Alternative Diffuse Lighting and Specular Reflection Approach Using YIQ Color Space for 3D Scene Visualization Using Programmable HLSL Shaders

Y. Kotsarenko*, F. Ramos

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Cuernavaca
Xochitepec, Morelos, México

ABSTRACT
In this work an alternative approach for diffuse lighting and specular reflections is presented that uses YIQ color space instead of traditional RGB color space. The classical illumination algorithms that rely on RGB color space may lead to unrealistic results either due to the fact that they cannot make the original color brighter (inherent limitation imposed by using the RGB color space) or produce incorrect shades when viewed from certain angles because in the calculation of the specular reflection the original texture color is not taken into account.

The approach proposed in this work gets around the problem by applying illumination in YIQ color space using its "luma" component (Y). In this novel approach the diffuse lighting is capable of increasing the perceived brightness of the source texture and the resulting color is always dependant on the surface's texture and produces reasonably realistic results when viewed from any possible angle. The algorithm and its HLSL shader code are described in this work along with the experiments that illustrate the problem and the solution. The performance benchmarks are also provided, showing that the proposed approach is a viable and realistic solution for applications running in real-time.

Keywords: visualization, illumination, shaders, lighting, reflection.

RESUMEN
En este trabajo se presenta un nuevo enfoque para iluminación difusa y reflexiones especulares que utiliza el espacio de color YIQ en vez del tradicional RGB. Los algoritmos clásicos de iluminación que utilizan el espacio de colores RGB pueden generar resultados no realísticos ya sea porque no pueden hacer el color de la textura original más brilloso (limitante que se origina por utilizar el espacio de color RGB) o pueden producir un tinte incorrecto en ciertos ángulos porque el cálculo de la reflexión especular no toma en cuenta el color de la textura original.

El enfoque propuesto evita los problemas anteriormente mencionados aplicando la iluminación en el espacio de colores YIQ utilizando el componente de "luma" (Y). Con este enfoque novedoso en la iluminación difusa se puede incrementar el brillo perceptual de un color, el cual siempre depende de la textura original. Por lo tanto, el resultado se ve razonablemente realístico desde cualquier ángulo. El algoritmo y su código fuente HLSL están descritos junto con los experimentos que muestran el problema y el resultado. Además, las pruebas de desempeño también están adjuntas mostrando que el enfoque propuesto es una solución viable y realística para aplicaciones corriendo en tiempo real.

1. Introduction

In many modern applications such as video games, movies and simulations where an artificial 3D scene is displayed, every attempt is made to make this scene look realistic and closer to the reality. In the movie industry, typically a large set of powerful computers is used to create computer generated video frames, a process which can take from few seconds 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 video cards having a graphics processing unit (GPU) capable of doing vast amounts of calculations per second – all with an effort of displaying more complex 3D scenes as closer to the reality as possible.

The rendering of 3D scenes on the computer display is typically implemented using one of the two technologies, or APIs – Direct3D developed by Microsoft and/or OpenGL managed by the Khronos Group. In the earlier years the entire rendering process was implemented by each of the APIs and the underlying hardware, but now it is possible to customize the rendering by writing programs that run on the GPU, or in other words – by using shaders. In the context of this work, the concept of shaders refers to high-level shader language (HLSL), a proprietary shader language developed by Microsoft for use with Direct3D API. OpenGL has analogous technology called GLSL. Although shaders written in HLSL for Direct3D can be ported to GLSL vice versa, the process is now covered here.

In a typical 3D application the objects are illuminated by one or more lights to generate shades that give the illusion of depth. When the lighting is implemented in shaders, it is generally separated into two steps – applying diffuse lighting and specular reflection. The diffuse lighting takes the fact that surfaces such as matte paint appear equally bright when viewed from any angle because of the rough surface, hence its brightness is only dependant on the angle between the actual light and the surface's normal [1]. On the other hand, the specular reflection works in a similar fashion to a mirror and is dependent on the eye (or camera) position [2]. In addition, a so-called ambient lighting is usually applied, which is simply a constant added to diffuse illumination component.

Considering that a 3D model is represented by a series of triangles consisting of vertices and vertex normals, the diffuse lighting component can be calculated using Lambert's Cosine Law:

$$L_D = \max\left(\vec{L} \cdot \vec{n}, 0\right) \tag{1}$$

where $\vec{L}$ is the vector pointing from the light source to the surface and $\vec{n}$ is the surface's normal. The above equation can be applied either to each vertex and/or pixel, depending on the implementation. It is common to add the value of *ambient lighting* to the diffuse lighting component to make the object appear brighter when it is partially lit. The value L_D is usually delimited within [0, 1] range. The specular reflection component according to Phong illumination model [3] can be calculated as:

$$L_S = \max\left(\vec{v} \cdot \left(\vec{L} - \left(2\vec{L} \cdot \vec{n}\right)\right), 0\right) \tag{2}$$

where $\vec{v}$ is the view vector pointing from the surface to the eye position, $\vec{L}$ is the vector pointing from the light source to the surface and $\vec{n}$ is the surface's normal.

The typical calculation of the final color (for each vertex or pixel depending on the implementation) can be generalized (according to [1], [2], and [4]) as:

$$F_i = C_i\left(L_A + L_D\right) + L_S,\ i = \mathrm{r,g,b} \tag{3}$$

where F_i is the resulting color (for each red, green and blue components), C_i is the color value from the original texture, L_A is the ambient light value, L_D is the calculated diffuse lighting component and L_S is the specular reflection component. The evident limitations of Equation (3) are that the original texture color cannot get brighter because RGB values are limited to [0, 1] range and that the specular reflection is added independently from the original texture's color.

The limitations described earlier are caused mainly by the usage of RGB color space. The authors of this work have previously proposed alternatives to use RGB color space for measuring color differences (or "similarities") [5], [6], which can be applied to illumination as well. This is done by using another color space that has different properties and does not manifest the limitations of the RGB color space.

2. Illumination approach using YIQ color space

In order to overcome the limitations of using RGB color space as was described earlier, a different color space can be used instead. In this work, YIQ color space was used to assist the illumination approach. This color space was introduced by the National Television System Committee (NTSC) [7], [8]. The color space is composed of the component *Y* called luma [8], [9], [10], which is proportional to the gamma-corrected luminance of a color and two other components *I* and *Q*, the combination of which describes both the hue and saturation of color. This particular color space was chosen because of the simple and fast conversion between RGB and YIQ color spaces.

The values of *Y*, *I* and *Q* can be directly calculated from the non-linear *r*', *g*' and *b*' components in the following way [8]:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.29889531 & 0.58662247 & 0.11448223 \\ 0.59597799 & -0.27417610 & -0.32180189 \\ 0.21147017 & -0.52261711 & 0.31114694 \end{bmatrix} \cdot \begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} \mathrm{r}' \\ \mathrm{g}' \\ \mathrm{b}' \end{bmatrix} = \begin{bmatrix} 1.00000000 & 0.95608445 & 0.62088850 \\ 1.00000000 & -0.27137664 & -0.64860590 \\ 1.00000000 & -1.10561724 & 1.70250126 \end{bmatrix} \cdot \begin{bmatrix} \mathrm{Y} \\ \mathrm{I} \\ \mathrm{Q} \end{bmatrix} \tag{5}$$

The value of luma *Y* is defined in range of [0, 1], *I* is defined in range of [-0.5957, 0.5957] and *Q* is defined in range of [-0.5226, 0.5226].

According to W. K. Pratt [8], the reasons for transmitting the YIQ components in television were that the *Y* signal alone could be used with existing monochrome receivers to display black and white video and that *I* and *Q* signals could have the transmission bandwidth limited without noticeable image degradation. In the context of this work, the transmission characteristics of YIQ color space are not relevant, but this color space has an important characteristic: the color is divided into main components – the "visible brightness" component (luma, Y) and the "color" component (hue and saturation coded into I and Q). This means that the illumination would affect only the *luma* (Y) but will leave the other components unchanged.

The illumination in YIQ color space therefore can be modified so that both diffuse lighting and specular reflection depend on the surface's texture:

$$Y_F = Y_T\left(L_A + L_D\right) + L_S \tag{6}$$

$$I_F = I_T \tag{7}$$

$$Q_F = Q_T \tag{8}$$

where Y_F, I_F and Q_F are the final color components specified in YIQ color space, Y_T, I_T and Q_T are the surface's texture color components also in YIQ color space, L_A is the ambient light value, L_D is the calculated diffuse lighting component, and L_S is the specular reflection component. As can be observed in Equations (6), (7) and (8), the color part of original surface's texture is preserved, while the *luma* is affected by all three components – ambient light, diffuse light and specular reflection. The cost of using the aforementioned equations is that the texture color must be either specified in YIQ color space or be converted on the fly. However, since the conversion between RGB and YIQ color spaces is a simple linear transformation, it can be done very quickly in shaders. The actual implementation of this approach is described in the next section.

3. Shader implementation

There are several ways of implementing the illumination using YIQ color space. The most efficient approach in terms of performance would be having the surface's texture pixels specified in YIQ color space, but doing so would be cumbersome when many textures are used. Since the linear transformation can be implemented very quickly in shaders, the conversion from RGB color space to YIQ color space and back can be made in the shader itself. The entire HLSL code that is fully compliant with NVIDIA FX Composer 2.5 [11] (a developer tool provided by one of the largest manufacturers of video cards) is described below.

```
float4x4 WorldInverseTranspose : WorldInverseTranspose <
string UIWidget="None"; >;
float4x4 WorldViewProjection : WorldViewProjection < string
UIWidget="None"; >;
float4x4 World : World < string UIWidget="None"; >;
float3   EyePos : CameraPosition < string UIWidget="None"; >;

float3 LightPos : Position <
  string UIName =  "Lamp 0 Position";
  string Object = "PointLight0";
  string Space  = "World"; > = {-0.5f, 2.0f, 1.25f};

float AmbientValue <
  string UIName   =  "Ambient Value";
  string UIWidget = "slider";
  float UIMin  = 0.0;
  float UIMax  = 1.0;
  float UIStep = 0.1; > = 0.1;

float SpecularPower : SpecularPower <
  string UIWidget = "slider";
  float UIMin = 0.0;
  float UIMax = 16.0;
  float UIStep = 1.0;
  string UIName =  "Specular Power"; > = 55.0;

texture ColorTexture : DIFFUSE <
  string ResourceName = "default_color.dds";
  string UIName       =  "Diffuse Texture";
  string ResourceType = "2D"; >;

sampler2D ColorSampler = sampler_state {
  Texture  = <ColorTexture>;
  FILTER   = MIN_MAG_MIP_LINEAR;
  AddressU = Wrap;
  AddressV = Wrap; };

struct OutputPhongVS {
  float4 Pos   : POSITION;
  float2 TexAt : TEXCOORD0;

float3 NormalWorld: TEXCOORD1;
  float3 PosWorld   : TEXCOORD2; };

OutputPhongVS PhongVertexShader(
 float3 InPos   : POSITION,
 float2 InpTex  : TEXCOORD0,
 float3 InNormal: NORMAL) {
  OutputPhongVS OutVS = (OutputPhongVS)0;

  OutVS.NormalWorld = mul(float4(InNormal, 0.0f),
   WorldInverseTranspose).xyz;
  OutVS.NormalWorld = normalize(OutVS.NormalWorld);

  OutVS.PosWorld = mul(float4(InPos, 1.0f), World).xyz;
  OutVS.Pos = mul(float4(InPos, 1.0f), WorldViewProjection);
  OutVS.TexAt = InpTex;

  return OutVS; }

float3 RGBtoYIQ(float3 InCol) {
  float3 OutCol;
  OutCol.y = 0.29889531f * InCol.r + 0.58662247f * InCol.g +
0.11448223f * InCol.b;
  OutCol.x = 0.59597799f * InCol.r + -0.27417610f * InCol.g + -
0.32180189f * InCol.b;
  OutCol.z = 0.21147017f * InCol.r + -0.52261711f * InCol.g +
0.31114694f * InCol.b;
  return OutCol; }

float3 YIQtoRGB(float3 InCol) {
  float3 OutCol;

  OutCol.r = InCol.y + 0.95608445f * InCol.x + 0.62088850f *
InCol.z;
  OutCol.g = InCol.y - 0.27137664f * InCol.x - 0.64860590f *
InCol.z;
  OutCol.b = InCol.y - 1.10561724f * InCol.x + 1.70250126f *
InCol.z;

  return saturate(OutCol); }

float4 LumaPhongDiffuse(float3 NormalWorld, float3 PosWorld,
float2 TexAt) {
  float3 ToEye = normalize(EyePos - PosWorld);

float3 LightVector = normalize(LightPos - PosWorld);
float3 LightVector = normalize(LightPos - PosWorld);

float3 Reflection = normalize(2.0f * NormalWorld *
dot(NormalWorld,
   LightVector) - LightVector);

  float Diffuse = saturate(saturate(dot(LightVector,
NormalWorld)) +
   AmbientValue);

  float Specular = pow(saturate(dot(Reflection, ToEye)),
SpecularPower);

  float3 TexCol = tex2D(ColorSampler, TexAt).rgb;

  float3 WorkCol = RGBtoYIQ(TexCol);

  WorkCol.y = WorkCol.y * Diffuse + Specular * 1.0 - 0.1;

  TexCol = YIQtoRGB(WorkCol);

  return float4(TexCol, 1.0); }

float4 LumaDiffusePS(OutputPhongVS InPS) : COLOR {
  return LumaPhongDiffuse(InPS.NormalWorld, InPS.PosWorld,
InPS.TexAt); }

technique PhongDiffuse < string Script = "Pass=p0;"; > {
  pass p0 < string Script = "Draw=geometry;"; > {
   VertexShader = compile vs_2_0 PhongVertexShader();

ZEnable = true;
   ZWriteEnable = true;
   ZFunc = LessEqual;
   AlphaBlendEnable = false;
   CullMode = None;
   PixelShader = compile ps_2_0 LumaDiffusePS();  }
```

Table 1. HLSL code for illumination using YIQ color space.

The code described in Table 1 according to NVIDIA FX Composer 2.5 uses approximately 40 instruction slots when compiled to assembly, which is relatively small considering the modern video cards. As it was mentioned in the beginning, the code can be converted to its GLSL equivalent with minimal effort and without sacrificing performance.

4. Visual appearance analysis

The proposed illumination approach described in this work was compared to the classical diffuse lighting and Phong specular reflection (from here referred to as Phong lighting model). The shader code for this model can be found in literature (see [1], [2], and [4]). A simple scene containing one point light source and a p-q torus knot [12] was used for rendering using the two aforementioned techniques. The resulting images are illustrated in Figure 1.

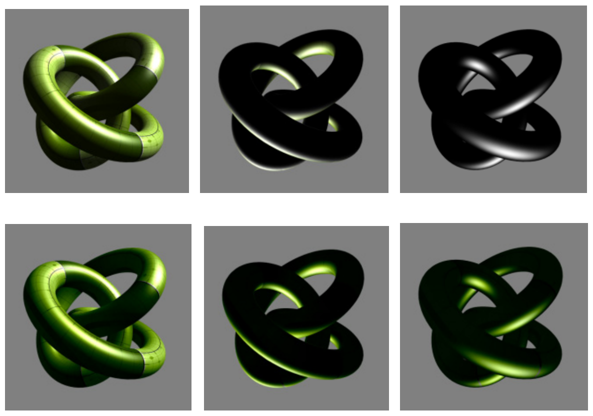

Figure 1. Torus Knot rendered with the light source at different angles and different rendering techniques. There are two groups with three images each – the first group uses Phong lighting while the second group uses the lighting approach described in this work. In each group of three images the first image the light source in front, the second image has the light positioned at the back and the third image shows specular reflection only.

As can be seen from the above figure, when the model is rendered with the light in front, the results are quite similar between Phong lighting model and YIQ-based lighting. It can be seen that the proposed model in this case is no worse than the classical one. However, when the light is on the back, the classical Phong lighting model generates grey shades as can be seen in the second image, while the YIQ-based lighting does not. In fact, when the specular reflection is rendered alone without diffuse component, in the classical Phong lighting model it is completely grey (no matter what the surface texture is), while in the YIQ-based lighting it depends on the surface and looks natural. It is evident from the sixth image that in YIQ-based lighting the specular reflection alone can be used without the diffuse component to simulate metallic surfaces.

5. Performance analysis

The real-time performance of the lighting approach described in this work was tested against the classical Phong lighting model on several computer configurations. In the performance test a more complex p-q torus knot was used, having 31529 vertices and 61440 triangles. The surface texture was an image of bricks with the size of 512x512. The resulting image is shown in Figure 2.

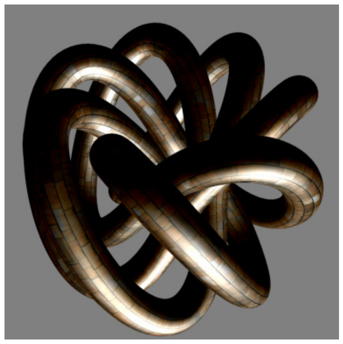

Figure 2. Torus knot rendered with a single light source using YIQ-based lighting model for performance testing.

The performance (in frames per second) along with the rendering size and multisampling is described in the table below. It is important to note that the final image size was 720x720 in most of the tests except on Eee PC 1000 HE, whose display was limited to 1024x600 resolution (in that case the size was 512x512).

Computer Configuration	**Size and Multisampling**	**Frame Rate (frames/second)**	
		Classical	**YIQ-based**
Intel Core 2 Quad Q660 2.4 Ghz, 4 Gb	720x720, No multisampling	*1240*	*1180*

RAM DDR2 800 Mhz (Cas Latency 5), Nvidia GeForce GTS 250	720x720, 8x multisampling	*389*	*385*
Intel Core 2 Duo E6300 1.86 Ghz, 4 Gb RAM DDR2 800 Mhz (Cas Latency 5), ATI Radeon X1950 GT	720x720, No multisampling	*672*	*453*
	720x720, 6x multisampling	*430*	*341*
Intel Atom 1.66 Ghz N280, 2 Gb RAM DDR2 533 Mhz, Intel GMA 950	512x512, No multisampling	*44*	*35*

Table 2. The performance of two lighting approaches on different computer configurations.

The performance results shown in Table 2 indicate that YIQ-based lighting is slower in all instances than the classical Phong lighting model, which is expected as the conversion from RGB to YIQ and back takes place. However, in the latest video card Nvidia GeForce GTS 250, the difference between the two is around 1% using 8x multisampling and around 5% without multisampling. The difference is larger for ATI Radeon X1950 GT and for Intel GMA 950, where it is approximately 20% (and can be as high as 32%, when tested on ATI video card with multisampling disabled). From the results, we conclude that YIQ-based lighting can be used in real time even in the low-end portable and cheap computer such as Eee PC 1000 HE.

6. Conclusions and future work

In this work an alternative diffuse lighting and specular reflection approach was proposed that uses YIQ color space for illumination instead of RGB color space used by most of the classical lighting models. The complete HLSL shader code was described to be ready to use in commercial applications. The advantage of the proposed approach is that it eliminates the restrictions derived from the use of RGB color space in the classical lighting models where the diffuse color cannot become brighter and where the specular reflection adds certain amount of gray regardless the source surface's texture.

The performance of the described approach was, in the worst case, 30% slower than the classical lighting model, and in the best case, it was around 1% slower in the computers used in the experiment. This indicates that there is room for improvement in terms of performance: first, the original texture can be specified in YIQ color space to reduce the number of conversions by half. Assuming that the color part (I and Q components) of the original color is not changed, the conversion from RGB to YIQ color space and back can be optimized to reduce the number of mathematical operations involved.

Although working directly in YIQ color space may not be possible for the entire rendering pipeline, because of programmable nature of today's GPUs, representing the texture in YIQ color space and working directly in this color space with shaders will provide the benefits of photorealism without compromising the performance. The current limitation is that the resulting color returned by the pixel shader still needs to be specified in the RGB color space, although this may be alleviated in the future versions of DirectX and OpenGL, especially considering the trend of general-purpose GPU programming.

In applications with significant 3D scene complexity such as video games, the visual artifacts of classical illumination approaches may not be easily apparent, while the performance plays a major role in decision making. However, in other industries where photo-realism is of higher importance, the proposed alternative may provide a significant improvement and produce real-looking scenes.

It is important to note that the approach described in this work is not only limited to diffuse lighting and specular reflections. For instance, it can be used to improve the overall quality of the Minneart lighting model (see [1]) by making the color brighter in some cases instead of making it darker each time. Many other lighting models such as Oren-Nayar or Cook-Torrance can benefit from using the described approach as well. Further work with these lighting models and the lighting approach proposed in this work is recommended.

There are other color spaces that have a “brightness” component in them such as YUV, CIELAB and CIELUV [13]. Although the last two color spaces might not be good choices for real-time processing (according to Poynton [9]), in applications where realism is required at the expense of performance, using these color spaces could be beneficial. There are other color spaces that have a brightness component modeled closely to the human perception, such as Guth’s ATD95 [14] and DIN99 [15], that can be used for rendering highly detailed 3D scenes with greater illumination accuracy based on the approach described in this work. Further research is required to determine whether these color spaces can be used for illumination in shaders.

References

[1] K. Dempski and V. Emmanuel, “Advanced Lighting and Materials with Shaders,” Boston MA, Jones & Bartlett Publishers, 2004.

[2] F. D. Luna, “Introduction to 3D Game Programming with Direct X 9.0c: A Shader Approach, 1st edition,” Boston MA, Jones & Bartlett Publishers, 2006.

[3] B. T. Phong, “Illumination for computer generated pictures,” *Communications of the ACM*, vol. 18, no. 6, pp. 311-317, 1975.

[4] E. Lengyel, “Mathematics for 3D Game Programming and Computer Graphics,” 2nd edition, Boston MA, Charles River Media, 2003.

[5] Y. Kotsarenko and F. Ramos, “*Perceptual Color Similarity Measurement*”, Master Thesis, Instituto Tecnológico y de Estudios Superiores de Monterrey, Morelos, México, 2006.

[6] Y. Kotsarenko and F. Ramos, “*Measuring perceived color difference using YIQ NTSC transmission color space in mobile applications*,” Programacion Matematica y Software, vol. 2, no. 2, pp. 28-43, Morelos, Mexico, 2010.

[7] D. Hearn and P. M. Baker, “Computer Graphics, C Version”, New Jersey, Prentice Hall, 1996.

[8] W. K. Pratt, “Digital Image Processing”, 3rd edition, New Jersey, Wiley-Interscience, 2001.

[9] C. Poynton, “Digital Video and HDTV Algorithms and Interfaces”, San Francisco, Morgan Kaufmann, 2003.

[10] C. Poynton, “*Frequenty-Asked Questions about Color*,” Charles Poynton Web Site available from: http://www.poynton.com/ColorFAQ.html (November 28, 2006, accessed on October 22, 2008).

[11] Nvidia Corp., “FX Composer 2.5”, NVIDIA FX Composer 2.5 GPU Shader Authoring Environment, available from http://developer.nvidia.com/content/fx-composer (accessed on August, 2012)

[12] C. C. Adams, “The Knot Book: An Elementary Introduction to the Mathematical Theory of Knots”, New York, W.H. Freeman & Company, 1994.

[13] J. Schanda, “Colorimetry: Understanding the CIE system”, New Jersey, Wiley-Interscience, 2007.

[14] L. S. Guth, “*Further Applications of the ATD Model for Color Vision*,” Proceedings SPIE (The International Society for Optical Engineering), vol. 2414, 1995, pp. 12-26.

[15] H. Büring, “*Eigenschaften des Farbenraumes nach DIN 6176 (DIN99-Formel) und seine Bedeutung für die industrielle Anwendung*,” 8 Workshop Farbbildverarbeitung der German Color Group, 2002, pp. 11-17.