软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software [doi: 10.13328/j.cnki.jos.007160]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

运动感知的移动游戏帧率调整算法*

梁宇智, 霍宇驰, 王 锐

(浙江大学 CAD & CG 国家重点实验室, 浙江 杭州 310058)

通信作者: 梁宇智, E-mail: yuzhiliang@zju.edu.cn

摘 要: 随着移动设备的广泛普及, 其图形处理器的性能也逐渐增强. 为了满足用户对卓越体验的不断追求, 移动设备屏幕分辨率和刷新率每年都在不断提升. 与此同时, 移动游戏中的可编程绘制流水线也变得日益复杂, 这导致游戏应用成为移动设备功耗的主要来源. 研究了移动游戏中的绘制流水线, 提出一种运动感知的绘制帧率调整方法, 以在节能模式下保证绘制质量. 与以往仅考虑历史帧绘制误差的预测模型不同, 该方法通过建立摄像机位姿和帧间绘制误差的非线性关系模型, 通过未来帧新的摄像机位姿预测其绘制误差, 实现更为精确的帧率调整策略. 此外, 该方法还包括一个轻量级的场景识别模块, 可根据玩家所处特定场景有针对性地调整误差阈值, 从而采用不同程度的帧率调整策略. 在定量对比上, 相比只考虑历史帧误差的预测模型, 构建的模型在游戏帧序列上的预测准确性提高 30% 以上. 同时, 在用户实验的定性对比上, 相同跳帧比例下该方法能够得到用户体验更好的绘制效果. 提出的算法融合了历史帧误差和摄像机位姿变化信息, 能够预测出更准确的未来帧误差. 算法结合预测结果和场景识别结果, 实现了更好的动态帧率调整效果.

关键词: 实时绘制; 能耗优化; 自适应帧率调整

中图法分类号: TP391

中文引用格式: 梁宇智, 霍宇驰, 王锐. 运动感知的移动游戏帧率调整算法. 软件学报. http://www.jos.org.cn/1000-9825/7160.htm

英文引用格式: Liang YZ, Huo YC, Wang R. Motion-aware Frame Rate Adjustment Algorithm for Mobile Game. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7160.htm

Motion-aware Frame Rate Adjustment Algorithm for Mobile Game

LIANG Yu-Zhi, HUO Yu-Chi, WANG Rui

(State Key Lab of CAD & CG, Zhejiang University, Hangzhou 310058, China)

Abstract: As mobile devices are widely used, the performance of their graphics processors has increasingly improved. To meet users' continuous pursuit of excellent experience, the screen resolution and refresh rate of mobile devices are constantly increasing every year. At the same time, the programmable shading pipeline in mobile games is becoming more complex, which leads to game applications becoming the main source of power consumption for mobile devices. This paper studies the rendering pipeline in mobile games and proposes a motion-aware rendering frame rate adjustment method to ensure rendering quality in power-saving mode. Unlike previous prediction models that only consider rendering errors of historical frames, this method builds a nonlinear model between camera pose and inter-frame rendering error and predicts error based on the new frame's camera pose, thus achieving more accurate frame rate adjustment strategies. In addition, the method also includes a lightweight scene recognition module that can adjust the error threshold according to the specific scene where the player is located, thereby adopting different degrees of frame rate adjustment strategies. Quantitatively compared with the prediction model that only considers historical frame errors, the proposed model improves the prediction accuracy on game frame sequences by more than 30%. At the same time, in the qualitative comparison of user experiments, under the same frame-skipping ratio, the proposed algorithm can achieve higher rendering quality and better user experience. The algorithm integrates historical frame errors and

* 基金项目: 国家自然科学基金 (61906090, U20B2064, 61773208); 江苏省自然科学基金 (BK20191287, BK20170809); 中央高校基本科研业务费 (30920021131); 中国博士后科学基金 (2018M632304)
收稿时间: 2023-09-27; 修改时间: 2023-11-07; 采用时间: 2024-01-15; jos 在线出版时间: 2024-04-29

camera information to predict more accurate future frame errors. It also combines prediction and scene recognition results to achieve better dynamic frame rate adjustment performance.

Key words: real-time rendering; power-optimization; adaptive frame rate adjustment

近年来, 移动设备的普及率逐渐提高, 伴随着图形处理器性能的不断增强. 移动游戏开发者充分利用图形处理器性能的提升, 通过增加场景模型的细节和外观渲染的复杂度, 实现更精美的游戏绘制效果, 以吸引更多用户. 然而, 这也导致移动图形处理器的性能始终无法满足开发者不断增长的需求. 另外, 移动设备屏幕分辨率也在逐年增加, 如 1080p (1920×1080) 已经成为标准配置, 而新款设备的分辨率已经达到 2K (2560×1440) 甚至 4K (3840×2160). 同时, 为了提升用户体验的流畅性, 移动设备的屏幕刷新率也不断提高, 从以前的 60 fps 增加到了 120 fps, 144 fps, 甚至 240 fps. 高分辨率和高刷新率使得每秒需要执行的像素着色次数显著提升, 而单次像素着色的计算成本随着场景和着色器的复杂度增加而上升. 这些因素都给绘制流水线带来了前所未有的压力. 与桌面设备不同, 移动设备缺乏稳定的电源供应, 其使用时间受到应用程序的能耗限制, 而游戏是影响移动设备能耗的主要因素.

从场景的输入到绘制结果的输出, 研究人员从不同角度提出了优化方法以降低游戏绘制流水线的能耗. 为了降低输入模型的复杂度, 细节层次结构技术对网格或着色器生成简化程度不同的版本, 绘制时根据摄像机距离选取对应层级[1–3], 从而在保证质量的前提下降低计算成本. 当模型输入到绘制流水线, 游戏中不同效果的参数对能耗产生显著影响, 例如高光质量, 阴影质量, 反走样质量等. 在这一领域, 研究者们也进行了广泛的研究[1–3], 提出了不同的自适应框架来调整绘制参数, 以实现高保真和低能耗的目标. 绘制流水线执行结束后, 生成了动态的绘制帧序列. 通过在屏幕空间上进行自适应采样, 可以降低单帧的像素着色率[4–6]. 而在时间上进行自适应采样则能够实现动态的帧率调整[5–9], 所有这些措施的目标都是在相同能耗下将重要的计算资源分配给关键信号.

本文提出了一种动态帧率调整方法, 用于改善移动游戏性能. 在游戏运行期间, 由于未来帧尚未渲染, 因此我们主要依赖历史帧的渲染结果进行预测. 然而, 采用线性方式来基于历史帧误差预测未来帧误差存在一定局限性. Hwang 等人提出的 RAVEN 框架[7]采用了此策略, 将误差值和预设阈值进行比较, 如果误差小于阈值, 则认为未来帧和当前帧相似度足够高, 可以跳过绘制. 这一策略在之前一代的简单移动游戏中表现良好, 这些游戏通常包含 2D 策略游戏, 即使是 3D 游戏, 其场景动态性较低, 因此线性误差预测方法仍然有效. 然而, 当前的大型多人在线竞技游戏 (MOBA) 具有复杂的动画和渲染效果, 简单的预测模型已经不再适用.

本文的研究针对当前流行的移动平台射击游戏提出了一种在节能模式下的动态帧率调整算法, 能够在节省能源的同时保证渲染质量. 具体而言, 我们提出了一种基于运动感知的误差预测模型, 通过建立相邻帧误差和摄像机位姿变化之间的非线性关系, 根据待绘制帧的新摄像机位姿外插出其渲染误差, 并根据误差判断是否可以跳过渲染该帧. 相较于仅考虑历史帧渲染误差的直接方法, 我们提高了预测准确性, 从而实现更精确的帧率跳过策略. 同时, 我们的方法还能够识别玩家角色所处的具体场景, 并根据不同场景自适应地调整误差阈值, 以实现不同程度的帧率降低策略. 综上所述, 本方法的主要贡献如下.

(1) 针对移动平台第一人称射击游戏的动态帧率调整算法;

(2) 运动感知的轻量级绘制误差预测模块;

(3) 快速场景预测模块, 能够根据场景动态调整跳帧阈值.

1 相关工作

1.1 实时绘制参数优化

游戏对场景的绘制包含多种不同效果, 例如阴影、基于物理的光照、反走样以及屏幕空间全局光照等. 每种效果都有各自可调节的参数, 用于平衡图形质量和性能. Wang 等人[10]在 2016 年提出了基于预计算的能源优化框架, 以八叉树的形式存储了场景中最优绘制参数对应的帕累托前沿, 并在实时绘制中进行查询和设置. 然而, 由于该框架需要长时间的预处理, 并且数据结构与静态场景紧密耦合, 因此在可扩展性和适用性方面存在明显的局限性. Zhang 等人[11]提出了一种在线能耗优化框架, 通过构建启发式的能耗和质量估计模型, 实时估算出最佳绘制参

数. 然而, 该方法的模型仍依赖于对特定视角和场景的实际测量数据, 存在一定局限性. Zhang 等人后续提出的 PowerNet[12]采用两个轻量级的神经网络, 分别用于估计特定参数下的绘制质量和绘制误差, 并根据能耗预算在实时绘制中选择最优的绘制参数组合. 这一方法在精度和图形质量方面相对于之前的方法都有所提升.

1.2 实时绘制自适应采样

近年来, 得益于硬件设备的支持, 出现了许多实时绘制的自适应算法, 这些算法能够动态调整帧率和屏幕空间分辨率. 在 NVIDIA 图灵架构[13]发布后, 硬件支持的可变着色率 (variable rate shading, VRS)[14]得到广泛应用. Yang 等人[4]提出的方法把前一帧绘制结果重投影到当前帧, 并通过计算信号频率来引导当前帧的绘制分辨率. Denes 等人[5]提供了一个同时考虑时空自适应采样的解决方案, 其模型评估了特定速度移动的物体在不同屏幕刷新率和分辨率下的视觉质量, 并在离线阶段预先优化了在不同数据带宽下的最佳刷新率和分辨率组合, 实时阶段则根据物体速度进行设置. Jindal 等人[6]在此基础上扩展了这一模型, 不仅考虑了自适应帧率, 还考虑了场景物体的纹理信息, 并支持了屏幕空间不同区域的适应性着色 (VRS). 然而, 这些方法都对硬件设备有一些特殊要求, 要求硬件能够在时间上支持屏幕动态刷新率 G-Sync 技术[9]或在空间上支持可变着色率 VRS 技术.

1.3 移动平台能耗优化

针对移动平台这一对能源消耗要求更高的设备, 前人已经进行了大量节省能耗的研究工作. Anttila 等人[15]提出了一种移动平台能耗模型, 该模型通过考虑绘制流水线的输入, 包括面片数、绘制批次数以及纹元总数, 来预测绘制的能源消耗. 然而, 这个模型需要与硬件紧密绑定, 因此需要为每个平台拟合一组专用的超参数. Choi 等人[16]提出的 LpGL 框架针对增强现实头戴式设备进行能耗优化. 他们通过封装图形接口 OpenGL 的绘制函数, 拦截应用层绘制命令并获取输入的网格模型和摄像机矩阵. 系统通过分析场景中物体的最大移动速度来调整绘制的帧率, 并根据物体在屏幕空间中的离心率 (Eccentricity) 自适应选择不同面片数的网格模型. Kim 等人[17]提出的框架根据相邻帧的 RGB 颜色差异来控制垂直同步频率, 进而在显示驱动层控制刷新率. 但该方法需要用户触屏信息的辅助才能达到较理想的效果, 当用户触屏时刷新率设置到最高, 在缺少该信息的情况下方法效果会有明显降低. Hwang 等人[7]发现在 RGB 颜色空间计算误差无法很好地反映相邻帧的视觉误差. 他们提出的 RAVEN 框架通过计算相邻帧在 YUV 色彩空间中的亮度误差, 更准确地逼近结构相似性度量准则 SSIM. 通过系数拟合, 他们得出了这两者之间的线性关系, 并用于调整绘制的帧率. Chang 等人[8]认为场景中物体的运动是影响人眼视觉的关键因素. 他们提出的 LSIM 相似度准则通过计算帧间物体在屏幕空间上的平移、旋转和缩放来衡量两帧之间的相似度. 然而, 无论是 RAVEN 还是 LSIM 都仅考虑了误差预测的部分因素, 前者只关注图像误差, 而后者则仅考虑了由场景仿射变换引起的误差, 这限制了这些模型对误差的预测能力. 我们提出的方法综合考虑了图像误差和摄像机位姿的影响, 通过构建非线性的预测模型来引导帧率调整, 在误差预测准确性上和帧率调整效果上都有所改进.

2 问题描述和算法概览

在上层应用的游戏线程中, 每个循环需要更新游戏逻辑, 包括虚拟物体的运动状态和位置、虚拟摄像机位置以及光源位置等. 应用把绘制请求发送给驱动层, 通常通过常见的图形接口, 例如 OpenGL (开放图形库)[18]. 驱动层负责实现图形接口的具体功能, 并将特定的 GPU 命令发送到硬件以执行实际的绘制操作. 我们的算法位于图形栈的驱动层, 在适当的时机, 我们决定跳过绘制命令以降低帧率. 当绘制命令被跳过后, 最终屏幕上的内容将保留上一次绘制的结果. 框架的时序逻辑如图 1 所示.

假设原始绘制序列为 $S=\{1,2,3,4,5,6,...\}$, 经过帧率调整后的序列为 $S'=\{1,1,3,4,4,6,...\}$, 则 $\{2,5,...\}$ 为被跳过的帧序列, 图 2 展示了这一样例序列. 图 3 呈现了我们方法的总览, 其中, 灰色方块代表算法模块, 算法被嵌入到游戏循环中, 在绘制命令产生后执行. 算法的输入包括 3 个部分: 当前帧 (第 N 帧) 的虚拟摄像机信息, 历史帧 (第 $N-K$ 帧, $K>0$) 的绘制误差及场景预测结果. 算法会综合不同模块的信息, 在经过快速计算后, 决定是否跳过绘制. 如果不能跳过, 则会继续执行正常的绘制流程, 调用接口将新的结果存储到帧缓存队列中. 如果可以跳过, 则会拦截所有绘制相关的接口, 中止 GPU 命令的传递, 并进入下一次游戏循环. 图中红色虚线框表示摄像机位姿提取

模块, 其输出包括虚拟摄像机的旋转速度和平移速度. 蓝色虚线框代表帧间误差计算模块, 它通过计算历史帧队列中相邻帧的绘制误差来推测未来帧的绘制误差. 黄色虚线框内的场景预测模块则通过分析已绘制结果的统计数据来判断当前玩家所处的场景. 在我们的方法中, 判断是否需要跳过绘制时, 会融合这三个模块的信息: 首先, 根据玩家角色所处的场景, 设置误差阈值. 然后, 根据摄像机位姿的变化和历史帧间的误差来预测新一帧的误差. 如果预测误差小于阈值, 就认为待绘制的帧与当前帧相似度很高, 允许跳过绘制, 直接重用当前帧的结果. 反之, 则继续执行正常的绘制流程. 在实际实现中, 为了避免预测误差过大, 我们最多只允许连续跳过一帧绘制. 如果第 N 帧被跳过, 那么必须执行第 $N+1$ 帧的绘制.

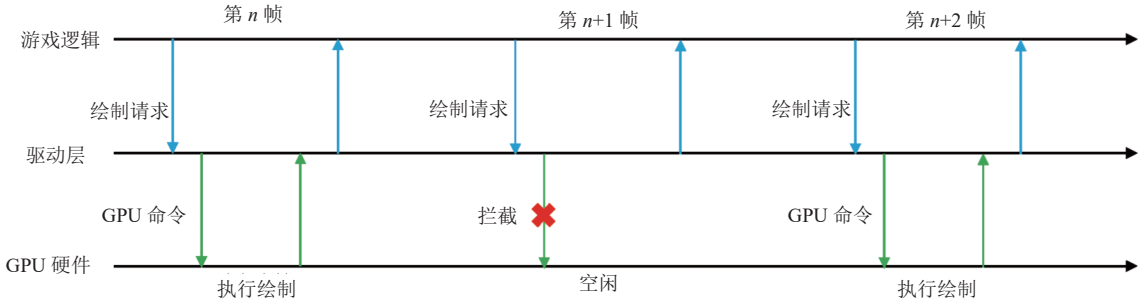


图 1　绘制命令时序图

图 2　跳帧样例序列

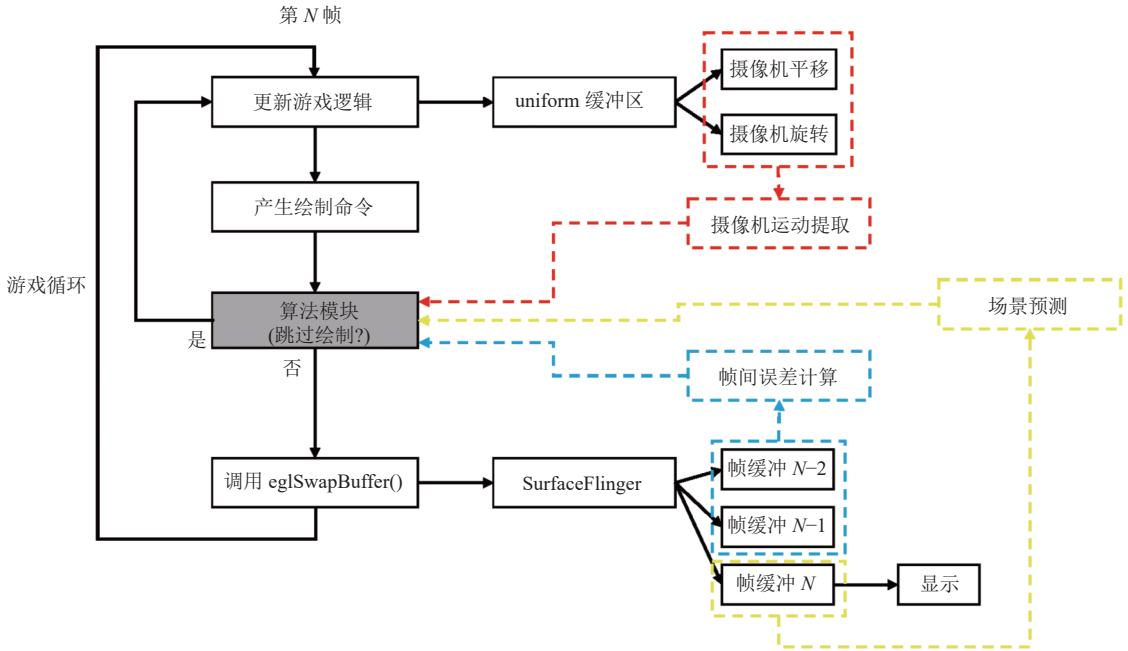


图 3　方法总览

3 运动感知的帧间误差预测

我们的方法专注于第一人称射击游戏. 在这类游戏中, 射击准心始终位于屏幕中心. 因此, 玩家的视线焦点通常集中在屏幕中心或其附近区域. 我们借鉴了凝视点绘制 (foveated rendering) 技术[19]中得出的人眼在屏幕空间敏感度分布模型, 以此来计算帧间误差时对不同屏幕区域进行加权. 如图 4 所示, 算法把屏幕空间划分为 3 个部分, 分别赋予不同的权重, 如下所示:

$$w(x,y)=\begin{cases} W_0, & dis(x,y)<r_0 \\ W_0\times\dfrac{r_1-dis(x,y)}{r_1-r_0}+W_1\times\dfrac{dis(x,y)-r_0}{r_1-r_0}, & r_0\leqslant dis(x,y)<r_1 \\ W_1, & \text{其他} \end{cases} \tag{1}$$

其中, W_0 和 W_1 分别表示最内层 (红色区域) 和最外层 (绿色区域) 的权重, 而 r_0 和 r_1 分别表示内层半径和外层半径. 位于过渡区域 (黄色区域) 的像素的权重会根据其距离屏幕中心的位置进行线性插值. Hwang 等人的工作[7]表明, 两帧绘制结果的亮度误差和结构性相似度 SSIM[20]存在较强的线性相关关系. 因此, 将亮度误差作为衡量两帧图像相似性的指标在一定程度上符合人眼的视觉感知. 图 5 展示了不同绘制序列的帧间亮度误差 Y_Diff 和结构相似性准则 SSIM 之间的关系: 结构相似度越高, 亮度误差越低. 同时, 两者呈现较高的线性相关关系. 图中不同颜色的数据来自不同绘制序列, 包括角色在公路和草地场景中行走以及角色驾驶汽车穿越公路和草地场景.

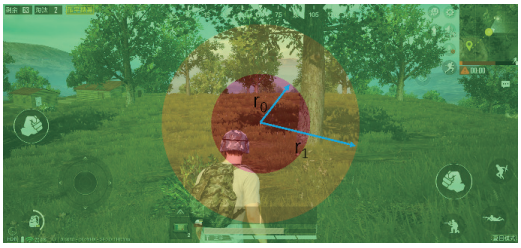


图 4 按凝视点距离计算权重

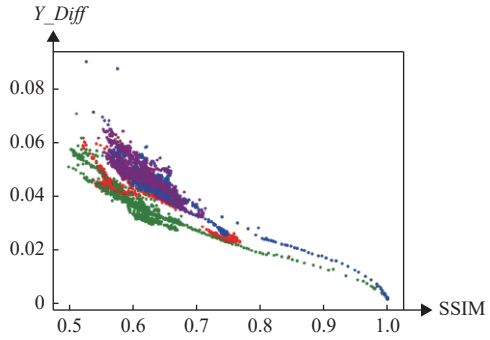


图 5 亮度误差和结构相似性的关系

我们使用如下公式计算相邻帧间的绘制误差:

$$Err(i,i+1)=\sum_{(x,y)\in\Omega} w(x,y)\times|L_i(x,y)-L_{i+1}(x,y)| \tag{2}$$

其中 Ω 表示屏幕空间, $L_i(x,y)$ 表示第 i 帧像素 (x,y) 的亮度值. 我们根据以下公式加权红、绿、蓝这 3 个通道得到亮度:

$$L=0.2126\times R+0.7152\times G+0.0722\times B \tag{3}$$

同时, 我们观察到, 在局部时间内, 当两帧之间的虚拟摄像机位姿变化较大时, 它们之间的帧间绘制误差也更大. 因此, 在较短的时间窗口内, 我们可以利用摄像机平移和旋转的速度与绘制误差之间的正相关关系, 来预测新一帧的绘制误差. 基于这一观察, 我们使用以下公式作为预测模型:

$$\frac{Err(i,i+1)}{(k_2\,\Delta\,T_{i,i+1}+\Delta\,R_{i,i+1}+eps)^{k_3}}=k_0\times\frac{Err(i-1,i)}{(k_2\,\Delta\,T_{i-1,i}+\Delta\,R_{i-1,i}+eps)^{k_3}}+k_1 \tag{4}$$

其中 $\Delta T_{i,i+1}$ 和 $\Delta R_{i,i+1}$ 分别表示第 i 帧和第 $i+1$ 帧之间的摄像机位移变化和朝向变化. k_0,k_1,k_2,k_3 都是模型超参数, 其中 k_0、k_1 用于描述相邻误差的线性关系, k_2 调控位移速度和朝向变化的权重, k_3 的作用是增加模型的非线性表达能力.

当系统即将绘制第 $i+1$ 帧时, 其摄像机信息在绘制前的更新游戏逻辑阶段已经被写入到 uniform 缓冲区中. 因此我们能够在实际绘制之前获取这些信息, 从而把其代入模型并预测得到该帧的绘制误差, 进而决定是否能够跳过该帧绘制.

4 快速场景识别

为了使算法能够适应游戏中的不同场景, 我们在执行帧率调整策略时考虑了场景因素, 并针对不同场景采用了不同的绘制误差阈值. 我们观察到, 用户对于游戏室内场景和室外场景的帧率敏感度存在差异. 当角色处于室外时, 用户对于绿色为主的场景, 如草地和森林等, 降低帧率的容忍度较高. 因此, 我们根据场景的特征将其分成 3 种情况, 并为每种情况选择特定的绘制误差阈值, 以执行不同程度的帧率调整操作.

具体而言, 在预处理阶段, 我们分析不同场景的绘制帧数据, 设计相应的场景识别模型, 并优化其超参数. 在游戏实际运行时, 算法根据绘制图像计算统计数据, 并将这些数据代入模型中以获得场景识别结果. 在实际实现中, 我们采用了轻量级线性模型作为二分类器, 用于执行场景识别操作, 以判断当前角色是否位于特定场景之中. 具体操作如下所示:

$$y_i = w_S^T \cdot x_i \tag{5}$$

其中 $x_i =< x_{i,0}, x_{i,1}, x_{i,2}, ..., x_{i,n-1}, 1 >$ 是第 i 帧绘制结果的统计数据. $w_s =< w_0, w_1, w_2, ..., w_{n-1}, w_n >$ 是场景 S 对应的识别器参数. 两者都是 $n+1$ 维向量, 最后一维的存在是为了考虑常数项. 模型的输出 y_i 是标量, 用来表示场景识别的结果. 当其值大于 0 时, 表示玩家角色处于场景 S 中, 否则不处于场景 S 中.

在判断角色第 i 帧处于室内还是室外时, 我们取第 i 帧之前 m 帧 ($m = 50$) 窗口内的绘制误差进行统计并计算其方差 v_i , 公式如下:

$$v_i = \sum_{j=i-m}^{i} [Err(j, j+1) - mean(Err)]^2 \tag{6}$$

当玩家处于室内时, 其搜索道具和探索房间的行为会导致虚拟摄像机位姿变化的情况与在室外情况有所不同. 相比于室外情况, 室内情况下绘制误差在时间上的方差更大. 我们通过对玩家的游戏数据进行分析发现, 这个值的分布在室内和室外可以大致通过一个阈值进行区分. 图 6 展示了 5 个序列中绘制误差的方差分布情况, (a)–(c) 是玩家在室内搜索的绘制序列, (d)、(e) 是室外绘制序列, 包括玩家在室外跑步和开车两种不同行为. 可以明显看到, 室内和室外的方差分布大致可以通过绿线所示阈值来区分. 对于室内外场景识别器, 我们取 $x_i =< v_i, 1 >$ 作为预测模型的输入, $w_{\text{indoor}} =< w_0, w_1 >$ 作为模型参数.

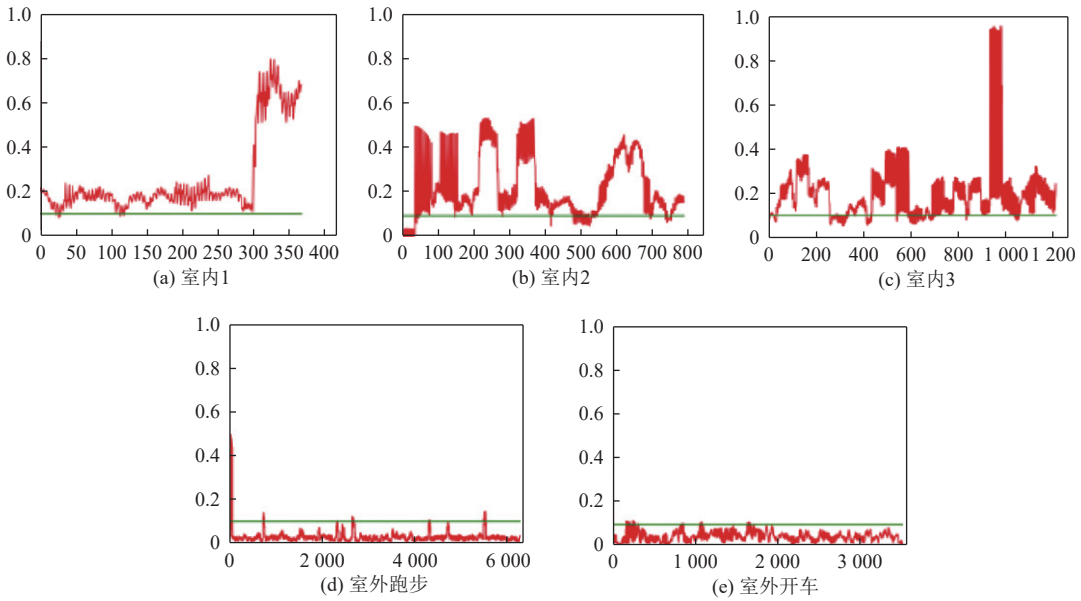
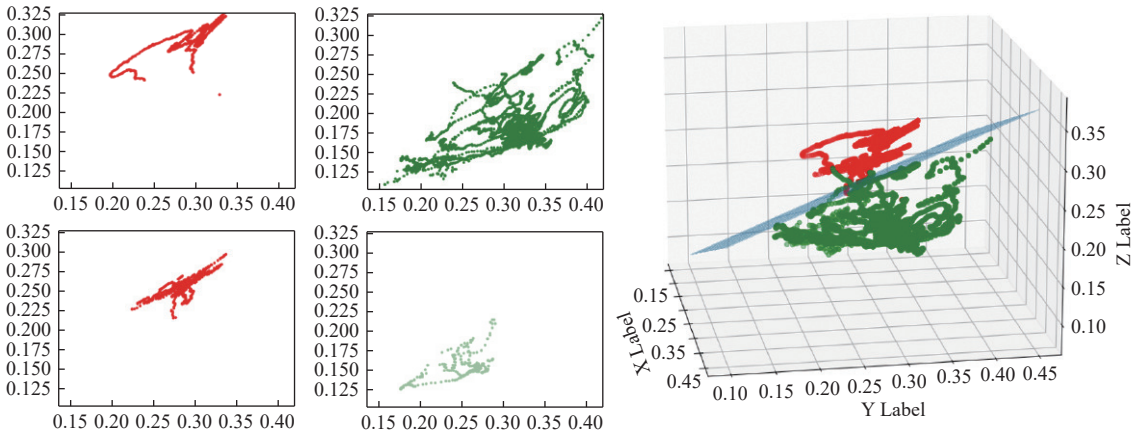


图 6 绘制误差在时序上的方差

当判断角色在第 i 帧处于室外草地场景还是室外非草地场景时, 我们截取屏幕空间特定采样窗口并计算其平均颜色 c_i :

$$c_i = \frac{1}{W \times H} \sum_{j \in \Omega} color_j \tag{7}$$

其中, j 是采样窗口 Ω 中的像素下标, W 和 H 分别是采样窗口的长和宽. 我们观察到, 当角色处于草地或森林场景时, 绘制序列对应的 c_i 在分布上能明显区别于除此以外的其他场景. 如图 7(a) 是两个数据集在采样窗口中计算的平均颜色分布, 上方是室外驾驶场景绘制序列, 下方是室外跑步数据集. 其中红色表示在草丛/森林中的数据点, 绿色表示除此之外的场景数据点, 每个数据点表示某一帧在采样窗口中平均颜色的 RG 通道. 图 7(b) 是所有数据点在三维上的可视化, 可以看到, 草丛/森林与其他区域的平均颜色近乎线性可分, 因此, 我们能够根据视窗的平均颜色在平面的哪一侧来判断当前所处场景. 对于草地/森林场景识别器, 我们取 $x_i =< c_i, 1 >$ 作为模型输入, $w_{\text{grass}} =< w_0, w_1, w_2, w_3 >$ 作为模型参数.



(a) 两个数据集在采样窗口中计算的平均颜色分布 (b) 所有数据点在三维上的可视化

图 7 采样视窗平均颜色分布

为了优化上述两个线性预测模型的参数, 我们使用感知器学习算法 (perceptron learning algorithm) 的一种变体: 即口袋算法 (pocket algorithm)[21], 通过迭代计算来获得最优参数 w_s . 下文图 8 展示了算法伪代码. 算法首先将模型参数初始化为随机值. 对于室内外场景识别器, 其参数维度为 2, 对于草地/森林识别器, 其参数维度为 4. 在每次迭代中, 算法会随机抽取训练集中的一个样本和相应的标签, 测试当前参数的预测结果并对参数进行修正. 同时, 它会在整个数据集上计算更新后参数的预测正确率, 并保留最优参数. 经过足够多次迭代后, 算法将得到接近最优的参数设置. 表 1 和表 2 分别记录了两个场景预测器的模型参数.

表 1 草地、森林场景识别器

参数	w_0	w_1	w_2	w_3
取值	6.6	−6.13	18.84	−0.8

表 2 室内外识别器

参数	w_0	w_1
取值	0.1	1.0

5 基于预测的帧率调整

帧率调整模块根据场景识别模块的结果来动态调整误差阈值, 并根据误差预测的结果来判断是否可以跳过绘制. 图 9 展示了算法的设置阈值过程. 算法首先执行室内外场景判定, 当角色处于室内时设置误差阈值为 t_{indoor} . 当处于室外时, 算法进一步判定角色是否处于草地/森林场景. 如果判定为真, 则将阈值设置为 t_{grass} , 否则将阈值设置为 t_{others} . 在用户实验中我们发现, 当角色处于室内或角色处于室外草地/森林时, 用户对帧率降低的敏感度更低, 我

们根据经验值把这两个场景的阈值设置得更宽松, 阈值间的关系如下:

$$\begin{aligned} t_{\text{indoor}} &= 1.5 \times t_{\text{others}} \\ t_{\text{grass}} &= 1.5 \times t_{\text{others}} \end{aligned} \tag{8}$$

另外, 算法在执行时可能处于两种不同状态: 1) 无跳帧状态, 即算法输入两帧在原始序列中相邻. 2) 已经处于跳帧状态, 即算法输入两帧在原始序列中相隔一帧. 跳帧状态变化如图 10 所示. 为了确保算法在这两种输入状态下都能够预测出最佳结果, 我们分别对这两种状态进行了预测模型参数的优化. 当输入状态为无跳帧时, 我们记预测模型参数为 $<k_0^{\text{noskip}},k_1^{\text{noskip}},k_2^{\text{noskip}},k_3^{\text{noskip}}>$, 而当输入已经处于跳帧状态时, 我们记预测模型参数为 $<k_0^{\text{skip}},k_1^{\text{skip}},k_2^{\text{skip}},k_3^{\text{skip}}>$. 表 3 和表 4 分别列出了这两种状态下预测模型的参数取值.

Algorithm 1 口袋感知学习算法 (pocket-pla)

```
iterations = Max_Iter // 设置最大迭代次数
// 初始化帧数据集和对应标签。
dataset = {x_1, x_2, ..., x_k}, labels = {y_1, y_2, ..., y_k}
// 随机初始化识别器参数, 维度为 n。
w = rand(n), best_w = rand(n)
best_num = CALC_FALSE(dataset, labels, w)
i = 0
while i ≤ iterations do
    rand_idx = rand(len(dataset)) // 随机抽取一个样本
    cur_data = dataset[rand_idx], cur_label = labels[rand_idx]
    cur_y = Dot(cur_data, w)
    // 遇到预测错误的样本, 修正参数 w
    if sign(cur_y) != cur_label then
        w = w + cur_data × cur_label
        // 计算当前参数下预测错误的样本数
        cur_false_num = CalcFalse(dataset, labels, w)
        // 更新预测结果最好的参数
        if cur_false_num < best_num then
            best_num = cur_false_num
            best_w = w
        end if
    end if
end while
return best_w
```

图 8 口袋感知学习算法伪代码

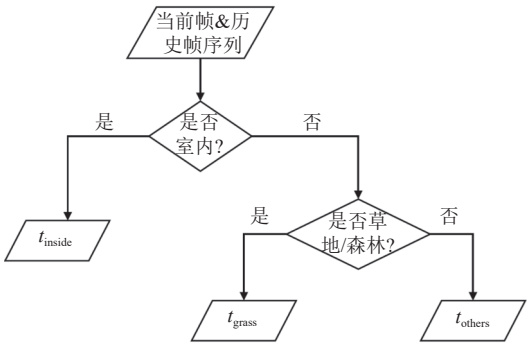


图 9 根据场景识别结果设置阈值

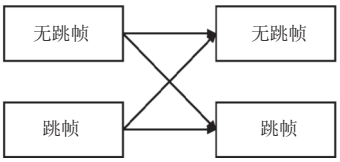


图 10 跳帧状态转换

表 3 跳帧状态模型参数

跳帧状态	k_0^{skip}	k_1^{skip}	k_2^{skip}	k_3^{skip}
参数值	0.948	0.000 103 6	487.15	0.257

表 4 不跳帧状态模型参数

不跳帧状态	k_0^{noskip}	k_1^{noskip}	k_2^{noskip}	k_3^{noskip}
参数值	0.866	9.354	164.03	0.294

6 实 现

6.1 误差计算带宽优化

算法在计算帧间误差时, 需要频繁地把相邻两帧 SurfaceFlinger 中的缓冲转换为 GPU 纹理, 并在片元着色器中读取. 当帧缓冲的分辨率很高时会造成很大的 IO 开销, 同时也会造成能耗提升. 为了缓解着色器访问显存的开销, 在计算绘制误差时我们只读取 1/4 分辨率的帧缓存, 即在横向和纵向上每隔一个像素读取一次, 从而把带宽降低到 1/4. 同时, 为了在时间上覆盖到整个屏幕空间, 我们采用抖动 (jitter) 的方式每 4 帧遍历一次像素的 2×2 邻域. 为了缓解帧缓冲转化为 GPU 纹理的开销, 我们预先在 GPU 中创建两个纹理缓冲区, 每次新的绘制结果到来时只更新到原来旧的纹理, 并把原来两个纹理缓冲区的新旧标签对换. 这样可以减少纹理更新的次数.

6.2 流水线状态保护

由于我们的算法实现在绘制流水线的驱动层, 处于游戏绘制的 OpenGL 上下文中, 而算法的执行也需要运行一次完整绘制批次 (render pass). 为了不影响游戏的运行, 我们需要在运行前后进行状态保存和状态恢复. 表 5 列举了需要保存和恢复的状态. 执行帧间误差计算前, 我们先调用每个状态对应的“glGet*”函数获取游戏当前状态值并保存在状态表中, 执行完误差计算后调用对应的“glSet*”函数进行状态恢复.

表 5 OpenGL 状态

状态	含义
VIEWPORT	绘制视窗大小
CURRENT_PROGRAM	当前绑定的绘制程序编号
TEXTURE_BINDING_2D	当前绑定纹理编号
VERTEX_ARRAY_BUFFER	当前绑定顶点属性数组
DRAW_FRAME_BUFFER	当前绑定帧缓存

6.3 虚拟摄像机信息获取

由于算法作用在驱动层且我们期望对上层应用透明, 因此无法直接通过上层接口得到虚拟摄像机的状态信息. 我们利用帧分析工具 MGD (mali graphic debugger) 抓取并分析帧序列. 针对游戏《和平精英》当时的版本, 我们发现在每帧对应的循环中, 顶点着色器的两个数据块有所更新. 第 1 个数据块存储该帧场景中所有物体共享的数据, 每帧只更新一次. 第 2 个数据块存储场景中每个物体的数据, 每绘制一个物体更新一次. 预测模型只需要获取全局的摄像机位姿, 因此我们定位到第 1 个数据块中绑定摄像机矩阵的命令, 在游戏对矩阵缓冲更新完毕后从中获取信息.

7 验 证

7.1 预测准确性

与仅使用历史帧误差来预测未来帧误差的方法[7]相比, 我们的预测模型考虑了摄像机位姿变化, 通过新一帧的摄像机位姿估计出其绘制结果的误差, 以实现更准确的预测. 图 11 对比了两个预测模型在帧序列上的预测误差. 蓝色曲线对应的预测模型仅考虑历史帧误差 (不考虑摄像机位姿), 而黄色曲线对应我们的模型. 由于前者完全没有未来帧的信息, 它完全依赖于历史帧信息来进行预测, 因此导致更大的预测误差. 而后者则考虑了新一帧的信息, 利用局部的线性关系进行外推, 从而有效降低了预测误差. 表 6 通过两个指标来展示我们方案的优越性. 第 1

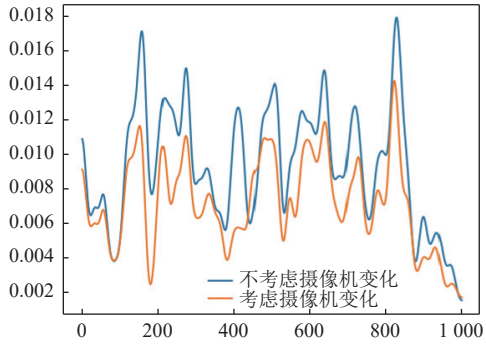


图 11 帧序列预测误差对比

表 6 两种方案预测数值对比

指标	两种方案	
	不考虑摄像机	考虑摄像机
相关系数	0.894	**0.933**
预测误差	0.000 21	**0.000 13**

个指标是相关系数, 用来描述历史帧误差和未来帧误差的相关性. 对于我们的方案, 根据公式的方式, 使用摄像机位姿变化量对亮度误差进行缩放, 并计算缩放后误差的相关系数. 对于不考虑摄像机位姿的方案, 我们只计算原始亮度误差的相关系数. 第 2 个指标是模型的预测误差, 我们对比了同一帧序列下两种方案的预测误差.

如图 10 所示, 算法执行时可能处于跳帧和非跳帧两种状态, 我们使用与特定状态对应的模型参数来预测绘制误差. 表 7 中展示了匹配状态的模型优势. 当采用通用模型时, 无论算法处于何种状态都使用相同的参数. 由于两种状态下输入两帧的误差分布不同, 通用模型无法灵活适配不同状态, 因此其预测准确性会降低. 另外, 当模型考虑摄像机因素, 其预测误差受输入状态的影响更小.

7.2 用户实验

为了验证算法效果需要进行用户实验. 我们使用了基于麒麟 980 芯片的移动设备, 并在其上进行了《和平精英》游戏的对比实验. 实验涵盖了 8 名测试人员, 其中包括两名专门负责游戏产品质量的专业测试人员, 其余 6 名测试人员具有第一人称射击游戏经验, 其中两名是《和平精英》的玩家. 实验包含 3 轮, 分别测试了以下 3 种方案: (1) 不考虑摄像机的方案[7] (仅使用历史帧误差作为模型输入), (2) 考虑摄像机位姿但不考虑场景的方案, (3) 完整方案: 考虑摄像机以及场景. 在实验开始前, 我们对每种方案误差阈值进行设置, 使得 3 个方案在相同路径下跳帧比例约为 20%.

在每一轮实验中, 测试者被要求进行两次游戏场景的探索, 第 1 次运行原始游戏版本, 第 2 次运行开启了跳帧算法的游戏版本. 在每次探索中, 测试者在场景中按指定路线行动, 以覆盖室内和室外场景以及公路和森林场景. 每一轮测试结束后, 测试者被要求提供分数反馈, 其中, 3 分表示测试者认为两次探索之间没有明显区别. 2 分表示测试者能够区别出开启算法的游戏版本, 但效果可以接受. 1 分表示测试者无法接受开启算法的游戏效果.

实验结果如表 8 所示. 相较于只考虑历史帧间误差的方案 (第 2 列), 我们改进的预测模型额外考虑了未来帧的摄像机位姿, 能够根据玩家实际操作预测出更准确的误差, 从而提升用户体验. 得益于场景识别模型的辅助, 算

表 7 跳帧状态下不同预测模型误差对比

模型	不考虑摄像机	考虑摄像机
通用模型	2.92×10^{-4}	1.68×10^{-4}
跳帧模型	2.54×10^{-4}	1.66×10^{-4}

表 8 用户实验结果得分

实验轮次	不考虑摄像机	考虑摄像机	考虑摄像机和场景
普通1	3	3	2
普通2	2	3	3
普通3	3	3	3
普通4	2	3	3
普通5	3	2	3
普通6	2	3	3
专业1	1	2	2
专业2	1	1	2
总分	17	20	**21**

法能够根据不同场景自适应调整阈值以改变跳帧策略. 因此, 完整方案在实验中获得了最高的总得分.

7.3 能耗收益

我们在麒麟 980 专用的能耗测量模块上测试了游戏《和平精英》在 60 fps 下开启算法的能耗收益. 测试结果如表 9 所示. 开启算法和原始流程采用完全相同的游戏回放. 能耗主要收益来自于 GPU, 能实现接近 70 mAh 时的能耗降低. 此外, 由于拦截了绘制命令, CPU 与 GPU 之间的通信也减少, 因此 CPU 能耗也有一定的下降.

表 9 游戏测试的能耗收益结果 (mAh)

	原始流程	开启算法
CPU功率	97.7	93
GPU功率	252.1	186.8
总功率	349.8	279.8

8 结 论

本文提出了一种基于运动感知的动态帧率调整算法. 与过去仅考虑帧间绘制误差的预测模型相比, 本方法在预测模型中创新地考虑了摄像机信息. 通过建模未来帧和历史帧之间的绘制误差与摄像机位姿之间的非线性关系, 该算法能够更准确地预测未来帧的误差, 从而实现更精确的帧率调整. 此外, 为了使算法能够根据游戏中不同的场景自适应调整, 我们还引入了场景识别模块, 通过轻量级的线性预测模型获得场景识别结果, 以辅助算法进行阈值调整. 从实验结果来看, 我们的方法相较于之前的预测模型具有更好的用户体验. 尽管仍然有一些测试人员能够识别出开启算法的游戏版本, 但我们的算法旨在节省能耗, 最终可作为游戏的一种节能模式运行. 例如, 在用户手机电量不足时, 用户可以主动选择启用节能模式来运行游戏, 在牺牲尽可能少的绘制质量的同时, 实现能源的有效节约.

References:

[1] Wang R, Yang XJ, Yuan YZ, Chen W, Bala K, Bao HJ. Automatic shader simplification using surface signal approximation. ACM Trans. on Graphics (TOG), 2014, 33(6): 226. [doi: 10.1145/2661229.2661276]

[2] Liang YZ, Song Q, Wang R, Huo YC, Bao HJ. Automatic mesh and shader level of detail. IEEE Trans. on Visualization and Computer Graphics, 2023, 29(10): 4284–4295. [doi: 10.1109/TVCG.2022.3188775]

[3] Garland M, Heckbert PS. Surface simplification using quadric error metrics. In: Proc. of the 24th Annual Conf. on Computer Graphics and Interactive Techniques. New York: ACM Press/Addison-Wesley Publishing Co., 1997. 209–216. [doi: 10.1145/258734.258849]

[4] Yang L, Zhdan D, Kilgariff E, Lum EB, Zhang YB, Johnson M, Rydgård H. 2019. Visually lossless content and motion adaptive shading in games. In: Proc. of the ACM on Computer Graphics and Interactive Techniques, 2019, 2(1): 6. [doi: 10.1145/3320287]

[5] Denes G, Jindal A, Mikhailiuk A, Mantiuk RK. A perceptual model of motion quality for rendering with adaptive refresh-rate and resolution. ACM Trans. on Graphics, 2020, 39(4): 133. [doi: 10.1145/3386569.3392411]

[6] Jindal A, Wolski K, Myszkowski K, Mantiuk RK. 2021. Perceptual model for adaptive local shading and refresh rate. ACM Trans. on Graphics, 2021, 40(6): 281. [doi: 10.1145/3478513.3480514]

[7] Hwang C, Pushp S, Koh C, Yoon J, Liu YX, Choi S, Song J. RAVEN: Perception-aware optimization of power consumption for mobile games. In: Proc. of the 23rd Annual Int'l Conf. on Mobile Computing and Networking. Snowbird: Association for Computing Machinery, 2017. 422–434. [doi: 10.1145/3117811.3117841]

[8] Chang YC, Chen WM, Hsiu PC, Lin YY, Kuo TW. LSIM: Ultra lightweight similarity measurement for mobile graphics applications. In: Proc. of the 56th Annual Design Automation Conf. Las Vegas: Association for Computing Machinery, 2019. 25. [doi: 10.1145/3316781.3317856]

[9] Nvidia. G-sync high dynamic range. 2017. https://images.nvidia.com/content/gsync/pdf/g-sync-hdr-whitepaper_v01.pdf

[10] Wang R, Yu BW, Marco J, Hu TL, Gutierrez D, Bao HJ. Real-time rendering on a power budget. ACM Trans. on Graphics, 2016, 35(4): 111. [doi: 10.1145/2897824.2925889]

[11] Zhang YJ, Ortin M, Arellano V, Wang R, Gutierrez D, Bao HJ. On-the-fly power-aware rendering. Computer Graphics Forum, 2018, 37(4): 155–166. [doi: 10.1111/cgf.13483]

[12] Zhang YJ, Wang R, Huo YC, Hua W, Bao HJ. PowerNet: Learning-based real-time power-budget rendering. IEEE Transactions on Visualization and Computer Graphics, 2022, 28(10): 3486–3498. [doi: 10.1109/TVCG.2021.3064367]

[13] NVIDIA. NVIDIA turing GPU architecture. 2018. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[14] Swaroop Bhonde. Turing variable rate shading in VRWorks. 2018. https://devblogs.nvidia.com/turing-variable-rate-shading-vrworks/

[15] Vatjus-Anttila JM, Koskela T, Hickey S. Power consumption model of a mobile GPU based on rendering complexity. In: Proc. of the 7th Int'l Conf. on Next Generation Mobile Apps, Services and Technologies. Prague: IEEE, 2013. 210–215. [doi: 10.1109/NGMAST.2013.45]

[16] Choi J, Park H, Paek J, Balan RK, Ko J. LpGL: Low-power graphics library for mobile AR headsets. In: Proc. of the 17th Annual Int'l Conf. on Mobile Systems, Applications, and Services. Seoul: Association for Computing Machinery, 2019. 155–167. [doi: 10.1145/3307334.3326097]

[17] Kim D, Jung N, Chon Y, Cha H. Content-centric energy management of mobile displays. IEEE Trans. on Mobile Computing, 2016, 15(8): 1925–1938. [doi: 10.1109/TMC.2015.2467393]

[18] Woo M, Neider J, Davis T, Shreiner, D. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2. Reading: Addison-Wesley Longman Publishing Co., Inc., 1999.

[19] Wang LL, Shi XH, Liu Y. Foveated rendering: A state-of-the-art survey. Computational Visual Media, 2023, 9(2): 195–228. [doi: 10.1007/s41095-022-0306-4]

[20] Wang Z, Bovik AC, Sheikh HR, Simoncelli EP. Image quality assessment: From error visibility to structural similarity. IEEE Trans. on Image Processing, 2004, 13(4): 600–612. [doi: 10.1109/TIP.2003.819861]

[21] Gallant SI. Perceptron-based learning algorithms. IEEE Trans. on Neural Networks, 1990, 1(2): 179–191 [doi: 10.1109/72.80230]

梁宇智(1993－), 男, 博士生, 主要研究领域为图形绘制中的自适应算法.

王锐(1978－), 男, 博士, 教授, CCF 专业会员, 主要研究领域为计算机图形学, 实时绘制, 真实感绘制, 基于 GPU 的计算以及三维显示技术.

霍宇驰(1987－), 男, 博士, 副教授, 主要研究领域为计算机图形学, 计算机视觉, 机器学习, 计算光学.