Improving The Efficiency Of 3D Graphics in ARM Architecture

By Rahul Padmanabhan
u15134683
University Of South Florida

Abstract—3D graphics have played a major role in the computer industry since the 80s. Ever since it was introduced, it found wide acceptance from the end user for being more aesthetically pleasing than 2D graphics. As computers evolved generation after generation, the complexity of this field grew by leaps and bounds and the advent of powerful mobile devices spurred a new race in 3D graphics. ARM (Advanced RISC Machine) architecture is the practically the standard for 32-bit instruction set architecture in mobile devices today. A vast majority of smart phones today use this type of technology. Mobile phones and devices have far more capabilities today than computers of the past did, however; resource and power constraints still apply. This paper highlights the optimization techniques that developers can use to create better 3D Rendering Solutions for ARM-based devices and also describes the limitations and restrictions faced.

Index Terms—ARM Architecture, 3D Optimization, ARM Limitations.

1 Introduction

It is hard to imagine life without mobile technology today. A few years ago, few would have fathomed the leaps and bounds this industry has taken to develop a powerful computer that fits into the palm of ones hand. These devices are rapidly evolving and as their performance grows, the ability to perform more complex operations on them (such as 3D games) grows as well. An example of a powerful device is the Samsung Galaxy S III which has a 1.4 GHz quad core Cortex-A9 which is a 32 bit multicore processor that has 4 cores, each of which uses the ARM architecture. This is coupled with a HD Super AMOLED display. Less than ten years ago, even the mid-range computer desktops did not have this kind of processing capability. This kind of power provides the capability of software rendering for 3D graphics which gives rise to the vast depth that is the 3D gaming ocean and provides a challenging, yet exciting arena for 3D game developers to explore. 3D gaming applications have long since been considered greatly intensive and required high-end systems to run on. They were so computationally intensive that they formed one end of a benchmark spectrum on a computers performance. High end graphics processors have a high pixel per second rate, high polygon count and triangles per second during 3D rendering. 3D gaming has now been made available on mobile devices but not without a few challenges such as power management, resource limitations etc. Several energy saving methods have been implemented but optimal power usage (coupled with resource constraints) still is a concern and a limiting factor. If we compare mobile devices to popular gaming consoles and PC platforms, we find a vast difference in several factors such as processing power, smaller caches, lower memory etc. Even the bandwidth is has various limitations due to a narrow bus and memory resource sharing.

- .
  E-mail: rahul3@mail.usf.edu
- *Rahul Padmanabhan - University Of South Florida.*

.

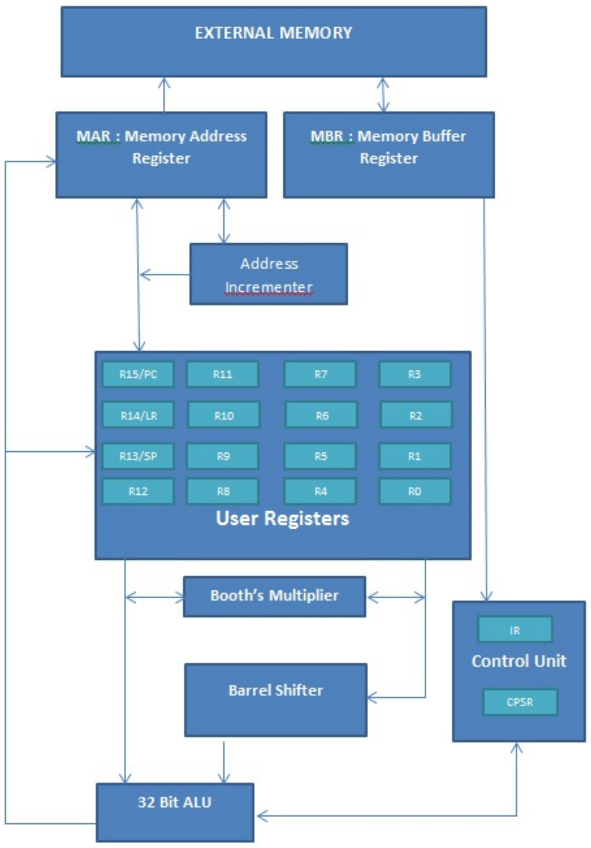


Fig. 1. Block Diagram Of ARM Architecture [1]

2 ARM

ARM is an abbreviation for Advanced RISC Machine. ARM architecture refers to a RISC 16/32-bit architecture, designed for compact yet powerful implementation of instructions for devices with low power and is rapidly becoming the industry standard in the world of mobile computing. [1] It is globally licensable and it is the most widely used 32 bit instruction architecture. Apple Computers and Acorn collaborated in this project to achieve a new breakthrough in microprocessor standards. A vast majority of mobile phones along with other hand-held devices use ARM-based processors. In ARM architecture, although 32-bit architecture is mainly used today, AMD mentioned that they would produce 64-bit chips in 2014.*[BBC Technology News]*

Some features of ARM are:

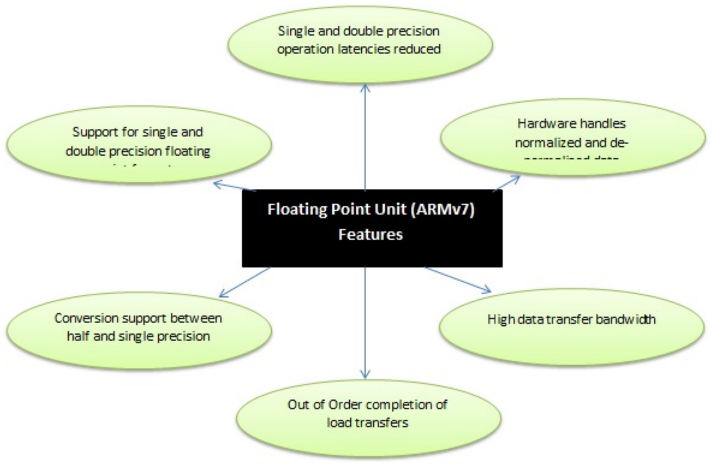

Single and double precision operation latencies reduced
Support for single and double precision floating
Hardware handles normalized and de-
Floating Point Unit (ARMv7) Features
Conversion support between half and single precision
High data transfer bandwidth
Out of Order completion of load transfers

Fig. 2. Features of FPU in ARMv7 architecture

Load/Store based architecture: Similar to RISC architecture, ARM architecture is that of a load/store type which in simple terms implies that the instructions that deal with processing data use the registers and they are segregated from the instructions that are of a memory access type. This is one of the factors that aids in the pipelining of the ARM processor. [1]

Single Cycle Instruction: Another feature of ARM which is also practically a RISC standard is that it requires an instruction execution stage to finish in a single cycle. (Data transfer instructions are one of the rare exceptions in this regard). [1] Mutiple Register Transfer Instruction Support: ARM lets upto 16 registers be stored or loaded simultaneously. Although the single cycle per instruction rule is not being held, the time to complete important operations like large data transfers is significantly decreased

3 3D Optimization in ARM

This paper describes the 3d optimizations made for ARM architecture as well as the limitations surrounding it. Some limitations to 3d optimizations are that many ARM based devices do not have dedicated floating point hardware (FPU) due to energy issues as well as from a cost standpoint. There are energy saving methods such as Dynamic Voltage and Frequency Scaling (DVFS) alongside system level power and performance optimization methods. However, to implement this in mobile devices it requires precise and speedy workload prediction.

3.0.1 *Newer ARM Processors*

However, it must be noted that the newer ARM processors (such as Cortex-A9) now have the FPU built in. Using hardware floating-point combined with multimedia processing capability, performance of imaging applications such as scaling, 2D and 3D transforms, font generation, and digital filters can be increased. Floating Point Units require more internal circuitry and there is an added cost to issues with SOC (System On a Chip) integration. The FPU contains support for arithmetic operations such as addition, subtraction multiplication, division, multiply and accumulate and also square root operations. We will talk about square root operations later in this paper because they are complex operations to perform and the addition of a hardware component to facilitate this speeds up the execution of a program. 3D graphics uses various arithmetic operations frequently and even a minor enhancement in one operation could prove to optimize performance to a significant level. [2] The FPU aids in conversion from floating-point data format to ARM integer word format and vice versa. The FPU also carries multiple operations in order for the conversion to be done in a round-towards-zero mode in order to support higher level languages. The FPU in the Cortex-A9 processor also contributes to the energy savings by having lower power consumption and a reduced die size. It supports out of order completion with respect to load transfers as well. It improves the overall performance by reducing execution time, lowers the power consumed and performs well for regular application usage.

3.1 Limitations Of ARM

Like everything else, ARM has its limitations and restrictions. Some of them are as follows.

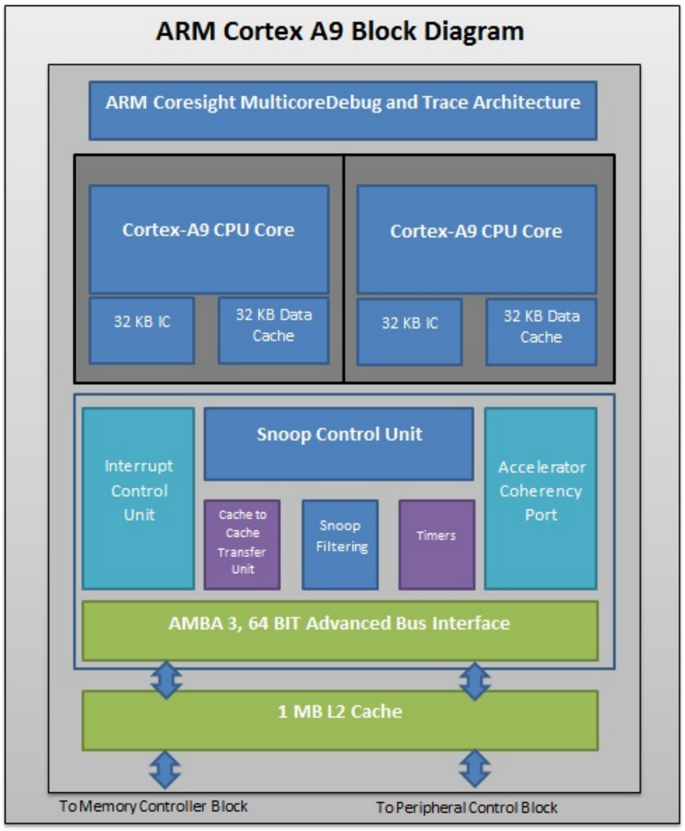

ARM Cortex A9 Block Diagram
ARM Coresight MulticoreDebug and Trace Architecture
Cortex-A9 CPU Core
32 KB IC
32 KB Data Cache
Cortex-A9 CPU Core
32 KB IC
32 KB Data Cache
Snoop Control Unit
Interrupt Control Unit
Cache to Cache Transfer Unit
Snoop Filtering
Timers
Accelerator Coherency Port
AMBA 3, 64 BIT Advanced Bus Interface
1 MB L2 Cache
To Memory Controller Block
To Peripheral Control Block

Fig. 3. ARM Cortex A9 Block Diagram

3.1.1 *Integer Divide Not Supported*

ARM hardware does not have support for integer division; it must therefore be performed from the software level. [2] Compilers call a library function when division is used in the code. For example, if the following code is used and compiled to Assembly level (for iPhones):

```
Code:

int HundredBy(int divisor
{
return 100/divisor;
}
```

The compiler will utilize a function called ___divsi3 which is a system function that uses the software to do the division. Division as an operation is a costly and is avoided for this reason
Let us consider an example of vector

normalization. Vector normalization is not essential but it is done for the purpose of equation simplification and can reduce the size of APIs. The normalized vector of N is a vector which the direction remains the same but of length 1. If |N| is the norm of N then the normalized vector N is

$$N = \frac{N}{|N|} \tag{1}$$

Vector normalization code for 16.16 format is as follows

$$d.a = (int)((\_\_int64)(d.a << 16)/(\_\_int64)norm);$$

$$d.b = (int)((\_\_int64)(d.b << 16)/(\_\_int64)norm);$$

$$d.c = (int)((\_\_int64)(d.c << 16)/(\_\_int64)norm);$$

Here the division operation is used. As we know that division is an expensive operation [2], division can be substituted by multiplying the dividend into the inverse of the divisor to get the quotient. We need to note the fact that if the division operations are replaced by multiplication operations then we would encounter a difference in the rounding which is normally termed as a rounding error. We define a value that computes the inverse of the divisor and all it "divisor_inverse".

$$divisor\_inverse = (1 << 16)/norm$$

$$d.a = (int)((\_\_int64)d.a * (\_\_int64)divisor_inverse);$$

$$d.b = (int)((\_\_int64)d.b/(\_\_int64)divisor_inverse);$$

$$d.c = (int)((\_\_int64)d.c/(\_\_int64)divisor_inverse);$$

3.1.2 Low Bus Bandwidth Availability

Due to the bus bandwidth on mobile devices not being very high, data sent to the code producing 3D graphics by the CPU could have delays due to the creation of bottlenecks that arise. The FPS (Frames Per Second) will be impacted negatively due to this causing a decline in frame rate which may make the game seem jerky or slow. Due to the low bus bandwidth, data path overloading is one aspect that developers need to be cognizant of. Efforts are being made to overcome this limitation. [3]

3.1.3 Lack of FPU

Many 3D rendering fundamental algorithms require real number representation of coordinate space systems; as a result, this feature is necessary in most 3D engines. The implementation of transform, lighting, culling and clipping require real number representation as they require a high level of precision and a wide dynamic range. Floating-point representation of real numbers has the ability to provide these features which is why it is generally preferred over integer representation. [4]

3.1.4 Older ARM Devices

Many ARM devices previously did not contain a dedicated FPU i.e. floating point hardware. Floating point operations were supported separately either using the software level through the floating-point library fplib that contained functions to be called that implemented floating point operations without reliance on separate floating point hardware. During the compilation of floating-point application code, the compilers function calls are sent to this library instead of floating point instructions. This enables the application code to take advantages of improvements made in the ARM code as well as it remains unaffected if an FPU hardware unit is added to the system. It is also recommended by ARM Inc. to have a floating-point library present in embedded systems.

Another way to do perform floating-point operations was by using the hardware VFP (Vector Floating Point) co-processor in conjunction with the ARM core to produce the floating point operations required. [1] [3] VFP is termed as a type of co-processor build that is involved in the implementation of IEEE floating point operations and it also supports single and double precision. Code compiled using the ARM floating-point library cannot use the following:

- extended precision
- rounding modes other than round to nearest
- inexact exceptions
- underflow exceptions
- rounding modes other than round to nearest

The implementation of floating point operations in VRP was done using a combination of hardware that executed common cases as well as software that dealt with the exceptions. The latest version of VFP is VFP9-S.

VFP9-S Benefits

The ability of the ARM VFP9-S in terms of vector processing offers enhanced performance in floating point operations used in applications (example. Imaging) such as 3D transforms graphics and scaling.

VFP9-S Applications Related To 3D Optimization

- 3D graphics
- Digital Consumer products (eg. Consoles)
- Imaging (eg. Digital Cameras)

3.2 Fixed-Point System

Fixed point is a methodology by which a number in a floating-point format is converted to represent a number in the integer format with a illusionary radix point separating the integer and the fraction part. The bits to the left of the radix point consist of the integer section of the number or value that is being used. The bits behave as weights for positive powers of 2. For example, the number 3.14 can be represented as 3140 in a fixed point system with a scaling factor of 1/1000. For better efficiency, the scaling factor is usually as power of 2. [2]

If we look at a simple example of an 8.8 fixed point format, the 8.8 implies that there are 8 bits before and after the radix. If the numbers are signed then the dynamic range of this covers the open interval as follows:

$$Openinterval = [-2^8, 2^8]$$

$$value = -bit_{15}2^7 + bit_{14}2^6 + ... + bit_9 2^1 + bit_8 2^0$$

$$+bit_7 2^{-1} + ... + bit_2 2^{-6} + bit_1 2^{-7} + bit_0 2^{-8}$$

where bit is a binary bit and its values are 0 or 1.

The original Sony Playsations 3D graphic engine used fixed point arithmetic to gain throughput without the hardware floating point unit. However, it must be noted that floating point calculations may have slight differences due to CPU, compiler and other such dependencies.

For the quickest run time while executing code to produce 3D graphics, we need to keep a few points in mind. Integer shift operations should be used. Division, square roots and trigonometric operations are expensive and should be used minimally or avoided altogether. Operations involving lookup tables consume memory access time and should be avoided. [2] Integer Add/Subtract/Multiply/Boolean Operations can be used. This will tend to be more complicated for the developer but we can draw a comparison to general computing where the most complicated way is generally the fastest. An example of this is Assembly Level Computing vastly outperforming High Level Programming. Trade-offs will need to be made in order to balance code complexity and functionality. [5]

3.2.1 Multiplication and Division

Multiplication and division operations in fixed-point system are more complex than the more simple arithmetic operations like addition and subtraction.

For example if we were to multiply two fixed point numbers a and b. The operation is listed as

$$(a * 2^x) * (b * 2^x) = (a * b) * 2^2 x$$

This example will overflow when |a * b| is greater than or equal to 1.

The scaling factor requires double the number of bits for the fraction. The result is scaled down in the case of multiplication

If x = 16 i.e. 32 bit, we capture the result into a 64-bit number:

$$(a * 2^1 6) * (b * 2^1 6) = (a * b) * 2^3 2$$

[6]

3.3 Dynamic Range and Precision

Dynamic range is defined as the range between the highest and lowest values of a dynamic or changeable quantity. When we talk about dynamic range and precision in a fixed point representation, we cannot ignore a problem that can be faced while arithmetic operations are run.

During the multiplication of two 16.16 fixed-point formatted unsigned numbers there is a 64 bit intermediate number generated which is listed in a 32.32 bit format. As we know in fixed-point formatting, this implies that there are 32 bits in the fractional section of the number and 32 bits in the decimal section of the number and they are separated by a radix. Developers need to ensure that there is proper accuracy while defining the data types because ARM supports this 64-bit intermediate number in different ways. Some instructions that it supports are as follows

An example of the multiplication earlier discussed is as follows

```
int Product(int x, int y)
{
return (int) (((__int64) x
* (__int64) y) >> 16);
}
```

While being able to prevent possible data losses, the 64-bit intermediate result cannot raise the threshold of the dynamic range and number precision.
There are two kinds of errors that can occur if the dynamic range and number precision is not monitored carefully [7]

1 Overflow error of the dynamic range
2 Underflow error of precision

3.3.1 Overflow Errors

An overflow error is one where the value is too large for the range of numbers represented. The risk of an overflow error of the dynamic range can be reduced by:

1 Restricting inputs
2 Use of localized space or normalized data
3 While rasterizing allot a smaller dynamic range

3.3.2 Underflow Errors

An underflow error is one where the value is too small to be of the least value in a number. The risk of an underflow of precision error can be reduced by:

1 Small vectors normalized for operations
2 Using lookup tables for approximation routines in mathematics
3 Depth buffer to use multiple precision formats

Care must be taken to strike a middle ground between the dynamic range and number precision keeping the level of complexity of the 3D graphics in mind.

3.4 Trigonometric Functions and Arithmetic Approximation Routines

3D graphics uses various operations such as division, square roots and trigonometric functions. These are usually time consuming functions in C due to their complexity. There can be an approximation done because as we have discussed in this paper previously, the display of a mobile device has limitations and is of a lower resolution, hence an approximation to a reasonable degree would not cause a major change in displaying 3D graphics on the mobile screen. [2]

Division is already known as an expensive operation and this is because it involves several iterations to get the result. Although there are different ways of doing it, a lot of computers internally still use the reciprocal of the divisor as a multiplicand. There are ways to do it using lookup tables and the Taylor series expansion however, compared to addition and subtraction, it is an expensive operation. While designing the processing unit, certain compromises may need to be made trading efficiency, functionality, cost and energy consumption with each other.

Square roots are one of the more complex arithmetic functions to be computed, however, they are commonly used in 3D graphics. Some processors have hardware to implement this because the method to implement this without hardware is one of the most computationally time intensive practices. The software approach to this is to first generate an approximate value with either a lookup table or an a set of instructions that gets the closest value, and then with a set of iterations, compute the value. Using a lookup table increases the execution speed because every time a value is looked up, the memory is accessed which takes up more time due to the slower speed of access. Multiple iterations also need multiple CPU cycles which are time intensive as well. 3D Optimization needs efficient code to increase its chances of being present in the instruction cache to reduce the execution time.

Trigonometric functions are computed like they are taught to most students in their introduction to trigonometry in school. Using lookup tables. As mentioned previously, lookup tables take up more time due to the time involved to access the memory. A snipped of code is given below where the functions sec(angle_val) and cot (angle_val) are internally computed by a reference made to a lookup table.

```
do
      {
         putpixel((int)(a+b+0.5),
         (int)(k-c+0.5),color);

         angle+=0.001;

         b=(rb*sec(angle_val));
         c=(rc*cot(angle_val));
      }

while(angle<=range);
```

3.5 Chip Caches small in comparison to PC architecture cache sizes

The latest ARM processors are built with either on-chip memory or on-chip L1 cache. [2] The cache is divided into data caches and instruction caches both of which are addressed virtually and are set-associative with a high degree of associativity. The instruction cache is restricted to being read-only while the data-cache is both read and write with a copy-back write strategy. A lot of ARM processors also provide the capability to lock down data in the cache such as critical sections of code or data stored. [3]

3.6 3D Rasterization Hardware Not Usually Present

Rasterization is the process by which a vector graphics formatted image is taken and converted to an image consisting of pixels and dots (known as a raster image) for output on a visual display or other similar types of display. The reason cell phone manufacturers stay away from implementing this is the prohibitive price as well as the power consumption of this hardware is high. Instead of using rasterization hardware, other software oriented methods are used such as libevas [8], which is a multi-platform library containing various routines for rasterization and image processing such as scaling, blending, drawing polygons, clipping images etc. [3]

3.7 Register Allocation

The ARM7 processor has 37 registers. The number of registers varies depending on the model of the ARM processor but it is always a fixed quantity. In order to optimize 3D graphics we need to ensure that memory swapping is as minimal as possible. [2] Take the case where there are more variables than registers available in the processor. The system is then forced to store some values of the registers in the memory and then swap them back again when needed. This is termed as a register-memory swap and takes a longer amount of time. In order to reduce the time of execution, there needs to be efficient register allocation where the swaps are reduced to a minimum thereby reducing execution time and improving 3D optimization. [1]

4 Branching and Predication

4.1 Branching

Branch: A branch is a segment of code that is executed based on a set of conditions and it's execution depends on the control flow at the branching point. [2] Some examples of branching are GOTO statements, IF-THEN-ELSE etc. The science of guessing the direction in which the branch is going to head is termed as branch prediction. The CPU has different stages as illustrated in the diagram.

As different instructions can be processed simultaneously, when an instruction is being read, the next instruction is being decoded and so on as per the stage cycle. Branch prediction is determining what the next instruction will be based on the condition decoded in the previous instruction. There may be multiple possibilities and this can pose problems for the

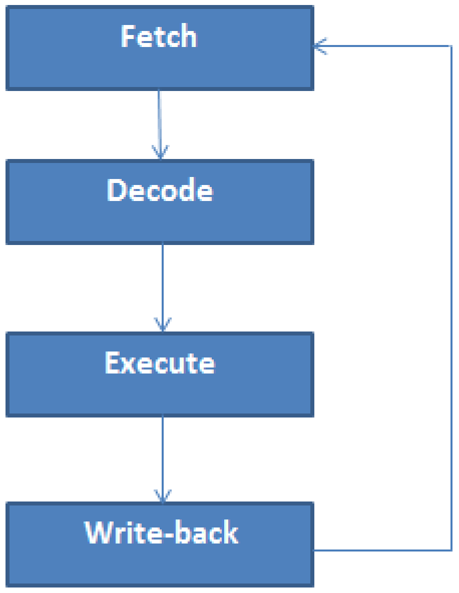


Fig. 4. CPU stages

processor pipeline. While branch prediction is extremely helpful in increasing overall system efficiency, we need to come to terms with the fact that there is a penalty for every miss where the penalty is equal to the number of steps the program has moved ahead based on the information given by the predictor for that instruction or section of code. For 3D optimization, the developer needs to be cognizant of control hazards that may occur with inefficient branching. Branching may limit the performance of the CPU as the incorrect prediction of a branch leads to a penalty that causes a decrease in performance. The unpredictability that sometimes surrounds branching may cause the compiler to make a more conservative decision that could affect system performance negatively. Control dependencies as well as pipeline stalls are also created as a result of branching. [2]

Let us take a look at the performance analysis of a loop shown in the example below

```
for (int n = 0; i < max; m++)
if (<condition>) total++;
```

Condition	Sequence	Time(ms)
(n&080000000) == 0	T repeated	322
(n & 0xffffffff) == 0	F repeated	276
(n & 1) == 0	TF alternating	760
(n & 3) == 0	TFFFTFFF	513
(n & 2) == 0	TTFFTTFF	1675
(n & 4) == 0	TTTTFFFFTTTTFFFF	1275
(n & 8) == 0	8T 8F 8T 8F	752
(n & 16) == 0	16T 16F 16T 16F	490

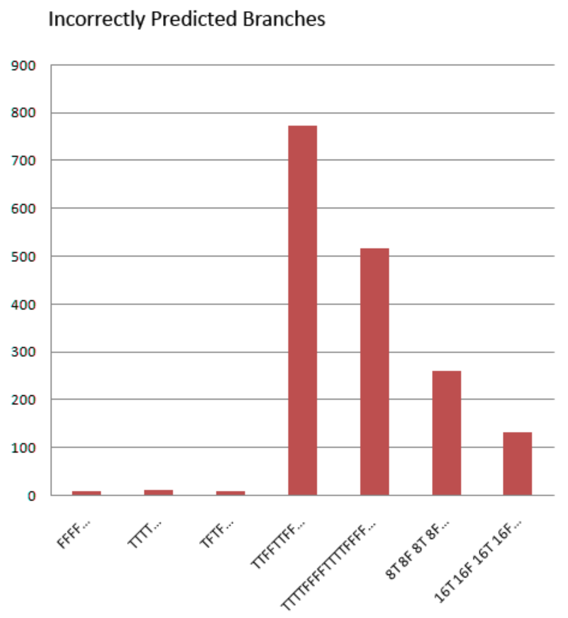


Fig. 5. Branch Prediction Errors

4.2 Predication

Predicated execution is a feature in computer architecture that exploits ILP (Instruction Level Parallelism) with control flow present. [9]It encompasses the conversion of program control flow into multiple conditional (or predicated) statements and this process is termed as if-conversion. The use of ILP is increasing by the day due to the extensive demand for hardware to meet its performance potential. Branching is one of the limiting factors for this because of the unpredictability involved. Branch misprediction can cause a decline in performance due to the penalties attracted for each failure as we have seen in this paper. Scheduling and compiler optimization are affected by branches. The model of predication has the advantage where instruction execution conditions in it do not have a complete dependency on branching. In ARM Architecture all the instructions are predicated, this is termed as Full Predication.

In predicated architecture, scheduling and optimization is done by a structure called the hyperblock. [9]In a hyperblock, control enters only from the top but it may have multiple points of exit. There is a single block designated as an entry block and the instructions in a hyperblock are only predicated instructions. Programs that are not branch heavy benefit from hyperblocks because they provide a less bounded framework for transformations during compilation.
There are issues that come up when we convert the control flow into multiple if statements. The regular approach is to do this at time of scheduling. This is usually beneficial because the constraints of the processing unit are known by the scheduler and an outline of the code details is also known. Therefore this makes the scheduler a favorable choice during decision time. Also, at the time of scheduling, no further changes can be made because enhancements to the code are already made by then. [10] [9]
However, a situation that comes up while converting to if-statements at the time of scheduling is that the complier is then restricted from convert the control flow into a predicate representation and also hinders it from optimizing different aspects of the code. This is not favorable because of the high level of complexity that is now brought into the picture. If there is a delay in converting to predicated statements then there are optimizations that may be left out and since a lot of optimizations and transforms are highly interdependent on each other and are usually applied repeatedly, there may be degradation in performance.

Effect Of Conditional statements: Although they decrease the number of branches in the code, they cause an increase in the code size. As a result compilers need to determine a balance between the two. We can see this in an example below. [9]

```
if ( x > y ) z = x;
else z = y;
```

Using branching, we can write the code as follows

```
CMP R0, R1
BLE _LowerThan
MOV R2, R0
B _NextCommand
_LowerThan:
MOV R2, R1
_NextCommand:
```

Now, if we were to re-write this code using conditional statements

```
CMP R0, R1
MOVGT R2, R0
MOVLE R2, R1
```

Compilers use various methods to strike a balance between code size and optimization requirements. For example, suppose an if-else block has the same possibility of execution and if the misprediction penalty is much greater than the number of clock cycles needed, then conditional coding can be utilized. [2]

4.3 Partial Reverse If-Conversion

Partial Reverse If-Conversion has three parts as referenced in the following figure.

Analysis Component
Before a determination of execution paths is made, a proper identification and classification of those paths must be identified in the conditional (predicated) code. A predicate flow graph (PFG) is used to display the structure and organization of the predicate

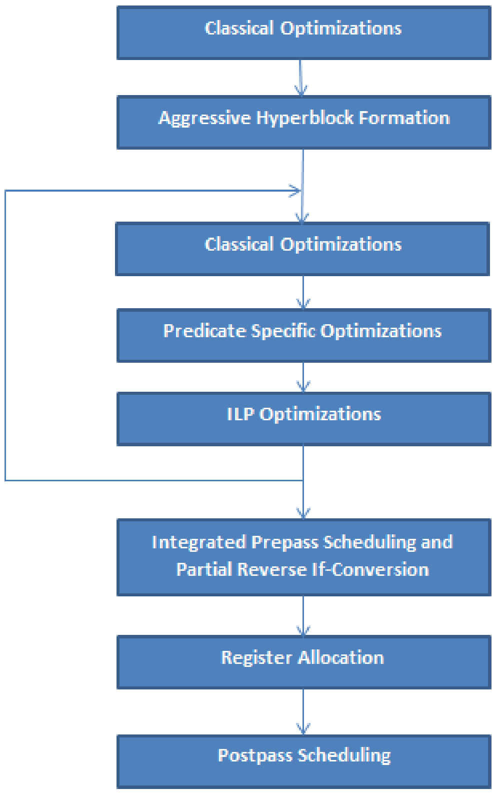

Classical Optimizations
Aggressive Hyperblock Formation
Classical Optimizations
Predicate Specific Optimizations
ILP Optimizations
Integrated Prepass Scheduling and Partial Reverse If-Conversion
Register Allocation
Postpass Scheduling

Fig. 6. Phase Ordering Diagram
[9]

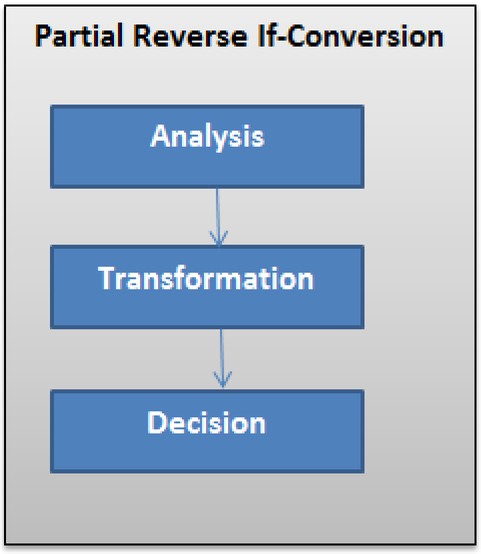


Fig. 7. Steps Involved in Partion Reverse If-Conversion
[9]

paths. A PFG is like a CFG (control flow graph) except that a CFG does not contain the predicate execution paths. [9] The PFG is also used in this component to also determine the dead operations so that they can be suitably removed. [9]

Transformation Component
This component aids in removing redundant code increases. It performs the reverse if-conversion at multiple points.

Decision Component
After the redundant bits of code are disposed of and the PFG created, the compiler then makes a calculated decision on which transformations to perform and which to ignore. This summarizes the Decision Component. [9]

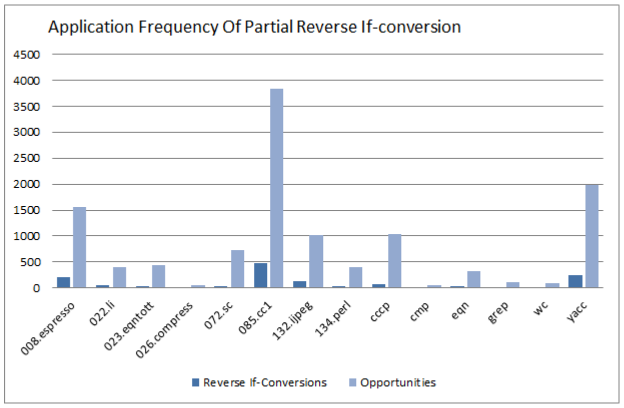
The table below displays the application frequency for the partial reverse if-conversion. It shows the number of reverse if-conversions that occurred throughout the testing process and the number of opportunities gives us the total number of reverse if-conversions that could have taken place.

Benchmark	Reverse If-Conversions	Opps
008.espresso	204	1552
022.li	50	393
023.eqntott	43	443
026.compress	11	56
072.sc	33	724
085.cc1	479	3827
132.ijpeg	134	1021
134.perl	42	401
cccp	77	1046
cmp	4	49
eqn	33	326
grep	3	103
wc	0	88
yacc	247	1976

[9]

4.4 Using Predication

A large amount of time is spent in loops by applications involving 3D Graphics. This is



[10]

Fig. 8. Application Frequency of Partial Reverse if-conversion graph

noticeably larger when the graphics engine carries support for multi-pass vertex processing. Using predication means that internally the loops are rolled out and divided into multiple conditional statements (if-conversion) and are then executed based on compiler analysis. Here we have an example where an incremental look transforms an object. We can see how a minor loop change can result in increased performance [2]

```
for ( int j = 0; j < Polycount; j++)
{ <-- Alt. Code--> }

MOV R0, #0 MOV R1, Polycount

Xform_Polynext:

{<  --- Alt. Code -- >}

ADD R0, R0, #1 CMP

R0, R1 BLT Xform_ Polynext
```

Now, getting the exit loop parameter check comparing to 0 using a decremental loop

```
for ( int j = 10; j != 0; j--)

 {<  --- Alt. Code -- >}
```

```
MOV R0, Polycount

Xform_Polynext:

SUBS R0, R0, #1 BNE Xform_Polynext
```

5 Impact of 3D Attributes

The 3D Pipeline has different stages:

1) The *geometry stage* is the stage where triangles are mathematically manipulated by the application of various geometric transformation functions onto them. [11] This stage is involved with the decision about which aspects/figures need to be displayed and how they need to be displayed.

2) The *triangle setup stage* is the stage which is associated with shading and texture. It is no surprise that the triangle is the fundamental block of 3D graphics. This is because it is a simple shape and every polygon can be represented as a union of multiple triangles. This is termed as triangulation. This stage determines the pixels that are present in the projection of every triangle. [11]

3) The *rendering stage* is the stage that gives color to the shapes by rendering each pixel to its specified color value. It eliminates the additional data that is present virtually in the image but not viewable to the observer in order to conserve space for speedier execution and less system complexity.

5.1 Resolution

The resolution is directly proportional to the execution time, if the resolution increases the rendering time increases as well which causes a bottleneck at high resolutions. The other stages are affected by the amount of time that the rendering stage takes. It is computationally expensive to have high resolutions rendered constantly. [11]

5.2 Lighting

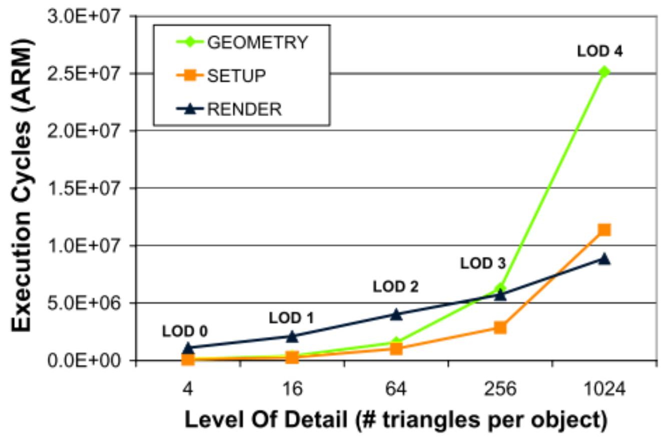
In our day to day lives, clarity of objects is but a representation of light falling on it. Objects are illuminated by light sources and produce a color which is within the spectrum of visibility of our eyes. While using 3D graphics, be it for ARM or PCs/Consoles, it needs to be generated using tools built in 3D software or coded to generate a perception to the human eye.
There is a high cost associated with using lighting while designing code for ARM architecture. This is unavoidable because in most 3D based applications, lighting is a mandatory requirement. In some cases, to optimize the running time, it is possible to use a mapping system that has already been lit and the complexity is moved to the render stage instead of the geometric stage. [11]

5.3 LoD (Level Of Detail)

If you have played computer games or console games, you probably are very familiar with this term already. Level of Detail is what makes the difference between the smoothness of an image which is aesthetically pleasing and one which is choppy and has pixels showing up on the outline of the image. In more technical terms, the Level of Detail or LoD is what is used to depict curved figures that have patches of triangles. The higher the level of detail is, the smoother the images appears to the human observer. The LoD primarily affects the triangle and geometry stage of the 3D graphics pipeline. [11] A balance must be maintained between the quality of the graphic and the execution time of the program. Figure 11 illustrates the impact of LoD on Execution Time.

5.4 Frame Rate

Frame rate is the number of frames displayed per second. Game developers design their code to achieve the maximum Frames per Second (FPS) possible because a low frame rate leads to objects motion appearing jerky and lagged. In more technical terms, if the frame rate is 30 fps, it means that thirty frames are being re-drawn every second. The more efficient the code and settings are, the faster the frames can be drawn leading to an increase in frame rate. If the frame rate is higher then the smoother the motion of animation appears to be as observed by the human eye.

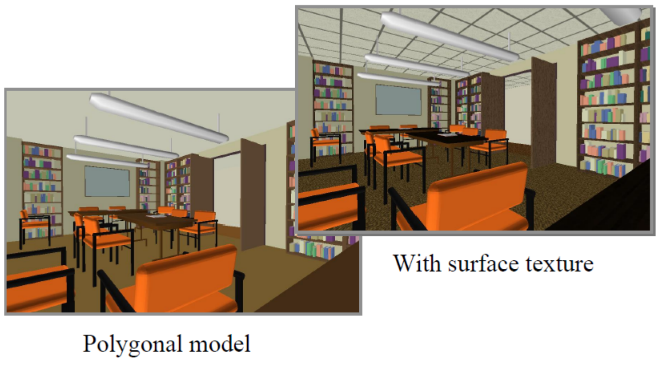


[11]

Fig. 9. Impact of Level Of Detail On Execution Time

5.5 Texture Maps

A texture map is added detail on a given area. The detail can be a texture, color, a picture/graphic or a combination. The texture is usually two dimensional and there are multiple methods of application of it. Pixels present in a texture map are terms as texels. Some texturing schemes are Nearest, Mipmap Nearest, Linear and MipMap Linear. [11]
[12] Figure 10 illustrates the difference in the aesthetic value of an image post texture mapping.



[12]

Fig. 10. Texture mapping to apply detail to 3D Objects

5.5.1 *Linear Mapping*

Linear mapping (also termed as affine texture mapping) and Mipmap nearest are two common texture mapping options. To understand *Linear mapping*, we need to first understand the concept of what the Manhattan distance is.

Manhattan distance: If we consider two points a and b where

$$a = (x_1, y_1)$$
$$b = (x_2, y_2)$$

The Manhattan distance between a and b is given by

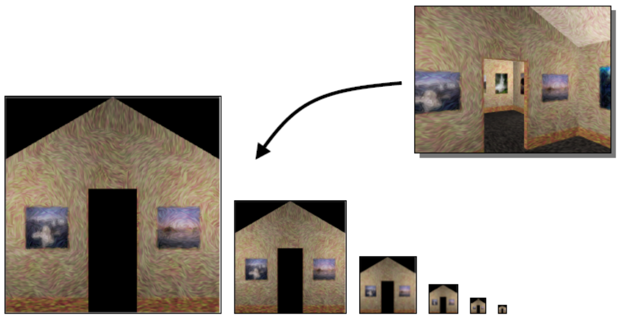
$$MH(a, b) = |x_1 - x_2| + |y_1 - y_2|$$

[13]

In Linear texture mapping, four of the closest texels to a given pixel are selected and the texturing is then done based on a weighted average of those four texels. The distance between the pixel and the texel is determined by the Manhattan distance between them [11]

5.5.2 *Mipmap Nearest*

Of the four texturing schemes mentioned above, Mipmap Nearest is generally preferred because Mipmap Nearest is the process in which several identical copies of the texture map are made. The difference between the copies is the resolution. The resolution of the first copy is twice as much as the second and so on. Based on the distance of the image/object from the observer the texture is changed from higher resolution to lower resolution or vice versa provided that the viewer has acceptable image quality. Changing the texture map to a lower resolution gives better performance without the user noticing the decline in texture quality. [12] This has little impact on execution time with varying texture quality. Hence, Mipmap Nearest is better for mobile devices with the ARM processor for 3D Graphics optimization. Figure 11 shows an image with the change in resolution during this type of texture scheme.



[12]

Fig. 11. Mipmap Nearest

6 Power Consumption

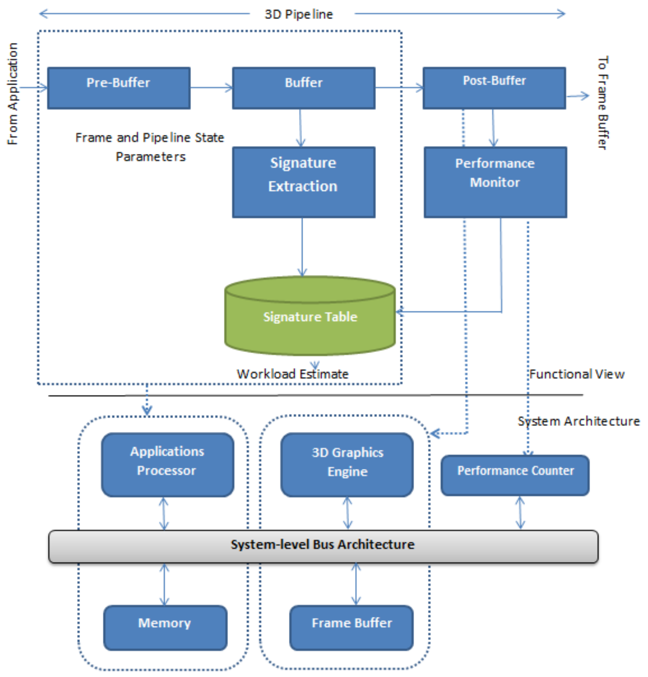


Fig. 12. Signature-basedworkload prediction scheme for mobile 3D graphics, depicting (a) a functional view (b) candidatemapping to a system architecture. [14]

From what we have already stated in this paper, there are multiple ways to implement 3D graphics in ARM. One of the major constraints however is power. Mobile devices usually are battery operated and the amount of time a battery lasts on a single charge is considered to be a defining point in the selling factor of the product in the market today. As a result of this, it is of utmost importance to keep power consumption in mind while developing an application with 3D graphics. Research indicates that taking the weight of a battery constant, the capacity of a battery will increase at the rate of 5% to 10% per year. [11] However, it is also noted that screen resolution and frame rate are constantly on the increase. Although mobile devices have smaller display screens which have lower resolutions compared to PCs or consoles, the mobile devices are usually held closer to the eye of the user thereby needing to render every single pixel with higher precision. The resolution of a screen is the number of pixels contained per inch or the sum of all the pixels contained in the given display unit. It is represented as the product of the width into the height and the standard unit is pixels. The current resolution for some popular mobile devices is as follows:

- Samsung Galaxy S III - WVGA 800 x 480 pixels
- Apple iPhone 4 640 x 960 pixels
- Motorola XT800 480 x 854 pixels

[Samsung, Apple and Motorola websites]

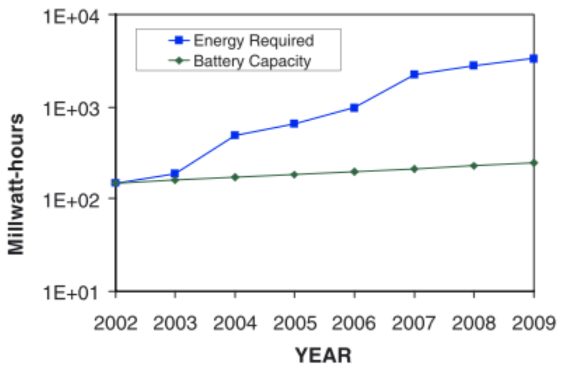


Fig. 13. The energy required for 3D graphics is increasing at a higher rate than the battery capacity rate

[11]

6.1 Impact Of 3D Attributes on Power Consumption

In this section, we will briefly discuss the impact of a few 3D attributes on energy consumption.

6.1.1 Resolution

Bottle neck issues arise as the resolution increases which causes an increase in the execution time. When the resolution is higher the dots per inch are higher. As we know that rendering is the color assigned to a texel which corresponds to the closest pixel, the rendering stage has a an increase in run time which makes it an outlier in comparison to the other stages.

6.1.2 Level of Detail(LoD)

The resolution has a direct impact on Level of Detail because, as the resolution increases, the pixels increase which leads to the ratio of the image increasing. Since the image is now larger, the number of triangles increases and the rendering time increases and could cause bottlenecks problems here as well. [11]

6.1.3 Texture Mapping

Although the MipMap Nearest method of texture mapping is preferred due its low impact on execution time linear mapping or affine texture mapping is also used. Linear mapping is more system instensive and usually at a higher cost than the mipmapping scheme.

The fluctuation in performance leads to bottlenecks and the application of DVFS (Dynamic Voltage and Frequency Scaling) can be made to optimize power consumption. [11]

7 CONCLUSION

Mobile gaming is a huge industry today and the success of several games such as Angry Birds is testimony to this fact. In this paper we have studied about the ARM architecture and we presented the benefits and setbacks of performance optimizations for 3D graphics. The paper also carries information about the benefits of having a hardware FPU.

Fixed-point arithmetic and its advantages and disadvantages have been depicted and we have mentioned the co-relation between fixed-point arithmetic and the balance to be maintained in precision and dynamic range. The significant advantages of Fixed Point arithmetic over Floating-Point is to be noted, considering that it is an unorthodox approach.

The functionality of various arithmetic operations was discussed, as well as their benefits and setbacks in 3D graphics code. Upcoming architectural aspects such as code predication have also been described in this paper. It is determined that predication has a definite edge over branching in more complex scenarios. 3D gaming aspects have also been covered and their impact on the speed and power usage of a mobile device. Aspects such as resolution, frame rate, Lod (Level of Detail), Lighting and Texture maps have been discussed with their impact on mobile device performance.

Overall, this paper encompasses a significant number of variables that play a role in improving the efficiency and optimizing 3D graphics in ARM Architecture.

REFERENCES

[1] L. Ryzhyk, "The ARM Architecture," pp. 1–14, 2006.

[2] G. K. Kolli, S. Junkins, and H. Barad, "3d graphics optimizations for arm architecture,"

[3] F. Chehimi, P. Coulton, and R. Edwards, "Evolution of 3D Games on Mobile Phones 3 . Main Limitations for 3D Games on Mobile Phones :," pp. 1–7, 2005.

[4] G. Frantz, B. D. Manager, and R. Simar, "Comparing Fixed- and Floating-Point DSPs," pp. 1–8.

[5] T. In-depth, "TECHNICAL SESSION Developing 3D Applications for PowerVR MBX Accelerated ARM Platforms," vol. 4, no. 3, 2005.

[6] J. Lauha, "The neglected art of Fixed Point arithmetic," vol. 2006, no. August, 2006.

[7] Y. Wang, D. Gu, D. Ruan, D. Liu, M. Wen, and J. Yang, "Research on Arithmetic Overflow and Underflow Vulnerabilities of Floating Point Special Numbers," vol. 10, pp. 3576–3584, 2011.

[8] D. Melnik, A. Belevantsev, and D. Plotnikov, "A case study : optimizing GCC on ARM for performance of libevas rasterization library,"

[9] D. August, W. Hwu, and S. Mahlke, "A framework for balancing control flow and predication," *Proceedings of 30th Annual International Symposium on Microarchitecture*, pp. 92–103.

[10] D. I. August, W.-m. W. Hwu, and S. A. Mahlke, "The Partial Reverse If-Conversion Framework for Balancing Control Flow and Predication," 1999.

[11] B. Mochocki and N. Dame, "Power Analysis of Mobile 3D Graphics," 2009.

[12] T. Funkhouser, "Texture Mapping Describe color variation in interior of 3D polygon," pp. 1–13, 2000.

[13] S. Sural, C. Histogram, E. Distance, M. Distance, V. Cosine, A. Distance, and H. Intersection, "Performance comparison of distance metrics in content-based Image retrieval applications,"

[14] B. Mochocki, K. Lahiri, S. Cadambi, and X. Hu, "Signature-based workload estimation for mobile 3D graphics," *2006 43rd ACM/IEEE Design Automation Conference*, pp. 592–597, 2006.

[15] B. Bauer, "The Manhattan Pair Distance Heuristic for the 15-Puzzle The Manhattan Pair Distance Heuristic for the 15-Puzzle," 1994.

[16] F. Chehimi, P. Coulton, and R. Edwards, "Evolution of 3D mobile games development," *Personal and Ubiquitous Computing*, vol. 12, pp. 19–25, Nov. 2006.

[17] F. Lamberti, C. Zunino, A. Sanna, D. Politecnico, A. Fiume, and M. Maniezzo, "An Accelerated Remote Graphics Architecture for PDAs," pp. 55–62, 2002.

[18] S. Mahlke, D. Lin, W. Chen, R. Hank, and R. Bringmann, "Effective Compiler Support For Predicated Execution Using The Hyperblock," *[1992] Proceedings the 25th Annual International Symposium on Microarchitecture MICRO 25*, pp. 45–54, 1992.

[19] S. E. Management, "School of Design , Engineering & Computing Peter Knaggs Stephen Welsh," 2004.

[20] M. Reddy, "Perceptually Optimized 3D Graphics," no. October, pp. 68–75, 2001.

[21] P. Robin, "Developing and Optimizing Linux on ARM," 2005.

[22] J.-h. Sohn and R. Woo, "Optimization of portable system architecture for real-time 3D graphics," *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No.02CH37353)*, vol. 1, pp. I–769–I–772.

[23] R. Yates, "Fixed-Point Arithmetic : An Introduction List of Figures List of Tables," 2009.

[24] "Fixed Point Arithmetic With Scaled Integers," 2010.