EGVE Symposium (2008)
B. Mohler and R. van Liere (Editors)

Posters

Improvement Rendering of Web3D Using the Shading Language

Masahiro Sakai[1,2] Noriyuki Ichijo[1] Yoshinori Dobashi[2] and Tshuyoshi Yamamoto[2]

[1]Micronet Co.,Ltd., Hokkaido, Japan
[2]Hokkaido University, Hokkaido, Japan

Abstract

Many Web3D systems for managing 3DCG in the contents of HTML pages use fixed shaders for graphics API for real-time 3DCG rendering, so it is difficult to simulate the global illumination. We propose a solution to the problem of global illumination that uses a Web3D environment map in our proprietary Web3D format, "3DX" and the GPU shading language.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation – Bitimap and Frame buffer operations I.3.7 [Computer Graphics]: Three–Dimentional Graphics and Realism – Color, Shading, Shadowing and Texture

1. Introduction

The three-dimensional spaces on the web has progressed since the 1990s. Such one technology domain is called Web3D. The first technology of this kind was VRML (Virtual Reality Modeling Language) in 1995, followed by a number of Web3D formats introduced and used on the Internet since 2000. Since the purpose of Web3D is real-time rendering, many systems used fixed shaders that depend on the graphics API of the operation system to display shapes. As a result, processes to display the effects of shading and illumination were restricted by graphics API and the representation of 3D images was dependent upon it. Also, not many attempts have been made to simulate dynamic global illumination in Web3D. This paper focuses especially on this problem and confirms the effect of improving the representation of Web3D by applying the shading language, which is currently appraised in a wide variety of computer graphics research fields, to our Web3D format, "3DX."

2. Related studies

Web3D technologies progressed dramatically once VRML was introduced. This section summarizes related studies by several topics. Web3D content consists mainly of two processes. The first is the content creation process. The second is the display process mentioned in the previous section.

2.1. Studies for improvements in the creation process

Representation became redundant when 3D graphics were described by polygon meshes, and this was a hindrance to the spread of the system in the 1990s as it caused inconvenience on the narrow-band Internet. While it was possible to display graphics smoothly by using a larger number of polygon meshes, long transfer time and high-performance graphics hardware were required [WYHC01]. Wakita et al. presented a lattice mesh creation method for creating freeform surfaces from initial simple polygons, and added a framework to handle freeform surfaces by extending VRML97 [WMYTC00].

2.2. Studies for improvements in the display process

For solution of the display process problem, the application of programmable shaders using graphic hardware resources has been presented as a direction for X3D, which is an upgrade from VRML. In the display process of conventional Web3D, the luminance distribution on the object surface was calculated in advance and was mapped on the surface as texture to create a 3D image representing the shading effect. Instead of these, the use of the shading function of graphics hardware for Web3D is currently being considered [GNMdC04].

3. Structure of the 3DX Web3D format

This section describes the 3DX Web3D format presented by authors used in the method proposed here.

3.1. File format

The file structure of 3DX can be divided into the model part, the file that puts animation data and link information with other Web sites together (.3dx) and the texture part used in the model part. While the model part of a 3DX file consists of form tags called "chunks" which represent geometry, it also includes animation-related information and texture-based animations.

3.2. Rendering method

Rendering was achieved by sending information on geometry, illuminant, view position (camera position) and texture obtained from the above-mentioned chunks to OpenGL 1.1, 1.3, 1.5 or other standard graphic libraries. Extensions by nVidia CgFX and Open GL 2.0 were also added in this study.

3.3. Browser plug-in

Different versions of the browser plug-in have been created and provided by implementation for Internet browsers to be compatible on various operating systems. The browser plug-in also has a function to interpret JavaScript to realize the interaction with HTML as mentioned below. This enables users to manipulate a 3D space on an Internet browser in an interactive manner.

3.4. Interaction with HTML

The browser plug-in is called up from HTML by describing the tag and rendering the display screen together with other information contained in the HTML page. It is also possible to embed JavaScript in HTML to realize interactivity with a 3D space.

4. Irradiance environment map used for graphics hardware

This paper presents a concrete method for improving the representation of Web3D by using the shading language of graphics hardware. Among the methods for achieving global illumination effects by dynamic illumination and objects, the method for calculating an irradiance environment map has been presented by King [Kin05]. Based on these methods, the authors implemented the method below.

4.1. Calculation of diffuse light intensity

The diffuse light intensity L from the surface where the normal vector is N can be expressed by the equation below on the assumption that the incident light intensity from direction ω is $L(\omega)$ and the solid angle in the direction is $d\omega$.

$$L = \int L(\omega) \cdot max(N \cdot \omega, 0) \cdot d\omega \tag{1}$$

Where, $L(\omega)$ is expanded in spherical harmony as shown below.

$$L(\omega) = \sum_{lm} L_{lm} \cdot Y_{lm}(\omega) \tag{2}$$

$$L_{lm} = \int L(\omega) \cdot Y_{lm}(\omega) \cdot d\omega \tag{3}$$

Where, $\sum_{lm}$ represents the total for all the bases of spherical harmonics. Next, the incident light from the entire globe surrounding the surface is substituted by an environment map.

$$L = \sum_{i} L(i) \cdot max(N \cdot \omega(i), 0) \cdot d\omega(i) \tag{4}$$

$$L(i) = \sum_{lm} L_{lm} \cdot Y_{lm}(i) \tag{5}$$

$$L_{lm} = \sum_{i} L(i) \cdot Y_{lm}(i) \cdot d\omega(i) \tag{6}$$

Where,$\sum_{i}$ represents the total for all the texels of the environment map, $L(i)$ is the intensity of texel no. i on the environment map, $\omega(i)$ is the direction and $d\omega(i)$ is its solid angle. The equations below can be obtained by substituting Eq. 5 for Eq. 4.

$$L = \sum_{lm} L_{lm} (\sum_{i} Y_{lm}(i) \cdot max(N \cdot \omega(i), 0) d\omega(i)) \tag{7}$$

$$L = \sum_{lm} L_{lm} \cdot A_{lm} \tag{8}$$

$$A_{lm} = \sum_{i} Y_{lm}(i) \cdot max(N \cdot \omega(i), 0) d\omega(i) \tag{9}$$

Because A_{lm} is an amount that depends only on the normal vector of the surface, it is calculated in advance for all the directions of N and set on the global map. L_{lm} is calculated in accordance with Eq. (6) by dynamically creating a dual-paraboloid environment map whenever there is a change of scene, and the result is stored as texture in the same way as the immediately aforementioned method. As represented by $Y_{lm}(i) \cdot d\omega(i)$ of Eq. 6, all the texels on both sides of the dual-paraboloid environment map are calculated in advance and stored as texture to improve the processing speed. Lastly, certain pixels are sampled from these two textures and their sum is used to calculate the diffuse light intensity.

5. Implementation

The irradiance environment map described in the previous section was implemented by the method below.

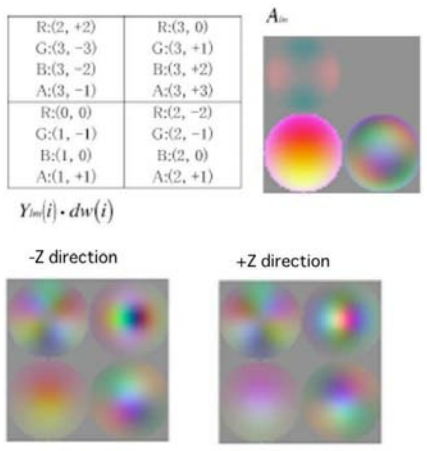
5.1. Changing the file format

First, improvements were made to increase shading language chunks and create a link to the shading language on the textured surface of a 3D object.

5.2. Implementation of the irradiance environment map

5.2.1. $A_{lm}, Y_{lm}(i) \cdot d\omega(i)$ texture creation

When calculating irradiance or other values on graphics hardware, the calculation result is stored according to the texture format. Fig. 1 displays an example of a created texture. While four textures were supposed to be created if there were 16 coefficients, the calculation result was stored by dividing one texture into four segments for the convenience of shader processing. Because the textures were floating-point ones, they were scaled to be within the range of 0 to 255 by giving a bias so that they could be displayed as 32-bit Bitmap files.



R:(2, +2)
G:(3, −3)
B:(3, −2)
A:(3, −1)
	R:(3, 0)
	G:(3, +1)
	B:(3, +2)
	A:(3, +3)

R:(0, 0)
G:(1, −1)
B:(1, 0)
A:(1, +1)
	R:(2, −2)
	G:(2, −1)
	B:(2, 0)
	A:(2, +1)

Figure 1: *A sample texture.*

5.2.2. Creation of the dual-paraboloid map

While it is possible to use a cube map as an environment map to approximate incident light on a surface, a dual-paraboloid map was used here since it requires six textures. The line of sight was changed to $-Z$ and $+Z$ from the center of the surface to be rendered.

5.2.3. L_{lm} texture creation

L_{lm} is the adding up all the texels on the dual-paraboloid map and those corresponding to the $Y_{lm}(i) \cdot d\omega(i)$ textures.

5.2.4. Final rendering

The diffuse light intensity of the rendering surface was found by sampling values from the A_{lm} texture using the normal vector of the surface as the texture coordinate, integrating the values with the corresponding texel value of the coefficient of the L_{lm} texture, and by calculating the total for all l and m values. A dynamic irradiance environment map shader created by the shading language is set for the globe at the center of the room.

5.3. Application to architectural content

Considering the use in an actual Web3D content, a "vase" subject to 2,496-polygon global illumination was placed in a 5,186-polygon apartment and an experiment was conducted using Intel Pentium4 (3.8 GHz) as its CPU and Nvidia Corporation's GeForce 6800 as its graphics hardware and image size (640 x 480) (Fig. 2). The measured value of imaging

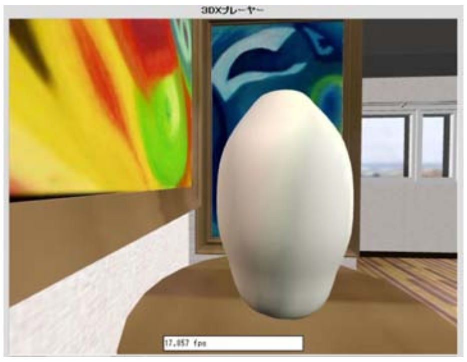


Figure 2: *A first step example with our proposal method.*

speed was a little lower than 18 fps indicating the possibility of application to actual contents.

6. Texture-based global illumination

In the previous section, multiple reflection was not taken into account. When the entire radiance is taken into consideration, the radiance from the surface where the normal vector is N can be expressed by the rendering equation below.

$$L = \frac{\rho}{\pi} \int L(\omega) \cdot max(N \cdot \omega, 0) \cdot d\omega \qquad (10)$$

By applying the Monte Carlo method with a sample number of N to this integral, the equation below can be obtained.

$$\begin{aligned} L &= \frac{\rho}{\pi} \int d\omega \cdot (\frac{1}{N}) \cdot L(\omega) \cdot max(N \cdot \omega, 0) \\ &= \frac{4\rho}{N} \sum L(\omega) \cdot max(N \cdot \omega, 0) \end{aligned} \qquad (11)$$

The one way to reuse the output from the shader in GPU is by rendering to a texture with consideration to the calculation result of global illumination found by a texture atlas using DirectX UVAtlas API [NAC04].

6.1. Texture atlas

Each texture atlas consists of position and normal atlases. As the initial radiance map, the rendering result obtained only with direct illumination is used.

6.2. Representation of shadows by the depth buffer shadow method

In this study, shadow was also drawn into the initial radiance map using the depth buffer shadow method described below. The depth buffer shadow method is used to judge whether a position is in shadow from a light source by storing the depth of the object as a color in a shadow texture using a camera positioned at the light source and comparing the rendered vertex with the depth at the time of rendering. First, the parallel light source was implemented to simulate by the depth buffer shadow method. Second, The Z coordinate was outputted to the fragment shader as a color.

6.3. Visible point determination using the depth separation method

Visible point determination is necessary for calculating global illumination. Hachisuka developed a method for very high-speed visible point determination by applying depth separation, which was originally proposed by Everitt of nVidia as a method for accurate translucency processing without presorting [Hac05]. This was adopted in this study to reduce the computation load. Depth separation is a method to express a scene by separated images (color texture) in the order of depth by storing the depth value in both the Z buffer and texture (depth texture) and repeating subsequent rendering processes several times without rendering the values that are the same or smaller than the depth value.

6.4. Texture synthesis based on the Monte Carlo integral of a diffuse light rendering

In this study, irradiances were sampled once using the method described in section 6.3, and the global illumination effect was calculated based on Eq. 11 using samples of 256 directions. The results of actual rendering performed by following the above procedure are shown below.

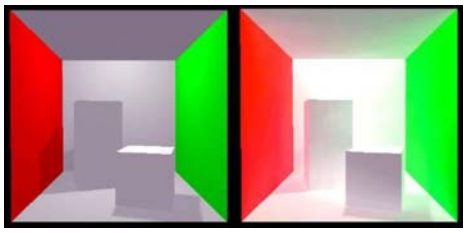


Figure 3: *Result with only direct OpenGL light (left) and that after the application of global illumination (right).)*

6.5. Experiment of the effects

By using the aforementioned method, it was possible to add the global illumination effect even to large objects. Considering the use in sample Web3D contents and an experiment was conducted using same PC configuration in section 5.3. The measured value of the global illumination computing speed was from 1.5 to 2.0 seconds to actual contents with our proposal method.

7. Conclusion

7.1. Summary of the effects

This paper presents a method for simulating global illumination by applying the shading language to Web3D and with the assistance of graphics hardware. Since it is also desirable to improve the representation of Web3D contents, the implementation of Web3D, with which global illumination can be simulated, is considered highly advantageous.

7.2. Future tasks

As a future task in the display process, it will be necessary to examine other method of the shading. At present, the speed problem cannot be solved even with the support of GPU in a 3D space consisting of a large number of polygons. This is due to current GPU not being capable of reading and writing simultaneously for one texture. Since it is possible to improve the entire throughput if the efficiency of this process can be increased even slightly, it will be necessary to consider different methods.

References

[GNMdC04] G. N. M. de Calvalho T. Gill T. P.: X3d programmable shaders. In *Proc. of the ninth international conference on 3D Web technology* (2004), pp. 99–108.

[Hac05] Hachisuka T.: *GPU Gem2*, japanese ed. Born Digital, 2005.

[Kin05] King G.: *GPU Gem2*, japanese ed. Born Digital, 2005.

[NAC04] N. A. Carr J. C. H.: Meshed atlases for real-time procedural solid texturing. *ACM Transactions on Graphics 2*, 21 (Feb. 2004).

[WMYTC00] Wakita A., M. Yajima T. H., Toriya H., Chiyokura H.: A compact and qualitied 3d representation with lattice mesh and surface for the internet. In *Proc. of VRML 2000* (2000), pp. 45–51.

[WYHC01] Wakita A., Yajima M., Harada T., Chiyokura H.: A compact and qualifred web3d representation based on a lattice structure(in japanese). *Journal of Information Processing Society of Japan 42*, 5 (May 2001), 1170–1181.