06软1 贾堃 06509250

基于Direct3D的实时阴影渲染实现

摘要：实时渲染技术是计算机绘图技术的一个重要分支，广泛应用于游戏、虚拟现实和虚拟3D社区中，是现在发展最快的图形渲染技术。本篇文章将讲解如何使用微软平台标准API接口DirectX中的3D图形组件Direct3D来实现在动态光照下物体的实时阴影效果。主要采用两种技术，阴影贴图和阴影体，两者都是现在的主流实现方法，前者主要是使用深度贴图的方式来实现，后者则是使用渲染阴影体和模板缓冲来实现。

关键词：Direct3D；实时渲染；实时阴影

**The Implement of Realtime Shadow Rendering Based on Direct3D**

**Abstract:** The Realtime Rendering technique which is used in Games, VR and virtual 3D communities widely is an important embranchment of Computer Graphics. It’s the fastest progress in graphics rendering techniques. In this article, we’ll explain how to implement the realtime shadow of objects under the dynamic lights on the 3D graphic component – Direct3D of the Microsoft standard platform API– DirectX. We mainly adopt two techniques – Shadow Map and Shadow Volume. All of them are popular. Shadow Map mainly uses Depth Map, Shadow Volume mainly uses volume rendering and Stencil Buffer.

**Keywords:** Direct3D; Realtime Rendering; Realtime Shadow

1 阴影贴图（Shadow Map）

1.1 基本原理

阴影贴图（Shadow Map）是Williams在1978年提出的，可以说是所有阴影处理中最简单的方法。基本原理如下：

场景中每个光源都相当于一个独立的摄像机，都有个一个独立的D3DFMT\_R32F灰度格式的阴影贴图。该贴图主要存储了一个深度值，用来表示光源到相应点的距离。然后在实际场景渲染过程中将当前顶点变换到光源坐标当中，并用其深度值与阴影贴图对应像素点的值进行比较，小于则被光照，大于则在阴影中，从而分别进行渲染。

1.2 具体实现

采用可编程管线的方式来实现，其优点是程序、数据和渲染过程分离进行，更为灵活。

在程序创建完D3D设备后，首先用IDirect3DDevice9::CreateTexture()创建用于渲染阴影贴图的2D纹理。然后分别加载用于渲染阴影贴图和正式场景的Shader，以及基于光源摄像机的世界、观察和投影矩阵。

接着，程序将进入阴影贴图的渲染函数。该函数首先先将原有的RenderTarget和深度/模板缓冲进行备份，并将阴影贴图纹理及其深度/模板缓冲设置为当前的RenderTarget和深度/模板缓冲。然后便对其进行渲染。

最后是正式场景的渲染函数。首先将备份的RenderTarget和深度/模板缓冲进行还原，然后设置标准的变换矩阵以及光源变换矩阵，并使用用于阴影比较的正式场景Shader进行最终渲染。

以下是阴影贴图Shader和正式Shader的核心代码，前者的PS返回的是一个32位灰度值，而后者的PS返回的是一个带有光照或阴影的正常颜色：

//Shadowmap Vertex Shader

VS\_OUTPUT VS(VS\_INPUT In)

{

VS\_OUTPUT Out;

Out.Position = mul(float4(In.Position, 1.0f), wvp\_matrix);

Out.ScreenPos = Out.Position;

return Out;

}

//Shadowmap Pixel Shader

float4 PS(VS\_OUTPUT In) : COLOR

{

float4 color = In.ScreenPos.z/In.ScreenPos.w;

return color;

}

//Main Vertex Shader

VS\_OUTPUT VS(VS\_INPUT In)

{

VS\_OUTPUT Out;

Out.Position = mul(float4(In.Position, 1.0f), wvp\_matrix);

Out.LightPos = mul(float4(In.Position, 1.0f), light\_wvp\_matrix);

Out.Normal = mul(In.Normal.xyz, (float3x3)light\_wv\_matrix);

return Out;

}

//Main Pixel Shader

float4 PS(VS\_OUTPUT In) : COLOR

{

float4 projected\_pos = In.LightPos/In.LightPos.w;

float fToCenter = distance(projected\_pos, float2(0.5f, 0.5f));

float fSpotLightEffect = 1.0f – saturate(fToCenter/0.4f);

float4 shadowmap = tex2D(texShadowmapSampler, projected\_pos.xy);

float fLit = projected\_pos.z<shadowmap.r;

float color = fSpotLightEffect\*InNormal.z\*fLit;

return color;

}

2 阴影体（Shadow Volume）

2.1 基本原理

Shadow Volume这种算法第一次被提出是在Franklin C.Crow在1977年写的一篇论文“SHADOW ALGORITHMS FOR COMPUTER GRAPHICS”里。其基本原理是根据光源和遮蔽物的位置关系计算并分别渲染出阴影体的正面和背面，同时通过模板缓冲来获得会产生阴影的像素区域，然后对所有物体进行光照/阴影渲染，被模板缓冲标记的地方将不会更新。主要有如下几个步骤：

1. 由于采用Direct3D9，而非Direct3D10/11，Shader本身无法生成新的顶点，因此先要对场景里所有物体进行插入分裂四边形的预处理。
2. 确定物体的轮廓边，并用Vertex Shader将这些边中的分裂四边形拉伸构成阴影体。
3. 分别渲染阴影体正反面，更新模板缓冲。
4. 渲染光照/阴影。

2.2 Z-Pass 和 Z-Fail

Z-Pass 是Shadow Volume一开始的标准算法，原理是渲染阴影体深度测试通过时更新模板缓冲。但是后来人们发现，如果观察点在阴影体内部的话，Z-Pass算法就完全失效了，渲染出来的阴影也会完全错误。Z-Fail算法的目的就是用来解决这个问题的，原理跟前者正好相反，是渲染阴影体深度测试失败时更新模板缓冲，因此Z-Fail算法更为实用。

2.3 具体实现

预处理的步骤是用分裂四边形（degenerate quad）来替换物体中的共享边，具体如下：

1. 遍历网格中所有的面（三角形）。
2. 计算当前面的面法线。
3. 遍历每个面的3个顶点。
4. 将每个顶点插入到一个检查表中。
5. 如果一个面的两个顶点与其他面的两个顶点构成的是同一条边，则进行如下处理：

* 检查两个面的面法线是否平行，如果不平行，则继续往下。
* 对其中插入分裂四边形。

值得注意的是e中的第一步可以大大提高效率，因为不是所有物体中三角形的边都会成为轮廓边的，如果两个面（三角形）的面法线平行，那么它们之间的共享边永远也不会成为轮廓边。

创建和渲染阴影体、更新模板缓冲等步骤都是在渲染函数中进行的，因此需要算法有较低的时间复杂度。确定轮廓边的方法如下：

1. 遍历网格中所有的面。
2. 计算dot(light\_direction,triangle\_normal)来确定该面是面向光源还是背向光源。
3. 对于面向光源的面，将所有的三条边压入一个栈，和里边的进行比较，如果发现重复的，则将这些边删除。
4. 检测过所有面的所有边以后，栈里面剩下的边就是当前光源/物体位置下的轮廓边了。

现在就可以使用Vertex Shader将其中的分裂四边形拉伸成阴影体了。

接着便是渲染阴影体。根据阴影体面相对于观察点的朝向将其分解为正面部分和反面部分，同时将模板缓冲清零并关闭Z缓冲的更新。渲染反面阴影体，并更新模板缓冲，使其增1；渲染正面阴影体，并更新模板缓冲，使其减1。当所有阴影体渲染完毕后，模板值为0的像素便是正常像素，而模板值不为0的像素便是带阴影的像素了。最后对模板值为0的像素渲染光照或对模板值不为0的像素渲染阴影就可以了。

由于Shadow Volume相对比较复杂，因此这里就不再给出相应实现代码。

3 总结

实时阴影技术是现在实时渲染中非常重要和基本的技术之一，它的实现将大大改善实时渲染的效果，使游戏和虚拟现实更加的逼真。或许将来会有更好的实时阴影算法，但是阴影贴图和阴影体仍然是现在的主流。关于实时渲染技术，还有很多话题，这里就不再说了。

参考文献：

[1] 彭国伦. 3D绘图程序设计——使用Direct3D 10/9和OpenGL 2.0[M]. 北京：科学出版社，2009：558-564 381-386.

[2] 信息产业部软件与集成电路促进中心，北京汇众益智科技有限公司. 网络游戏客户端编程[M]. 北京：电子工业出版社，2007：188-199.

[3] Wolfgang F.Engel. Direct3D游戏编程入门教程[M]. 第二版. 北京：人民邮电出版社，2005：183-195.

[4] 李剑鹏. Shadow Map在DirectX9.0 SDK Sample 的实现方法[EB/OL]. GameRes游戏开发资源网，（）[2010-1-18]. http://dev.gameres.com/Program/Visual/3D/ShadowMap.mht