Real-Time Rendering Framework in the Virtual Home Design System

Pengyu Zhu[1], Mingmin Zhang[1], and Zhigeng Pan[2,*]

[1] State Key Lab of CAD&CG, Zhejiang University, Hangzhou, 310027
[2] Digital Media and HCI Research Center, Hangzhou Normal University, Hangzhou, China
{zhupengyu24,zhigengpan}@gmail.com

Abstract. This paper introduces a home design system with its great functions and framework design, including the scene management based on the Cell&Portal system, improved variance shadow mapping and the recently popular real-time rendering framework called deferred lighting. In the implementation details, we put in some useful improvements, such as compressing the Geometry Buffer and Lighting Buffer to decrease the video memory and bandwidth occupation with which the multi-render-target limitation has been dislodged, using the light volume stencil culling which is similar to the shadow volume algorithm to identify the lit pixels and modifying the physically correct shading model based on Fresnel term to adapt to the deferred lighting framework.

Keywords: Scene Management, Soft Shadow, Deferred Lighting, Light Volume.

1 Introduction

With the real estate industry blooming as well as the housing prices continuing higher in recent years, in order to live a life with comfort and economy, people have to consider a variety of factors before purchase and renovation of housing, such as the size and practicality, the cost estimate of decorating and the beauty and comfort of the room layout. Meeting the above various needs, we have developed a user friendly, flexible and powerful virtual home design software. Distinct from the traditional design systems such as AutoCAD, 3ds Max which not only require the users to have a strong design skills but also need a few hours' design cycle, there already exist some home design systems with foolproof operations and powerful functions such as 72xuan, Sweet Home 3D. However these systems still have some weakness, including lack of efficient scene management and poor rendering effects. Premise of powerful features, our system aims at providing users with comfortable operations and efficient rendering effects.

When there are a lot of geometries in the scene, the real-time rendering system needs to cull the geometries which are outside the view frustum efficiently to minimize the geometric information dumped into the pipeline. There exist lots of visibility

* Corresponding author.

Z. Pan et al. (Eds.): Transactions on Edutainment IX, LNCS 7544, pp. 213–224, 2013.

algorithms, such as bounding volume test, space partition [1]. Haumont [2] has come up with a visibility test algorithm called Cell&Protal Graph which is especially suitable for interior scenes.

Real-time shadow can greatly improve the image authenticity, but a majority of home design software generally uses the model-bounded texture to simulate pseudo shadow. The most popular real-time shadow generation algorithms include Shadow Volume [3] and Shadow Mapping [4]. The shadow volume algorithm needs to calculate the models' silhouette edges, whose time complexity is associated with the geometric information among the scene, which may become the bottleneck of loading models. The shadow mapping is more widely used among real-time rendering areas due to its simplicity, generality and high speed, but it suffers from aliasing artifacts because of perspective projection and under-sampling errors. Simple solution is to increase the shadow map resolution or use Percentage Closer Filter (PCF) which may affect performance. Donnelly [6] came up with the Variance Shadow Map (VSM) algorithm in 2006, which can produce soft shadow efficiently by blurring the shadow map immediately. VSM is exciting but still has some problems, such as light-bleeding artifact due to variance jittering when the scene is complicated. Wojciech [6] has reduced the lighting-bleeding artifact greatly by combining VSM with Exponential Shadow Map (VSM).

The time complexity of pixel-lighting in traditional real-time rendering framework (forward rendering) is $O(m * n * s)$ where m represents geometry number, n represents light number and s represents pixel number. Therefore it's unable to meet the real-time need when there're multi lights in the scene. Deferred Shading was a real-time rendering framework published by Deering [7] in 1988 which has been widely used in games such as Startcraft2, Battlefield and Dota2 in recent years [8-10]. Deferred Shading stores pixel-related position, normal and material information in multiple textures called Geometry Buffer (GBuffer). The time complexity of pixel-lighting decreases to $O(m + n * s)$ because of decoupling lights and geometries. However, Deferred Shading still has some weakness:

- Fat GBuffer costs so much video memory and bandwidth.
- Rendering to GBuffer needs Video Cards with Multi-Render-Target feature.
- Hard to handle semi-transparency geometries.
- Unable to utilize hardware accelerated MSAA.
- The whole rendering pipeline can only use single rendering equation.

The rendering equation among Deferred Shading can be expressed as follows:

$$L_o(v) = \sum_{i=1}^{n} f_{shade}(B_{L_i}, l_i, v, n, C_{diffue}, C_{specular}, S) \quad (1)$$

B_{L_i}: light's intensity and color information, l_i: incident light vector, v: vertex position, n: vertex normal vector, C_{diffue}: surface diffuse material, $C_{specular}$: surface specular material, S: shininess.

According to Equation (**2**), the GBuffer needs to store normal, position and material. This information may conquer 3 to 4 viewport-sized floating textures, which consume so much video memory and bandwidth. How to slim down the GBuffer becomes the key of improving the performance of Deferred Shading. Our system implements a meliorated framework called Deferred Lighting which was proposed by Naty [11] in 2009. Deferred Lighting has been widely used in video games and game engines such

as Crysis2 and Unreal3 [12-13]. Compared to Deferred Shading, Deferred Lighting takes an additional forward geometry pass but reduces the GBuffer size. It also stores lighting result into textures called Lighting Buffer, with which we can use multiple rendering equations among shading. We take some optimizations when implementing this framework to decrease GBuffer and Lighting Buffer to single texture each and eliminate the Multi-Render-Target limitation.

2 Cell&Portal Scene Management

Our system uses Cell&Portal to do scene management. In the implementation details, we maintain a room list and a model list. At First, we compute the room areas as Cells in terms of the walls' information, then identify the doors and windows belong to the associated rooms as Portals, refer to figure 1. Using Equation (**2**) we can efficiently identify the model's belonging room:

$$\bigvee_{k=1}^{m}\{\bigwedge_{i=0}^{n-1}(\boldsymbol{v}_{k,i}-\boldsymbol{p})\times(\boldsymbol{v}_{k,(i+1)\%n}-\boldsymbol{p})>0)\} \tag{2}$$

×: represents cross product, m: convex polygon in the room, n: vertex number, $\boldsymbol{p}$: viewpoint position, $\boldsymbol{v}$: represents vertex position.

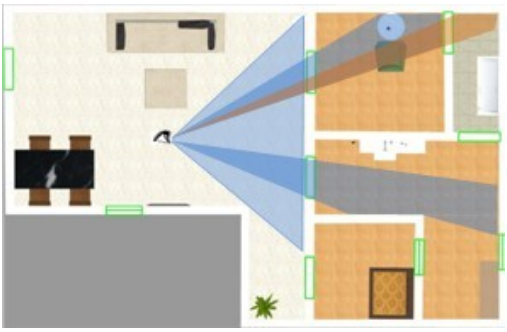


Fig. 1. Cell&Portal in the System

At last, we use the Gribb [14] to extract the world space's frustum information from View-Projection Matrix. With above information, we can compute the models that need to be bumped into the pipeline recursively as the following pseudo code:

```
program Cell_Portal (cell, frustum): model_list
    var model, portal, new_frustum;
  begin
    for(model in cell.model_list)
      if(Intersect(model.OBB, frustum))
        model_list.add(model);
    for(portal in cell.portal_list)
      if(Intersect(portal, frustum))
        new_frustum = Cull(frustum, portal);
        Cell_Portal(portal.other_cell, new_frustum);
  end
```

3 Real-Time Soft Shadow

Variance Shadow Mapping can produce plausibly soft shadow. Due to our small range indoor scene, one median sized resolution shadow map is sufficient. VSM also allows filtering the shadow map immediately, which is much more efficient than the other shadow mapping algorithms that need multi sampling.

We supply three different resolution shadow map due to the current scene range, including 256*256, 512*512 and 1024*1024. VSM uses Chebyshev Inequality (Equation (**3**)) to simulate the shadow factor, so we can filter the shadow map immediately (Mipmap, anisotropic filtering and Gaussian blur) .

$$\begin{aligned} &P(\mathrm{O} \geq \mathrm{R}) \leq p(\mathrm{R}) = \frac{\sigma^2}{\sigma^2+(R-u)^2}, u < R \\ &u = E(\mathrm{O}) \\ &\sigma^2 = E(O^2) - E(\mathrm{O})^2 \end{aligned} \tag{3}$$

R: pixel depth, O: associated texel depth, σ^2: texel depth variance, u: texel average depth.

Then we can present the shadow function as follows:

$$\mathrm{S}(\mathrm{R}) = \begin{cases} p(\mathrm{R}) & u < R \\ 1 & u \geq R \end{cases} \tag{4}$$

When $\mathrm{u} \geq \mathrm{R}$ the pixel is lit.

3.1 Light Bleeding

When the occlusions are complex in the scene, light can emerge in some wrong shadow areas due to depth variance jittering, refer to figure 2.

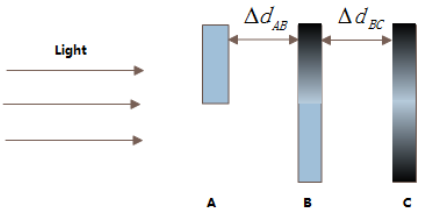


Fig. 2. Light Bleeding: the penumbra of object B bleeds onto C because of variance jittering when $\Delta d_{AB}/\Delta d_{BC}$ becomes big

We add a negative tail value after $p(\mathrm{R})$, this can reduce the bleeding light's intensity, but may cause band artifact, refer to Equation (**5**).

$$p'(\mathrm{R}) = \max\left(\frac{\sigma^2}{\sigma^2+(R-u)^2} - \gamma, 0\right), u < R \tag{5}$$

Wojciech solves this problem by combining another shadow mapping algorithm based on statistics called Exponential Shadow Map (ESM) with VSM. This combination can handle most situations except when VSM and ESM both fail, but that hardly happens. Then we get the shadow function as follows:

$$
\begin{aligned}
S'(R) &= \begin{cases} p''(e^{cR}) & u < R \\ 1 & u \geq R \end{cases} \\
p''(e^{cR}) &= \max\left(\frac{\sigma'^2}{\sigma'^2 + (e^{cR} - u')^2} - \gamma, 0\right) \\
\sigma'^2 &= E((e^{cO})^2) - E(e^{cO})^2 \\
u' &= E(e^{cO})
\end{aligned}
\tag{6}
$$

c is a control parameter, its value is as big as possible in theory, but too large c may cause floating point precision, we set c 30 in our system. Taking the exponential of depth makes $\Delta d_{AB}/\Delta d_{BC}$ gets smaller.

4 Deferred Lighting

Modified from Equation (**1**), we get a set of equations as follows:

$$
\begin{aligned}
\mathbf{L_{diffuse}} &= \sum_{i=1}^{n} f_{diffuse}(B_{L_i}, \mathbf{l_i}, \mathbf{n}) \\
\mathbf{L_{specular}} &= \sum_{i=1}^{n} f_{specular}(B_{L_i}, \mathbf{l_i}, \mathbf{n}, \mathbf{v}, S) \\
\mathbf{L_o} &= \mathbf{C_{diffuse}} \circ \mathbf{L_{diffuse}} + \mathbf{C_{Specular}} \circ \mathbf{L_{specular}}
\end{aligned}
\tag{7}
$$

°: tensor product, $\mathbf{L_{diffuse}}$: pixel's diffuse light intensity, $\mathbf{L_{specular}}$: pixel's specular light intensity.

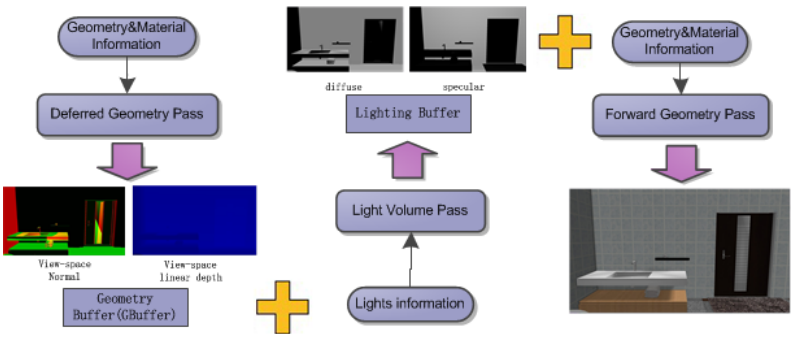


Fig. 3. The Framework of Deferred Lighting

According to Equation (**7**), our GBuffer only needs to store n, v, S, and we can get the two material terms $C_{diffuse}$ and $C_{Specular}$ which occupy two textures in the naïve deferred shading framework from the second geometry pass. Because of the decoupling of lighting pass from shading pass, Deferred Lighting is also called Partial Deferred Shading, refer to figure 3.

In terms of figure 3, there is an additional rendering pass in order to get the Lighting Buffer which stores $\mathbf{L_{diffuse}}$ and $\mathbf{L_{specular}}$. The naïve memory layout still needs two textures which consumes so much video memory and bandwidth, we must compress it into single texture with some optimizations.

4.1 Buffer Compression Optimization

In the implementations, GBuffer uses GL_RGBA16F pixel format and Lighting Buffer uses GL_RGBA8. GBuffer contains $\mathrm{n}, \mathrm{v}, \mathrm{S}$, among which n and v occupy *xyz* three terms each. We adopt "Spheremap Transform" [16] from CryEngine3 to compress n into two terms and extract the original when needed refer to Equation (**8**).

$$\begin{aligned} \boldsymbol{G_n} &= normalize(\boldsymbol{n_{xy}}) * \sqrt{\frac{\boldsymbol{n_z}*2+1}{2}} \\ \boldsymbol{n_z} &= length\left(\boldsymbol{G_{n_{xy}}}\right) * 2 - 1 \\ \boldsymbol{n_{xy}} &= normalize\left(\boldsymbol{G_{n_{xy}}}\right) * \sqrt{1 - \boldsymbol{n_z}^2} \end{aligned} \tag{8}$$

G_n : the compressed normal information, n : the original normal vector. This compression scheme consumes 18 pixel shader instructions, but compared to some other schemes, it has a better balance between performance and effect.

In fact we can easily reconstruct the view position from pixel's *xy* information in NDC space. So we just store the view space linear depth in the GBuffer, and construct View-Ray to rebuild the view position by similar triangle theorem in very small errors (Equation (**9**)). By means of the above compression schemes, the GBuffer is compressed into a single texture.

$$\boldsymbol{v} = \frac{\boldsymbol{r}*\boldsymbol{f_z}*\boldsymbol{v_z}}{\boldsymbol{r_z}} \tag{9}$$

f_z: the far clip plane's depth in view space, r: the View-Ray which is produced by the rasterization from the far clip plane corners when rendering full screen quad or the pixel view position when rendering light volume.

Lighting Buffer stores pixels' lighting intensity information, including $\mathbf{L_{diffuse}}$ and $\mathbf{L_{specular}}$ which contains three terms each. Assuming that $\mathbf{L_{diffuse}}$ and $\mathbf{L_{specular}}$ have the same color information, we use an approximate solution which just stores the luminance of $\mathbf{L_{specular}}$, and reconstruct its color information from $\mathbf{L_{diffuse}}$ (however specular color information is lost when $\mathbf{L_{diffuse}}$ and $\mathbf{L_{specular}}$ have different color information).

$$\mathbf{L_{specular}}' = \mathbf{L_{diffuse}} \circ \left(\frac{\mathrm{lum}(\mathbf{L_{specular}})}{\mathrm{lum}(\mathbf{L_{diffuse}})+\varepsilon}\right) \tag{10}$$

ε represents a small constant (0.0001 in our implementation) in case of dividing zero.

4.2 Light Volume

In the Deferred Lighting framework, the lighting computation of global lights such as the parallel light from sun is fulfilled by a full screen quad rendering pass. However, there also exist some local lights which may just affect limit number of pixels, it'll be a waste of time to render a full screen quad pass as well. General solution is to use light volume (sphere as point light and comicalness as spot light) to mark out the pixels affected by local lights. Fabio [16] made use of the hardware accelerated sissor test to mark out pixels by projecting the light volume onto screen viewport, but this method may mark out some pixels incorrectly. Our system uses a method similar to the shadow volume algorithm which is able to identify the pixels affected by lights accurately no matter the eye point is inside or outside the light volume. This algorithm can be divided into two passes whose pipeline states are taken out in Table 1.

Table 1. The pipeline states of Light Volume

State	Pass 1	Pass 2
Backface Culling	Back	Front
Color Mask	None	RGBA
Depth Writable	False	True
Depth Test	LessEqual	GreateEqual
Stencil Test	True	False
Stencil Op	Z-Fail Incr	Equal Ref=0
Clear Stencil	False	True

According to the rendering states in Table 1, in Pass 1 we cull the back face of light volumes, the blue pixels pass the depth test and keep the stencil buffer, red pixels fail the depth test and increase the stencil buffer value. In Pass 2 front face is culled, the green pixels pass the depth stencil test and marked as lights affected pixels refer to figure 4-a. When eye point is within the light volume, this method still marks out the pixels correctly seen from Figure 4-b.

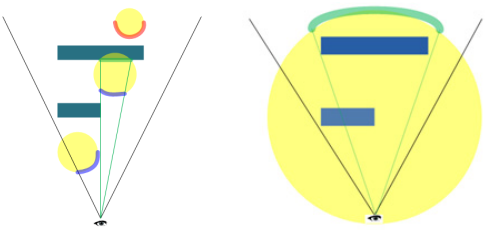


Fig. 4. Light Volume algorithm: (a) eye point outside the light volume (b) eye point inside the light volume

4.3 Rendering Equation

We use the rendering equation with physical correctness published by Sloan [1] to do shading:

$$f_{shade}(B_L, \mathbf{l}, \mathbf{n}, \mathbf{v}, S) = B_L{}^{\circ}\left(\mathbf{C_{diff}} + R_F(\mathbf{C_{spec}}, \mathbf{l}, \mathbf{h})\frac{8+S}{8}(\mathbf{n}\cdot\mathbf{h})^S\right)(\mathbf{n}\cdot\mathbf{l}) \quad (11)$$

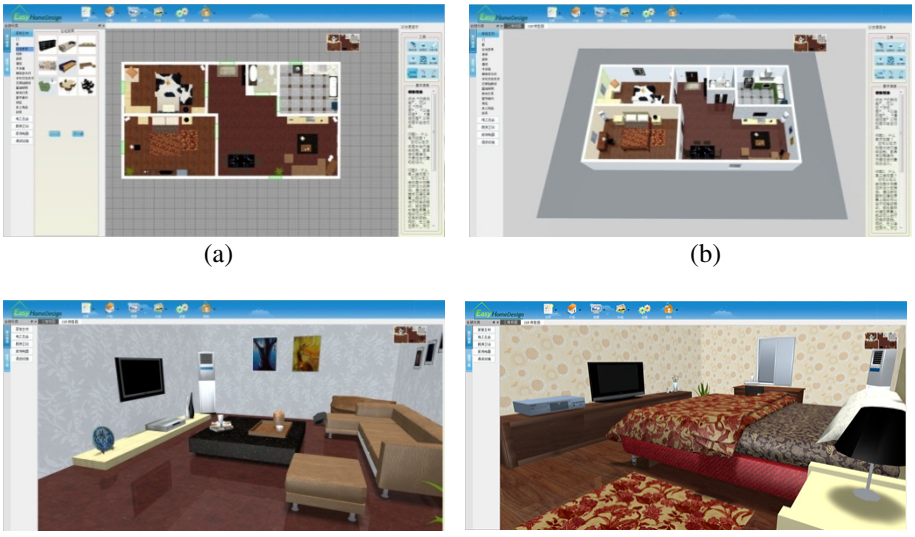
B_L: the light diffuse and specular reflection intensity, $\mathbf{l}$: the incident light, $\mathbf{v}$: the eye vector, $\mathbf{h}$: the half vector in Blinn Phong model, R_F: Fresnel reflection function.

Due to the fact Fresnel function R_F depends on $\mathbf{l}, \mathbf{h}$, which also depend on the light information, this leads the failure to decouple lighting from shading, so we use an approximate solution by modifying R_F as follows:

$$R_F{}'(\mathbf{C_{spec}}, \mathbf{n}, \mathbf{v}) \approx \mathbf{C_{spec}} + (\mathbf{1} - \mathbf{C_{spec}})(1 - \mathbf{n}\cdot\mathbf{v})^5 \quad (12)$$

Although this alterative solution is not physically correct, when $\mathbf{v}$ is coinciding with $\mathbf{l}$, the value of $\mathbf{n}\cdot\mathbf{v}$ is the same as $\mathbf{h}\cdot\mathbf{l}$. Then $R_F{}'(\mathbf{C_{spec}}, \mathbf{n}, \mathbf{v})$ no longer depends on the light, it can be applied to our shading framework with unnoticeable artifact.

5 Results



(a) (b) (c) (d)

Fig. 5. (a) top orthogonal design view (b) 3D design view (c)(d) 3D navigation view

Our software has three different views (figure 5), including the top orthogonal view for wall construction, operating models, laying floors, etc, 3D design view for operating models, pasting wallpaper, etc, and the 3D navigation view for walking through the

indoors. The results below are running on a PC with Intel Core2 Quad 2.5G Hz CPU, 4GB RAM, Nvidia GTX460 video card and Windows7 OS.

5.1 Scene Management

We take three variations as experiments when Cell&Portal system is on and off. The results show that the frame rate has a big raise when Cell&Portal is on though the performance still depends on the scene distribution, refer to table 2.

Table 2. Frame rate test under Cell&Portal Management

House Type	Triangle Number	On	Off
West Lake Type A	46987	102 fps	42 fps
Wanan Court B	34959	105 fps	56 fps
Wanan Court C	29871	151 fps	65 fps

5.2 Soft Shadow

In the implementation details, the system automatically choose the shadow map resolution from three candidates (256*256, 512*512 and 1024*1024) according to the scene range. And by setting the value of γ and c reasonably, the light bleeding is alleviated effectively. Seen from figure 6-left, when the scene's occlusion relationship is complex, the original VSM introduces noticeable light bleeding artifact due to variance jittering. On the other hand, too big γ may cause banding as well (figure 6-middle).

Fig. 6. Light Bleeding: left: VSM causes light bleeding middle: VSM with $\gamma = 0.2$ causes banding right: VSM+ESM with$\gamma = 0.1, c = 30$

5.3 Deferred Lighting

The video memory consumed by Deferred Shading and Deferred Lighting with our compress optimization respectively is displayed below (the resolution of viewport is 1072*768). According to table 3, Deferred Lighting uses just the half memory comparing to Deferred Shading and needs only one texture each rendering pass which saves bandwidth.

During the process of rendering, GBuffer contains a GL_RGBA16F float point texture which uses Spheremap Transform to compress normal into RG channels and stores the linear depth in channel G and shininess in channel A, refer to figure 7.

Table 3. Memory Consumption of Deferred Shading and Deferred Lighting

Buffering | Channels	Deferred Shading	Deferred Lighting
Normal | 2	√	√
Linear Depth | 1	√	√
Shininess | 1	√	√
Diffuse Material | 3	√	X
Sepcular Material | 3	√	X
Emissive Material | 3	√	X
Diffuse Lighting | 3	X	√
Specular Lighting | 3	X	√
Textures	GL_RGBA16F*3	GL_RGBA16F*1 GL_RGBA8*1
Memory(MB)	18.84	9.42

(a) (b) (c)

Fig. 7. Geometry Buffer (GBuffer) (a) Spheremap Transformed normal (b) Linear Depth (c) Shininess

Lighting Buffer contains a GL_RGBA8 texture which stores diffuse lighting information in RGB channel and specular lighting luminance in channel A (figure 8).

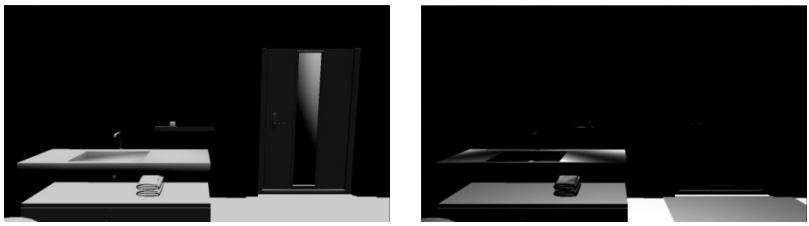


Fig. 8. Lighting Buffer (a) Diffuse Intensity (b) Specular Luminance

When entering the shading pass, geometries, materials and textures are dumped into pipeline, and Lighting Buffer is viewed as a shader resource. After the pixel lighting and shadow generation, the result of the bathroom is showed below:

Fig. 9. Rendering Result

6 Conclusion

This paper introduces a virtual home design system, and describes the next-gen indoor rendering framework with its optimizations in details. In addition to pixel lighting, we can add some features effectively such as screen space ambient occlusion (SSAO), high dynamic rendering (HDR) and depth of field (DOF) in the future.

Acknowledgements. This research work is co-supported by the following NSFC projects: grant no: 61003197, 60970076, 61170318.

References

1. Tomas, M., Haines, E.: Real-Time Rendering, 3rd edn. A.K. Peters Ltd. (2008)
2. Haumont, D., Debeir, O., Sillion, F.: Volumetric cell-and-portal generation. J. Computer Graphics Forum 22(3), 303–312 (2003)
3. Crow, F.C.: Shadow algorithms for computer graphics. In: Proceedings of SIGGRAPH 1977, pp. 242–248. ACM Press, Barzel (1977)
4. Williams, L.: Casting curved shadows on curved surfaces. In: Proceedings of SIGGRAPH 1978, pp. 270–274. ACM Press, Atlanta (1978)
5. Donnelly, W., Lauritzen, A.: Variance shadow maps. In: Proceedings of the 2006 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 161–165. ACM Press, New York (2006)
6. Wojciech, S.: Variance Shadow Maps Light-Bleeding Reduction Tricks. GPU Pro2 Advanced Rendering Techniques. A.K. Peters Ltd. (2011)
7. Deering, M.S., Winner, B., Schediwy, C., Duffy, Hunt, N.: The Triangle Processor and Normal Vector Shader: A VLSI system for High Performance Graphics. J. Computer Graphics 22(4), 21–30 (1988)
8. Shishkovtso, O.: GPU Gems 2: Deferred Rendering in S.T.A.L.K.E.R. Addison-Wesley Professional (2005)
9. Valient, M.: Deferred Rendering in Killzone 2. Presentation, Develop Conference, Brighton (2007)

10. Filion, D., Mcnaughton, R.: Effects & techniques. In: SIGGRAPH 2008: ACM SIGGRAPH 2008 Classes, pp. 133–164. ACM Press, New York (2008)
11. Naty: Deferred Lighting Approaches, http://www.realtimerendering.com/blog/deferred-lighting-approaches/
12. Mittring, M.: Finding Next-Gen: CryEngine 2. In: SIGGRAPH 2007: ACM SIGGRAPH 2007 Courses, pp. 97–121. ACM Press, New York (2007)
13. Samaritan: Unreal Engine 3 Showcase. Technical report, Epic Games (2011)
14. Gribb, G., Hartmann, K.: Fast Extraction of Viewing Frustum Planes from the World View Projection Matrix. Online document (2001)
15. Martin. M.: A bit more Deferred – CryEngine 3. Technical report, Triangle Game Conference (2009)
16. Fabio, P., Franciso, F.: Deferred Shading Tutorial, http://fabio.policarpo.nom.br/docs/Deferred_Shading_Tutorial_SBGAMES2005.pdf