基于 GPU 的后置处理运动模糊算法①

陈淑彬

(上海海事大学 信息工程学院, 上海 201306)

摘 要: 针对虚拟现实系统中运动模糊的真实感绘制问题, 着重研究了一种基于图形处理器(GPU)后置处理的运动模糊改进算法. 该算法通过将场景渲染时前后两帧全分辨率纹理图的深度值存入深度缓存区, 使用其深度信息与当前帧的映射矩阵提取物体的世界中心坐标, 并获取运动的速度矢量; 沿速度矢量进行采样、高斯滤波, 最后引入模糊因子, 并对线性化采样值进行加权融合, 力求实现最真实的运动模糊效果. 实验结果表明: 所提算法不仅不会影响大型虚拟现实系统的渲染流畅度, 更改善了过分重影或者拖影, 具有良好的实时性和真实感.

关键词: 运动模糊; 模糊因子; 深度缓存; 虚拟现实

Post-Processing Blur Algorithm Based on GPU

CHEN Shu-Bin

(College of Information Engineering, Shanghai Maritime University, Shanghai 201306, China)

Abstract: In order to enhance the immersive realism of motion blur in virtual reality system, an improved post-processing algorithm of motion blur based on Graphics Processing Unit (GPU) is proposed. Depth value of full resolution texture maps in before and after frames of scene rendering are stored in the depth buffer in this algorithm. It uses its depth information and mapping matrix of the current frame to extract the world coordinate of the object, and gets the velocity vector. Then along speed vector it samples Gaussian filters. Fuzzy factor is introduced and fused weighted linear sample values to achieve the most realistic motion blur effect. The results show that: the proposed algorithm withgood real-time and realism effect will not only not affect the fluency of rendering large virtual reality system, but also solve over ghosting or streaking.

Key words: motion blur; fuzzy factor; depth buffer; virtual reality

当相机拍摄物体时, 相机的自身移动或拍摄物体的移动都会产生相对运动, 而相机的快门曝光是有延迟的, 因此在这段时间范围内, 相机会对物体进行捕捉, 产生多次渲染, 即在底片上的成像是会跟随运动物体移动而移动的, 这种成像效果就是运动模糊. 而对于计算机图像引擎来说, 大多数采用的都是针孔模型相机, 几乎不考虑快门时间, 因此并不会产生运动模糊的效果, 这不仅使现有的虚拟现实场景缺乏真实沉浸感, 也会让用户对运动物体的速度有较难的把握. 随着对运动模糊算法的不断研究与深入, 使用户操纵、观赏时更易投入, 能够真正意义上地实现虚拟环境的真实体验.

Korein 与 Badler[1]、Dippe 与 Wold[2]等人的早期研究, 为后人探索运动模糊特效奠定了良好的基础. 运动模糊算法大致可分为三类. 第一类是由 Korein 与 Badler[1]等人提出的基于时间域的方法, 先将单帧拆分为多帧渲染, 再融合为单帧输出到屏幕, 在时间域上对场景进行过采样, 有着较高的精度, 在运动物体速度不是很快的情况下会有较好的效果, 但是由于它会将渲染都存入缓存区, 很难具有实时性且会严重降低

① 基金项目:国家自然科学基金(61171126);交通运输部应用基础研究项目(2014329810060)
收稿时间:2016-01-14;收到修改稿时间:2016-03-08 [doi:10.15888/j.cnki.csa.005366]

系统帧率. 第二类是由 Wloka[3]与 Zeleznik 等人基于物体空间提出的相关算法, 其中心思想在于对物体进行表面拆分, 再构造出连接两部分的多边形作为拖影部分, 以扫过风格实现运动模糊. 从空间的角度出发解决问题, 对物体的运动范围有较清晰的模拟, 但它的拓展性差, 对较为复杂曲线轨迹便会束手无策. 最后一类是由 Simon Green[4]等人提出的基于 GPU 后置处理的解决方案, 其经典算法由 Shimizu[5]提出, 通过缓存帧中的纹理信息求出物体运动矢量, 从而只依赖屏幕的分辨率而减少与场景复杂性的相关度. Rosado[6]将输入改为深度信息, 通过投影矩阵计算生成速度图. Padget[7]则脱离了对速度图的依赖, 使渲染效果大幅提升, 但却不适应复杂环境. 更有 Schmid[8]等人提出通过线性插值渲染出模糊线条, 以此来模拟运动模糊轨迹的算法出现. 近年来还基于 REYES[9,10]渲染架构进行了一系列研究工作. 综上所述, 为了满足系统的各项指标要求, 本文对传统的 GPU 后置处理算法进行了改进, 加入了对运动顶点的预判, 改善了过分拖影的问题, 对采样点进行高斯滤波, 保证了真实性和流畅性两者的平衡, 并且通过大量的实验数据, 分析得到最符合系统效果的模糊因子, 最后将构造出的模糊因子与采样值加权融合从而实现良好且不影响系统运行帧率的运动模糊特效.

1　基本原理

1.1 运动模糊原理

首先通过下列四张图, 了解基于静止图像的运动模糊实现原理. 第一幅旨在将目标图放大倍数, 添加更多的像素点, 在更高的分辨率下画出一个失真的圆; 第二张图则将图像分割为许多 4*4 小正方形; 第三张是对于每个小正方形, 取其所有像素的平均值, 再用此值来填充小正方形; 最后缩小还原, 完成一张具有运动模糊的反失真圆形图.

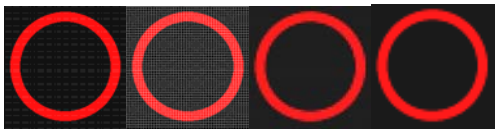

图 1　运动模糊的合成原理

1.2 可编程渲染管线

通过使用图形硬件编程实现算法, 将大量计算都放在 GPU 中进行, 这样不仅提高了运行速率, 也大大减少了多帧图像融合错误的概率. 相较在 CPU 固定渲染管线计算的坐标转换, 利用 Shader 实现模糊, 会使效果更为逼真, 也更容易与底端渲染引擎融合. 相关框架流程如下图 2 所示.

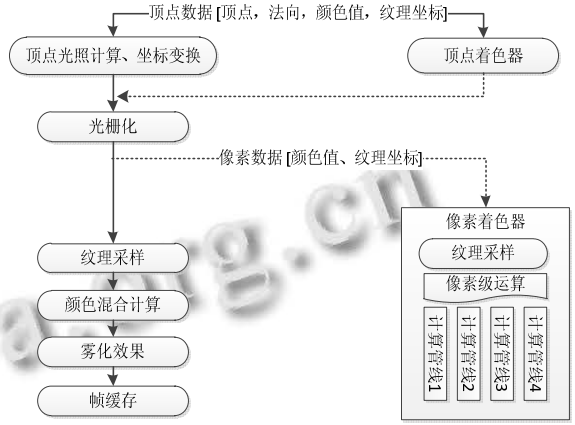

图 2　可编程流水线总体构架图

2　基于后置处理运动模糊算法

运动模糊必须满足实时性和真实性, 同时还不能影响系统的流畅渲染. 本文通过处理渲染纹理得到物体运动矢量, 再根据运动矢量来进行运动模糊的模拟, 并且保证物体点的模糊程度尽可能与真实情境保持一致.

本文的算法实施步骤如下:

1) 将场景的前后两帧全分辨率渲染纹理存入深度缓存区, 利用 GPU 顶点着色器计算出该运动物体顶点的运动矢量, 并根据前后两帧的视图矩阵, 视图投影矩阵提取出物体的世界坐标;

2) 对 1 中的顶点作条件判断, 输出顶点坐标信息传送至像素着色器, 插值出每个像素速度和坐标;

3) 沿运动矢量对场景渲染纹理进行采样, 高斯滤波;

4) 构造模糊因子;

5) 将线性化采样点与模糊因子进行加权融合.

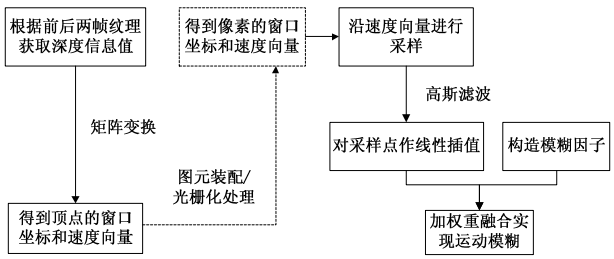

图 3　算法流程图

2.1 利用顶点及像素编程获得运动矢量及顶点预判

算法先提取出已存入纹理缓存区的深度值, 把视口坐标与视图投影逆矩阵相乘, 得到相应的世界坐标, 再利用前一帧视图投影矩阵转化求来的矩阵进行差值平均, 计算出顶点的速度矢量. 假设给定像素 A 的视口坐标为(x, y, z), 其中x, y的取值范围在为$(-1,1)$, z 为该像素的深度值, 原点在屏幕的正中央, 坐标为$(0,0,0)$, H为转换矩阵, M为这个像素的世界坐标, 且 m 设为 1, 则有如下计算公式:

$$A=(\frac{x}{m},\frac{y}{m},\frac{z}{m},1) \tag{1}$$

$$A\times M^{-1}=(\frac{mX}{mH},\frac{mY}{mH},\frac{mZ}{mH}),mH=B \tag{2}$$

$$H=\frac{B}{B\cdot m} \tag{3}$$

得到前后两帧的窗口坐标, 再通过差值平均的方式求出物体顶点的运动矢量.

根据上述已经求出的运动方向与顶点法向量的夹角来判断顶点处于物体的哪个表面. 物体表面的划分定义如下, 将夹角小于 90°归类为前表面, 反之则归类于后表面. 在进行运动模糊效果模拟时, 我们往往可以发现, 在移动时物体的每个点均会发生拖尾模糊, 这样营造出的效果就不符合真实情况, 因此, 为了弥补此类不足, 本文对顶点采取了预判, 让处于物体前表面顶点的像素值保持基本不变, 模糊因子尽可能增大, 相反情况则尽可能减小模糊因子, 使模糊拖影现象明显.

假设运动向量坐标为(x_1,y_1,z_1), 顶点法向量坐标为(x_2,y_2,z_2), 当前像素值为 A, 前一帧像素值为 B, 构造的数学模型如下:

$$\theta=\arccos(\frac{x_1x_2+y_1y_2+z_1z_2}{\sqrt{x_1^2+y_1^2+z_1^2}\cdot\sqrt{x_2^2+y_2^2+z_2^2}}) \tag{4}$$

$$A=(\theta<90^0)?A:B \tag{5}$$

由此模型所构造出的最佳效果如图 4, 这不仅能够明确地知晓运动方向, 也能让成像时模糊效果更具层次感, 例如近在眼前的物体与视野内较远物体相比, 模糊感要强烈得多.

图 4 最佳运动模糊效果图

同时所求的顶点信息会被传递给像素着色器进行下一步的处理, GPU 会进行图元装配和光栅化处理, 插值出每个像素的速度和坐标.

2.2 构造合适的模糊因子

在快门曝光的时间内, 拍摄物体或者相机正在移动, 便会导致成像中的物体变模糊, 但是相同运动方向, 场景中的各个物体点的模糊程度却是不尽相同. 因此不同规模的系统需要根据自身需求采用合适的模糊因子. 模糊因子一般的取值范围为$(0,1)$. 在选取过程中, 结合了前一步骤中对顶点的预判, 具体计算公式如下所示:

$$\alpha=\begin{cases}0.08, Motion_{vector}\bullet Normal>0\\ 0.04, Motion_{vector}\bullet Normal<0\end{cases} \tag{6}$$

其中, $Motion_{vector}$ 表示运动向量, $Normal$ 表示顶点法向量. 对处于物体前表面的顶点, 模糊因子取值相对大些, 让模糊效果不那么明显, 处于后表面的点则相反. 关于模糊因子参数的设置, 本文对模糊因子、系统帧率、模糊效果三个因素进行了综合分析, 如图 5 所示: 发现两条曲线的交叉点在α=0.04 处时, 系统帧率为 82 帧/秒, 用户在模糊视觉效果上打了 7 分, 而在α=0.08 时, 系统帧率为 90 帧/秒, 模糊视觉效果给分是 4, 模糊程度已经相对微弱, 而运行却相对流畅. 因此选用α=0.04/0.08 作为最终结果.

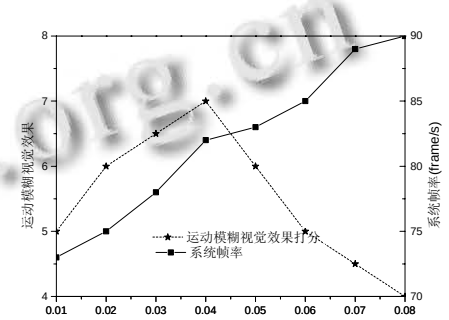

图 5 运动模糊效果与模糊因子、系统帧率的关系

2.3 采样点处理与权重融合

根据每个像素的窗口坐标和速度值, 沿着运动方向, 结合不同的运动速度进行采样, 并对采样点进行高斯滤波. 同时需要注意的是采样点的数量也不宜过多, 一般 8 个即可, 虽然越多肯定能够达到更好的效果, 但会对场景进行过多次的纹理渲染, 导致系统帧率严重下降, 甚至出现渲染不流畅等不良现象. 因此为了避免这类问题, 可采用分离式二维高斯滤波, 数

学模型如下所示:

$$A=(\sum_{x=1}^{n}\sum_{y=1}^{n}B_{xy}\times C_{xy})\Big/D \tag{7}$$

其中, A 表示高斯滤波后的值, B_{xy} 表示二维高斯矩阵, C_{xy} 表示 B_{xy} 对应的高斯系数, D 表示高斯矩阵中所有系数的总和, n 表示矩阵的维数. 采用标准卷积的计算方式对采样点进行高斯滤波, 即分别沿运动水平方向 X、运动垂直方向 Y 轴作一维卷积.

将经过处理的采样值与构造好的模糊因子, 以公式(8)进行权重融合.

$$F(N)=\begin{cases}R(0)+\alpha(1-\tau)R(N)+\sum_{n=1}^{N-1}\alpha\tau^{N-n}(1-\tau)R(n), N>0\\ R(N), N=0\end{cases} \tag{8}$$

其中, $F(N)$ 表示当前像素的最终颜色渲染值, $R(N)$ 为采样点颜色值, $N=0$ 表示初始帧, 对清晰场景纹理采样时得到的顶点颜色值, τ 表示融合权重, α 表示归一化模糊因子.

3　试验结果与分析

本文实验的硬件环境如下:

CPU: i7-3770(3.40GHz), 内存: 4GB, 显卡: NVIDIA GeForce 605, 分辨率: 1280*1024. 软件配置为 Window 7(64bit)操作系统, 利用 DirectX 中的高阶着色器语言(High Level Shading Language, HLSL)编写具体算法, 同时根据物体顶点位置的预判调节模糊因子, 在运动矢量方向上进行采样高斯滤波, 最后将线性化处理过的采样值与模糊因子进行插值融合, 改善前人算法中不足的地方.

系统旨在营造良好的沉浸感, 因此系统所有建模都使用了 1024*1024 等高清纹理贴图, 导致系统的显卡使用率大幅增加. 在模糊时必须考虑到帧率的影响, 否则只能顾此失彼.

图 6 给出了同一情境(模拟直升机正常俯冲速度)不同模糊因子下的运动模糊效果对比(α =0.02/0.08, α =0.04/0.08, α =0.06/0.08).


(a)上海海事大学图书馆　(b) α =0.02/0.08


(c) α =0.04/0.08　(d) α =0.06/0.08

图 6　不同参数设置下的运动模糊效果

观察发现, 在 α =0.04/0.08 时效果为最佳, 既能判断直升机的运动状态, 又能将视野范围内较远物体清晰渲染, 符合真实情境. 前者拖影现象过为严重, 无法看清物体的真实样貌, 在 3D 荧幕投影下会产生晕眩的不良反应; 后者运动模糊效果不明显, 体验者无法感受到俯冲而下的视觉冲击.

本文系统运行帧率大致在 73~82fps 范围内, 也尝试使用参考文献中的算法进行性能比较, 虽然顶点预判、纹理采样等处理会引起一部分的计算时耗, 但就效果来说, 是更为真实, 更具有针对性和节省内耗的, 同时也提高了整个算法的执行效率. 由表 1 可知, 本文算法对帧率的影响要远小于其他算法, 并且在视觉效果上也有较好地体现.

表 1　算法实时性比较

帧率单位(frame/s)	最低帧率	最高帧率	平均帧率
文献[6]的算法	62	71	65.5
本文的算法	73	82	77.5

4　总结与展望

本文提出了一种基于 GPU 后置处理的运动模糊算法, 在模拟直升机俯冲视角的过程中, 研究了渲染帧率、模糊因子、视觉效果的三者关系, 基于速度缓存的思想方法, 提出对顶点所处表面的预判, 并将模糊因子与线性化采样值加权融合的改进办法, 使最终结果更贴近现实生活中的真实情况. 实验结果表明, 本算法在不影响系统运行速度的前提下, 为原本静止的画面增添了一份动感.

就现今社会而言, 人们对于虚拟现实的视觉感受以及真实感、沉浸感方面的需求是越发强烈, 因此未来还需要在运动模糊等特效处理上做到精益求精, 力求能达到更为平滑流畅的视觉体验.

参考文献

1 Korein J, Badler N. Temporal anti-aliasing in computer

generated animation. Computer Graphics (Proc. SIGGRAPH), 1983, 17: 377–388.

2 Dippe M, Wold E. Antialiasing through stochastic sampling. Computer Graphics (Proc. JSIGGMPH), 1985, 19: 369–381.

3 Matthias W. Implementing Motion Blur&Depth of Field Using DirectX 8. Meltdown, 2001: 488–493.

4 Green S. Stupid openGL shader tricks. Game Developers Conference 2003. California, USA: Gama Network, 2003.

5 Shimizu C, Shesh A, Chen BQ. Hardware accelerated motion blur generation. Minnesota, USA: University of Minnesota, 2003.

6 Rosado G. Motion Blur as a post-processing effect. Hubert Nguyen, ed. GPU Gems 3. New Jersey: Addison-Wesley, 2007: 495, 575–581.

7 Padget B, Games R, Diego S. Efficient real-time motion blur for multiple rigid objects, 2012.

8 Schmid J, Summer R, Bowles H. Programmable motion effects. ACM Trans. on Graphics (Proc. SIGGRAPH), 2010, 29(4): 57.

9 Hou Q, Qin H, Li W. Micropolygon ray tracing with defocus and motion blur. ACM Trans. on Graphics, 2010, 29 (4): 64.

10 Heinzle S, Wolf J, Kanamori Y. Motion blur for EWA surface splatting. Computer Graphics Forum, 2010, 29 (2): 733.

11 Navarro F, Serón FJ, Gutierrez D. Motion blur rendering: State of the art. Computer Graphics Forum, 2011, 30(1): 3.