第41卷 第12期 西 安 交 通 大 学 学 报 Vol.41 №12
2007年12月 JOURNAL OF XI′AN JIAOTONG UNIVERSITY Dec. 2007

基于球体追踪的动态视差遮挡映射算法

刘晓东，寻亮，马栋，陶海霞
（西安交通大学电子与信息工程学院，710049，西安）

摘要：提出了一种利用球体追踪进行逐像素光照计算的动态视差遮挡映射算法.该算法先对纹理的高度图进行预处理，为纹理高度空间内的任意点到高度图中保存的实际表面建立最短距离映射，并在图形处理单元(GPU)像素着色器上利用距离映射进行球体追踪，获得准确的纹理视差偏移值，同时加入纹理的自遮挡效果，生成软阴影.算法还利用可编程GPU管线进行硬件加速，以满足实时渲染的要求.与在高度图上进行固定步长线性采样的动态视差遮挡映射算法(DPOM)相比，所提算法的处理效率可提高3.3倍，并且不会出现DPOM算法在步长较大时出现的纹理漂浮现象.

关键词：动态视差；球体追踪；软阴影；实时渲染
中图分类号：TP391 **文献标识码**：A **文章编号**：0253-987X(2007)12-1401-05

Dynamic Parallax Occlusion Mapping Algorithm Based on Sphere Tracing

Liu Xiaodong, Xun Liang, Ma Dong, Tao Haixia
(School of Electronics and Information Engineering, Xi′an Jiaotong University, Xi′an 710049, China)

Abstract: A dynamic parallax occlusion mapping algorithm based on per-pixel sphere tracing is proposed, in which the height map of the texture is preprocessed to build a shortest distance map for the practical surface saved in the height map from any points within texture height space. Then, the sphere tracing is carried out to obtain the accurate parallax offset of the texture in a pixel shader of graphic processing unit (GPU) by using distance map. Meanwhile, the self occlusion effect of texture is added to form the soft shadow. Applying the programmable GPU pipeline, the algorithm adopts hardware acceleration techniques to satisfy the needs of real-time rendering. Comparing with the dynamic parallax occlusion mapping (DPOM) which uses linear sampling with fixed step in the height map, the efficiency of the proposed algorithm can be increased by 3.3 times, and the texture floating phenomenon that exists in DPOM when step is bigger will not appears.

Keywords: dynamic parallax; sphere tracing; soft shadow; real-time rendering

纹理映射是通过定义纹理与物体之间的映射关系，将纹理图像映射到物体的几何表面来表征物体外观属性的过程.常用的颜色纹理映射难以表现物体表面凹凸细节.1978年，Blinn提出了一种纹理映射技术——凹凸映射[1]，利用一个扰动函数扰动物体的表面法向量来模拟有随机法向量的粗糙表面纹理.随后很多人对凹凸映射进行了深入研究，也提出了很多的改进算法.2001年，Kaneko等人提出了视差映射[2]，使用高度信息来偏移纹理，但容易产生失真.2004年，Welsh在可编程GPU上实现了带有位移限制的视差映射[3]，但在倾斜的观察角度上，仍会产生严重失真.2006年，Tatarchuk提出了动态视差遮挡映射算法(DPOM)[4]，通过在高度图上进行固定步长的线性采样来获取纹理偏移位置信息，并根据视线和几何表面的夹角来决定固定步长的大小，但这种方法不可避免地具有线性采样算法的缺点，

收稿日期：2007-04-29. 作者简介：刘晓东(1976～)，男，副教授. 基金项目：国家高技术研究发展计划资助项目(2003AA209021).

在提高精确性的同时降低了渲染的效率.

本文提出一种基于球体追踪的带有自遮挡、软阴影的动态视差映射算法.算法利用球体追踪来进行纹理偏移计算,根据纹理自遮挡情况实时计算软阴影,最后使用法向量图进行光照计算.

1 视差映射算法及其分析

1.1 视差偏移

当观察具有高度起伏的凹凸表面时,会出现视差效果,而普通的颜色纹理映射则无法表现这种效果.假设从图1中所示的 E 位置进行观察,观察向量与多边形表面的交点为 $\boldsymbol{T}_0$(即实际表面上的 A 点),但观察的结果应该是看到实际表面上的点 C,普通的颜色纹理映射会直接渲染 A 点,于是视差效果便消失了.

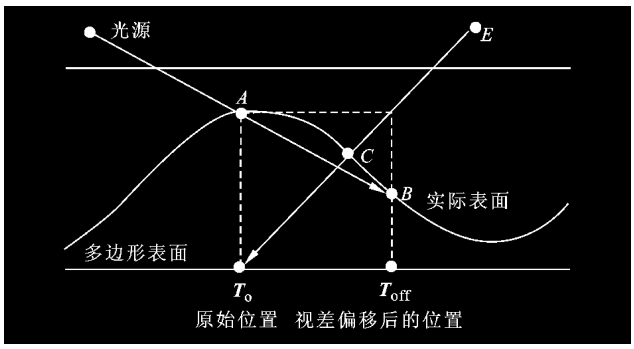


图1 Kaneko[2]的视差偏移算法原理图

视差映射通过沿视线方向偏移当前点的纹理位置来获得视差效果,偏移的大小由当前点的高度信息决定.

Kaneko[2]通过使用观察向量与多边形表面法向量的点积与点 A 的高度相乘得到近似的视差偏移量,求得点 B 的纹理位置信息 $\boldsymbol{T}_{off}$,最终使用 $\boldsymbol{T}_{off}$ 来映射纹理(见图1).这种方法可以近似表现纹理的视差偏移现象,但这样求出来的交点不一定在真正的几何体凹凸表面,并不能够提供很好的效果.随后,Tatarchuk[4]、Policarpo等人[5]都提出了一些改进的方法来求取准确的交点.这些方法都是在高度图上进行固定步长的线性采样算法,只要采样的步长大于一个像素,就可能会遗漏处于采样点之间的交点.如图2所示,在计算交点的过程中,由于步长较大,p_3 和 p_4 都位于多边形表面外侧,算法认为视线与表面没有相交,于是导致细节丢失的现象.

1.2 自遮挡

纹理本身高度值大的部分会对其他部分产生遮挡作用,并在有光照的情况下对自身产生阴影.例如

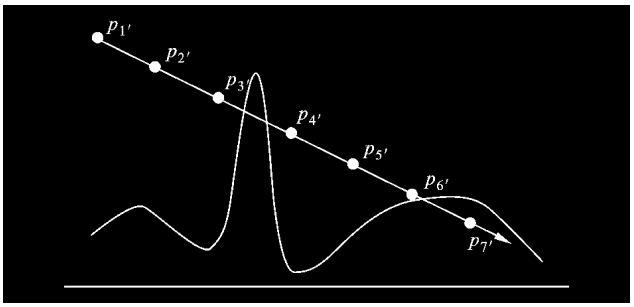


图2 线性采样算法原理图

图1中 C 点由于受到自身纹理的遮挡,无法被光源照射到,所以在渲染纹理时应该在 C 点生成阴影.

2 算法实现

本文算法通过定义表面的距离映射,使用球体追踪方法来获取纹理视差偏移位置信息,并且通过计算像素点的遮挡状态,生成软阴影.整个算法分为坐标系变换、纹理视差偏移、计算自遮挡生成的软阴影和光照计算4个阶段.

2.1 坐标系变换

首先建立切空间坐标系.假设当前处理的顶点是 I 点,从物体几何模型中检索到此点所对应的法向量 $\boldsymbol{N}$,在表面上沿纹理 u(或者 v)轴生成切向量 $\boldsymbol{T}$,$\boldsymbol{B}=\boldsymbol{N}\times\boldsymbol{T}$,其中×运算表示向量的外积.用单位化后的3个向量 $\boldsymbol{T}$、$\boldsymbol{B}$、$\boldsymbol{N}$ 组成基矩阵 $\boldsymbol{M}$.

获得矩阵 $\boldsymbol{M}$ 之后,可以将 $\boldsymbol{M}$ 与世界坐标系中的光线向量和视线向量进行外积计算,就可将世界坐标中的向量变换到切空间之中.在GPU的顶点着色器(Vertex Shader)上定义一个额外的因子,来调整纹理偏移的比例.计算光线向量、视线向量的公式如下

$$\left.\begin{aligned}\boldsymbol{V}_{lw}&=\boldsymbol{P}_{light}-\boldsymbol{P}_{vertex}\\ \boldsymbol{V}_{ew}&=\boldsymbol{P}_{eye}-\boldsymbol{P}_{vertex}\\ \boldsymbol{V}_{lt}&=\boldsymbol{V}_{lw}\times\boldsymbol{M}\\ \boldsymbol{V}_{et}&=\boldsymbol{V}_{ew}\times\boldsymbol{M}\times[1,1,\alpha,1]\end{aligned}\right\}\tag{1}$$

式中:$\boldsymbol{P}_{light}$是光源位置信息;$\boldsymbol{P}_{vertex}$是顶点位置信息;$\boldsymbol{P}_{eye}$是视点位置;$\boldsymbol{V}_{lw}$和$\boldsymbol{V}_{ew}$分别是世界坐标系中的光线方向向量和视线方向向量;$\boldsymbol{V}_{lt}$和$\boldsymbol{V}_{et}$分别是切空间中的光线方向向量和视线方向向量;α 是调整纹理偏移大小的控制因子.这个坐标系变换的工作是放在顶点着色器中来完成的,之后,所有数据经过渲染管线的裁剪阶段后传入GPU的像素着色器.

2.2 纹理视差偏移

本文计算纹理视差偏移位置信息时使用球体追踪的方法不断逼近几何体表面,经过少量的采样就

可以计算出精确的纹理视差偏移位置信息.

首先，定义表面的距离映射，即对于纹理空间的任何给定点 p 和表面 $\boldsymbol{S}$，定义函数

$$D_{\min}(p,\boldsymbol{S}) = \begin{cases} \min\{d(p,q) \mid q \in \boldsymbol{S}\} & p \text{ 点位于 } \boldsymbol{S} \text{ 的外侧} \\ -\min\{d(p,q) \mid q \in \boldsymbol{S}\} & p \text{ 点位于 } \boldsymbol{S} \text{ 的内侧} \end{cases} \tag{2}$$

式中：q 是表面 $\boldsymbol{S}$ 中的任意一点；$d(p,q)$ 是点 p 到点 q 的距离；$D_{\min}(p,\boldsymbol{S})$ 是点 p 到表面 $\boldsymbol{S}$ 的最短距离.

计算距离映射可以采用 Danielsson[6] 提出的关于 $O(n)$ 复杂度的算法[6]，这里的 n 是映射中像素的数目. 算法创建一个三维的距离映射，并初始化所有位于表面内侧像素的距离变量为 0，所有位于表面外侧像素的距离变量为正无穷，然后执行一个三维邻域内的连续交换，每个像素根据邻近像素的距离变量计算出自己的距离变量.

通过以下定义的公式，可以获得所需精度的纹理视差偏移位置信息

$$\boldsymbol{p}_n = \begin{cases} [\boldsymbol{T}_0,1] + D_{\min}([\boldsymbol{T}_0,1],\boldsymbol{S})\boldsymbol{V} & n = 1 \\ \boldsymbol{p}_{n-1} + D_{\min}(\boldsymbol{p}_{n-1},\boldsymbol{S})\boldsymbol{V} & n > 1 \end{cases}$$

$$\boldsymbol{T}_{\text{off}} = U(\boldsymbol{p}_n) \tag{3}$$

式中：$\boldsymbol{T}_0$ 是源像素点的纹理位置信息；$\boldsymbol{T}_{\text{off}}$ 是纹理视差偏移位置信息；$\boldsymbol{V}$ 是单位化的视线方向向量；$U(\boldsymbol{p}_n)$ 表示点 $\boldsymbol{p}_n$ 在坐标系 XY 平面上的投影；n 是控制算法进行距离纹理采样的次数.

视差偏移算法的步骤如下.

(1)把当前 $\boldsymbol{T}_0$ 赋值给 $\boldsymbol{T}_{\text{off}}$，把 $\boldsymbol{V}_{\text{et}}$ 单位化，为预先计算好的 3D 距离纹理和像素着色器中的 3D 纹理对象 $\boldsymbol{D}_{\text{tex}}$ 建立映射，根据实际情况设置算法的精度控制因子 $\beta=n$，初始化 $i=0$.

(2)如果 $i>\beta$，跳到步骤(5)，否则往下执行.

(3)使用 $\boldsymbol{T}_{\text{off}}$ 对 $\boldsymbol{D}_{\text{tex}}$ 进行采样，获得当前位置到表面之间的最短距离 $d_{\min}$.

(4)令 $i=i+1$，利用公式(3)，沿 $\boldsymbol{V}_{\text{et}}$ 对 $\boldsymbol{T}_{\text{off}}$ 进行偏移，偏移距离为 $d_{\min}$，如果 $d_{\min}$ 大于误差阀值 e，跳回到步骤(2).

(5)输出当前获得的 $\boldsymbol{T}_{\text{off}}$，算法结束.

图 3 中的例子采用以上算法计算 5 次就可找到足够逼近表面的点.

2.3　计算自遮挡生成的软阴影

软阴影情况如图 4 所示. 首先沿视线 $\boldsymbol{V}$ 查找到 $\boldsymbol{T}_{\text{off}}$，即交点 K，然后利用光源到交点 K 的光线 $\boldsymbol{L}$ 和距离映射来进行阴影计算，如果 $\boldsymbol{L}$ 与表面 $\boldsymbol{S}$ 有交点，就说明发生了自遮挡，当前点位于阴影中.

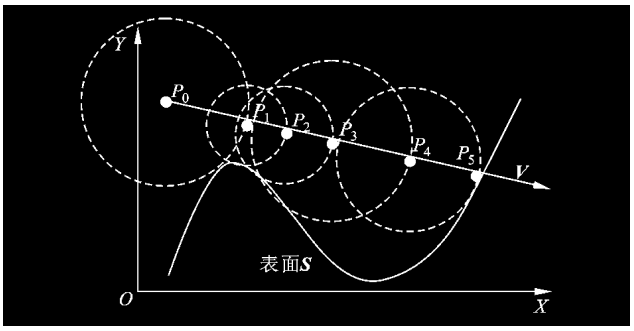


图 3　球体追踪逼近表面

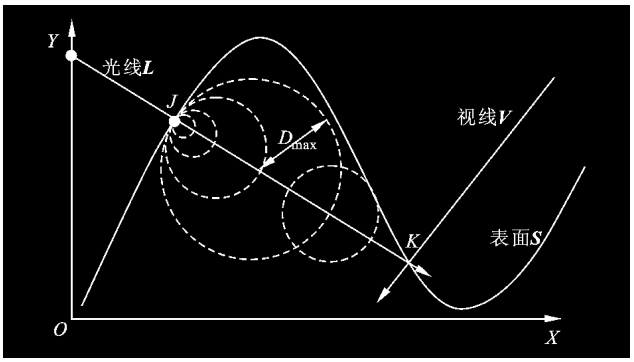


图 4　利用球体追踪计算遮挡体的体积

软阴影由以下公式计算

$$O_c = (D_{\max}/D_{\text{tsd}})(1-d) + d \quad d \in [0,1] \tag{4}$$

式中：d 是软阴影的影响因子；$D_{\max}$ 是遮挡体的体积；D_{tsd} 是阴影控制阀值；O_c 是最终进行光照计算时候的阴影系数.

软阴影算法的步骤如下.

(1)获取 $\boldsymbol{P}_{\text{light}}$，视差偏移算法中计算出来的 $\boldsymbol{T}_{\text{off}}$，3D 距离纹理对象 $\boldsymbol{D}_{\text{tex}}$，纹理高度图；根据 $\boldsymbol{T}_{\text{off}}$ 和对应的纹理高度 k，生成交点 K 的位置 $\boldsymbol{P}_k=[\boldsymbol{T}_{\text{off}},k]$，计算出 $\boldsymbol{V}_{\text{lt}}$；初始化 $D_{\max}=0$.

(2)利用距离映射，利用公式(3)求出光线与表面的交点 J，如果点 J 与点 K 之间的距离小于阈值 e，则认为点 J 与点 K 为同一点，没有发生遮挡，跳转到步骤(7).

(3)以点 J 为起点，起始 $d_{\min}$ 为 $2e$，使用公式(3)计算新点位置向量 $\boldsymbol{P}_{\text{temp}}$，初始化 $i=0$.

(4)如果 $i>\beta$，跳转到步骤(7)，否则往下执行.

(5)使用 $\boldsymbol{P}_{\text{temp}}$ 对 $\boldsymbol{D}_{\text{tex}}$ 进行采样，获得当前位置到表面之间的最短距离 $|d_{\min}|$.

(6)计算 $\boldsymbol{P}_{\text{temp}}+|d_{\min}|V_{\text{lt}}$，将获得的新的点赋值给 $\boldsymbol{P}_{\text{temp}}$，如果 $|d_{\min}|$ 值大于 $D_{\max}$，$D_{\max}=|d_{\min}|$；$i=i+1$；跳转到步骤(4).

(7)如果 $D_{\max}=0$，则 $O_c=1$；否则，$O_c=(D_{\max}/D_{\text{tsd}})\cdot(1-d)+d$，输出 O_c，算法结束.

2.4 光照计算

利用视差偏移算法和软阴影算法计算的结果进行光照计算. 计算光照的公式如下

$$C_{\text{final}} = C_{\text{base}} I_{\text{diff}} O_{\text{c}} = C_{\text{base}}(k_{\text{a}} I_{\text{a}} + k_{\text{d}} I_{\text{l}}(\boldsymbol{N} \cdot \boldsymbol{L}))O_{\text{c}} \quad (5)$$

式中:C_{base}是使用$\boldsymbol{T}_{\text{off}}$对纹理颜色图进行采样的颜色值;$I_{\text{diff}}$是漫反射光强度;$k_{\text{a}}$是环境光反射系数;$I_{\text{a}}$是环境光强度;$k_{\text{d}}$漫反射系数;$I_{\text{l}}$是光源强度;$\boldsymbol{N}$是表面单位法向量;$\boldsymbol{L}$是指向点光源的单位方向向量;符号·表示向量的内积.

2.5 本文算法整体描述

(1)输入三维空间中的物体的网格数据(包括顶点数据集$\boldsymbol{I}$、顶点纹理坐标集$\boldsymbol{T}$、顶点法向量集$\boldsymbol{N}$)、光源位置$\boldsymbol{P}_{\text{light}}$、视点位置$\boldsymbol{P}_{\text{eye}}$、纹理颜色图$\boldsymbol{C}_{\text{tex}}$、纹理高度图$\boldsymbol{H}_{\text{tex}}$、纹理法向量图$\boldsymbol{N}_{\text{tex}}$、纹理偏移控制因子$\alpha$、算法精度控制因子$\beta$、距离映射精度因子$\gamma$、软阴影的影响因子$d$.

(2)输出经过动态视差遮挡映射算法处理过的屏幕渲染帧,并存储到显示卡的帧缓存中.

本文算法描述如图5所示.

```
ParallaxMapping Algorithm(...) //输入数据如前述.
    使用距离映射算法对H_tex进行预处理获得3D纹理D_tex;
    将网格数据、P_light、P_eye、C_tex、H_tex、N_tex、D_tex、α、β、
        d 送入显存;
    for I中的每一个顶点
        计算顶点X的变换矩阵M;
        根据公式(1), 使用M和α计算V_lt和V_et;
        if 顶点可见
            把V_lt、V_et送入像素着色器;
            把X的位置信息I_x、纹理信息T_x、法向量N_x送入像素
            着色器;
        end if
    end for
    for 显示卡帧缓存的每一个像素
        以T_x、V_et、D_tex为参数调用视差偏移算法获得T_off;
        以V_lt、P_light、T_off、D_tex、H_tex为参数调用软阴影算法
        获得O_c;
        使用T_off对D_tex、N_tex采样获得C_base和法向量N;
        根据公式(5)计算出最终的帧像素颜色;
    end for
    进行雾化操作;
    进行模板/深度/alpha检测;
    送入帧缓存;
    end
```

图5 本文算法整体描述

下面进行算法复杂度分析. 顶点着色器对每个需要处理的顶点进行操作,算法复杂度是$O(n)$,n为物体顶点数. 像素着色器使用的视差偏移算法以及软阴影算法的复杂度取决于对纹理采样进行的次数,也就是设定的算法精度控制因子β,所以这2个算法的复杂度是$O(\beta)$,设屏幕像素分辨率为θ,那么像素着色器的算法复杂度为$O(\theta\beta)$. 由于现代图形硬件对于矩阵运算是很高效的,而纹理采样则是相对昂贵的,所以算法的整体负载在像素着色器上.

3 结果与分析

本文算法采用DirectX9.0C实现,运行环境为PC机(P4 3.0 GHz,内存1 GB,显卡Geforce 6600,显存128 MB). 输入数据包括一个纹理的颜色图、高度图以及法向量图,几何物体为只有4个顶点的正方形,控制因子α、β、γ分别为2、8、16,屏幕分辨率θ为1 024×768像素.

图6给出了本文算法、普通的凹凸纹理映射(BM)以及Welsh的带有位移限制的视差映射(PMOL)的实现效果. 可以看出,BM算法不能够处理由于视角变化引起的纹理透视深度的变化,也不能够处理纹理自身的遮挡效果,而PMOL在计算砖块缝隙的纹理偏移信息时,由于高度变化过快,导致计算出来的信息不准确,产生严重的失真. 本文算法实现的效果是对每块砖的细节表现得更加具体,可以明显地感觉到每一块砖头都是向外突出的,并且在凹凸有致的排列中会产生自遮挡效果,自阴影效果十分明显.

(a)BM (b)PMOL (c)本文算法 (d)DPOM

图6 不同算法的纹理映射效果

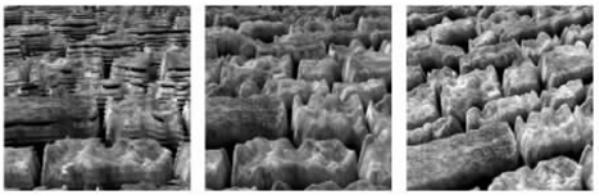
Tatarchuk实现的动态视差遮挡映射算法(DPOM)是基于高度图进行固定步长线性采样的算法,DPOM在顶点着色器上的复杂度是$O(n)$,在像素着色器上的复杂度是$O(\theta\delta)$,其中δ为1/固定步长,尽管DPOM采用了层次细节(LOD)决定运行时步长的改良做法,但是算法的复杂度仍然是$O(\theta\delta)$. 与本文算法复杂度$O(\theta\beta)$相比,由于本文算法利用预处理计算距离映射信息,可以大大减少纹理采样的次数. 为了达到同样的纹理偏移计算精度,DPOM必须使用极小的步长,也就导致δ明显大于β. DPOM实现的效果如图6d所示. 当选用很小的步

长时，DPOM 实现的效果和本算法的效果类似，但是采样次数明显增多. 由于本文算法的纹理采样次数取决于所设定的精度值，进行光线求交仅需要 8 次纹理采样就可以达到图中效果，而 DPOM 算法的纹理采样次数取决于所设定的采样步长，如果设定采样步长为 $1/n$，那么所需要的纹理采样次数就会是 $y(y\in n)$次，实验结果表明 DPOM 在选用步长为 1/130 的时候，所需要的平均纹理采样次数为 52. 正因为这种进行逐像素光线求交过程中纹理采样次数的巨大差别，导致两者在渲染效率上面也有巨大的差别. 如表 1 所示，本文算法的平均渲染帧速为 43.79 帧/s，而 DPOM 在步长为 1/130 时的平均渲染帧速仅为 13.25 帧/s，本文算法相比 DPOM 算法效率提高了 3.3 倍.

表 1　各种凹凸纹理映射算法对本文实例渲染的帧速比较

	BM	PMOL	本文算法	DPOM (步长 1/8)	DPOM (步长 1/130)
帧速/帧·s^{-1}	98.23	94.46	43.79	47.31	13.25

本文算法除了在效率上与 DPOM 算法相比有很大提高之外，在渲染效果上的差别也很大. 正如前面分析的结果一样，像 DPOM 这种在高度图上进行固定步长线性采样的算法所实现的效果会受到步长的影响. 图 7 比较了本文算法与 DPOM 算法的实现效果与渲染帧速.



(a)DPOM(1/8)　(b)DPOM(1/130)　(c)本文算法

图 7　DPOM 算法与本文算法的渲染效果比较

由图 7 可以看出，当 DPOM 算法采用步长为 1/8 的时候，会导致严重的纹理走样，产生了纹理漂浮现象，这种现象产生的原因就是前文中提到的由于步长过大，在采样点之间的交点会被遗漏.

4　结　论

本文提出一种基于球体追踪的动态视差映射算法. 先对纹理高度图进行预处理，获取距离映射信息，然后利用距离映射进行球体追踪，获得准确的纹理视差偏移值，很好地表现了纹理的自遮挡属性，避免了纹理漂浮现象，并利用软阴影真实地表现了物体表面的凹凸细节. 算法主要工作在可编程 GPU 端，充分利用了可编程 GPU 管线的硬件加速能力，减轻了 CPU 的负担，较好地满足了实时渲染的要求，与同类算法 DPOM 相比，效率可提高 3.3 倍.

参考文献：

[1] Blinn J F. Simulation of wrinkled surfaces [J]. Computer Graphics, 1978, 12 (3): 286-292.

[2] Kaneko T, Takahei T, Inami M, et al. Detailed shape representation with parallax mapping[C]//Proceedings of the ICAT 2001. Tokyo: ICAT, 2001: 205-208.

[3] Welsh T. Parallax mapping with offset limiting: a per-pixel approximation of uneven surfaces[M]// Engel W. ShaderX3: Advanced Rendering with DirectX and OpenGL. Boston, USA: River Media, 2004: 89-95.

[4] Tatarchuk N. Dynamic parallax occlusion mapping with approximate soft shadows[C]// Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. New York: ACM Press, 2006: 63-69.

[5] Policarpo F, Oliveira M M, Comba J. Real-time relief mapping on arbitrary [C]//Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics Polygonal Surfaces. New York: ACM Press, 2005: 359-368.

[6] Danielsson P E. Euclidean distance mapping [J]. Computer Graphics and Image Processor, 1980, 14 (2): 227-248.

(编辑　刘杨　苗凌)