On the impact of using mixed real number representations in a graphics pipeline

OVIDIU SICOE
Politehnica University of Timisoara
Department of Computer Engineering
Timisoara
ROMANIA
ovidiu.sicoe@gmail.com

MIRCEA POPA
Politehnica University of Timisoara
Department of Computer Engineering
Timisoara
ROMANIA
mircea.popa@upt.ro

Abstract: This paper aims to analyse how the accuracy of rendering 3D models on 2D surfaces is affected by using different real number representation inside different stages of a graphics pipeline. Also, an optimal format that would not alter the final product in a significant way was a target of our research. Additionally, a great number of 2D projections for each 3D Model is targeted. In order to achieve our goals, we have modified an own software implementation of the OpenGL ES 1.1 specification so that it could be easily specified which real number formats are used in each stage.

Key–Words: Fixed-Point, Floating-Point, Graphics pipeline, instrumentation

1 Introduction

Modern graphics processing units(GPUs) used in personal computers provide huge data throughput[1]. They are general purpose GPUs, designed to perform in all cases. Improvements can be made by specialized GPUs, designed to perform for particular applications. Such GPUs could be implemented using FPGAs[2].

In this paper we analyze the impact of using different formats for representing real numbers along the stages of a graphics pipeline implemented on a FPGA. The main goal is performance optimization, but also resource consumption optimization is in as a secondary objective.

2 Graphics pipeline

Most of the graphics hardware works based on a multistage pipeline in which each step is specialized in doing a certain operation. Initial graphics pipelines were fixed[3], meaning that the user could use a limited set of standard transformations that could be applied to the processed data. Nowadays, the trend is to make the pipeline more and more flexible[3], allowing more complex effects to be applied during the data flow.

In order to use and control the graphics pipeline, certain application programing interfaces(APIs) were conceived. Two of the most known and used are OpenGL[4] and DirectX[5]. Those APIs are the interface to the graphic hardware and come as an abstraction of its capabilities. They are implemented in the driver, thus having little differences between each vendor.

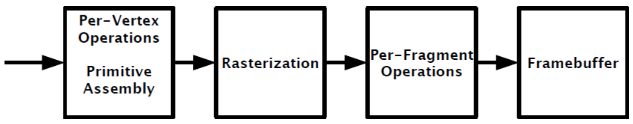


Figure 1: OpenGL ES 1.1 pipeline[4]

Being an open specification, OpenGL was quickly adopted by different platforms and operating systems, allowing for easy, cross-platform graphics content deployment. We have chosen to instrument our own software implementation of the OpenGL ES 1.1 specification due to the simplicity of the concept and because it targets embedded systems.

Figure 1 presents the main stages of the abstraction of a graphics pipeline described by the OpenGL ES 1.1 specification.

In a fixed pipeline, like the one described by OpenGL ES 1.1 specification, most of the heavy real number computations happen in the Per-Vertex Operations stage, mainly implying matrix to matrix or matrix to vector computations[6], followed by the Rasterization stage.

The OpenGL ES 1.1 specification allows 3D input objects to be described as triangles(polygons with three vertices). Figure 2 presents how such an input object would look like. Its surface is described as being composed of multiple polygons of different sizes.

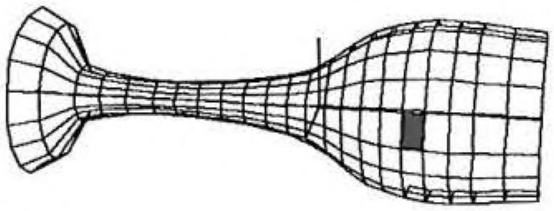


Figure 2: Polygon based surface of a glass[7]

3 Real numbers representations

Accurate real numbers can not be represented in binary computers so approximations are used. For this, there are several formats that suite different needs. Two of the most known and used are floating-point format and fixed-point format.

3.1 Fixed-Point numbers

The fixed-point number format always has a given number of bits for both the fractional part and the integer part. For example, the number 2.5 can be represented on a 16:16 format(16 bits for integer part and 16 bits for the fractional part) as 0000000000000010.1000000000000000.

The general rule is that the integer bits have ratios starting from 2^0 to 2^{i-1}, where i is the number of integer bits. On the other hand, the fractional bits have ratios from 2^{-1} to 2^{-f}, where f is the number of fractional bits. This can be better observed in Figure 3.

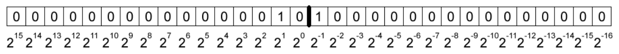


Figure 3: Fixed-Point Format

It can be easily observed that the precision of a fixed-point number is 2^{-f}. The main advantage of fixed-point representation is that the operations can be executed using the integer arithmetic and logic unit(ALU), so they are faster than the floating-point operations.

3.2 Floating-Point numbers

For floating point, the point doesn't have a stationary position, thus the name floating. The general form of such a number is depicted in Figure 4.

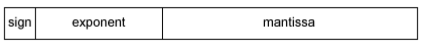


Figure 4: Floating-Point Format

The mathematical interpretation of those bits according to the IEEE 754 specification[8] is $x = -1^{sign} \cdot 1.mantissa \cdot b^{exponent-bias}$, where b is the numeration base of the representation and bias is $2^{size(exponent)} - 1$. Following this, 2.5 is represented as 0.011111111.01000000000000000000000 as a single precision floating-point number.

4 Graphics Pipeline instrumentation

We have conceived a software implementation of the OpenGL ES 1.1 specification so that we could easily change the number representation across the logical stages of the pipeline. For this, we have provided hooks that are executed at the beginning and at the end of each stage, respectively.

We have also implemented an abstraction for real numbers and provided two concrete implementations, one for fixed point numbers and one for floating point numbers. This way, we could easily exchange the two different representations at any point of the execution, even at runtime.

For the fixed point representation, we have used 5:5, 10:10, 15:15, 16:16, 20:20 and 30:30 formats, while for the floating point representation, we have only used the IEEE 754 single precision format, with the exponent represented on 8 bits and the mantissa on 23 bits.

We have chosen the following instrumentation points, according to Figure 1, taking into consideration the operations that were involved in between them:

- the start of the pipeline, before the Per-Vertex Operations stage
- after the the Per-Vertex Operations stage and before the Rasterization

The Per-Vertex Operations stage is the most computation intensive, so we tried to see how a loss in precision during this stage would influence the final output. For this, we have considered that the reference images would be the ones that result from using the floating point representation across the whole pipeline. The next steps have been to replace the floating point number format during the Per-Vertex Operations stage with the fixed point format, taking into consideration different precisions. Table 1 presents the resulting combinations that we have used.

	Per-Vertex Operations	Rasterization
1	FP[1]	FP
2	5:5[2]	5:5
3	10:10	10:10
4	15:15	15:15
5	20:20	20:20
6	30:30	30:30
7	10:10	FP
8	15:15	FP
9	16:16	FP
10	20:20	FP
11	30:30	FP

Table 1: Precision Combinations

For each such combination we have generated 100 images, with the drawn object rotated by two degrees around the vector $(-1, -1, -1)$ each frame. For example, Figure 5a shows the first reference image by using floating point representation along the complete pipeline, Figure 5b represents the output of the same three dimensional model when using 5:5 fixed point along the pipeline, while 5c depicts the difference between the two images.

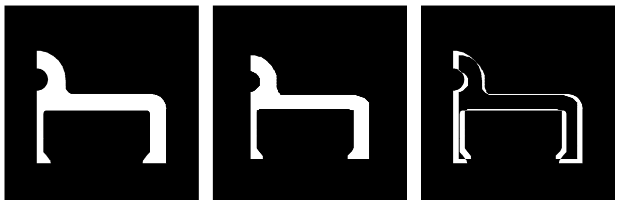

(a) FP (b) 5:5 (c) difference

Figure 5: Frame 0 images

Additionally, we have created difference images for each fixed point image, relative to the floating point counterpart, as briefly depicted in Figure 5.

From this first round of generated images, we have isolated several formats, based on their accuracy loss that we would further use in order to extend the analysis:

- complete 10:10 fixed point across the pipeline
- complete 16:16 fixed point across the pipeline
- complete 30:30 fixed point across the pipeline
- 16:16 fixed point combined with floating point

[1]Floating Point

[2]Fixed point format precision as integer size : fraction size

	3D Model	Polygons
1	ArrowCookieCutter	233
2	Bulldozer	7013
3	Dice	280
4	FirstTrophy	238
5	House	1406
6	Jack-O-Lantern	1668
7	Model	24302
8	PlusCookieCutter	312
9	Rocket	21827
10	Six	3932
11	SoccerTrophy	13652
12	SpaceShuttle	17859
13	SpookyTree	5776
14	TrainCarCaboose	2484
15	TrainEngine	3940
16	TrophyBase5	1222

Table 2: Used Objects

For the second round, we have mainly generated extra images, following the same big scenario: rotate the 3D model by two degrees around the vector $(-1, -1, -1)$ each frame. The main difference is that this time we have taken into consideration more objects and we have rendered 180 frames for each one, so that we have finally achieved a complete 360 degree rotation. The used objects as well as their characteristics are presented in Table 2. A visual representation of those objects is presented in Table 3 from Appendix A.

5 Results

In the first phase, we have generated for a 3D model, using the precision combinations from Table 1 a total of 1100 images. By analysing those images, we have chosen two combinations that would yield big differences and two that would produce more accurate results, in order to see how they would behave on other models.

In the second phase, we have generated projections of the sixteen 3D models presented in Table 2 by using our instrumented pipeline. During this phase, we have generated a total of 11520 images, a total of 720 for each input object, by using four real numbers format combinations.

All the results we have obtained had shown that a full use of fixed point numbers across the complete pipeline would need a rather big increase in the size of the format, the 30:30 format being the only one that

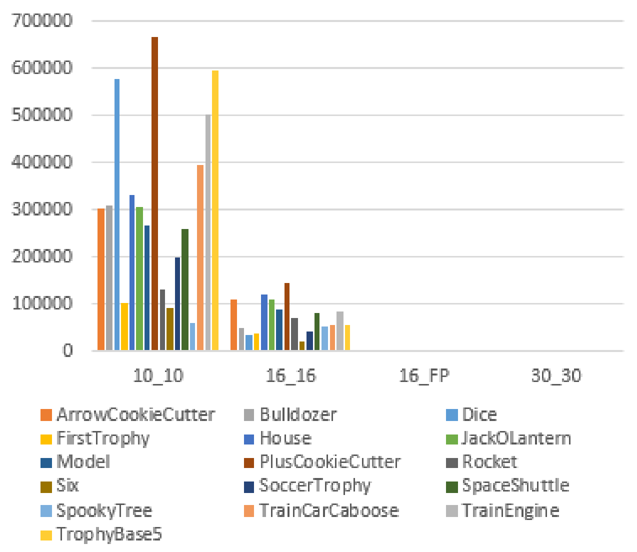


Figure 6: Chart for pixel difference using the four mentioned formats

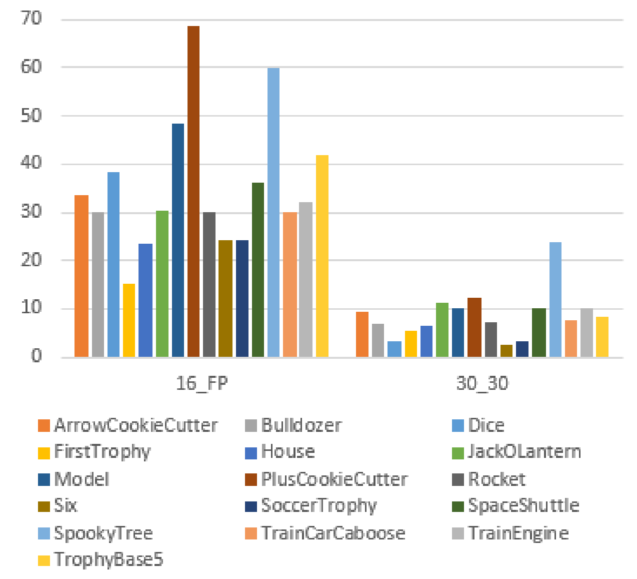
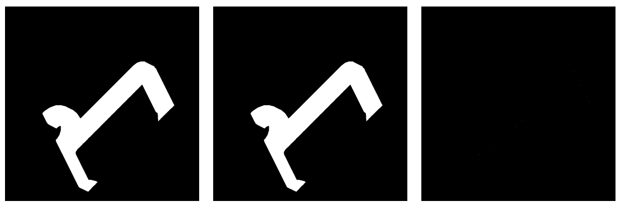


Figure 7: Detailed chart for pixel difference using 30:30 and 16:16 with floating point



(a) FP (b) 30:30 (c) difference

Figure 8: Frame 30 images

provided results almost identical to the original full FP rendering, as depicted in Figure 8. Although the Figure 8c seems black, not highlighting any difference, there are actually 8 pixels different. To support this, the chart in Figure 6 presents the differences in pixels obtained for each 3D model by using the various formats. This only highlights that the small sized fixed point formats used across the whole pipeline yield great output differences, although they are somehow dependant on the configuration of the model, not necessarily on its size. The same figures suggest that the rendering of a compact model, with fewer irregularities is more appropriate for a pipeline using a small sized fixed point representation.

As depicted in Figure 5 and sustained by the chart in Figure 6, the differences are quite significant, so we tried to find a tradeoff between the output accuracy and the used formats presented in Table 1. The output difference images pointed out that the formats presented at line 9 in the same Table 1 would yield outputs pretty close to the original floating point images. The only fixed point format that behaves better is the 30:30, but that means a big increase in the memory footprint; actually real numbers of double size. To enhance this, the chart in Figure 7 presents a more detailed situation that can not be observed in Figure 6 because the values are too small. The average difference for the 16:16 fixed point combined with floating point for all the models is 35.46, while the average difference for full 30:30 fixed point usage is 8.64.

To sustain the idea that a pipeline using the mixed 16:16 and floating point formats produces an acceptable output Figure 9 presents the biggest difference produced by this pipeline relative to the image produced by a full floating point pipeline.

6 Conclusions

We succeeded in comparing the outputs of a graphics pipeline that used different real number representations, either in a single format across the whole pipeline, either using mixed fixed and floating point representations along the stages.

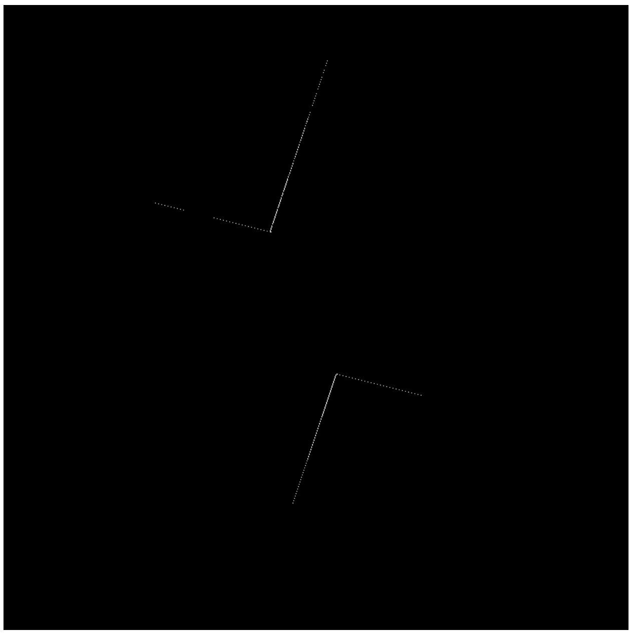


Figure 9: Biggest difference produced by the pipeline using mixed 16:16 fixed point and floating point

One of the main conclusions is that the best mix of real number representations across the pipeline, without a significant increase in resources, but still providing a result similar to the original output, that would not be noticeable by the human eye, would be represented by a 16:16 fixed point representation being used in the Per-Vertex Operations stage and a floating point format being involved in the Rasterization and onward stages.

Another conclusion is that for the graphics pipeline to be accurate when using a fixed point representation across all the stages, this representation would have to be somewhere around 64 bits in size, meaning an increased memory footprint. Although this would represent a bigger resource consumption, it may come with a performance improvement since integer operations are generally faster and easier to implement.

One of the drawbacks of our OpenGL ES 1.1 software implementation is that it doesn't allow to measure realistic times of execution for the different formats, as the algorithms are software implemented and would differ substantially from the hardware implemented counterparts.

One of the other conclusion that we drew is that although using small sized fixed point formats along the stages yields outputs significantly different from the reference images, for a particular shape of the input 3D model, they are not so bad.

As a future analysis, it would be interesting to determine a threshold in the value of the density of pixels from the difference image that would highlight a significant difference for the bare human eye, relatively to the reference image.

References:

[1] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU," in *37th International Symposium on Computer Architecture (ISCA 2010), June 19-23, 2010, Saint-Malo, France*, A. Seznec, U. C. Weiser, and R. Ronen, Eds. ACM, 2010, pp. 451–460. [Online]. Available: http://doi.acm.org/10.1145/1815961.1816021

[2] S. Franchini, A. Gentile, F. Sorbello, G. Vassallo, and S. Vitabile, "An embedded, fpga-based computer graphics coprocessor with native geometric algebra support," *Integration*, vol. 42, no. 3, pp. 346–355, 2009.

[3] D. P. Luebke and G. Humphreys, "How gpus work," *IEEE Computer*, vol. 40, no. 2, pp. 96–100, 2007. [Online]. Available: http://dx.doi.org/10.1109/MC.2007.59

[4] T. K. G. Inc, "Opengl ES Common/Common-Lite Profile Specification," https://www.khronos.org/registry/gles/specs11.

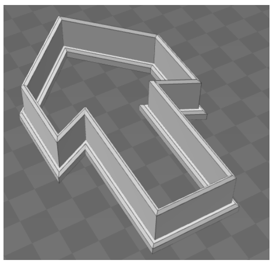
[5] Microsoft, "DirectX Graphics and Gaming," https://msdn.microsoft.com/en-us/library/windows/desktop/ee663274(v=vs.85).aspx.

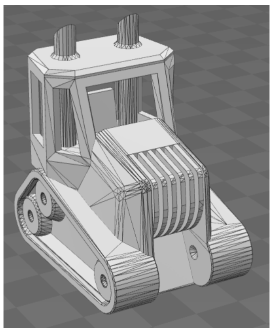
[6] J. F. Hughes, A. van Dam, M. McGuire, D. F. Sklar, J. D. Foley, S. K. Feiner, and K. Akeley, *Computer Graphics: Principles and Practice (3rd Edition)*. Addison-Wesley, 2013.

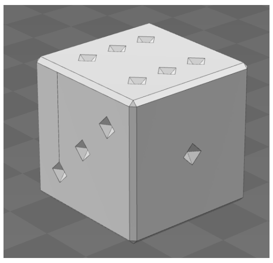
[7] A. H. Watt, *3D Computer Graphics with Cdrom*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

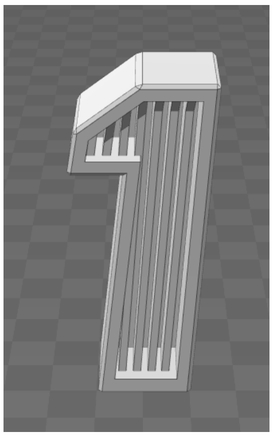
[8] P. W. Markstein, "The new IEEE-754 standard for floating point arithmetic," in *Numerical Validation in Current Hardware Architectures, 6.1. - 11.1.2008*, ser. Dagstuhl Seminar Proceedings, A. A. M. Cuyt, W. Krämer, W. Luther, and P. W. Markstein, Eds., vol. 08021. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2008. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2008/1448

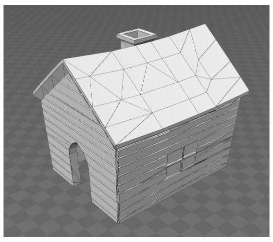
A 3D Models

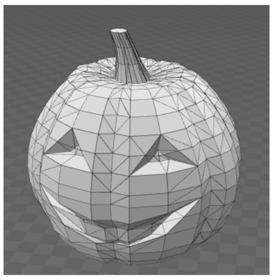

(a) ArrowCutter

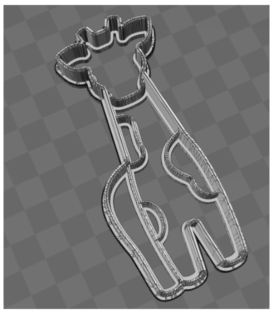

(b) Bulldozer

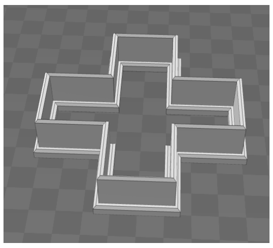

(c) Dice

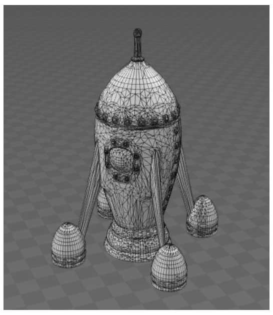

(d) FirstTrophy

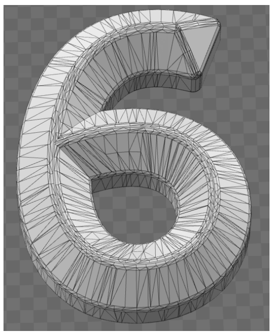

(e) House

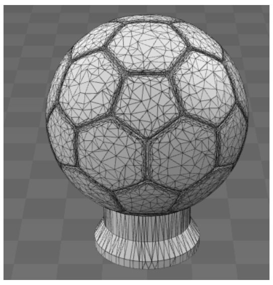

(f) Jack-O-Lantern

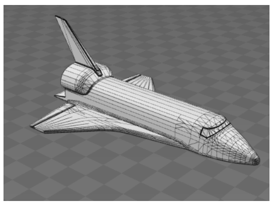

(g) Model

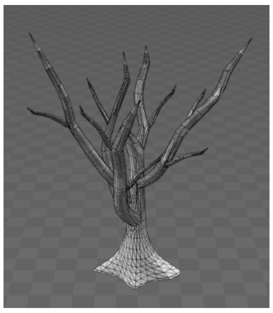

(h) PlusCookieCutter

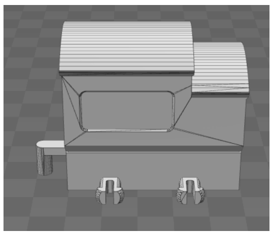

(i) Rocket

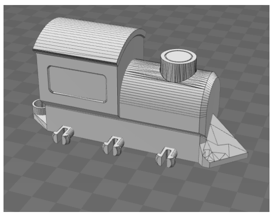

(j) Six


(k) SoccerTrophy


(l) SpaceShuttle


(m) SpookyTree


(n) TrainCarCaboose


(o) TrainCarEngine

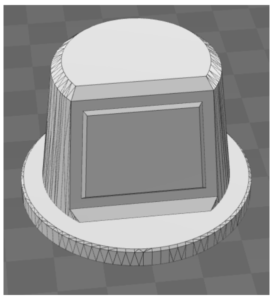

(p) TrophyBase5

Table 3: 3D Models