Journal of Theoretical and Applied Information Technology
15th September 2017. Vol.95. No.17

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

RENDERING SPEEDS OF DYNAMIC AND STATIC OBJECTS WITH TANGENT SPACE NORMAL MAPPING ON 3D GAMES

YOUNGSIK KIM
Dept. of Game and Multimedia Engineering, Korea Polytechnic University, Republic of Korea
E-mail: kys@kpu.ac.kr

ABSTRACT

In 3D games, bump mapping is an efficient way to provide high-resolution bumpy lighting features in textures using only low-resolution meshes during runtime. This paper developed two 3D games based on Unity3D and Direct3D using normal mapping, which is a typical one among bump mapping methods. In particular, dynamic objects in 3D games require tangent space normal mapping, which requires much computation per vertex. The performance of dynamic and static objects with or without normal mapping in 3D games is analyzed using various screen resolutions as well as eight simulation models in terms of the rendering speed like frames per second (FPS). The rendering speeds of the models Gu and Gd on Unity3D and Direct3D based games, where using normal mapping to all objects among the eight simulation models, can be improved by up to 79.7% and 19.2%, respectively, compared with the models Bu and Bd without normal mapping. The tangent space normal mapping on both dynamic and static objects in 3D games has a large effect on rendering speed.

Keywords: *Bump Mapping, Tangent Space Normal Mapping, Rendering Speed, Frames Per Second (FPS), Direct3D, Unity3D*

1. INTRODUCTION

Bump mapping has been widely used in the latest 3D computer games to represent bumpy surfaces in real time [1,2,3,4,5]. To provide the high-resolution bumpy features with low-resolution meshes, bump mapping has been used with texture mapping. Blinn invented bump mapping in 1978 [5]. Bump mapping is a texture-based rendering approach to provide wrinkle effects by applying perturbed patterns on low-resolution macro meshes according to lights. By representing such perturbed surfaces in texture maps, bump mapping provides a bumpy lighting features without both the high-resolution meshes and the complex true geometric perturbations using a height field function as shown in Figure 1.

One of the typical bump mapping techniques is normal mapping, which uses normal information stored in a texture. Normal mapping has begun to be implemented in the game since the early 2000s when pixel shaders appeared. The most important tasks of pixel shaders are lighting and texture processing. Normal mapping, which performs lighting calculations using a special texture with perturbed normals stored, was not possible in real-time before pixel shaders appeared [3].

In [6], bump mapping is applied to cartoon rendering using silhouette detection method in image space. In [2,7], bump mapping hardware approaches have been proposed for high-end 3D graphics hardware. Bump mapping hardware in [2,7] supports per-fragment lighting operations. In [2], the costly per-pixel steps are eliminated by reconstructing a tangent space and perturbing the interpolated normal vector. In [8,9], the VISA+ hardware architecture is proposed as a new generation of graphics accelerators designed primarily to render bump-, texture-, environment- and environment-bump-mapped polygons. Visa+ bump-mapping in [8,9] determines the respective lighting diffuse and Phong specular lighting contributions, which uses vector offset maps to encode the bump map perturbations and uses two cube maps. In [10], Gouraud bump mapping is developed as another approach. In [11], a new real-time terrain rendering approach combines hardware tessellation and parallax mapping [12].

Unlike static objects in 3D games, normal mapping must be computed per vertex when applying normal mappings to dynamic objects. That

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

is, we need to define a tangent space with the surface normals along the z axis for each vertex. Therefore, tangent space normal mapping applied to dynamic objects requires more computation than static objects. Parallax mapping [12] is known as an advanced method compared with normal mapping. This paper applies tangent space normal mapping to both static and dynamic objects on two kinds of proprietary games like Unity3D and Direct3D based game. The rendering speeds of two games with or without normal mapping are compared where changing screen resolutions as well as simulation models in terms of frame per seconds (FPS).

The composition of this paper is as follows. In Section 2, three bump mapping algorithms are introduced. Section 3 describes the tangent space normal mapping. Section 4 introduces two proprietary 3D games based on Unity3D and Direct3D that use tangent space normal mapping. Section 5 compares and analyzes the rendering speed of various resolutions with respect to the normal mapping application of static and dynamic objects in a two-game 3D game environment. Finally, Section 6 concludes this paper.

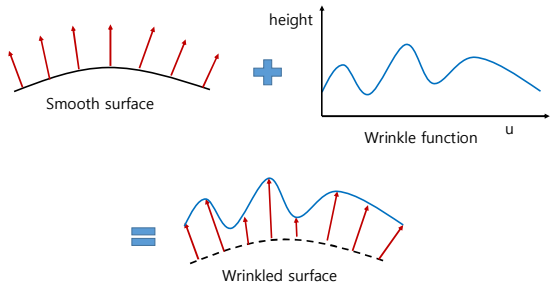


Figure 1. A surface perturbed by a height field makes a bumpy surface [5]

2. BUMP MAPPING METHODS

Bump mapping is a technique that uses low-resolution meshes to handle bumpy surfaces in real time, while high-resolution meshes are stored in normal map textures for runtime use. Bump mapping uses a special texture called a height map, which is simply generated during preprocessing. The height map expresses the high resolution model as a height value from a flat bottom surface. The height map was visualized by interpreting the height value as a grayscale color. For example, if the height is in the range [0,255], the lowest value 0 is expressed in black, and the highest value 255 is expressed in white.

The bump mapping using the height map is largely divided into two tasks. First, calculate the perturbed normals of the bumpy surface using the height map. And the second performs lighting calculations on a fragment basis using a perturbed normal. Three methods for handling bump mapping using a height map are 1) normal mapping, 2) parallax mapping, and 3) displacement mapping. 1) Normal mapping is the oldest and most used in the game. The effect of normal mapping is excellent, but since it is a kind of gimmick, its limitations are also clear. First, in the preprocessing step, the height map is used to calculate the perturbed normal of the bumpy surface and store it in a special texture called the normal map normal map. The irregular normal stored in the normal map is used as the surface normal of the macro structure at runtime. The macro structure is intact, but it is rendered using normals in the normal map to simulate the bumpy surface by lighting. Normal mapping is implemented as a pixel shader.

Various pixel shader algorithms have been developed to replace normal mapping. A typical example is 2) parallax mapping [3] [12]. Parallax mapping performs a simple ray tracing algorithm on the height map at runtime. Unlike normal mapping, it is possible to express that the bump is actually hidden. Parallax mapping is implemented as a pixel shader. That is, one ray is emitted per pixel to calculate the point at which this ray hits the height map, and the color of that point is determined. As GPU performance continues to evolve, ray tracing, which was only implemented in the non- real time domain, can be implemented in real time (although in a simplified form). 3) Displacement mapping does tessellation operations on the macro structure and then actually moves the vertices using a height map [3]. It is more attractive because it supports tessellation in shader model 5. Displacement mapping is implemented by the tessellator and the domain shader.

3. TANGENT SPACE NORMAL MAPPING ALGORITHM

As shown in Figure 2, the surface normal, n_p, for a point on P with texture coordinate (u,v) is defined in the object modeling step. It is assumed that 'perturbed n_p' is stored in the normal map. (n_p = unit vector in the z-axis direction of world space (0,0,1)). Texturing is to spread the texture on the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

object surface. The $n(n_p, v_p)$ taken from the normal map replaces n_p. Therefore, the normal of the normal map is not 'perturbed n_p' but 'perturbed surface normal'. Consider a tangent space with a z-axis surface normal at each point on the object surface. Each normal of the normal map is defined in the tangent space of the surface point to which the normal is applied, not world space. In this sense, normal is called tangent space normal map.

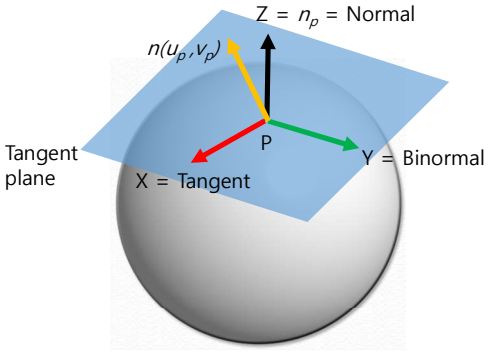


Figure 2. Tangent Space Normal Map [3].

The tangent space basis is defined as {T, B, N}. The vertex normal, N, is defined per vertex in the modeling step. Tangent, T, and Binormal, B, need calculation. The surface normal, N, for a point on P with texture coordinate (u,v) is defined as equation (1) [1]. Usually Tangent, T, is taken from the partial derivative of the u texture coordinate, and Binormal, B, is taken from the partial derivative of the v texture coordinate with respect to a point P(u,v) in the world space x, y, z as defined as equations (2) and (3) [4]. Figure 3 shows pseudo shader programs for tangent space normal mapping [3].

$$N(u,v) = \frac{dP(u,v)}{du} \times \frac{dP(u,v)}{dv} \quad (1)$$

$$T \propto \frac{du}{dP} = \left(\frac{du}{dx}, \frac{du}{dy}, \frac{du}{dz} \right) \quad (2)$$

$$B \propto \frac{dv}{dP} = \left(\frac{dv}{dx}, \frac{dv}{dy}, \frac{dv}{dz} \right) \quad (3)$$

4. UNITY3D AND DIRECT3D BASED GAMES WITH TANGENT SPACE NORMAL MAPPING

Pseudo Program: Tangent Space Normal Mapping

```
void VS_Tangent_Space_Normal
( float4 Pos : POSITION,
  float3 Normal : NORMAL,
  float3 Tangent : TEXCOORD0,
  float2 Tex : TEXCOORD1, // normal map
  out float4 oPos : POSITION,
  out float2 oTex : TEXCOORD0,
  out float3 Light : TEXCOORD1,
  uniform float3 LightPos,
  uniform float4x4 ViewProj)
{
  oPos = mul(ViewProj, Pos);
  oTex = Tex;
  light = LightPos – Pos.xyz;
  float3 Binormal = cross(Normal, Tangent);
  Rotation=float3(Tangent,Binormal,Normal);
  Light = mul(Rotation, light);
}

void PS_Tangent_Space_Normal
(
  float2 oTex : TEXCOORD0,
  float3 Light = TEXCOORD1,
  out float4 Color : COLOR,
  uniform sampler2D shadowMap)
{
  float3 LightDir = normalize(Light);
  float3 Normal = tex2D(NormalMap, oTex).xyz;
  Normal = normalize(Normal*2.0 – 1.0);
  Color = dot(Normal, LightDir);
}
```

Figure 3. Pseudo Shader Program for Tangent Space Normal Mapping [3].

This paper developed two 3D games like Unity3D and Direct3D based games for performance evaluation. The normal mapping rendering effects were applied to the static and dynamic objects of 3D games of own production, and the performance was analyzed at various screen resolutions.

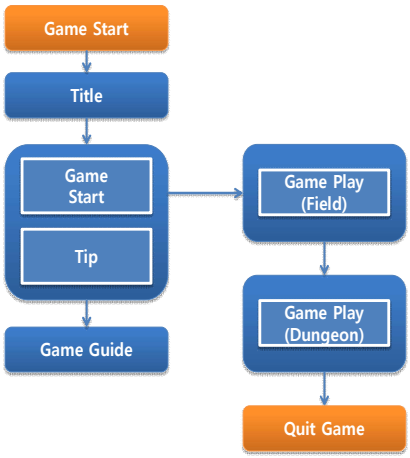
The Unity3D based game used in the first experiment is a Massive Multiplayer Online Role Playing Game (MMORPG) using Unity3D game engine. This Unity3D based game uses a free camera viewpoint and needs to defeat field and dungeon monsters. It is a game to nurture the character's skill and power and to suppress the oak king boss monsters on the top stage of the dungeon. Second, the Direct3D based game used in the experiment is an online action Role Playing Game (RPG). This Direct3D based game uses the third person's back view to kill the monsters in the map with the party member and finally needs to kill the boss to clear the game.

Figure 4 (a) and (b) illustrate the operation flows of our own Unity3D and Direct3D based

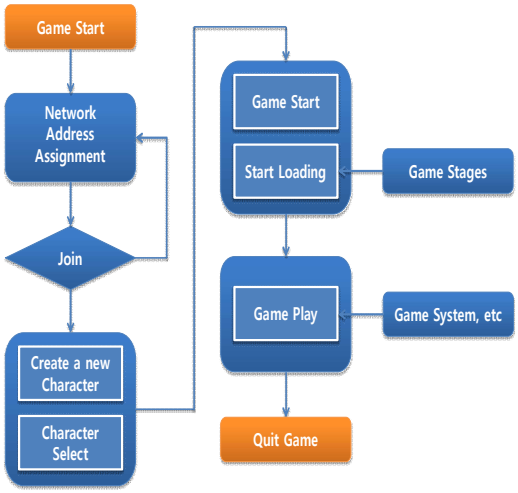
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

games, respectively. Especially, since Direct3D based game is a network based MMORPG, the network address assignment and join steps are included in the game control flow as shown in Figure 4 (b).

Table 1 (a) and (b) show the number of vertices and triangles of Dynamic and Static Objects used in our own Unity3D and Direct3D based games, respectively. In Unity3D based game, dynamic objects are Monster and Player, and static objects are Bonfire, Log, Fence1, and Fence2 as in Table 1 (a). In Direct3D based game, dynamic objects are Female Player, Male Player, Lizard, and Hyena and static objects are Ruin1, Ruin2, Ruin3, Ruin4, and Barrel as in Table 1 (b). All objects in Unity3D and Direct3D based games are shown in various screen shots of Figure 5.



(a) Unity3D based Game



(b) Direct3D based Game

Figure 4. Game Control Flow.

Table 1. Dynamic and Static Objects in 3D Games

(a) Unity3D based Game

	Objects	No. of Vertices	No. of Triangles
Dynamic Objects	Monster	692	1218
	Player	1426	2478
Static Objects	Bonfire	112	144
	Log	368	398
	Fence1	86	112
	Fence2	504	546

(b) Direct3D based Game

	Objects	No. of Vertices	No. of Triangles
Dynamic Objects	Player (Female)	4939	5941
	Player (Male)	4942	4954
	Lizard	4542	5984
	Hyena	3450	3996
Static Objects	Ruin1	7422	5560
	Ruin2	6024	4641
	Ruin3	3660	2633
	Ruin4	2942	2127
	Barrel	930	780

5. PERFORMANCE EVALUATION

In this Section, The performance of dynamic and static objects of 3D games with tangent space normal mapping was analyzed by measuring the rendering speed (FPS: frame per second). First, for the Unity3D based game, the computer used in the experiment is the processor: Inter (R) Core (TM) i5-4670 CPU @ 3.40GHz 3.40GHz, memory: 8.00GB, 64bit operating system, graphics card: Geforce Nvidia GTX 660. Also, for the performance analysis of Unity3D based game, the screen resolution of the device was changed to 640x480, 800x600, 1024x768, 1152x864, and 1280x960. Second, for the Direct3D based game, the computer used in the experiment is the processor: Inter (R) Core i5 4210M @ 2.60GHz 2.60GHz, memory: 4.00GB, 64bit operating system, graphics card: Nvidia Geforce 940m. Also, for the performance analysis of Direct3D based Game, the screen resolution of the device was measured while changing to 640x360, 1024x576, 1366x768, 1600x900, and 1920x1080.

Table 2 shows eight simulation models Au / Ad, Bu / Bd, Cu / Cd, Du / Dd, Eu, / Ed, Fu / Fd,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 2. Eight Simulation Models in Unity3D and Direct3D based Games

(a) Unity 3D based Game

Simulation Model	Au	Bu	Cu	Du	Eu	Fu	Gu	Hu
Number of Dynamic Objects	2	4	2	4	2	4	2	4
Normal Mapping on Dynamic Objects	Yes				No			
Number of Static Objects	14							
Normal Mapping on Static Objects	Yes		No		Yes		Yes	

(b) Direct3D based Game

Simulation Model	Ad	Bd	Cd	Dd	Ed	Fd	Gd	Hd
Number of Dynamic Objects	16	32	16	32	16	32	16	32
Normal Mapping on Dynamic Objects	Yes				No			
Number of Static Objects	20							
Normal Mapping on Static Objects	Yes		No		Yes		Yes	

Table 3. Rendering Speeds (FPS) of Eight Models in Various Screen Resolutions

(a) *Unity3D based Game*

Models / Screen Resolutions	Au	Bu	Cu	Du	Eu	Fu	Gu	Hu	AVR	STDEV
640x480	67.5	58.3	94.1	81.7	71.2	69.1	114.9	93.7	81.3	18.6
800x600	63.1	54.4	82.2	72.1	63.7	61.1	98.2	81.8	72.1	14.5
1024x768	52.8	47.8	75.7	64.7	59.4	57.5	77.7	68.1	63.0	10.6
1152x864	41.8	35.6	62.4	57.6	48.1	43.3	67.6	59.2	52.0	11.3
1280x960	34.2	32.1	59.1	39.1	43.2	40.8	51.7	44.7	43.1	8.9
Average(AVR)	51.9	45.6	74.7	63.0	57.1	54.4	82.0	69.5	62.3	12.4
Standard Deviation(STDEV)	14.0	11.5	14.4	16.1	11.4	12.0	24.9	19.1	15.3	4.6

(b) Direct3D based Game

Models / Screen Resolutions	Ad	Bd	Cd	Dd	Ed	Fd	Gd	Hd	AVR	STDEV
640x480	635.7	585.7	698.0	629.0	641.1	594.4	700.3	643.6	641.0	41.7
800x600	490.5	448.0	536.4	488.3	503.0	467.9	540.5	507.5	497.8	31.5
1024x768	324.7	300.2	353.5	328.7	334.2	308.9	361.2	335.0	330.8	20.4
1152x864	268.5	249.3	287.8	265.4	271.1	254.7	293.9	272.7	270.4	15.0
1280x960	226.4	210.0	240.1	223.8	229.0	215.3	241.1	229.7	226.9	10.8
Average(AVR)	389.2	358.6	423.2	387.0	395.7	368.2	427.4	397.7	393.4	23.8
Standard Deviation(STDEV)	170.5	155.7	190.4	168.6	172.4	158.8	189.9	173.4	172.4	12.6

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

(a) Model Au for Unity3D based Game

(b) Model Bu for Unity3D based Game

(c) Model Cu for Unity3D based Game

(d) Model Du for Unity3D based Game

(e) Model Ed for Direct3D based Game

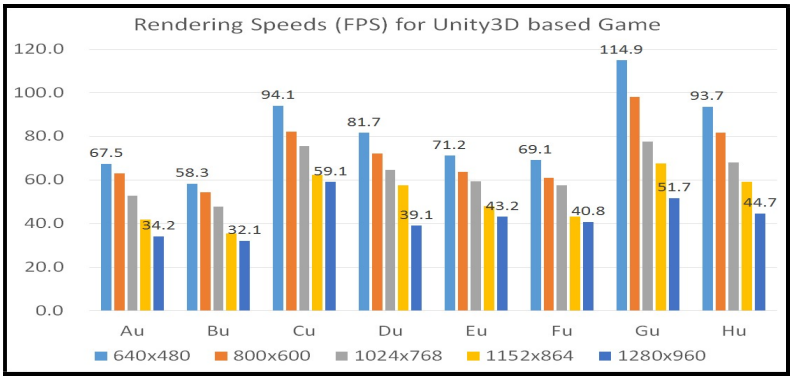
(f) Model Fd for Unity3D based Game

(g) Model Gd for Direct3D based Game

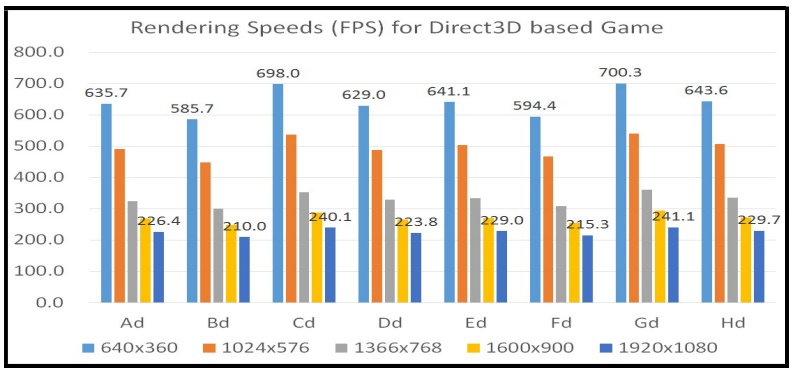
(h) Model Hd for Direct3D based Game

Figure5. Screen Shots of Various Models for Unity3D and Direct3D based Games.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



(a) Unity3D based Game



(b) Direct3D based Game

Figure 6. Rendering Speeds (FPS) for Unity3D and Direct3D based Games.

Gu / Gd and Hu / Hd, respectively, with varying numbers of dynamic and static objects for each of the two games (Unity3D and Direct3D based games). Figure 5 (a) - (g) illustrate the screenshots of the eight simulation models in Table 3 in turn.

Table 3 and Figure 6 compare the rendering speeds of the eight simulation models. As shown in Table 4 and Figure 6, the lower the screen resolution, the better the rendering speed in both games in all eight models. In Unity3D based games, the average rendering speed of 640x480 resolution was 47.0% higher than that of 1280x960 resolution. Standard deviations were 18.6 and 8.9 at 640x480 resolution and 1280x960 resolution, respectively, with a standard deviation of up to 52.1% at lower resolutions. In Direct3D based games, the average rendering speed was 64.6% higher than the 1920x1080 resolution and 640x360 resolution. Standard deviations were 41.7 and 10.8 at 640x360 resolution and 1920x1080 resolution, respectively, with a standard deviation of up to 74.1% at lower resolutions.

The average rendering speeds of model Gu and model Gd were 82.0 FPS and 427.4 FPS, respectively, according to various simulation models. The reason that the rendering speed of models Gu and Gd is higher than other models is because normal mapping is not applied to both dynamic objects and static objects, and the number of dynamic objects is also small. The average rendering speeds of the models Gu and Gd were improved by 79.7% and 19.2%, respectively, compared to Bu and Bd models which applied normal mapping to both dynamic objects and static objects.

The average rendering speed of A: B, C: D, E: F, and G: H models according to the number of dynamic objects was 13.8% and 8.2% respectively on Unity3D and Direct3D based games, respectively. The average rendering speeds of A: E, B: F, C: G, and D: H models with or without tangent space normal mapping applied to dynamic objects were 12.3% and 2.0% respectively on Unity3D and Direct3D based games, respectively. The average rendering speed of A: C, B: D, E: G, and F: H models with and without tangent space normal mapping applied to static Objects was 38.4% and

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

8.2%, respectively, in Unity3D and Direct3D based games, respectively.

The tangent space normal mapping does not represent a realistic bump representation compared to parallax mapping or displacement mapping. However, normal mapping is the most cost effective method of representing bumps.

6. CONCLUSION

In 3D games, normal mapping is a typical bump mapping method to use low-resolution meshes and store high-resolution features of bumpy surfaces in textures for runtime use. In this paper, we apply tangent space normal mapping to static and dynamic objects using two Unity3D and Direct3D based games, and analyze performance at various screen resolutions. Performance analysis showed that the average rendering speed was 47.0% and 64.6% higher in the two games on the low resolution screen than on the high resolution screen. And the rendering speed of the model Gu / Gd, which applied normal mapping to all objects among the eight simulation models, was improved by 79.7% and 19.2%, respectively, compared with the model Bu / Bd without normal mapping. The normal mapping has the greatest effect on rendering speed.

The implication is that the number of dynamic objects and static objects and the rendering speed according to the screen resolution should be considered whether or not tangent space normal mapping is applied in Unity3D or Direct3D based 3D games. In the future, research can be expanded by applying parallax mapping and displacement mapping.

ACKNOWLEDGEMENT

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. 2016-0-00204, Development of mobile GPU hardware for photo-realistic real time virtual reality).

REFERENCES:

[1] Mark J. Kilgrd, “A Practical and Robust Bump-mapping Technique for Today’s GPUs”, *Game Developers Conference*, 2000.

[2] Mark Peercy, John Airey, and Brian Cabral, “Efficient Bump Mapping Hardware”, *Computer Graphics (Proc. Siggraph ‘97)*, 1997, pp. 303~306.

[3] JungHyun Han, *3D Graphics for Game Programming*, CRC Press, 2011.

[4] Wofgang Engel, *Shader X5 Advanced Rendering Techniques, 2.6 Normal Mapping without Precomputed Tangents*, Charles River Media, 2007.

[5] James Blinn, “Simulation of Wrinkled Surfaces,’’ *Computer Graphics (Proc. Siggraph ’78)*, pp. 286-292, Also in Tutorial: Computer Graphics: Image Synthesis, 1978, pp. 307-313.

[6] Won-Kyu Lee, Sun-Young Lee, and In-Kwon Lee, “Cartoon Rendering for Bump Mapped Object”, *Journal of the Korea Computer Graphics Society* 12(1), pp. 15~18.

[7] Michael Cosman, Robert Grange, 1996, “CIG Scene Realism: The World Tomorrow,” *Proc. Of I/ITSEC on CD-ROM*, 2006, pp. 628.

[8] K. Bennebroek, I. Ernst, H. Rüsseler, O. Wittig, “Design Principles of Hardware-based Phong Shading and Bump Mapping,” *11th Eurographics Workshop on Graphics Hardware, Poitiers, France*, 1996, pp. 3-9.

[9] I. Ernst, D. Jackèl, H. Rüsseler, O. Wittig, “Hardware Supported Bump Mapping: A Step towards Higher Quality Real-Time Rendering,” *10th Eurographics Workshop on Graphics Hardware*, Maastricht, Netherland, 1995, pp. 63-70.

[10] I. Ernst, H. Rüsseler, H. Schulz, O. Wittig, “Gouraud Bump Mapping,” *Proc. 1998 Eurographics/Siggraph Workshop on Graphics Hardware*, Lisbon, Portugal, 1998, pp. 47-53.

[11] González, Cesar, Mariano Pérez, and Juan M. Orduña. "A hybrid GPU technique for real-time terrain visualization." *Proceedings of Computational and Mathematical Methods in Science and Engineering*, 2016.

[12] Wofgang Engel, *Shader X5 Advanced Rendering Techniques, 2.3 Practical Parallax Occlusion Mapping with Approximate Soft Shadows for Detailed Surface Rendering*, Charles River Media, 2007.