High-Performance "Fake" Specular Lighting for Real-Time Rendering on Budget Hardware and GPUs

Yuriy Kotsarenko[1] , Humberto Andrade[1], Fernando Ramos[1],

[1] ITESM Cuernavaca Morelos, México
ykot@inbox.com, humbertoandrade@gmail.com, fernando.ramos@itesm.mx

Abstract. In this work two novel techniques are proposed for 3D lighting computed in real-time on dedicated video hardware (GPU). Classical techniques such as Phong specular reflections are computationally heavy when executed on budget hardware, performing poorly in real-time and reducing battery life. The proposed alternatives are defined in simpler terms yet produce realistically looking results similar to the classical techniques. Numerous experiments are provided implementing the proposed techniques in hardware running both on GPU and CPU. The provided performance benchmarks show that the proposed techniques boost the performance significantly on budget equipment. The experiments were made on many different computers both on 32-bit and 64-bit platforms using single-threaded and multi-threaded approaches to evaluate the real-time performance accurately.

Keywords: 3D lighting, real-time illumination, specular reflections, GPU programming, 64-bit platform, multi-threading.

1 Introduction

In computer software applications where interactive 3D scenes are rendered it is common to work with shaders running directly on the dedicated video hardware, specifically the graphics processing unit (GPU). Such applications include but not limited to video games, interactive simulations, entertainment, scientific experiments and medical support. In many cases the budget is limited so the applications must scale to hardware with different processing power, starting from budget mobile netbooks to powerful high-end workstations. In addition, every attempt is made in the applications to produce the 3D as much realistically looking as possible. In fact, in the movie industry a large set of high-end computer stations is used to create computer-generated video frames, a process which can take from few minutes to several days. In the video game industry, an application is commonly run on a personal computer or dedicated hardware (such as gaming console) that has limited computational capabilities. During the last few years the computer industry evolved to assist these applications with powerful GPUs capable of doing heavy computations so that more complex 3D scenes can be rendered as fast as possible yet looking closely to the reality.

M.A. Cruz-Chávez (Ed): CICos 2011, ISBN. 978-607-00-5091-6. pp. 115 – 125, 2011

The rendering of 3D scenes is commonly made by using one of the two popular technologies, or APIs – Direct3D provided by Microsoft [1] and OpenGL managed by Khronos Group [2]. In earlier years the rendering process was implemented in the majority by each of the APIs and the underlying hardware, but now it is possible to customize the rendering using shaders that are executed directly on GPU. In this work, high-level shading language (HLSL) was used, a proprietary shader language developed by Microsoft for use with Direct3D API. OpenGL has similar technology called GLSL. Although shaders written in HLSL for Direct3D can be ported to GLSL, the process is not covered in this work.

In typical 3D lighting approaches, the resulting color is produced by a sum of two (or more) components, including diffuse term and specular reflection, typically calculated as in Phong or in Blinn-Phong shading models [1], [3-4]. In the calculations RGB color space is typically involved (some other color spaces are covered in [5-8]), although other implementations have been proposed [9-10]. The calculation of traditional diffuse lighting and specular reflections is one of the most popular techniques used in the industry.

1.1 The diffuse and specular reflection

The diffuse lighting technique simulates materials such as matte that appear equally bright when viewed from any angle because the light is scattered in many directions from the surface so the viewed brightness is only related to the angle between the light's source and the surface normal. In the case of specular reflection, which simulates metallic and shiny materials, the viewed brightness changes depending on the viewer's position respective to the object and the light's origin. The difference between the two is illustrated on the figure below.

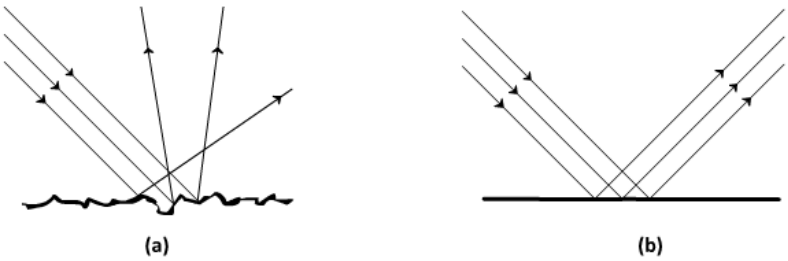

Fig. 1. (a) A rough surface reflects light diffusely (b) A plane surface produces specular reflection.

It can be observed on the Fig. 1 that the diffuse lighting assumes the scattering of the light while for specular reflections most of the light's energy is transferred to the observer. The geometry of specular reflections is similar to the one of the mirror, bounds to few basic rules as shown on Fig. 2. First, the incident ray and the reflected ray are always located on the same plane; independently of the incident surface be it plane or a curved one, the point where the incident ray hits is uniquely defined as it can be seen below. The angle of light's ray incidence and the angle of reflections are equal.

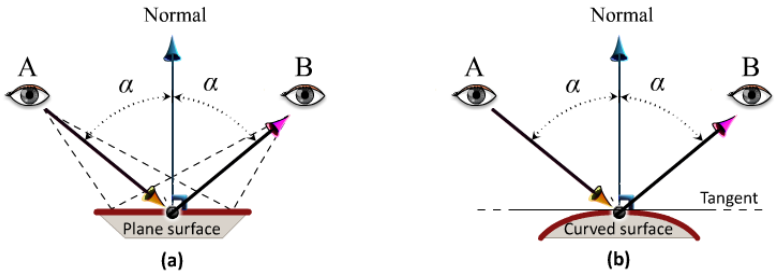

Fig. 2. (a) Geometry of reflection on a planar surface (b) Geometry of reflection on a curved surface.

The implementation of shaders for diffuse lighting and specular reflections is discussed in great detail in the popular literature (e.g. [1] and [3]), commonly denoted as Phong lighting technique, although other lighting techniques exist [4].

2 "Fake" Specular Technique

In this novel lighting technique the resulting colors can be calculated for each individual vertex of the given 3D mesh inside vertex shader, or for each individual pixel in the pixel shader. In the context of this work the lighting calculations will be calculated for every pixel as in typical 3D shader applications. In this configuration the components such as vertex position and vertex normal are interpolated for each pixel. The first step in the lighting technique is to calculate the incident angle between the light and the surface it hits:

$$\alpha = \arccos\left(\vec{L}_0 \cdot \vec{N}_W\right) \tag{1}$$

where $\vec{L}_0$ is the light direction and $\vec{N}_W$ is the interpolated surface normal transformed into world space. If the light source is defined as point light (or "omni light"), the light direction can be calculated as:

$$\vec{L}_0 = \left|\vec{P}_0 - \vec{P}_W\right| \tag{2}$$

where $\vec{P}_0$ is the light's position in world space and $\vec{P}_W$ is the interpolated surface position transformed into world space. The resulting light vector is normalized to make sure it is a unity vector. The angle described earlier assumes that the normal vector is also normalized. The diffuse component in this lighting technique can therefore be calculated as the following:

$$D_{term} = \text{saturate}\left(\frac{\pi/2 - \alpha}{\pi/2}\right) \tag{3}$$

where saturate() is a function in HLSL that clamps the result in [0, 1] range and is usually optimized for performance when the shader code is compiled to shader assembly. The diffuse color component can be calculated as:

$$C_D(r,g,b) = C_{\text{in}}(r,g,b) \cdot D_{term} \tag{4}$$

where Cin is the source interpolated color either taken from the vertex or sampled from the texture. The fake specular reflection component is calculated using the same angle α as the following:

$$S_{term} = \text{saturate}\left(\frac{p_0}{1-\alpha} - k_0\right) \tag{5}$$

where p0 is the apparent reflection strength and k0 is the strength adjustment parameter; both parameters are calibrated to produce the desired result that resembles Phong specular reflection. In the context of this work, the parameters were chosen as p0=0.2 and k0=0.5. The final color component can be calculated as:

$$C_{final}(r,g,b) = C_{\text{D}}(r,g,b) + S_{term} \tag{6}$$

It is important to note that this lighting technique works directly with angles between the surfaces and not the dot product itself as in the case of classical diffuse lighting and Phong reflections. This may impact performance to a lesser degree because of arccosine calculation but the result is more accurate.

3 Fast "Fake" Specular Technique

It was mentioned earlier that the proposed lighting technique is more accurate in diffuse component because it works with angles between the surfaces as opposed to raw dot product found in many classical lighting techniques. For the sake of performance the arccosine in the calculation can be dropped and the entire process can be made using dot products instead. The new diffuse component is calculated as:

$$P_0 = \text{saturate}\left(\vec{L}_0 \cdot \vec{N}_W\right) \tag{7}$$

The diffuse color component is therefore calculated as:

$$C_D^{'}(r,g,b) = C_{\text{in}}(r,g,b) \cdot P_0 \tag{8}$$

It can be seen in the above equation that it is similar to how diffuse component is calculated in the classical techniques. The fake specular component can be calculated in two steps:

$$P_0^{'} = \max\left(\left(P_0 - 0.5\right), 0\right) \cdot \left(2\gamma'\right) \tag{9}$$

where $P_0^{'}$ is adjusted lighting component, max is the function that returns the maximum of two values in HLSL and γ' is an alpha calibration component that is used to tune up the produced result so it roughly matches the more accurate lighting version described earlier. In the context of this work, γ'=0.9. The new specular component can be calculated as:

$$S_{term}^{'} = \text{saturate}\left(\frac{p_0}{1 - P_0^{'}} - k_0\right) \tag{10}$$

The final color equation simply combines the newly introduced components:

$$C_{final}^{'}(r,g,b) = C_D^{'}(r,g,b) + S_{term}^{'} \tag{11}$$

It is evident that this technique is very simple in mathematical terms and can be calculated with minimal effort on GPU.

4 Experimental Results

The lighting techniques described earlier were subject to numerous experiments with throughout performance benchmarks. In the experiments many different computers were used ranging from ultra-mobile netbooks to high-performance workstations. The specification of all testing machines is provided below.

Table 1. The specification of all testing machines with their abbreviations that are used as references in the experimental result tables.

Abbreviation	Machine Specification
Eee8G	Asus Eee PC 8G, Intel Celeron 630 Mhz, DDR2-570 Mhz, Intel GMA900
Eee10HE	Asus Eee PC 1000HE, Intel Atom N280 1.66 Ghz, DDR2-667 Mhz, Intel GMA950
Eee15PN	Asus Eee PC 1015PN, Intel Atom N570 1.66 Ghz, DDR3-1333 Mhz, Intel GMA 3150
Eee15PNi	Asus Eee PC 1015PN, Intel Atom N570 1.66 Ghz, DDR3-1333 Mhz, Nvidia ION 2
LatD830	Dell Latitude D830, Intel Core 2 Duo T7700 2.4 Ghz, DDR2-667 Mhz, Intel GMA X3100
Vost1500	Dell Vostro 1500, Intel Core 2 Duo T7250 2.0 Ghz, DDR2-667 Mhz, Nvidia Geforce 8600 GT
Prec4500	Dell Precision M4500, Intel Core i7 Q840 1.86 Ghz, DDR3-1333 Mhz, Nvidia Quadro FX 1800
Cor2-18	Desktop Core 2 Duo E6300 1.86 Ghz, DDR3-1066 Mhz, Nvidia Geforce 7600 GS
Cor4-24	Desktop Core 2 Quad Q6600 2.4 Ghz, DDR2-800, Nvidia Geforce 250 GTS
Cor4-25	Desktop Core 2 Quad Q8300 2.5 Ghz, DDR2-1333, ATI Radeon HD 4870
MacMin	Apple Mac Mini Core 2 Duo 2.0 Ghz, DDR2-667, Nvidia Geforce 9300 GS

Cel1200	Desktop Celeron Dual-Core E1200 1.6 Ghz, DDR2-667, Intel GMA X3100
Stu1535	Dell Studio 1535, Core 2 Duo T5800 2.0 Ghz, DDR2-800, Intel GMA X3100
DvT4200	HP Pavilion DV4 , Pentium Dual-Core T4200 2.0 Ghz, DDR3-1066, GMA 4500M

Three different applications were used in the experiments. The first one used HLSL shaders for rendering P-Q Torus Knot [11] filled with a single diffuse color (no texturing); the model has 16929 vertices and 32768 faces. The resolution of the rendered image was 512x512 with 8x multisampling, if supported by a certain video card. The model was chosen so that the polygon count is low and the vertex shader will contribute little to the performance benchmarks. The shape of the model is also important as it can illustrate the shading from many different angles. The same model was used with three different pixel shaders illustrating the two proposed lighting techniques and the classical Phong implementation.

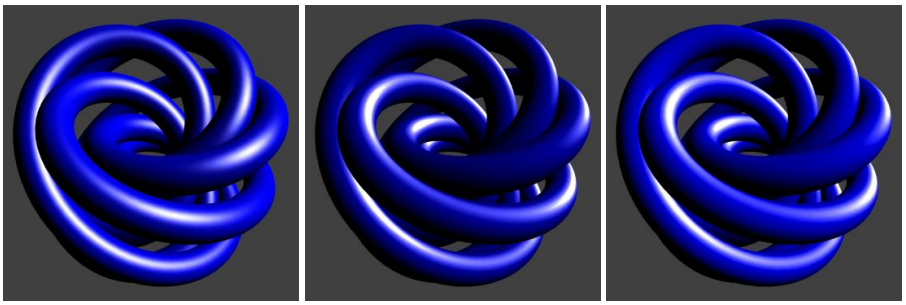

Fig. 3. P-Q (7-4) Torus Knot rendered using a) Phong reflections, b) "Fake" specular and c) Fast "fake" specular.

From the images on **¡Error! No se encuentra el origen de la referencia.** it can be seen that although the general location of the specular reflection is different, all three images look similarly shining and produce an illusion of metallic look. Should an unprepared viewer watch each of these images, it would be difficult if not impossible to realize whether or not the real Phong shading is being used. In addition, it is important to note that the diffuse lighting used in "Fake" specular produces higher contrast and more metallic look; as it was mentioned earlier, the "Fake" specular uses more accurate diffuse approach.

Table 2. Performance benchmarks for the first application with different shading techniques running directly on GPU using shaders. The values are specified in frames per second.

Machine	Phong reflections	"Fake" specular	Fast "fake" specular
Eee8G	33	30	49
Eee10HE	58	53	75
Eee15PN	73	65	95
Eee15PNi	78	88	102

LatD830	65	94	123
Vost1500	180	183	182
Prec4500	406	402	406
Cor2-18	264	232	287
Cor4-24	1028	1032	1032
Cor4-25	1655	1654	1657
Cel1200	95	91	109
Stu1535	52	72	96
DvT4200	105	105	127
Minimal Benefit		**88%**	**100%**
Maximum Benefit		**145%**	**189%**
Average Benefit		**104%**	**128%**

As it can be seen from the results on Table 2, the computers having weaker and cheaper video cards are in greater benefit of the proposed alternatives, while high-end video cards have little benefit. In some cases, up to 89% increase in performance is observed when using the fast "fake" specular technique.

In the second application, the different lighting techniques were implemented in a more advanced approach: per-pixel bump mapping in shaders. In this approach, a different P-Q Torus Knot model was rendered textured using brick-shaped metallic texture. The model has 10593 vertices and 20480 faces, again being few polygons so that the majority of work is made in pixel shader. The rendered image resolution was 512x512 with 8x multisampling. On desktop machines the 960x960 resolution was used instead. The resulting rendering is shown below.

Fig. 4. P-Q (3-4) Torus Knot rendered using per-pixel bump mapping with a) Phong reflections, b) "Fake" specular and c) Fast "fake" specular.

From the images on Fig. 4 it can be seen that the proposed techniques produce realistically looking results which to an inexperienced viewer make look the same. The performance, however, when using the proposed techniques is a different story. It can be noted that the "fake" specular technique, which uses more accurate diffuse

shading produces image with higher contrast and more metallic look. The performance results are shown below.

Table 3. Performance benchmarks for the second application with different shading techniques running directly on GPU using shaders illustrating per-pixel bump mapping. The values are specified in frames per second.

Machine	Phong reflections	"Fake" specular	Fast "fake" specular
Eee8G	41	41	44
Eee10HE	71	67	83
Eee15PN	93	88	112
Eee15PNi	126	140	148
LatD830	110	141	154
Vost1500	256	252	256
Prec4500	633	633	636
Cor2-18	135	134	136
Cor4-24	626	625	626
Cor4-25	1448	1446	1450
Cel1200	102	101	105
Stu1535	85	113	123
DvT4200	191	185	204
Minimal Benefit		**94%**	**100%**
Maximum Benefit		**133%**	**145%**
Average Benefit		**104%**	**112%**

It can be seen from the results on Table 3 that the proposed techniques produce better real-time performance in more complex situations as well. The benefit margin is smaller because more weight is put on pixel shaders by sampling the texture and calculating bump-mapping parameters. Still, the majority of low-end to mid-end video cards benefit from the proposed techniques with up to 45% increase in performance.

The previous two applications were executed in shaders running directly on GPU. However, in certain mobile devices the GPU acceleration is not available. Therefore, in the last application the different lighting techniques were implemented in software running directly on computer's CPU. The application was compiled in Embarcadero Delphi XE 2 with compiler optimizations enabled and all debugging information disabled. In total four different approaches were used: 1) single-threaded approach on 32-bit platform, 2) single-threaded approach on 64-bit platform, 3) multi-threaded approach on 32-bit platform and 3) multi-threaded approach on 64-bit platform. In the single-threaded approaches, the program calculated the final color for one million vertices. In the multi-threaded approaches, the program calculated the final color for a million of vertices in each of the 64 threads that were run simultaneously. The execution time was evaluated for each instance. The benchmarking results for single-threaded variant are shown below.

Table 4. Performance benchmarks running single-threaded lighting techniques for million pixels on 32-bit and 64-bit platforms. The data values are specified in milliseconds indicating the total execution time of experiment; grey cells indicate that the 64-bit feature was not tested on that machine.

	Phong	**"Fake"**	**Fast "fake"**	**Phong**	**"Fake"**	**Fast "fake"**
Machine		**32-bit**			**64-bit**	
Eee8G	881.34	707.47	328.30			
Eee10HE	868.73	507.57	244.23			
Eee15PN	849.42	491.10	243.94			
LatD830	201.40	176.61	69.07	154.61	185.35	46.77
Vost1500	263.24	217.16	88.05			
Prec4500	140.47	120.18	55.65	92.33	103.50	36.50
Cor2-18	268.80	242.17	94.14			
Cor4-24	231.57	193.50	78.85	163.13	195.46	49.68
Cor4-25	158.84	146.39	64.91	125.73	150.23	41.39
MacMin	277.22	203.90	102.20			
Cel1200	348.14	313.85	117.62			
Stu1535	471.05	392.97	163.30	327.30	393.72	98.94
DvT4200	381.52	317.84	150.23			
Minimal Benefit		**109%**	**245%**		**83%**	**253%**
Maximum Benefit		**173%**	**356%**		**89%**	**331%**
Average Benefit		**127%**	**288%**		**85%**	**309%**

The results shown on Table 4 illustrate the performance increase with the proposed techniques. It is evident that in some cases the performance is increased dramatically (sometimes three times faster and more!) by using one of the proposed lighting techniques. An interesting observation is that on 64-bit platform the "fake" specular technique seems slower, most likely due to arccosine function being the bottleneck. The above table can be used as a performance benchmark of different CPUs when used in single-threaded mode. However, for a more objective study it is important to use multi-threading capabilities of CPUs, which over past few years have grown substantially. The results for multi-threaded approach are shown below.

Table 5. Performance benchmarks running multi-threaded lighting techniques for million pixels in each thread on 32-bit and 64-bit platforms. The total number of threads was 64. The data values are specified in milliseconds indicating the total execution time of experiment; grey cells indicate that the 64-bit feature was not tested on that machine.

	Phong	**"Fake"**	**Fast "fake"**	**Phong**	**"Fake"**	**Fast "fake"**
Machine		**32-bit**			**64-bit**	
Eee8G	929.46	713.46	339.17			
Eee10HE	555.89	298.64	166.40			
Eee15PN	282.22	157.67	85.28			
LatD830	107.71	97.29	41.41	82.85	100.03	26.30
Vost1500	133.89	113.36	50.03			
Prec4500	50.14	39.07	24.82	33.58	36.67	16.54
Cor2-18	144.82	125.96	50.48			
Cor4-24	56.99	49.08	24.20	46.84	56.92	16.14
Cor4-25	41.19	37.34	24.49	32.79	39.56	16.00
MacMin	141.10	105.48	55.13			
Cel1200	165.85	141.23	58.21			
Stu1535	223.57	198.16	81.70	169.87	203.63	55.30
DvT4200	177.95	157.82	72.66			
Minimal Benefit		**110%**	**168%**		**82%**	**203%**
Maximum Benefit		**186%**	**334%**		**92%**	**315%**
Average Benefit		**129%**	**263%**		**85%**	**264%**

As it can be seen on Table 5, the performance increase from using the proposed techniques is also significant when multi-threading is used, although to a lesser degree (most likely due to cache contamination on CPUs with small cache). It is also important to note that multi-threading greatly increases the performance in some cases. The above table can also be used as a definitive performance guide to each of the individual CPUs. For instance, from the table it can be seen that Core i7 Q840 can illuminate 62.5 million vertices (or pixels) using fast "fake" technique per second on 64-bit platform; in other words, filling the entire 1024x768 screen (and calculating the fast "fake" lighting for each pixel) it can achieve rendering speed of 794 frames per second using multiple threads.

Conclusions and Future Work

In this work several alternatives were proposed to the classical lighting techniques. The performance achieved with the proposed lighting techniques is significantly

higher both on GPU and on CPU with the similar perceived results. Although the proposed techniques are not a full replacement to high quality specular reflections using traditional approaches, they can be used for performance-critical applications and video games running on budget hardware. The proposed "fake" specular lighting produces diffuse-lit colors that are more accurate than in the traditional lighting techniques; although somewhat slower in some special circumstances (such as on 64-bit CPUs) and faster in others (GPU, 32-bit CPUs) it produces realistic results. The fast "fake" lighting technique is drastically faster than the traditional Phong technique. A special LOD-based approach can be used to mix both one of the proposed alternatives and the classical Phong technique for a hybrid approach where distant objects use faster alternative and closer objects are rendered with a slower classical technique. In the majority of cases it is difficult to determine visually for an inexperienced viewer that the used technique is not a true Phong reflection; the only way to figure it out would be looking at the light's origin and then at object to see that the reflection actually goes back to the light's origin. The last issue can be possibly mediated by using two light origins per single light, one being the original position for diffuse component while another being calculated as the average between the viewer and light's origin to be used for specular component, simulating the moving reflection.

References

[1] Luna, Frank D. *Introduction to 3D Game Programming with Direct X 9.0c: A Shader Approach.* 1st edition. Jones & Bartlett Publishers, 2006.

[2] OpenGL Architecture Review Board. *OpenGL(R) Reference Manual.* 4th Edition. Edited by Dave Shreiner. Addison-Wesley Professional, 2004.

[3] Dempski, Kelly, and Emmanuel Viale. *Advanced Lighting and Materials with Shaders.* Jones & Bartlett Publishers, 2004.

[4] Lengyel, Eric. *Mathematics for 3D Game Programming and Computer Graphics.* 2nd edition. Charles River Media, 2003.

[5] Hearn, Donald, and Pauline M. Baker. *Computer Graphics, C Version.* Prentice Hall, 1996.

[6] Hill, Francis S. *Computer Graphics using OpenGL.* Prentice Hall, 2000.

[7] Lindbloom, Bruce J. "Accurate Color Reproduction for Computer Graphics Applications." *Computer Graphics* 23, no. 3 (July 1989): 117-126.

[8] Schanda, Janos. *Colorimetry: Understanding the CIE system.* Wiley Interscience, 2007.

[9] Kotsarenko, Yuriy, and Fernando Ramos. "Simple perceptual color space for color specification and real-time processing." *22nd General Congress of the International Commission for Optics (ICO).* Puebla: SPIE, 2011.

[10] Kotsarenko, Yuriy, and Fernando Ramos. "Measuring perceived color difference using YIQ NTSC transmission color space in mobile applications." Edited by Marco Antonio Cruz Chávez. *Programación Matemática y Software* 2, no. 2 (December 2010).

[11] Adams, Colin Conrad. *The Knot Book: An Elementary Introduction to the Mathematical Theory of Knots.* W. H. Freeman & Company, 1994.