used for hidden surface removal also called the Z-buffer (z-coordinate stores distance), accumulation buffer that is used for composing and blending images and stencil buffer Used for masking operations.

Regarding to reading and writing buffers, OpenGL has functions like `glReadPixels (x, y, width, height, format, type, *pixels)`, `glRasterPos2i (x, y)`, `glDrawPixels (width, height, format, type, *pixels)` and `glCopyPixels (x, y, width, height, format, type, buffer)`, where format can be `GL_RGB`, `GL_RGBA`, `GL_RED`, `GL_GREEN`, `GL_BLUE`, `GL_ALPHA`, `GL_COLOR_INDEX`, `GL_DEPTH_COMPONENT` etc., type can be `GL_UNSIGNED_BYTE`, `GL_UNSIGNED_SHORT`, `GL_FLOAT`, etc. and buffer can be `GL_COLOR`, `GL_DEPTH`, `GL_STENCIL`.

This is a brief introduction of how OpenGL works in practical programming manner. OpenGL is the fundamental API for many 3D application development software tools and game engines, its current version is OpenGL 4.4.

3.2. OpenGL ES

OpenGL ES (Open Graphics Library for Embedded Systems) is the 3D graphics API especially designed for embedded systems. It is developed from OpenGL by removing redundant and computationally expensive functionalities which are not absolutely necessary, while at the same time keeping it as compatible with OpenGL as possible and adding features needed for embedded systems.

The API of OpenGL ES is quite similar to OpenGL. OpenGL ES now has 3 versions - OpenGL ES 1.x, OpenGL ES 2.x and OpenGL ES 3.x [14]. Taking OpenGL ES 1.0 as an example, the main differences between OpenGL and OpenGL ES are:

- Removing redundant API: In OpenGL, the system has many different functions to do the same tasks considering the flexibility of the language and the only difference is the parameter types. For instance, the function of colors, OpenGL has over 30 functions with the same function name `glColor()`, but OpenGL ES removed the most of them and only left those with common and supportive data types.
- Removing redundant functions: OpenGL ES has only kept the most effective functions in computing due to the scale and capability of the whole system is not as large as the platform OpenGL is based on.
- Restrict the functions with high costs: There are many important functions with high computing cost in OpenGL, such as texturing. OpenGL ES remove some and make some optional.
- Remove some data types: OpenGL ES does not support double data type. If there are some functions only using double data type, OpenGL ES will convert them into float such as `glTranslatef`, `glRotatef`, `glScalef`, `glFrustumf`, `glOrthof`, `glDepthRangef`, `glClearDepthf`. OpenGL ES simplify some API' supporting data types for some functions as well.
- There is no equivalent to OpenGL libraries like GLUT or GLU for OpenGL ES.

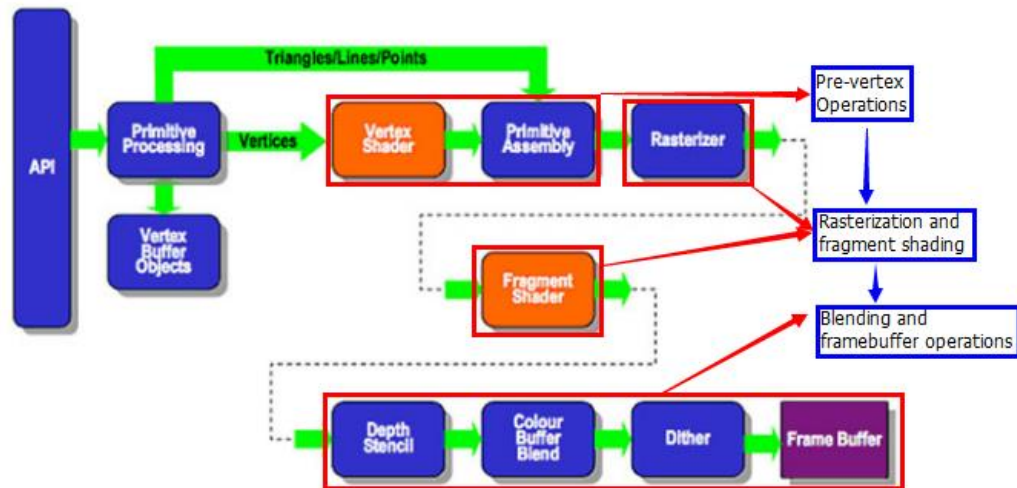


Figure 3.5. Pipeline 2.0

The basic structure of the OpenGL ES (Figure. 3.5) pipeline is similar with OpenGL [15], there is no single key step missing in the pipeline, but OpenGL ES is a bit different from OpenGL that there is no longer fixed build-in support for lighting, fog, multi-texturing and vertex transformations. These features here must be implemented with customised shaders. For example, before primitive assembling in OpenGL ES 2.0, vertex shading replaced fixed function for transformation and lighting in Open GL ES 1.x, whose main task is to provide vertex positions for next stage of fragment shading [16]. It is possible to translate, rotate, and pass texture coordinates and calculate lighting parameters. Similarly in fragment operation, fragment shader replaced fixed functions for texture, color and fog whose main task is to provide color values for each output fragment and assign textures and fog parameters.

The latest version is OpenGL ES 3.0 and it is backwards compatible with OpenGL ES 2.0. This new version makes it friendlier to developers with more flexible hardware requirements and more features and functions from OpenGL 3.3 and 4.x to mobile devices. One of the key new feature is the new texture algorithm which makes better compression and others like instance rendering, occlusion queries and transform feedback that accelerate the hardware. For example, previously developers usually create different texture files for different devices in APK (Android application package file) is the package file format used to distribute and install application software and middleware onto Google's Android operating system), but with the new algorithm they can create one file that can be used on both PC and mobile platforms. One of the exciting features for OpenGL ES 3.0 is that it gives better graphic performance with longer battery time.

The OpenGL ES 3.1 specification was publicly released in March 2014 and it provides new functionalities in like compute shaders, independent vertex and fragment shaders and indirect draw commands. OpenGL ES 3.1 is backward compatible with OpenGL ES 2.0 and 3.0, thus enabling applications to incrementally incorporate new features.

OpenGL ES is widely used in diverse range of mobile devices where it makes 3D graphics possible and perform better on mobiles.

3.3. WebGL

WebGL (Web Graphics Library) is a JavaScript 3D computer graphics API which has no need for plugins, but it can only run on supported browsers. So far the supported browsers are Google Chrome 9.0+, Mozilla Firefox 4.0+, Safari 5.1+ (On Mac OS X), Opera 12.0+ (planned) on desktop and iOS Safari, Opera Mini, Opera Mobile, Android 2.3, Firefox Fenec (beta) for mobile [17]. WebGL is based on OpenGL ES, its syntax is nearly identical to OpenGL and it has minimal set of features for compelling content. WebGL is purely shader-based and has no fixed functions. The pipeline in Figure 3.6 shows the process of turning commands from the function drawScene (in the next code clip) to pixels that are displayed on canvas.

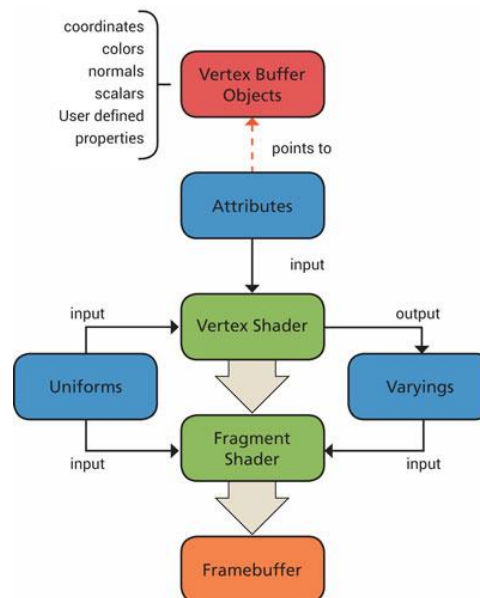


Figure 3.6. WebGL Rendering Pipeline

From the top, the data WebGL is transferring to vertex shader is in the form of “Attributes” which are describing vertex coordinates, colors, norms, scalars and customized properties and Uniform which stores model viewport matrix and projection matrix. The vertex shader is called every time after finishing the construction of attributes while uniform does not change during the process and input projection matrix. Then, the shader stores the results in varying variables within which there is a special variable called `gl_Position` storing the coordinates of vertices. WebGL will translate the described 3D objects into 2D images and call fragment shader for pixels in the images. Finally, WebGL will output the resulted pixel onto the screen through frame buffer. Just like OpenGL, general matters in graphics engineering can be also realized in WebGL

such as coloring, texturing, animating, camera viewport, lights, user interaction and rendering [18].

As is known, HTML5 is a mark-up language used for structuring and presenting content for the World Wide Web and a core technology of the Internet. However, WebGL is not exactly the same as HTML5 but an extension of it utilizing canvas element [19]. Canvas is a new concept in HTML5 that is supporting Javascript to draw 2D image and WebGL for 3D images. Developers only need to set simple properties for canvas and put what they want to display in Javascript function called `webGLStart`.

```
function webGLStart() {
    var canvas = document.getElementById("lesson01-canvas");
    initGL(canvas);
    initShaders();
    initBuffers();
    initTexture();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

    drawScene();
}
```

After creating canvas, the shader should be initialized and the 3D images delivered to it that would be drawn on canvas model, then initialize some arrays with `initBuffers` to store the details of triangles and rectangles and load textures. The canvas should be initialized with color black and enable depth test so that the objects behind will be blocked by the fronts. These configurations are all realized by calling functions of graphics library. Finally, `drawScene` function is called to draw objects from the arrays. Next clips of codes are in the `initBuffers` function with arrays of vertex coordinate information and color information of each vertex. The demo below shows briefly the creation of 3D object in WebGL:

```
var pyramidVertexPositionBuffer;
var pyramidVertexColorBuffer;

pyramidVertexPositionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexPositionBuffer);
var vertices = [
    // Front face
    0.0, 1.0, 0.0,
    -1.0, -1.0, 1.0,
    1.0, -1.0, 1.0,
    // Right face
    0.0, 1.0, 0.0,
    1.0, -1.0, 1.0,
    1.0, -1.0, -1.0,
```

```

        // Back face
        0.0,  1.0,  0.0,
        1.0, -1.0, -1.0,
        -1.0, -1.0, -1.0,
        // Left face
        0.0,  1.0,  0.0,
        -1.0, -1.0, -1.0,
        -1.0, -1.0,  1.0];
gl.bufferData(gl.ARRAY_BUFFER,          new          Float32Array(vertices),
gl.STATIC_DRAW);
pyramidVertexPositionBuffer.itemSize = 3;
pyramidVertexPositionBuffer.numItems = 12;

pyramidVertexColorBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexColorBuffer);
var colors = [
    // Front face
    1.0, 0.0, 0.0, 1.0,
    0.0, 1.0, 0.0, 1.0,
    0.0, 0.0, 1.0, 1.0,
    // Right face
    1.0, 0.0, 0.0, 1.0,
    0.0, 0.0, 1.0, 1.0,
    0.0, 1.0, 0.0, 1.0,
    // Back face
    1.0, 0.0, 0.0, 1.0,
    0.0, 1.0, 0.0, 1.0,
    0.0, 0.0, 1.0, 1.0,
    // Left face
    1.0, 0.0, 0.0, 1.0,
    0.0, 0.0, 1.0, 1.0,
    0.0, 1.0, 0.0, 1.0];
gl.bufferData(gl.ARRAY_BUFFER,          new          Float32Array(colors),
gl.STATIC_DRAW);
pyramidVertexColorBuffer.itemSize = 4;
pyramidVertexColorBuffer.numItems = 12;

```

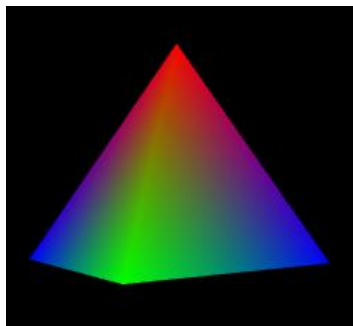


Figure 3.7. Output of WebGL

WebGL indeed accelerates 3D graphics on web browsers and mobile without any plugins and simplifies the ways for developers to build web-based games and applications.

However, there are drawbacks that WebGL's rendering does not completely match native graphics processing loads, it requires high processing power in hardware and it may have some security issues. The future of 3D web graphics is quite promising for the development of WebGL that the web served video games and online 3D games (web-based) and highly advanced web design might be possible which would be quite competitive with the traditional graphics field which is based on local graphics.

3.4. OpenCL

OpenCL (Open Computing Language) is a programming framework for heterogeneous compute resources to accelerate the graphic performance of devices in hardware level. The working principle of OpenCL is as shown in Figure 3.8. [20]

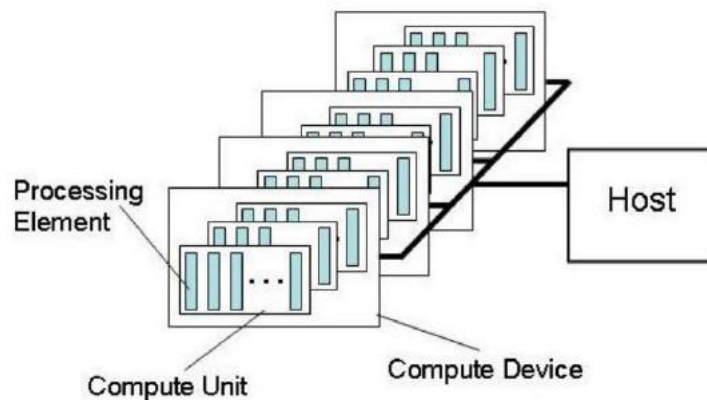


Figure 3.8. Architecture of OpenCL

There is one host connecting to one or more compute devices and each compute device consists of one or more compute units and each compute units is divided into one or more processing elements. The execution model of OpenCL defines N-dimensional computation domain and execute a kernel at each point in computation domain. The kernels are written in subset of ISO C99 (C-based and derivative language), which is basic executable codes and API to discover devices and distribute work to them. For example:

```
kernel void vectorMult(global const float* a,global const float* b,
                      global float* c){
    int id = get_global_id(0);
    c[id] = a[id] * b[id];
}
```

The target devices for parallel computing can be GPUs, CPUs, DSPs, embedded systems, mobile phones and even FPGAs. Program collects kernels and is analogous to a dynamic library. In command queue, applications queue kernels and data transfers and

perform in or out of order. Work-item is an execution of a kernel by a processing element. Work-group is collection of related work-items executing on a single compute unit (core). In image processing specific, work-items are pixels and work-groups are tiles.

In Figure 3.9, the memory model of OpenCL is shown. The blue blocks are private memories and they are computed per work-item. The green is local memory that is at least 32KB split into blocks with each available to any work-item in a given work-group. The orange is the global and constant memory. Till now, the working unit is on a single computing device. The red is the host memory on CPU to coordinate tasks and data when working. The memory management is clear in that the application must move data from host to global memory, then to local memory and back.

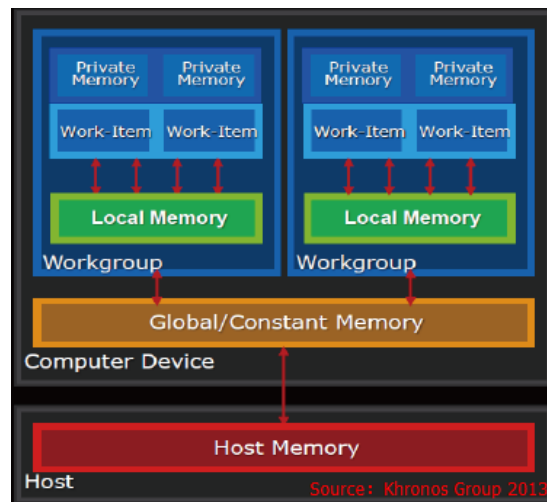


Figure 3.9. OpenCL Memory Model

The executing of OpenCL programs generally follows the procedure below [21]:

1. Query host for OpenCL devices;
2. Create a context to associate OpenCL devices;
3. Create programs for execution on one or more associated devices;
4. From the programs, select kernels to execute;
5. Create memory objects accessible from the host and/or the device;
6. Copy memory data to the device as needed;
7. Provide kernels to the command queue for execution;
8. Copy results from the device to the host.

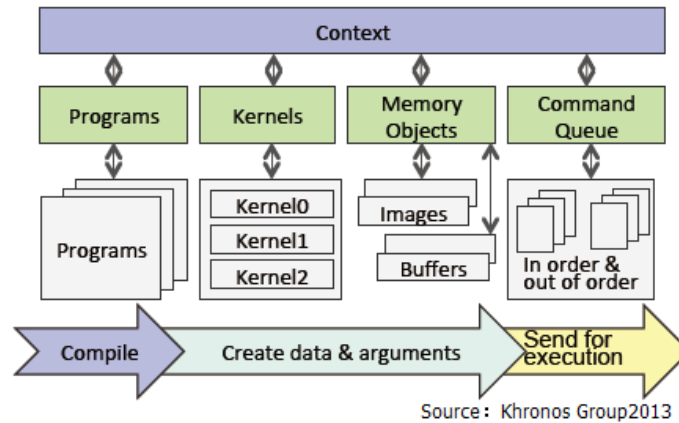


Figure 3.10. Executing OpenCL Programs

OpenCL 1.0 consists of a parallel computing API and programming language aiming at this kind of computing and it can be efficiently interact with API from OpenGL, OpenGL ES and other graphics API. OpenCL 1.1 is backward compatible with OpenCL 1.0 with the new features of supporting new data types, 3D vectors and more image formats, multi hosts and cross device buffering which can read from and write in 1D, 2D, 3D triangle regions and improve the interaction with OpenGL. OpenCL 2.0 improves the capability of shared virtual RAM, dynamic parallel computing and image buffering.

3.5. WebCL

First announced in March 2011, WebCL (Web Computing Language) is a JavaScript binding to OpenCL enabling acceleration of web applications especially allowing compute-intensive applications such as big data visualization, augmented reality, video processing, computational photography and 3D games through high performance parallel processing on multicore CPU & GPGPU. It also provides a single coherent standard across desktop and mobile devices. WebCL is close to OpenCL standard (Figure. 3.11) so that it preserves the familiarity and facilitates adoption with developers, allow them to translate their OpenCL code to web environment and keep both two synchronized as are evolving. The requirements for running WebCL are a browser supporting WebCL and hardware, driver and runtime support for OpenCL. WebCL intended to be an interface above OpenCL by facilitating layering of higher level abstraction.

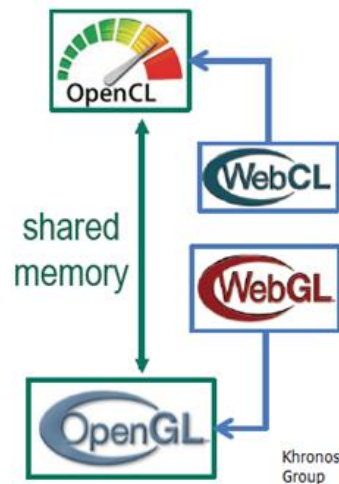


Figure 3.11. OpenGL, OpenCL, WebGL and WebCL relations

Similar to OpenCL, the procedure of programming WebCL is to initialize a working environment, and then create and run kernel. Then creating image objects by programming with WebGL from Uint8Array, , <canvas>, or <video>, WebGL vertex buffer and WebGL texture and animate them [22]. Basically, there are no more coding specifics to explain which is same with WebGL but there are more grammar manner in writing codes with WebCL for certain CL signs.

Here are two examples from Samsung WebCL Demo named N-body simulation and Deformation [23]. The test platforms for both demos are hardware: MacBook Pro (Intel Core i7@2.66GHz CPU, 8GB memory, Nvidia GeForce GT 330M GPU) and software: Mac OSX 10.6.7, WebKit r78407. N-body simulation calculates the positions and velocities of N particles and animates them and simulates them in JavaScript and WebCL with 2D/3D rendering option. Deformation calculates and renders transparent and reflective deformed spheres in skybox scene. The result turns out that WebCL computing speed is significantly faster than the original Javascript. It will be a main 3D tool for web and mobile 3D application in the future.

Table 3.2. Performance comparison of JavaScript vs. WebCL

Demo Name	JavaScript	WebCL	Speed-up
	5-6 fps	75-115 fps	12-23x
N-body simulation (1024 particles)	<p>N-body simulation (left: JavaScript, right: WebCL)</p>		

	~ 1 fps	87-116 fps	87-116x
Deformation (2880 vertices)			

WebCL standardizes portable and efficient access to heterogeneous multicore devices from web content, defines ECMAScript API for interaction with OpenCL and designs for security and robustness. It is based on OpenCL 1.1 Embedded Profile, implementations may utilize OpenCL 1.1 or 1.2. The standardization is interoperability between WebCL and WebGL through an extension and initialization simplified for portability. However, WebCL also has some restrictions such as that kernels do not support structures as arguments, kernels name must be less than 256 characters, mapping of CL memory objects into host memory space is not supported, binary kernels are not supported and restrictions on buffer access for security.

4. 3D GRAPHICS DEVELOPMENT TOOLS

This chapter is devoted to software tools facilitating 3D graphics applications development. Two types of tools are needed for 3D applications: object/scene modelling software and animation software. As animation software special software called game engine can be used, which has an advantage and it is highly optimized. The use of software tools is illustrated on an example process specific application development. We use software package Maya for modelling and Unity game engine for animation. Application selected is Web shop for clothing and fashion using 3D graphics for cloth fitting.

4.1. Web shop with mobile 3D graphics application

We illustrate the process of developing mobile 3D graphics applications using generic case of Web shop where the usage of 3D would be very useful and attractive. This is the case of Web shop for buying cloth and fashion items. Buying cloth over the Web is complicated by the fact that cloth has to fit so it looks well on the user's body. In traditional clothing shops users are testing how the cloth is fitting by wearing it and looking in a mirror. This is not possible in the Web shop so there is significant risk that after ordering an item it will not fit and will have to return. Many Web buyers are trying to solve this problem by ordering items in multiple sizes and colors. After finding the one which fits they return the other items using the law that Web shops have to accept any returns without any costs to buyers. But returns bring significant added costs to Web shops involved in clothing making them unprofitable in the worst case. Using 3D graphics may potentially offer good improvement for this situation. For this users would have her/his 3D model ready (such detailed model could be created using e.g. Microsoft Kinect-type 3D scanner) and the Web shop system would have software which could place the cloth item on a model to check if it fits. User could then look at his/her 3D model from different positions to check the fitting. Implementation of this idea requires sophisticated 3D graphics hardware and software to accurately model the 3D fitting with realistic cloth material properties like e.g. softness and flexibility. But with the recent advances in graphics hardware this should be possible to realize in the current high-end mobile devices. In a near future every mobile device will have such capability and then the 3D graphics applications should become widespread.

The problem of virtual clothing fitting to human body model is not simple and trivial since the modelling in this case should be as realistic and life-like as possible, preserving delicate cloth properties. This requires visualizing the dynamic effect of cloth material

and interaction between the user model and clothing. Simulation system of cloth fitting should thus include construction of human-like model, the model animation, and realistic model of cloth including texture, physical properties, animation and lighting.

The overall aim of building such 3D graphics model is to visualize the static and dynamic effects of clothes on characters. In real-life, human body and any clothes put on it will not stay still, i.e. there will be dynamic visual effects in any cloth categories such as shirts, pants, jeans, dresses, jackets and so on. For the process of building cloth models, one can notice that most of clothes are complicated in design, they may have thousands of meshes in different sizes and directions. In terms of cloth simulation, there are two main aspects one should look into: texture materials and physical animations. First, the material description in 3D model is its texture which covers the surface of 3D objects. In 3D graphics, textures are bitmaps with the colors and patterns of material, textures of materials can be acquired by taking photographs of them. Second, the physical animations require taking into account impacts due gravity, collisions between different clothing parts and folds. These are common effects that are visible and can happen to any kind of clothes. But for garments of different kinds, the mass, coefficient of friction, extent of folds, tolerance of tearing and many other effects are varying widely. These factors add serious complexity to modelling but luckily modern modelling and animation software have tools to deal most general physical parameters of a cloth and allows users to manipulate and interact with the cloth to create the ideal effects they want to have for the clothes [24]. This will be covered later in the chapter. The textures and animations are not independent from each other, in other words, each category of material has its own textures and physical characteristics in animation which should be considered as a whole. For instance, silk clothes are light and bright in reflection, and when simulating silk, the gravity factor should be set with low value and texture should be set with high reflection value. In animation, such cloth will fit the body more smoothly and softly. But for a leather jacket, which is heavy and dark, the parameter settings should be totally different. In the example in the thesis considering best visualization and easiness in construction, the cartoon character “*Batman*” and the type of cloth “*Cloak*” are chosen to explain the idea of virtual clothes fitting with cloth dynamics. This example also shows how tools developed for 3D gaming can be used in this application. The software tools which are used are Maya for modelling and Unity for animation.

4.2. 3D Modelling Software Tools

Generally, 3D modelling is the method to build 3D models or objects in virtual 3D world by using 3D modelling software. There are two major approaches for constructing models: NURBS (Non Uniform Rational B-spline) and polygon meshes. NURBS is very demanding on details and is suitable for complex models, while polygon meshes is suitable for build complex scenes and animations. Building models from scratch would

be very complicated and time consuming. Many software tools are available to simplify this task. The major 3D modelling software commonly used in the industry are 3DS max, Maya, Softimage|XSI and Rhino. Before specifying the tools, let's take an overview of the four 3D modelling tools in short.

3Ds max is advanced 3D animation software which performs perfectly in 3D modelling, animation and rendering and meets highest demands for the quality. It is designed to run on less powerful personal computers with lower hardware requirements. Regarding to the software itself, the user interface is easy to learn and it supports quite many plugins with rich 3D effects such as fur, hair and skin. It also has strong character animation functionality and its own controlling script- "Maxscript" allows users to customize functions.

Maya is 3D computer graphics and animation software from Autodesk which runs on Windows, Mac OS and Linux to create interactive 3D applications including video games, films, TV series and visual effects. Maya is a mature 3D modelling engine with diverse visual effects working with particles, cloth, human animation, camera, fur, hair and so on. Developers can simulate almost all the physical effects in real world in the engine. Thousands of video games, 3D games and movies are made using Maya with rich contents and effects, for example, the movie "The Spider Man", "Ice Age", "Harry Potter" and "Avatar".

Softimage|XSI is developed by digital media company AVID. It is the first 3D software that introduces the concept of non-linear editing changing the animation development process, which makes it very suitable for animation production in films. Actually, Softimage has been the major software in animation production in Hollywood for years. The most famous of Softimage|XSI is the Mental Ray super renderer containing rich 3D algorithms and high quality imaging.

Rhino is widely used in 3D animating, industrial design, scientific research, architecture, mechanical and automation design. It can be easily integrated with partial 3Ds max and Softimage functions and it is suitable for 3D NURBS models. The exported formats can be OBJ, DXF, IGES, STL, and 3dm allowing team works.

Of course, all the mentioned 3D modelling software are qualified for the model construction in the early stage. Based on the familiarity and previous experience, Maya is used to create 3D model and animation in this thesis. The reason for this is that Maya has very friendly user interface, offers great deal of effects with easy operations, it has accurate simulation of human body kinetics for animation as well as intuitive polygon modelling. Additionally, Maya provides vast range of key-frame, nonlinear editing and advanced character animation which is not very demanding for programming skills. That is much easier for many developers who are not experienced programmers. In gen-

eral, there are multiple ways of constructing object meshes in Maya. Developers can build objects parts separately and then put them together in 3D world with proper position to make a whole object. When building detailed parts, for example head, developers can modify model from a basic sphere by editing meshes, vertices and edges. Because human body is symmetrical only building body half part is enough, etc.



4.3. Game Engines for Mobile Applications




3D game engines are animation software tools to make a “real world” on computers to display physical interaction. There are very many game engine tools which widely differ in scope and capabilities. In most cases this refers to a tool which game developers use to build 3D characters, game scenes and 3D objects which are the basic elements of games. The meaning and values of 3D engines differ much since there are thousands of projects and different users, it is hard to define 3D engines overall. However, when seen from the functionality level, there are some common features:

- Data Management which consists of scene management, sequences management, exchange and interaction with external tools and constructional 3D data organization and display.
- Renderer is a key part for 3D engine since the performance of the 3D are partially judged by the capability of renderer such as the algorithms of lighting, reflecting, surface diffusion and so on.
- Interactivity means the interaction with other developing tools like scene editor, script editor and particle editor.

Basically, if an engine has the 3 functions above, it can be a simple 3D engine. Most popular and often used 3D developing engines are listed in Table 4.1 to compare them based on the aspects such as available platforms, price, language, graphic ability and developed games. The five game engines are chosen because of the good developer reviews and there are already some well-developed games using these game engines.

Table 4.1. Most popular 3D developing tools/engines for mobile devices

	Plat- forms	Orienta- tions	Price	Skill	Lan- guage	Games
Corona SDK		2D 3D	Trial is Free, Indie \$199/Year, Pro is \$349/Year	Interme- diate	Lua	11 Games
ShiVa3D Game Engine		3D	Web: Free, Advanced:\$1000, Basic:\$200, Educational:\$670	Interme- diate	Lua	6 Games

SIO2 Engine		2D 3D	Trial is Free, Win/Mac:\$199.99, Android/IOS:\$399.99	Interme- diate	C++	7 Games
UNIGINE Engine		3D	Case-by-case (about \$30,000 USD/project)	Beginner	C++	1 Games
Unity3D		3D	Unity:Free, Unity Pro:\$1500	Interme- diate	.NET (Mono) Scripting: JavaS- cript, C#, Python	14 Games (Temple Run)

Unity is an engine that all developers, can afford. There are two versions of Unity. Basic one is free to all and Pro version is for more advanced developers, it contains more features and effects that make a game rich in contents and vivid in effects. The key feature of Unity is that it is cross-platform. The development work is done on the engine which is platform independent. If users are familiar with 3D construction engines, there won't be any difficulty for them to start with. After construction, users can choose a platform to render and run the 3D models on. As mentioned above, the options cover all the current OS in the market - Mac, PC and Linux desktop computers, Windows Store the Web, iOS, Android, Windows Phone 8, game consoles PS3 and Xbox 360. Users do not have to acquire the knowledge of development for a specific platform, Unity will allow building for the chosen Operating System provided all configuration environment is set correctly.

However, if users want more controls and interactions in the games or applications, they can write some scripts to manipulate the objects and control the views. In this case, the developers should have some programming skills. Even if programming is required, it is done with script languages such as Javascript and Boo (Unity own script) which are much simpler than languages like C++.

Unity is thus a good choice, taking into account of the following aspects:

- Price of the engine. For basic version, it is free of charge. Basic version can meet the most demands and requirement for game developers. For Unity Pro with some extraordinary effects, it will cost customers 1500 dollars per year.
- Language. The control script languages are familiar to developers such as C# and Javascript.

- Usability. The user interface of Unity3D is similar to 3D software on computers which will make for developers easy to adapt to the developing environment. It is easy to use and can make professional level effects with very simple operations.
- Platform. Games can developed on Unity3D for many platforms including PC, Mac and Linux, iOS, Android, Xbox 360, PS3 and Wii. It means applications developed with this engine can run on any platform.

Unity uses OpenGL ES to render for Android and iOS devices and versions from Unity 4.2 uses the new features of OpenGL ES 3.0.

The layout of Unity user interface is quite similar with other 3D modelling software with menu, four construction view, game view, inspector, editing tools, file section and hierarchy session in the current scene. Example of developing an application will be provided later.

4.4. 3D Graphics Application Generation

Basic process of developing 3D mobile graphics application is composed of the following steps:

1. Importing models/characters from outside (e.g. made in Maya or other 3D modelling software), with or without animation, to Unity and mapping materials.
2. Building 3D physical scene (terrain, buildings and other objects for collision) with materials and set natural factors like lights and skybox
3. Set animation with user input interaction controlling with mouse, touch or keyboard
4. Simulate visual and audio physical effects from real world
5. Build user interface controlling system
6. Export to mobile devices, PC or other platform of choice

Take a simple game in which a character is roaming on a terrain with fixed edges and skybox as an example. The desired goal of the game is to display the character in different running animations and roam on a terrain such as mountains with different orientations using keyboard control [25].

First, developer should construct a model in 3D modelling software with different animation such as running to the left, forward and right and jump. The poses are all set in

3D modelling software (Maya) and exported together with the 3D model itself. Then import the character with animation clips to the game engine (Unity) and make configurations with the model. Actually, Unity can recognize original formats of 3D software like .3ds and .mb as well as formats with plugins like .fbx, .obj. Unity will automatically translate them into the format that can be read and written by it.

Secondly, developer should build 3D scene, in this example, a terrain with fixed edges and draw some trees onto the “panel”; a skybox as the sky background and a main camera as the game view. The camera should be right behind the character and should follow the character along the way.

Thirdly, users should be able to control the character to run freely in the scene as they want with user control, for instance, controlling by keyboard inputting different keys. This is realized by attaching scripts to the objects in the scene. In this situation, attaching the script to the objects combining the model itself and the main camera makes that if user presses a certain key on a keyboard, there will be corresponding reaction with the model’s running poses and the position of the model and the camera will be transformed in the scene as well. The controlling script can be in C#, Javascript or own Unity script. The Unity system also provide whole functionality package for keyboard, touch and mouse interaction.

Next step is adding audio and visual effects if developer want the game to be more precise and rich in contents. Then putting user interfaces for controlling and trigger some events in the scene. It is common in mobile games to have touch interaction since there are no keyboard and mouse available.

Finally, the whole program is ported to a selected platform for example Android. This is an automatic process if developer has the correct configuration environment for the application development.

We will illustrate the process of building the application based on the example of Web cloth shop with virtual clothes fitting and body animation as described before.

4.4.1. Modelling in Maya

Modelling in Maya uses the way of separated construction of body parts. The body construction is symmetric, so we only need to construct half of the body parts. Maya provide the function of mirror symmetric editing that the objects built and the modifications made will be seen in the other half. Generating detailed objects like head, arms and chests starts from original primitive objects like sphere, capsule, etc. Making shapes

of the objects is done by dividing meshes into more pieces as required and changing the position of vertices, surfaces, and edges. Then, adding proper material texture to the objects will make them look real, e.g. body skin. The construction method is by rooting from the hip of the body for both upper part and lower part of the body. For upper body, create joints from hip to waist, chest, clavicles, shoulders, elbows and hands, going up to neck, chin and forehead. Thus basic control of the arms, heads and turns of chest can be realized with the joints and bones. For lower part of the body, the construction is easier which also rooted from hip then goes to left and right with the exactly same structure from thighs to knees, and till heels and toes. However the lower half of the body is more complicated since the movements of legs should obey the rules of kinematics enabling bend of knees for example with the “IK Handle Tool” in Maya.

When the construction in Maya is done, in order to make the character act like human, the very first thing is to animate the model, i.e. add human movement to the model because what is built in the 3D software are only the meshes with skin texture of the model. So “Skeletons” should be added to the model so that the textured meshes can move with the skeletons to simulate body movement and all controls can be done with the “bones” when the meshes are combined with skeletons.

The construction of skeletons should be at least to cover basic human body parts which are needed for movements. For cloth fitting, the body setting is idle standing with random and slight movements of the body.

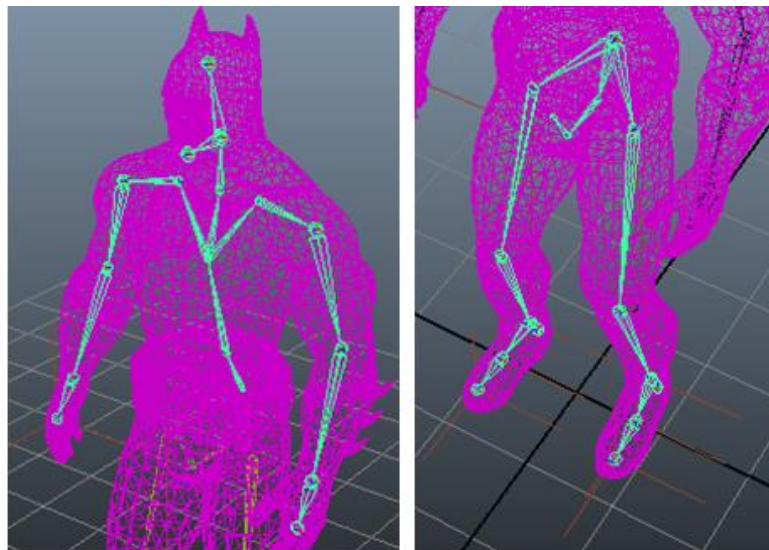


Figure 4.1. Snapshot of Body Skeleton Construction with Upper(Left) and Lower(Right) Part of the body

Finally, the built skeletons should be combined with meshes to make the model move like a real human character. Animation is made of 24 frames per second with loops for slight up and down movements with arms swaying and then key frame simulation is

baked, and the whole scene exported in FBX file format which can be read by Unity. In this way all with Maya is done.



Figure 4.2. Snapshot of Whole Body with Skin(Left) and Skeleton(Right)

4.4.2. Animation in Unity

After baking in Maya, the FBX file is containing animations which can be recognized by Unity. The first step of simulation is done and the imported model can make movements in Unity. Till now, the model can move naturally like a human with swinging arms and random movements of legs. Next and also the core part is the cloth simulation and it is realized by the function in Unity called “Interactive Cloth”. The Interactive Cloth class is a component that simulates a "cloth-like" behavior on a mesh. This component is used if developer wants to use cloth in the scene. The virtual clothes fitting are done with the similar concept but clothes act in a more detailed manner since cloth is not “decoration” but the main objects that are going to be viewed in details. The main components of the “interactive cloth” are the physical factors that are added to the cloth mesh as shown in Table 4.2.

Table 4.2. Interactive Cloth Properties in Unity

Properties	Explanation	Setting
Bending Stiffness	Bending stiffness of the cloth.	0.75
Stretching Stiffness	Stretching Stiffness of the cloth.	1
Damping	Damp cloth motion.	0
Thickness	The thickness of the cloth surface.	0.2
Use Gravity	Should Gravity affect the cloth simulation?	<input type="checkbox"/>
Self Collision	Will the cloth collide with itself?	<input checked="" type="checkbox"/>
External Acceleration	A constant, external acceleration applied to the cloth	X 0 Y -5 Z 0
Random Acceleration	A random, external acceleration applied to the cloth	X 0 Y 0 Z -15
Mesh	Mesh that will be used by the interactive cloth for the simulation	Batman_Year1_
Friction	The friction of the cloth.	0.5
Density	The density of the cloth.	0.001
Pressure	The pressure inside the cloth	0
Collision Response	How much force will be applied to colliding rigid-bodies?	12.9
Attachment Tear Factor	How far attached rigid bodies need to be stretched, before they will tear off.	100
Attachment Response	How much force will be applied to attached rigid-bodies?.	0.2
Tear Factor	How far cloth vertices need to be stretched, before the cloth will tear.	2000
Attached Colliders	Array that contains the attached colliders to this cloth	<ul style="list-style-type: none"> Attached Colliders Size 5 Element 0 <ul style="list-style-type: none"> Collider L_shoulder (Spl) Two Way Inter: <input checked="" type="checkbox"/> Tearable <input type="checkbox"/> Element 1 <ul style="list-style-type: none"> Collider L_shoulder1 (C) Two Way Inter: <input checked="" type="checkbox"/> Tearable <input type="checkbox"/> Element 2 <ul style="list-style-type: none"> Collider R_shoulder (Spl) Two Way Inter: <input checked="" type="checkbox"/> Tearable <input type="checkbox"/> Element 3

Detailed explanations of factors that are useful in the case of virtual cloth fitting are listed below in the order of priority [27]:

- **Mesh:** In general, interactive cloth can apply to previously constructed meshes. In other words, users can first build the meshes by another software and import them to Unity then can create an object of interactive cloth and apply the mesh to the cloth
- **Use Gravity:** This option enable the cloth simulate gravity effects on the mesh as a whole object
- **Attached Colliders:** If users want the clothes to hang on the human model, it should be attached to colliders on the model for tow effects: hanging and collision which are keys to simulation of clothes fitting

- **External Acceleration and Random Acceleration:** External Acceleration is like gravity which is external force to the cloth. Random acceleration here can simulate the wind field since the force is not constant and comes from mainly one direction all the time, i.e. there will be slight and random changes in the magnitude of the force

Regard of other factors, users should adjust them several times and try them in player view to see whether the effects reach the ideal simulation effects.

In real life, the clothes can never go through the body. As mentioned before, 3D model is a mesh but not a collider, which means there is no collision effect between each other. So the first step is to make the 3D model into a big collider or rigid body for collision. There are different options in Unity to create collider such as box collider, sphere collider, capsule collider, mesh collider, wheel collider and terrain collider. If the model is static, the solution would be very simple to create a mesh collider based on previously built 3D body mesh and all the collision effects can be realized. But the output display is then quite clumsy with a cloak fluttering behind a static body. So the mesh collider may not be suitable for this situation by two reasons even though the collider fits the body perfectly. First, mesh collider is not dynamic and cannot move with the body and second even if dynamic mesh collider can be created, the computational load can be very heavy since the mesh may contain thousands of vertices and facets. With every frame refreshing, there will be difference that should be calculated. After refreshing with a certain period of error accumulation, this would affect the display effects.

To solve the problems of moving colliders, the solution is to bond the colliders to the skeleton joints which are moving with the body so that the colliders can move with the body too. Other colliders are simple in shapes and they contain much less meshes. The only and fatal drawback is that simple shaped colliders cannot cover all the surface of the models, and it makes the clothes pass through some parts of the model. The perfection and precision of collision effect is depending on the numbers of skeleton and joints that are constructed on the model. The more joints are built, the more colliders one can build upon them and the more perfect the collision effect it will be. In Figure. 4.3, the collider bonding screenshot of the model is shown.



Figure 4.3. Model with colliders and output in the engine

4.4.3. Mesh Rendering

One goal of the application is the online clothes changing and fitting function. In Unity System, there are mainly two ways to implement this with mesh renders:

1. The model has fully developed meshes with all the clothes and other accessories on the body. To put on clothes means showing the hidden part of the model in render with codes, but this has some drawbacks that rendering with the whole model sometimes will slow down the speed since the facets of the meshes will increase the calculation load and the times of draw will also increase.
2. In most 3D games, developers prefer to separate the body parts and deal with a single object and it is also the solution used in this thesis for the application of clothes changing. Taking the change of cloaks as an example, the solution is that the 3D model construction and geometric structures of cloak are the same, the only slight difference might be the softness factor and other minor nature of the cloth which are not making big difference in visual output. The most distinct visual difference should be the patterns, colors on the clothes, interpreted into 3D terms by “Materials” property of the clothes. In a 3D game engine, materials are referring to a picture or bitmap of the patterns on the clothes and the reflection parameters of the materials. With the simulation of these two factors, the objects with rough meshes become life-like clothes or any other objects developers would like them to be.

In our demo, the mesh of the cloth and the whole dynamic animated outfit are not changing and changes are only made on the surface of the materials. In the engine, users can set in “cloth render” inspector and choose from the material asset file in the project. But in the application, users do not have access to back stage of development and they can only choose from the user interface. So there should be a pre-defined material li-

brary for users to choose from and fit it to the model. This is illustrated by a code snippet [28]

```

System.Random random = new System.Random();
public int SelectedNumber;
// Set the materials in the inspector
public Material[] myMaterials = new Material[12];

// Use this for initialization
void Start () {
    // Assigns a random material at start
    gameObject.renderer.material = myMaterials [random.Next(0,myMaterials.Length)];
}

// Update is called once per frame
void Update () {
    renderer.material = myMaterials[SelectedNumber];
}

```

This is the code from the script that is binding to the game object, in the case the interactive cloth. The command “*public Material[] myMaterials = new Material[12]*” creates a list of 12 materials and “*renderer.material = myMaterials[SelectedNumber]*” in update function change in the object with the selected material.

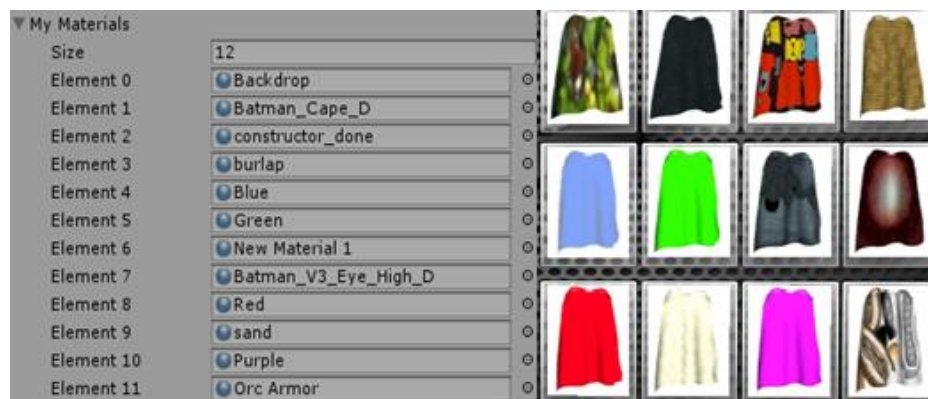


Figure 4.4. Material Inspector and Corresponding UI Layout

As can be seen from the figure, there are textures on the image buttons with corresponding indexing pictures. In order to make the buttons texture more intuitive, the images are made with the real look of cloak by taking snapshot. The message sending and event system are the same with that of button “Rotate” and “Move” by using UI Event Listener.

Seen from a general perspective of clothes fitting and changing, the processes can be divided into:

- Building meshes for different categories and styles of clothes. One has to take into account that the construction of tops, pants, T-shirts, skirts, dresses, coats, jackets is totally different. Building basic meshes for each category is essential.
- Modifying of the “general” mesh for each type and styling it into a specific and designed manner. The changes would not be so huge for one category, such as pants in meshes construction, thus creating unique meshes for series of clothes.
- Mapping material to the meshes with the designed patterns, colors and designs.

There is quite a big work load when the application grows into a big industrial use with the need for creating thousands of different styles materials and the construction of meshes.

As mentioned before one way to make changes in clothes is to make the invisible meshes by disabling the mesh renderers which are binding the meshes to render colors and textures. Though the method is not taken in clothing changing process, it can be used to hide the character body to make users see the single piece in details. By pressing the buttons for enabling and disabling the renderer, the body can be visible and invisible at users’ preference. The following code can be add to the start and update function to any script [28]:

```

if(flag%2 == 1) {
    SkinnedMeshRenderer[] marr=target.GetComponentInChildren
    <SkinnedMeshRenderer> (true);

    foreach(SkinnedMeshRenderer m in marr){
        m.enabled = true;
    }
}
else{
    SkinnedMeshRenderer[] marr = target.GetComponentInChildren
    <SkinnedMeshRenderer>(true);

    foreach(SkinnedMeshRenderer m in marr){
        m.enabled = false;
    }
}

```

Here is a short explanation of the code. Flag is the variable in UIButton event response function that counts the clicks of the button. SkinnedMeshRenderer is the name of the renderer binding with the mesh. In general, the name can be changed into any kind of renderer as long as it is in the Render class in Unity. GetComponentInChildren includes all the children of the object which is the target in the codes. So one can create an array for the objects and disable all renderers with `m.enabled = false` and enable with

true. In general case, one can make any meshes with renderer invisible with above commands.

4.4.4. User Interface

After the construction and animation simulation, the application should have a way to interact with users, i.e. users are supposed to manipulate the model and view the clothes in details.

First of all, in creating user interface, with every camera, the Unity provides GUI in inspector so user can easily create widgets-like buttons. But there is also a powerful UI system in Unity called NGUI (Next-Gen UI kit) written in C#. The reason to call it powerful is that it is fully integrated with inspector and users can directly see the results on Game view without hit Play, and it has flexible event system and UI response.

Below is a simple example for building two buttons on the application UI layer to view and manipulating the model. The building route for the UI is

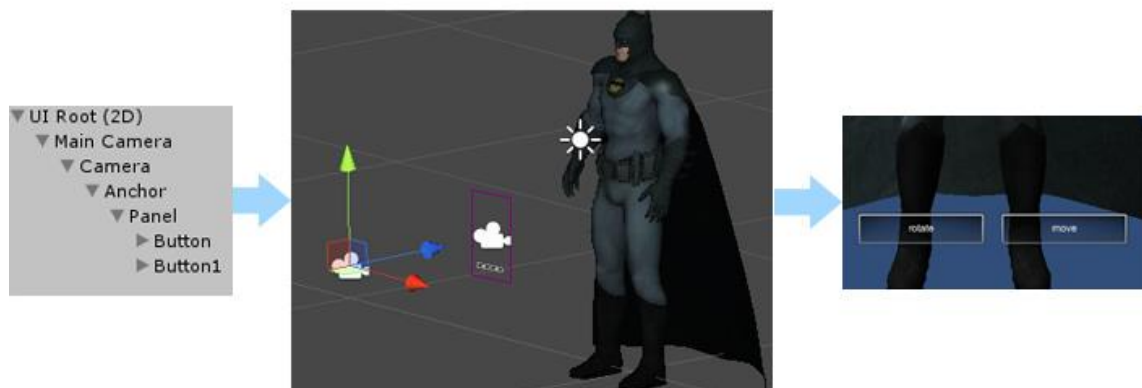
1. UI Root (2D); It is the root of the UI and determines the nature of the UI which can also be in 3D
2. Main Camera; Usually, the main camera is not supposed to be in the UI layer, but in most 3D games and in the case of our application, the UI of the scene is not changing as the movement of the character and components stay still on the screen. The solution is to move the main camera to the UI layer so that the UI layer will move with the main camera which is seen as staying still on the screen.
3. Camera; The camera here is not a real camera but a tool to render the UI components that have been created and catches various kinds of UI events and response.
4. Anchor; It is a tool to make the relative position of the UI elements on the screen.
5. Panel; It is the drawing board that users can have any kind of UI widgets on it.
6. Button; or any other UI elements. It contains background, label and some scripts dealing with the event on the buttons. There are some functions to set the attributes, such as `UIButtonColor`, `UIButtonScale`, `UIButtonOffset`, `UIButtonSound`, `UIButtonPlayAnimation` and `UIButtonMessage`. Users can customize some events on `MonoBehaviour` to response to more events.

In Table 4.3 there is list of mouse and interaction functions in Unity [27].

Table 4.3. *Functions for Mouse and User Interaction*

void OnHover (bool isOver)	Sent out when the mouse hovers over the collider or moves away from it. Not sent on touch-based devices.
void OnPress (bool isDown)	Sent when a mouse button (or touch event) gets pressed over the collider (with 'true') and when it gets released (with 'false', sent to the same collider even if it's released elsewhere).
void OnClick()	Sent to a mouse button or touch event gets released on the same collider as OnPress. UICamera.currentTouchID tells you which button was clicked.
void OnDoubleClick ()	Sent when the click happens twice within a fourth of a second. UICamera.currentTouchID tells you which button was clicked.
void OnSelect (bool selected)	Same as OnClick, but once a collider is selected it will not receive any further OnSelect events until you select some other collider.
void OnDrag (Vector2 delta)	Sent when the mouse or touch is moving in between of OnPress(true) and OnPress(false).
void OnDrop (GameObject drag)	Sent out to the collider under the mouse or touch when OnPress(false) is called over a different collider than triggered the OnPress(true) event. The passed parameter is the game object of the collider that received the OnPress(true) event.
void OnInput (string text)	Sent to the same collider that received OnSelect(true) message after typing something. You likely won't need this, but it's used by UIInput
void OnTooltip (bool show)	Sent after the mouse hovers over a collider without moving for longer than tooltipDelay, and when the tooltip should be hidden. Not sent on touch-based devices.
void OnScroll (float delta)	Sent out when the mouse scroll wheel is moved.
void OnKey (KeyCode key)	Sent when keyboard or controller input is used.
void OnHover (bool isOver)	Sent out when the mouse hovers over the collider or moves away from it. Not sent on touch-based devices.

We describe just a rough and basic path to create a simple UI with a few elements on the panel (Figure. 4.5), but the working principle is the same with bigger UI projects.

**Figure 4.5.** *Logical Process of Building UI and its output*

Taking button as an example of monitoring events that are taking place in NGUI, there are three ways to realize it:

- The simplest way is to bind the script directly to the button to monitor events.

```
void OnClick( ){
    Debug.Log("Button is Click!!!");
}
```

In this case, every time the button is clicked due to the OnClick function, there will be “Button is Click!!!” printed out in debug window .

- Using SendMessage

In NGUI, there is a function called Button Message (Figure. 4.6) which is used to send messages when events are happening on the button. It is more complicated than the first method as it contains some parameters that should be set by developers.

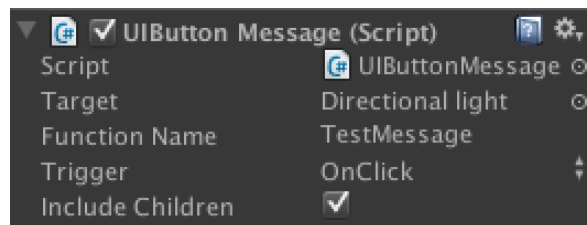


Figure 4.6. UIButton Message Inspector

As can be seen from the Figure 4.6, Target is the object that is receiving message from the button, Function Name is the function that should be contained in the script which is binding to the object so that it can be recognized in script and Trigger is the action that triggers the event, such as click, press, mouse over, double click and so on. This method is more flexible than just binding a simple script on objects and developer can add customized code to the script to allow more effects.

- Using UIListener

The most flexible way is to use the UIListener. This is done by adding the component -NGUI-Internal -Event Listener to the buttons that one wants to interact with and it does not have any parameter to set up. Then, in any script in the game scene, one adds the following functions, taking OnClick as an example:

```
void Awake ( ) {
    //Get the button objects that should be listened to
    GameObject button = GameObject.Find("UI Root
    (2D)/Camera/Anchor/Panel/LoadUI/MainCommon/Button");
    //Set the action of the button to the function in the same
    //class ButtonClick
    UIEventListener.Get(button).onClick = ButtonClick;
}
```

```

void ButtonClick(GameObject button){
    Debug.Log("GameObject " + button.name);
}

```

The reason of using different buttons in the UI system is for the finger gesture manipulation of the 3D model which will be explained next.

4.4.5. Finger Gesture and Mouse Interaction

As mentioned before, the reason to set different button action is for viewing of 3D models. This is different from 3D games in that the manipulation is mainly focusing on movements of the characters. The interaction with 3D model is to view the model or the clothes on model in details, so it is more like viewing a photo in 3D. According to the interaction gesture experience, users are used to the gestures like drag, swipe, tap, double tap, pinch and spread and the effects are rotation, moving, zoom in and zoom out [26]. The reason to use separated buttons is for the rotation and movement since the most frequent gesture is drag with single touch. So with different button, users can rotate and move the view with the same drag gesture. Other gestures like double tap and spreading fingers correspond to zoom in and out.

In the Unity C# scripts, there are some general start codes, the control of finger gesture is realized by some functions of Input class and movement of objects in Unity are realized by “transform” class such as rotation and translation. The variables and functions are shown in Table 4.4 [27].

Table 4.4. Variables and Functions in Unity for Interaction

General Starting Codes	
using UnityEngine; using System.Collections;	
void Start ()	Use this for initialization
void Update ()	Update is called once per frame
Input Interaction	
Static Variables	
multiTouchEnabled	Property indicating whether the system handles multiple touches.
touchCount	Number of touches. Guaranteed not to change throughout the frame. (Read Only)
touches	Returns list of objects representing status of all touches during last frame. (Read Only) (Allocates temporary variables).
mousePosition	The current mouse position in pixel coordinates. (Read Only)
Static Functions	
GetTouch	Returns object representing status of a specific touch. (Does not allocate temporary variables). Touch phases are TouchPhase.Began, TouchPhase.Moved and also it can calculate

	delta position-Input.GetTouch(0).deltaPosition.
GetMouseButton	Returns whether the given mouse button is held down.
GetMouseButtonDown	Returns true during the frame the user pressed the given mouse button.
GetMouseButtonUp	Returns true during the frame the user releases the given mouse button.
Object Transformation	
Variables	
Transform.position	The position of the transform in world space.
Functions	
Transform.Translate	Moves the transform in the direction and distance of translation. void Translate(float x, float y, float z, Space relativeTo);
Transform.RotateAround	Rotates the transform about axis passing through point in world coordinates by angle degrees. void RotateAround(Vector3 point, Vector3 axis, float angle);

After this preparatory knowledge for writing a script to control and view 3D models, we will now provide short explanation for the scripts to control the model and interact with the UI system.

When first considering finger manipulation separating rotating and moving as mentioned before, one idea is to rotate the model itself and moving with the camera in order to separate the corresponding objects. If the object is a simple cube or sphere, the rotation is smooth and requires less calculation. In the real case, the model is complex consisting of thousands of meshes. If the model is rotating by itself, then when refreshing every frame, the computational work will be very heavy due to the animation, random dynamic physical effect and so on. There will be some errors as well for real-time calculation with the clothes visual output [29]. So if the model rotates around by itself, the result could be as in Figure 4.7.



Figure 4.7. Distorted cloth when rotating model itself

So the solution is to move only the main camera in the scene and keep the model still in one position only to deal with the physical animation. The interaction can be realized by putting two main functions- moving and rotation in one script. The switches between buttons can be realized with two variables.

4.5. Exporting project

Exporting project is the easiest step in the development since Unity is helping developers with the basic coding and construction of the application for many main platforms such as Windows, Mac, Linux, IOS, Android, Windows Phone, and others just by selecting environment as shown in Figure. 4.8. In the case of Android application development, after installing the development environment for Android system, developers just have to configure for players and device and then Unity will do all other work required. This means that developing with Unity does not require any Android developing experience. One just needs some C# or Javascript programming skills and knowledge of specific Unity functions.

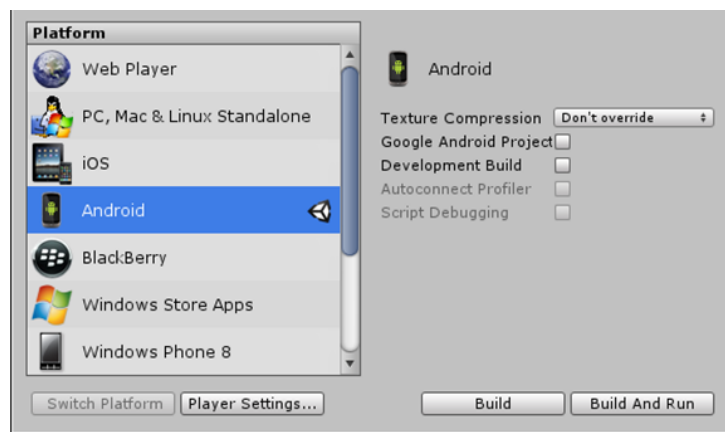


Figure 4.8. Export Setting and Platform Options

5. RESULTS AND DISCUSSION

In this chapter, there will be description of the demo as an output of development and the discussion of improvements towards the application to have a better result.

5.1. Results

The output of the demo is an application with 3D characters. Below is a snapshot of the organization of game objects and user interfaces in the 3D world scene.



Figure 5.1. Scene Organization in 3D engine

There are three main parts consisted in the whole application –

- Scene; in the “Scene”, there are all game objects, lights, main camera and background images. It can be seen from the left hierarchy list that batman_cover is a capsule collider covering the whole model, bat_anim is the FBX file imported to Unity with animation which has three children, the interactive cloth, the body mesh and the skeleton build within the skinned meshes. In “Pivot”, there are two directional lights and main camera with a child of “backdrop” which is the background image. The reason to set two lights in front and behind is for the visualization of the clothes in both two sides and to put background image under main camera so that the image would move with the main camera which turns out in the game view is a steady background.
- 2D UI; 2D UI group using NGUI is a group of buttons with the function of finger gesture interaction like rotate and move.

- 3D UI; 3D UI is also using NGUI for display the thumbnails of all clothes. The reason to use 3D UI is that it can realize various kinds of 3D effects since they are like objects in the scene can move in three dimensions and also with full functionality of UI elements in 2D UI. Another reason is that it is easy to manage different UIs separately in 2D and 3D since it will be complicated if there are thousands of garments which will mess up the UI system.

Following snapshot is the output of the demo application running on PC simulator. As is shown that left side is the model with interactive buttons and right side is the virtual model based on the user size for fitting.



Figure 5.2 Application output UI snapshot

The right part is supposed to produce design following the structure of smartphone UI which is a container with banner title on the top and content lists below. Users should be familiar with the user interface to navigate within all categories of clothes in the system. Every single button with its thumbnail is clickable and it will fit onto the body on the left.

On the left side, when click on rotate button, the model will rotate by itself and when click on move, users can zoom in and out with double click on the model. It is both designed for interaction with mouse and finger touch. When click on fit on/off button, users can hide the body or make it visible depending on whether they want to view the clothes in details or with the body as a whole and the character will come to its original position.

The construction of the models and animation are complemented with Maya (3D construction engine) and Unity (3D game engine). The working principle of the UI system is based on the UIButton event response system and message sending principle. Buttons for changing clothes are actually selecting corresponded materials for the data stack where materials are previously stored for indexing. Buttons for mouse and touch interaction is sending messages to the main camera in the scene to change the location and angle of the camera.

5.2. Discussion

5.2.1. Colliders in Animation

The model in the demo is with animation to simulate the real-life experience in both the body itself and the clothes fitting on the body. In animation making process, the movements of body, in fact is the movements of skeletons which are built within the body in the structure of human body. It contains joints, the connection between joints, kinematics and ergonomics effects that can be added to the skeletons to make human movements like standing, walking, jumping and other general movements. The basic animation in frames is made in 3D constructing engines like Maya and then developers import the model with animation to Unity. Unity can read models in most 3D formats and translate them into the language that Unity can understand. Animations and skeletons will also be translated.

As mentioned in the “Animation in Unity” subchapter, in order to make collisions between human bodies and the clothes, colliders should be placed in human body to make collisions with the clothes which are also colliders. However, the solution of adding a body mesh collider with the exact mesh of the body turns out to be quite clumsy. So it is better to bind colliders to the joints of the skeleton since they are moving with the skeletons and joints. Though there are several collider types to choose from, they are not perfectly matching with the surface of the body that most of them will exceed the skin a little bit, or else collision between body and clothes will show up with some moments that surface of body skin will be out of the clothes which are unrealistic. Even though it solves part of the problems, still the colliders did not cover the whole body surface in some cases.

In my opinion, the solution for this problem is to make every collider into the exact same shape with the mesh colliders. Starting from the constructing process, export some of the key body part meshes, such as shoulders, arms, hands, back, waist, hip, knees and foot, from the whole mesh and store them separated for making the mesh colliders in Unity. Note that the joints have orientations and binding mesh to a mesh collider also has a particular rule with facing orientation of the meshes. Thus, the collision effect will be more accurate and realistic, but the consumption of memory in device will be also

heavier. But with more advanced device with fast CPU and GPU, this won't be a problem.

5.2.2. Material on Meshes

To be accurate on mapping images to the meshes, in real practice, the images of the clothes should be cut into the exact shape with the meshes to fit in every single piece of clothes. Below is an example of a shader implemented through a Material.

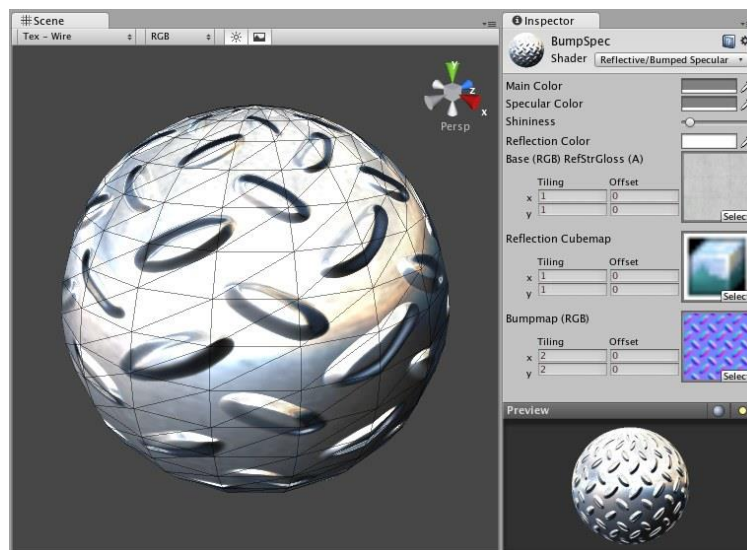


Figure 5.3. Material Setting Example

There are two parameters in texturing: offset and tiling. Offsets are defining the shifts of the texture image around the objects, for instance, offset of 0.5 means to shift by half of the current tile size and tiling controls how many times the texture is duplicated across your surface of the objects.

For instance, the shader BumpSpec has three imaging properties: Base RGB Gloss for basic diffusing mapping, Reflection image for the reflecting effect and Bumpmap. Bumpmap is a method to render shadows and highlights of the materials with different color depth without increasing the number of polygons, thus to increase the visual detail and realism to the original textures and models. The principle is to provide an exact same size color depth image to calculate the 3D vision effect. It can be used to create vivid vision effect for cloth with special material like knitted and flax clothes which have a strong need in bumping vision.

5.3. Designing of Overall System

The development of the application demo is one part of the system of clothes fitting application on mobile devices. In Figure 5.4 one logical solution for the whole system is shown.

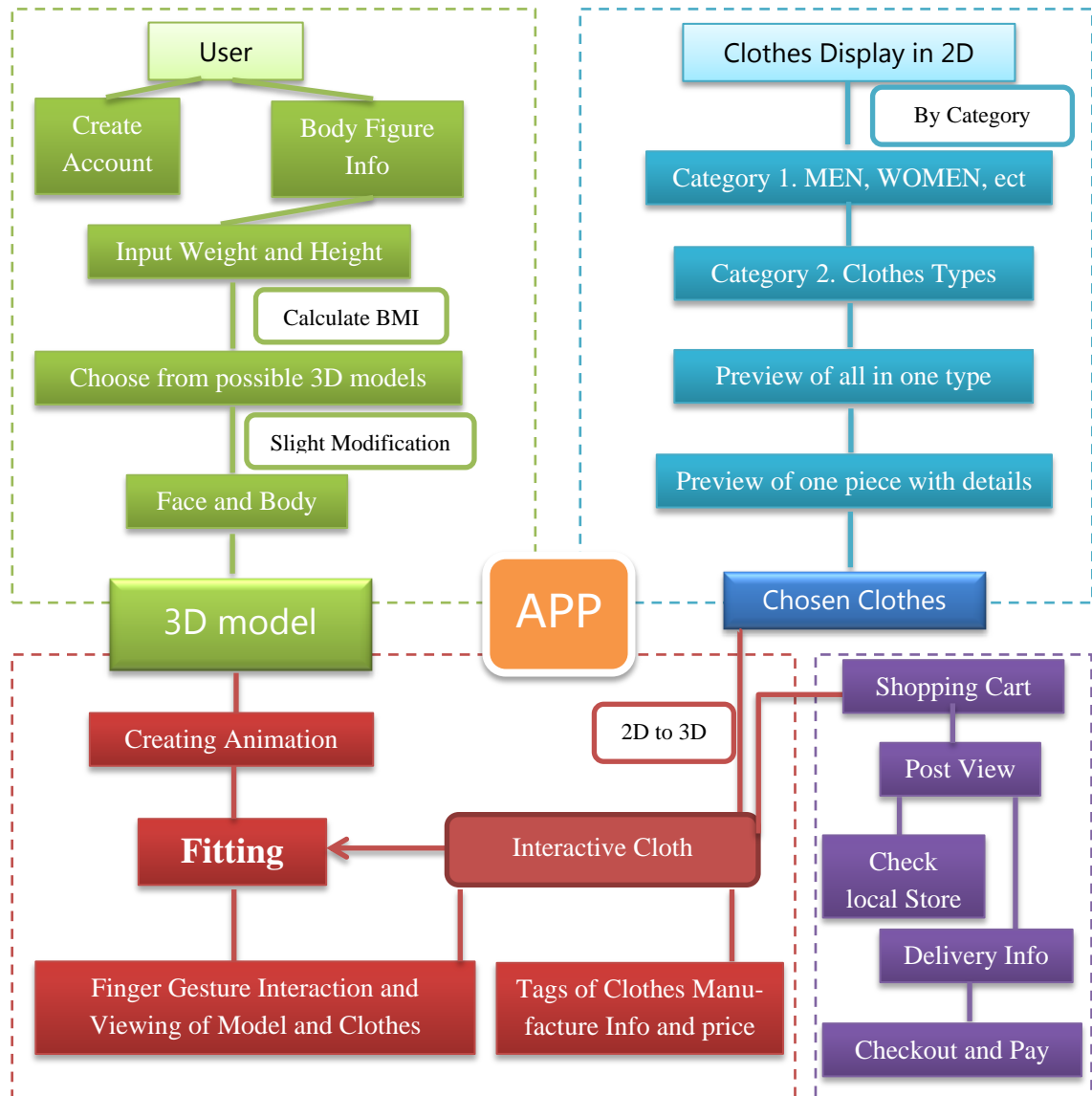


Figure 5.4. The Whole Application System Structure

This system is a complete design for the clothes fitting application from creating account to final mobile online shopping. The whole working load for the implementation is huge and concerns some different aspects of software developing skills such as Android development, database enquiry, 3D model engineering and cloud services. In the Figure 5.4, the part in red is the work done in the demo and how it is cooperating with other modules in the application logically. Here is short explanation of the whole system in the assumed design:

- Module 1: User

The aim of the application is to create a virtual human body that is similar in shape and size with the real users' body size so that they can do online virtual fitting on their mobile devices. Other than the general register process to ask users to input their personal information, there is another important procedure to input their body figure numbers. For most people, they may not know their own sizes of their waists, chests, hips and lengths of arms and legs as these numbers are not usually measured unless you are fitting in a tailor's shop. But for most of people, they should know their heights and weights. There is a measure for human body shape based on an individual's mass and height called BMI (body mass index = $\text{mass}(\text{kg})/(\text{height}(\text{m}))^2$). With this BMI number, a basic body shape can be created in the engine. There may some difference in the shapes even with the similar BMI, the system should provide users with diverse body shapes to choose from. In any cases, user will succeed in choosing one model that is closest to his or her own body. Of course, the option for direct input detail body size numbers is available. Then, users can modify the facial appearance just for personalization which is realized by merging meshes in 3D engine. Finally, the personal and unique model is finished and system will add animation and colliders to the body for further fitting.

- Module 2: Clothes Display in 2D

The process of choosing one specific clothes is very similar with other online shopping sites: first list is sorted by men, women, kids; secondary list is the different types of clothes; third list is the lists of images of the chosen type. When choosing one item, price and manufacture details are also listed besides. In order to release the system from heavy 3D calculating load, the preview of the clothes can be in 2D. 2D image is also in very high quality and users can view in a very detailed way with the exact same picture viewing method on all mobile devices. Noted that in the system, the memory fragments after calling images from database should be deleted or else the memory will run out of space.

- Module 3: Fitting

As described in "Results", users are able to choose from the clothes gallery and try them on the virtual body they build based on the body figures of their own. Users are also able to make finger gesture interaction with the model to view the details. Of course, the fitted clothes are dynamic with the help of "interactive cloth" function in Unity. For a 3D web shop, the detailed information of the clothes will also be in the interface.

- Module 4: Check Out and Payment

Concerning the business concept, the application will be used in mobile online retail shops. So there should be ways to motivate users to buy the items they are interested is the final goal. Here there are provided two ways to encouraging customers to purchase. Suppose you have one coat that you are interested and you finished with the virtual fit-

ting. The system has chosen the suitable size for you. You can either choose the nearest local retail store to have a real fitting or you can just place your order online and pay. Why these two options? It is for two kinds of consumers and also due to the nature of the application itself. As for people who do not trust the system very much, they can still benefit from the very first goal of the application that they can get updated with the new arrivals and try them on. The only worry of them is that they are afraid if they order online, the clothes they receive won't fit them with the perfect size. However, they can still go to the real local retail store to try it on. The application help customers save the time selecting ones they love, i.e. pre-filtering. As for people who are willing to trust the system with the fitting effect, they can place the order right away and receive the item at home. It may take some time for customers to get used to the new way of shopping. It also needs the system to be more accurate and precise about body virtual fitting. Anyway, it may become a new shopping way with mobile devices.

6. CONCLUSIONS

The development of mobile graphic technology happens at a very fast speed and it only took one decade that the graphic processing power became mature. The breakthrough of inventing smartphone with advanced 3D graphics brings lots of possibilities to users due to many applications relying on the basis of high graphic processing quality.

Based on that background, the thesis focused on developing 3D graphics application for web shop with the use of dynamic virtual clothes fitting on avatars with animations. The Maya modelling and Unity engine chosen for the development allows for both large developing teams and individual developers to build applications with easy-to-use user interface especially if developers have former experience with 3D game engines. The practical illustration of results of the thesis is a demo showing the dynamic model with animations, user finger gesture interaction on mobile device and changing model mesh skin rendering.

Investigating into users' finger gestures interaction on mobile device has revealed that users like really simple gestures like tap, double tap and drag other than complex gestures such as two finger spreading and pinch. Providing buttons with specific functions releases the heavy operating load for users since users only have to click on the button first and use only simple gestures afterwards. Moreover, in the cases here, isolated buttons avoid conflicts between two visual effects with the same gesture.

Regarding to put the result into practical market use, the idea comes from the market of mobile online shopping which just takes the advantage of mobile 3D graphic to realize virtual fitting and entertain users in the shopping process. This provides value in the market from the perspectives of both customers and clothing companies. The demo presented is not the final product, and it rather illustrates the basic idea and development for both the model construction and programmable interaction on mobile device. For commercial application, the construction of models or persons and clothes should be more precise and detailed. In the database, mapping of materials should be also set carefully. Simulated animation of persons can be richer with more actions and adding more backgrounds and lights can simulate different dressing environment.

However, the limitation of the application is that it is heavily relying on the quality of the mobile device especially the graphic process power. Additionally, the lack of mobile purchasing habit of customers would slow down spreading of the application. The good

news is that Unity allows diverse platforms not only for mobiles but also on the Web and PC platforms. But the high speed of smartphone development and the growing number of users in mobile shopping may accelerate spreading of the application. The topic is worth further research both in theory and practice.

REFERENCES

- [1] Capin, T., Pulli, K., & Mödler, T. A., The State of the Art in Mobile Graphics Research, IEEE Computer Graphics and Applications, Vol. 28, Issue 4, 2008, pp. 74–84.
- [2] Wright, C., A Brief History of Mobile Games, [WWW], 23.12.2008, [accessed on 10.11.2013] Available at:
<http://www.pocketgamer.biz/r/PG.Biz/A+Brief+History+of+Mobile+Games/feature.asp?c=10619>
- [3] Lawrence, J., 3D Polygon Rendering Pipeline [WWW], University of Virginia, 22.10.2012, [accessed on 15.4.2014] Available at:
<http://www.cs.virginia.edu/~gfx/Courses/2012/IntroGraphics/lectures/13-Pipeline.pdf>
- [4] Woligroski, D., Graphics Beginner's Guide, Part 2: Graphics Technology. Tom's Hardware, (July 31, 2006).
- [5] Introduction of GFXBench, [WWW][accessed on 10.10.2013] Available at:
<https://gfxbench.com/gfxbench.jsp>
- [6] Five Main Mobile GPU Providers [WWW] [accessed on 20.4.2014] Available at:
<http://www.expreview.com/24705.html>
- [7] Tegra K1 – The world's most advanced mobile processor [WWW][accessed on 18.4.2014] Available at: <http://www.nvidia.com/object/tegra-k1-processor.html>
- [8] Maher, K., Peddie, J., OpenGL 3.2 Arrives with Siggraph and Khronos Steams Ahead [WWW] [accessed on 8.10.2013] Available at:
http://www.catiacommunity.com/print_article.php?cpfeatureid=45601
- [9] Neider, J., Tom, D., Mason, W., OpenGL Programming Guide: The Official Guide to Learning OpenGL, Release 1, Addison-Wesley, Reading, Massachusetts, 1993 (ISBN 0-201-63274-8).
- [10] Benj L., Overview of OpenGL [WWW] [accessed on 13.4.2014] Available at:
http://web.cs.wpi.edu/~matt/courses/cs563/talks/OpenGL_Presentation/OpenGL_Presentation.html

- [11]CMSC 498M: Chapter 3 Overview of OpenGL [WWW] [accessed on 4.4.2014]
Available at: <http://www.cs.umd.edu/class/spring2011/cmsc498m/Lects/chapt03-opengl.pdf>
- [12]OpenGL and Graphics Hardware, Overview of OpenGL Pipeline Architecture Alternatives [WWW] [accessed on 4.4.2014] Available at:
<http://www.cs.cmu.edu/~djames/15-462/Fall03/notes/02-opengl.pdf>
- [13]Pulli, K., Aarnio, T., Miettinen, V., Roimela, K., Vaarala, J., Mobile 3D Graphics with OpenGL ES and M3G, USA 2008, Morgan Kaufmann Publisher, 446 p
- [14]Moller, R., State-of-the-Art 3D Graphics for Embedded Systems, Proceedings of the 6th International Caribbean Conference on Devices, Circuits and Systems, Mexico, Apr. 26-28, 2006, pp. 339-343
- [15]Mali GPU, Application Optimization Guide, Copyright © 2011 ARM, [WWW] [accessed on 4.4.2014] Available at:
http://infocenter.arm.com/help/topic/com.arm.doc.dui0555a/DUI0555A_mali_optimization_guide.pdf
- [16]Ginsburg, D., OpenGL ES 2.0: Shaders Go Mobile, [WWW] [accessed on 8.4.2014] Available at: <http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/GDCMobile2006-Ginsburg-OpenGL-ES2.0.pdf>
- [17]Harmon, P., WebGL: The HTML5/JavaScript 3D Computer Graphics API [WWW] [accessed on 7.4.2014] Available at: http://www.phillihp.com/wp-content/uploads/WebGL_Demos_and_Intro_to_Development.pdf
- [18]Ben, B., Dave, D., Chris, H., Daniel, K. O., Visualization, Analysis, and Workflows Over the Web for Geosciences Using UVCDAT, VTK, and VisTrails [WWW] [accessed on 14.4.2014] Available at: <http://www.kitware.com/source/home/post/125>
- [19]Trevett, N., WebGL, WebCL and Beyond, Proceedings of HTML5 Developers & Designers Conference December 7-8, 2011, Santa Clara, CA. pp 45.
- [20]Neil, T., OpenCL WebGL and WebCL [WWW] [accessed on 15.4.2014] Available at: https://www.khronos.org/assets/uploads/developers/library/2012-pan-pacific-road-show/OpenCL-WebGL-WebCL-Taiwan_Feb-2012.pdf
- [21]Brutch, T., Eliuk, S., Torok, M., Sevenier, M. B., Khronos WebCL: Enabling OpenCL Acceleration of Web Applications, Proceedings of SIGGRAPH Asia Dec12, 2012, pp 28. [WWW] [accessed on 6.4.2014] Available at:

<https://www.khronos.org/assets/uploads/developers/library/2013-march-meetup-WebCL/WebCL-March-Meetup-2013.pdf>

- [22] Jeon, W., Brutch, T., Gibbs, S., WebCL for Hardware-Accelerated Web Applications, TIZEN Developer Conference May 7-9, 2012, [WWW] [accessed on 8.4.2014] Available at:
https://download.tizen.org/misc/media/conference2012/tuesday/ballroom-b/2012-05-08-1415-1455-webcl_for_hardware-accelerated_web_applications.pdf
- [23] Tasneem, B., Samsung's WebCL Prototype Implementation: Overview of APIs [WWW] [accessed on 15.4.2014] Available at:
<https://webcl.googlecode.com/files/Samsung%20WebCL%20Prototype%20APIs.pdf>
- [24] Wu, Y.Y., Mok, P.Y., Kwok, Y.L., Fan, J.T., Xin, J. H., An investigation on the validity of 3D clothing simulation for garment fit evaluation, Proceedings of the IM-ProVe 2011 International conference on Innovative Methods in Product Design, June 15 – 17, 2011, Venice, Italy, pp. 463-468
- [25] Yang, K., Jie, J., Haihui, S., Study on the Virtual Natural Landscape Walkthrough by Using Unity 3D, IEEE International Symposium on Virtual Reality Innovation 2011, 19-20 March, Singapore, pp. 235-238
- [26] Ruiz, J., Li, Y., Lank, E., User-Defined Motion Gestures for Mobile Interaction, Proceedings of CHI 2011 • Session: Mid-air Pointing & Gestures May 7–12, 2011 • Vancouver, BC, Canada, pp. 197-206
- [27] Unity User Reference Manual [WWW] [accessed on 11.12.2013] Available at:
<https://docs.unity3d.com/Documentation/Components/index.html>
- [28] Unity Script Reference [WWW] [accessed on 11.12.2013] Available at:
<http://docs.unity3d.com/Documentation/ScriptReference/index.html>
- [29] Theodorou, A., Optimizing Unity Games for Mobile Platforms, Unite 2013, 28th-30th August [WWW], [accessed on 20.9.2013], 39 p. Available at:
http://malideveloper.arm.com/downloads/Unite_2013-Optimizing_Unity_Games_for_Mobile_Platforms.pdf