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）
21301057	于蕊宁	7	27	17	37	88

北京交通大学

计算机图形学中的高洛德着色技术及其应用

计算机图形学课程论文

学　　院：软件学院

专　　业：软件工程

学生姓名：于蕊宁

指导教师：吴雨婷

北京交通大学

2024 年 6 月

目录

摘要

本文是计算机图形学课程实践的总结和延申，总结了所学到的知识和本人感兴趣的着色器技术，选取第二次实验中组内没有使用的高洛德着色技术（Gouraud Shading）为主要研究目标。

高洛德着色技术是一种通过在顶点计算光照并在多边形表面插值颜色的方法，显著提高了图像的平滑度和真实感。

本文首先介绍计算机图形学的发展历史和 3D 建模、渲染和光照模型等基本概念。作为计算机科学的一个重要分支，计算机图形学对现代影视、游戏等领域产生了深远的影响。

其次，详细讨论了高洛德着色的基本原理，其中重点讲解顶点光照计算和颜色插值方法，并附代码和图片展示该算法的具体实现步骤。

此外，本文还探讨和对比了高洛德着色在视频游戏和电影中的实际应用，通过案例分析展示其在游戏和电影实时渲染中的优势。也在这个基础上分析了高洛德着色的局限性，比如在处理高光和复杂光源场景时的不足。

最后，本文展望了高洛德着色技术的未来发展方向，主要分为与其他着色技术的结合、在基于物理的渲染中的应用、实时渲染技术和 AI 在图形学中的前景四部分。

关键词：高洛德着色，计算机图形学，实时渲染，颜色插值，顶点光照

引言

背景介绍

计算机图形学作为计算机科学的一个重要分支，对我们的生活产生了深远的影响[2]。电影效果和游戏场景的应用使人们能够在虚拟世界中进行可视化和互动。计算机图形学研究如何使用计算机技术生成、处理和显示图形。随着硬件和算法技术的进步，图形学不断打破传统限制，提供更丰富、更逼真的视觉效果。

3D 技术是计算机图形的重要组成部分，我也对课堂上讲的着色器（Shader）部分非常感兴趣，我选取了小组第二次实验没有用到的高洛德着色法进行深入研究。Gouraud 着色是一种有效的着色方法，它可以计算顶部多边形照明的效果，并扭曲表面的颜色，以实现平稳的颜色过渡。与着色平面相比，高洛德着色提供了更准确、更逼真的视觉效果，以及更高的计算效率。虽然高洛德着色有几十年的历史，但它在某些场景中具有不可替代的优势。

本文将探讨高洛德着色技术的原理、实现方法、优缺点，并结合具体应用案例，分析其在现代图形学中的实际效果。通过研究高洛德着色技术，可以更好地理解其在图形渲染中的重要作用及其未来发展潜力和应用前景。

研究目的

计算机图形学是现代计算机科学中的重要领域，广泛应用于多个行业[1]。高洛德着色作为一种重要的渲染技术[7]，通过在顶点计算光照并在多边形表面插值颜色，显著提高了图像的平滑度和真实感。以下是本人的研究目的，也是本论文的文章结构：

讨论实施原理和方法：详细分析顶点光照计算和颜色插值的工作机制。

评估高洛德着色的优缺点：将高洛德着色的优缺点与其他着色技术进行比较[6]，评估其应用和局限性。

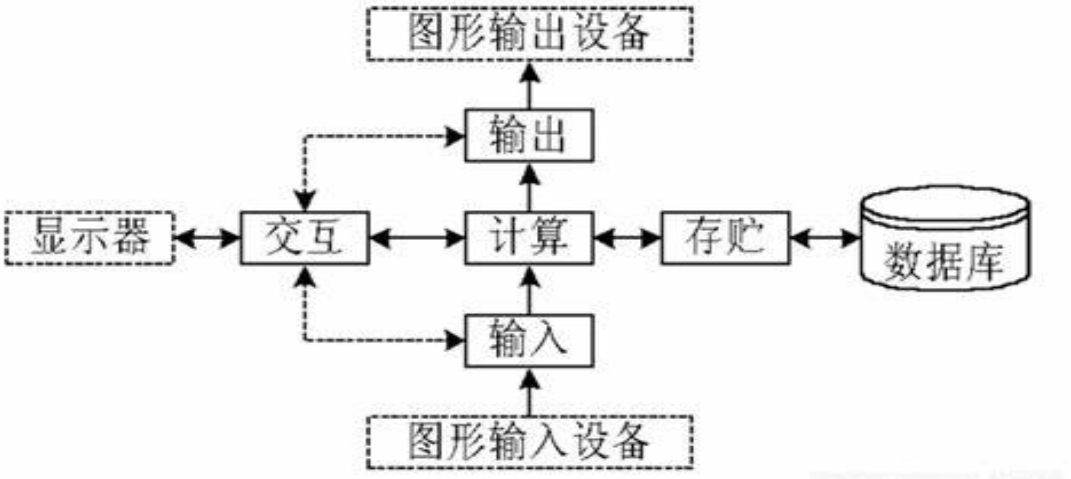
案例分析与应用：讨论在视频游戏、电影制作中的实际应用的结果，分析其在这些领域的重要性和贡献。

讨论未来的发展方向：结合当前图形处理技术的发展，展望高洛德着色在未来图形学中的潜力和应用前景。

第一章：基础梳理

计算机图形学的定义与发展

计算机图形学是研究如何通过计算机生成、操作和显示图形图像的学科（如图 1-1），其核心目标是将数字信息转化为可视化的图像。本部分是本人对于这门学科和课堂内容的总结和个人理解。



1-1

基本概念：3D 建模、渲染、光照模型

3D 建模：使用计算机软件创建三维物体。3D 模型由顶点、边和面的集合组成，表示复杂的形状和结构。

渲染：将 3D 模型转换为二维图像，包括计算光照、颜色、纹理等信息。分为实时渲染和离线渲染两种。

光照模型：用于模拟光线与物体表面相互作用的数学模型。通过计算环境光、漫反射光和镜面反射光的贡献，生成逼真的光照效果[1]。

图形渲染管线概述

图形渲染管线是图形处理单元（GPU）用于将 3D 场景转换为二维图像的基本步骤如下[1]：

顶点处理：这一阶段分为顶点着色、几何变换和视图变换三部分。计算出每个顶点的颜色和光照效果后，进行的几何变换是将模型坐标转换为世界坐标，视图变换是将世界坐标转换为视点坐标 。

图元组装：顶点被组合成的基本图元，是构成 3D 物体的基本结构。

光栅化：将图元转换为像素。首先将图元投影到二维屏幕上，然后确定哪些像素被图元覆盖。

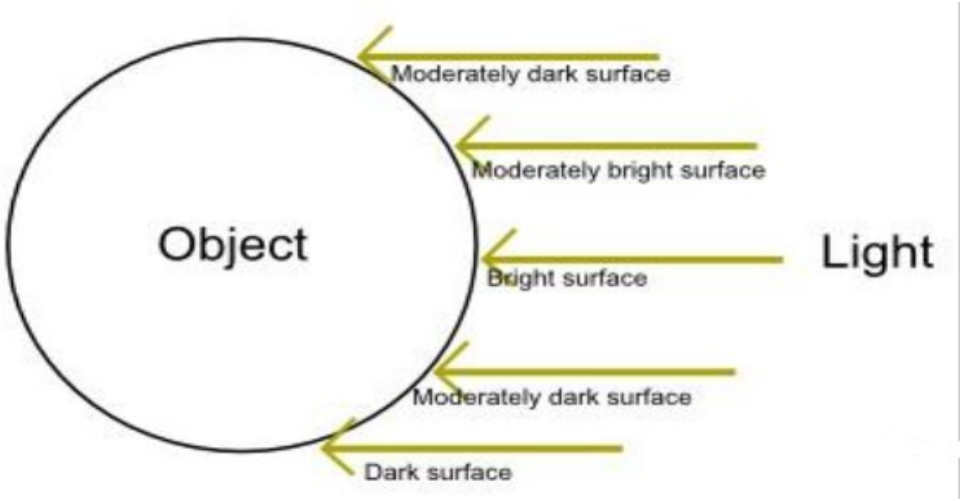
片段着色：计算每个像素的最终颜色值，如纹理映射、光照计算和颜色插值等操作。

输出合成：经过处理的像素被写入帧缓冲区，生成最终的图像。这些图像会在屏幕上显示或保存为图像文件。

第二章：高洛德着色技术

高洛德着色的基本原理

高洛德着色（Gouraud Shading）是一种通过在顶点计算光照并在多边形表面插值颜色的方法（如图 2-1）。基本思想是先在每个顶点上计算光照效果，然后在多边形的内部像素间进行颜色插值，使多边形表面显示出平滑的颜色渐变[3]。这种着色器提高了视觉效果的平滑度，且计算效率较高，非常适合实时渲染应用[4]。



2-1

顶点光照计算

顶点光照计算是高洛德着色的第一步。光照模型（如冯氏光照模型[6]）用于计算每个顶点的颜色。光照模型考虑了环境光、漫反射光和镜面反射光等因素，计算出顶点在当前光照条件下的颜色值[3]。公式如下：

$$I = I_{ambient} + I_{diffuse} + I_{speclar}$$

其中，$I_{ambient}$是环境光，$I_{diffuse}$是漫反射光，$I_{speclar}$是镜面反射光。

颜色插值方法

在计算出顶点颜色后进行颜色插值。通过在多边形表面上插值顶点颜色来实现，将每个顶点的颜色值按照距离权重分布到多边形的每个像素上，能够使得多边形表面颜色变化更加平滑[3]。

高洛德着色的算法实现

在第二次实验过程中，由于我们使用的模型都是高精度处理光照和阴影的，没有大场景非细节的部分，因此没有使用高洛德着色。因此本人选择高洛德着色算法进行个人的尝试。高洛德着色的算法实现主要包括以下几个步骤：

1. 计算每个顶点的光照值。
2. 对每个像素进行颜色插值。
3. 显示最终的渲染结果。

```
#include <GL/glut.h>
#include <iostream>
#define _USE_MATH_DEFINES
#include <math.h>
using namespace std;

GLfloat w = 800;
GLfloat h = 600;

float cam_x = 0.0, cam_y = 50.0, cam_z = 160.0;
float lightPos[] = { 0.0, 60.0, 0.0, 1.0 };  // Initial position of the point light (roof light)
float sunDirection[] = { 0.0, -1.0, -1.0, 0.0 }; // Initial sun direction

int lightMode = 0; // 0 for point light, 1 for directional light (sun)

void initRendering() {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glClearColor(0.53, 0.81, 0.92, 1.0); // Sky blue background
    glShadeModel(GL_SMOOTH); // Use smooth shading for Gouraud Shading
}

void setLights() {
    glDisable(GL_LIGHT0);
    if (lightMode == 0) { // Point light
        glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, new GLfloat[4]{ 1.0, 1.0, 1.0, 1.0 });
        glLightfv(GL_LIGHT0, GL_SPECULAR, new GLfloat[4]{ 1.0, 1.0, 1.0, 1.0 });
    }
    else { // Directional light
        glLightfv(GL_LIGHT0, GL_POSITION, sunDirection);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, new GLfloat[4]{ 1.0, 1.0, 1.0, 1.0 });
        glLightfv(GL_LIGHT0, GL_SPECULAR, new GLfloat[4]{ 1.0, 1.0, 1.0, 1.0 });
    }
    glEnable(GL_LIGHT0);
}
```

多面体函数：

```
void handleResize(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (double)w / (double)h, 1.0, 200.0);
}

void drawPolyhedron() {
    glColor3f(0.7, 0.7, 0.7);
    glPushMatrix();
    glScalef(10.0, 10.0, 10.0); // 放大模型
    glutSolidIcosahedron(); // Draw a solid icosahedron
    glPopMatrix();
}
```

球形函数：

```
void handleResize(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (double)w / (double)h, 1.0, 200.0);
}

void drawSphere() {
    glColor3f(0.7, 0.7, 0.7);
    glPushMatrix();
    glScalef(10.0, 10.0, 10.0); // 放大模型
    glutSolidSphere(1.0, 50, 50); // Draw a solid sphere
    glPopMatrix();
}

void drawScene() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(cam_x, cam_y, cam_z, cam_x, 0.0, -1.0, 0.0, 1.0, 0.0);

    setLights();

    drawSphere();

    glutSwapBuffers();
}
```

```
void handleKeypress(unsigned char key, int x, int y) {
    const float stepSize = 10.0;
    switch (key) {
    case 'w':
    case 'W':
        cam_z -= stepSize;
        break;
    case 's':
    case 'S':
        cam_z += stepSize;
        break;
    case 'a':
    case 'A':
        cam_x -= stepSize;
        break;
    case 'd':
    case 'D':
        cam_x += stepSize;
        break;
    case 't':
    case 'T':
        lightMode = 0; // Switch to point light
        setLights(); // Update light settings
        break;
    case 'y':
    case 'Y':
        lightMode = 1; // Switch to directional light
        setLights(); // Update light settings
        break;
    case 'p':
    case 'P':
        if (lightMode == 1) sunDirection[2] -= 0.1; // Adjust sun direction forward
        setLights(); // Update light settings
        break;
    case 'l':
    case 'L':
        if (lightMode == 1) sunDirection[2] += 0.1; // Adjust sun direction backward
        setLights(); // Update light settings
        break;
    case 27: // Escape key
        exit(0);
    }
    glutPostRedisplay();
}
```

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(w, h);
    glutCreateWindow("Gouraud Shading Example with Sphere");
    initRendering();
    glutDisplayFunc(drawScene);
    glutKeyboardFunc(handleKeypress);
    glutReshapeFunc(handleResize);
    glutMainLoop();
    return 0;
}
```

```
void initRendering() {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glClearColor(0.53, 0.81, 0.92, 1.0); // Sky blue background
    glShadeModel(GL_SMOOTH); // Use smooth shading for Gouraud Shading
}
```

在 drawPolyhedron 函数中，绘制多面体模型，通过顶点的颜色插值实现高洛德着色效果：

```
void drawPolyhedron() {
    glColor3f(0.7, 0.7, 0.7);
    glPushMatrix();
    glScalef(10.0, 10.0, 10.0); // 放大模型
    glutSolidIcosahedron(); // Draw a solid icosahedron
    glPopMatrix();
}
```

如图 2-2 和图 2-3 所示，图 2-2 是未进行高洛德着色法着色的多面体，图 2-3 是进行过高洛德着色的多面体，可以明显看出，经过高洛德着色的图形更加的平滑。

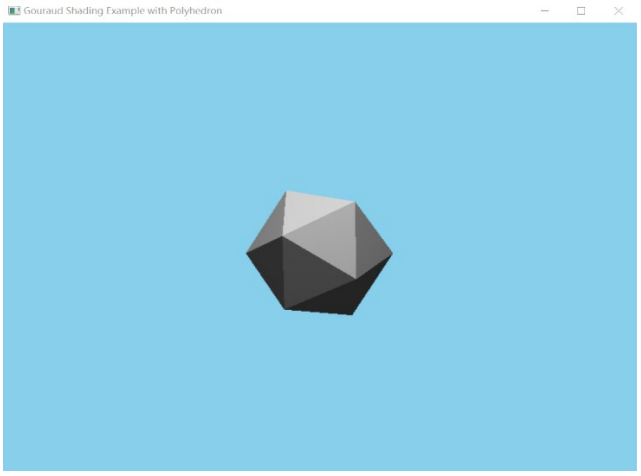


图 2-2

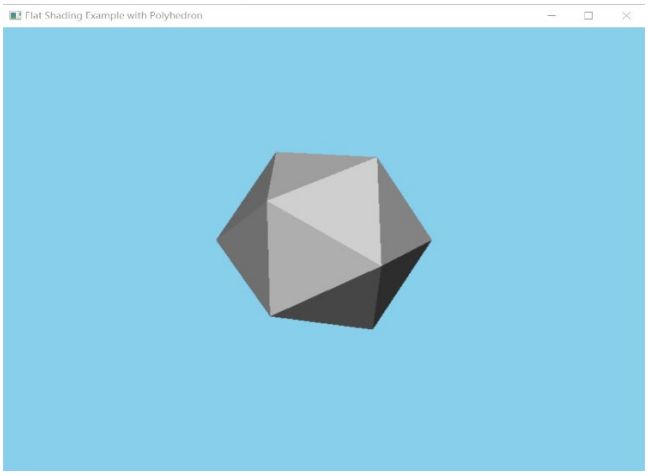


图 2-3

本人又继续使用了球形进行着色器的使用展示，图 2-4 使用的是平面着色，可以看出着色并不平滑。图 2-5 使用高洛德着色，启用平滑着色。

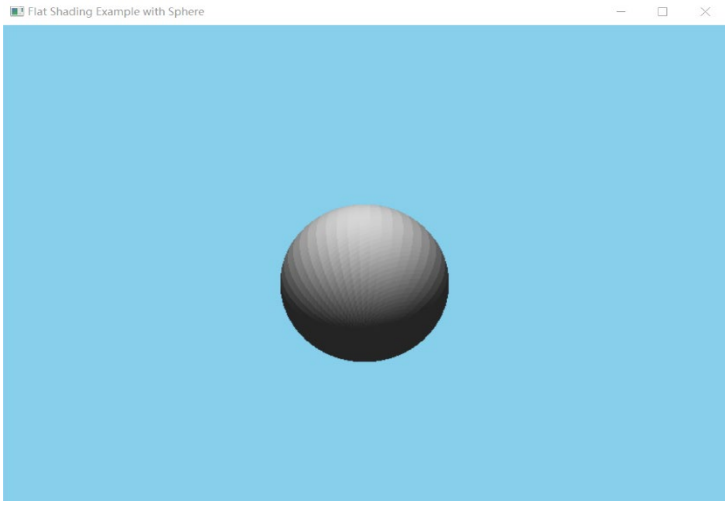


图 2-4

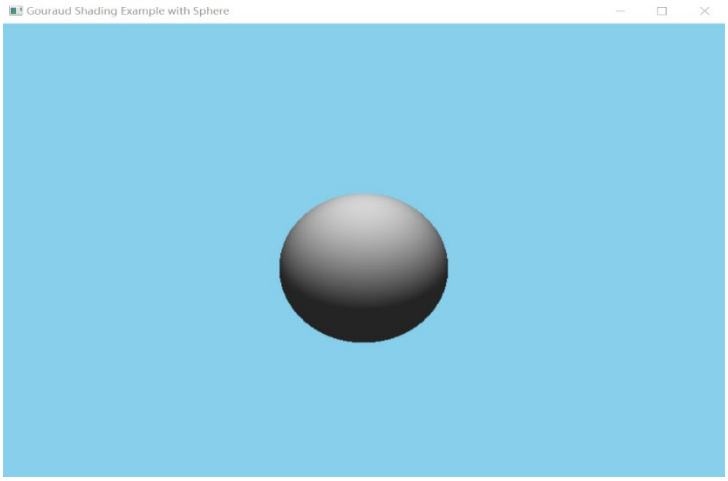


图 2-5

2-6 是高洛德着色，图 2-7 是冯氏着色。

高洛德着色(Gouraud Shading), 通过计算顶点颜色, 然后通过插值方法完成着色, 在一个面上可以有不同的颜色。每一个顶点做一次光照计算，然后每一个三角形根据组成它的三角形进行插值给片段进行着色，

冯氏着色(phone Shading)，包含 ambient,diffuse,specular 部分，每一个图元逐像素的法线进行光照计算。

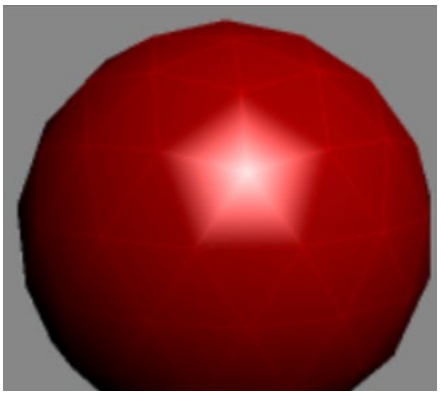


图 2-6

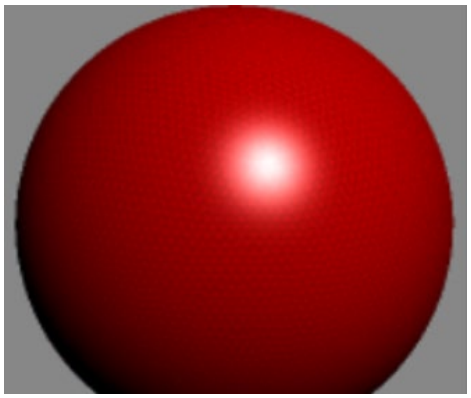


图 2-7

视频游戏中的应用

实时渲染的优势

在视频游戏中，因其高效的计算能力和良好的视觉效果而被广泛应用[3]。视频游戏通常需要实时渲染大量的图形场景，高洛德着色快速光照计算和颜色插值使其能够在保持高帧率的同时提供平滑的图像效果[7]。高帧率和流畅的视觉效果可以显著提升互动和沉浸感，因此对于玩家的游戏体验会有所提升。

案例分析：游戏中的高洛德着色实现

在《巫师 3：狂猎》（The Witcher 3: Wild Hunt）中，高洛德着色被用来处理环境光照和大面积的自然景观。通过在顶点进行光照计算，再在表面进行颜色插值，游戏实现了广袤的森林、山脉和海洋的平滑过渡效果。

电影和动画中的应用

提高视觉效果的实现

在电影和动画制作中，高洛德着色也起着重要的作用。电影制作一般追求高质量的视觉效果，高洛德着色用于处理大面积背景和次要物体，提供平滑的颜色过渡和光照效果，从而减少计算负担，将更多资源用于主要角色和细节处理[3]。

案例分析：电影中的高洛德着色技术

在电影《阿凡达》的制作过程中，高洛德着色被用来处理潘多拉星球的大量自然景观和次要角色。通过在顶点计算基础光照并在表面进行插值，影片实现了广阔的森林和山脉的平滑光照效果，为观众呈现出逼真的自然环境。

第四章：高洛德着色的优势与局限

优势分析

计算效率：在渲染过程中，高洛德着色只需在每个顶点进行一次光照计算，然后通过插值计算多边形内部的像素颜色。相比于需要在每个像素进行光照计算的冯氏着色，高洛德着色的计算量大大减少。

视觉效果：高洛德着色通过顶点颜色插值，可以在多边形表面上实现平滑的颜色过渡，提供较为真实的视觉效果。

局限性探讨

光照精度问题：由于光照计算仅在顶点进行，而不是在每个像素进行，则会导致光照效果不够精确，尤其在处理高光和复杂光源场景时。

处理高光和阴影的不足：高光效果在顶点计算，如果高光落在多边形的中心位置而不是顶点，高洛德着色可能无法准确表现，如图 4-1，高光没有扩散到该多边形的任何顶点，使用 Gouraud 着色法就不会渲染出任何效果，若周围的顶点没有得到这个计算值，这块也就不可能产生高光效果。由于阴影效果需要更加精细的光照计算，高洛德着色在阴影处理上也较为有限。

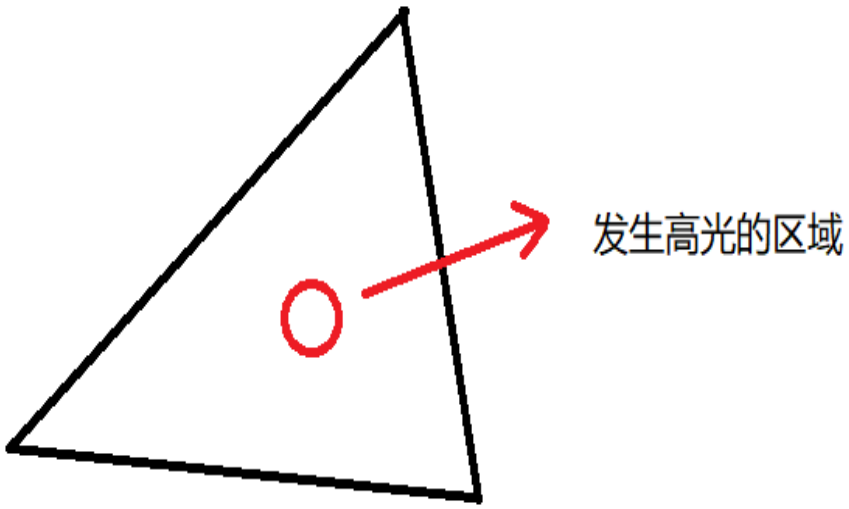


图 4-1

第五章：高洛德着色的改进与发展

与冯氏着色的结合

结合的必要性

高洛德着色在计算效率和视觉效果方面具有明显优势，但其在处理高光和阴影方面存在不足。冯氏着色通过在每个像素进行光照计算，可以提供更精确和真实的光照效果[6]。

高洛德着色与冯氏着色的比较

1. **高洛德着色**：

   优点：计算效率高，通过顶点插值实现平滑的颜色过渡，适合实时渲染。

   缺点：光照精度较低，难以处理复杂的高光和阴影效果。

2. **冯氏着色**：

   优点：光照精度高，通过每个像素计算光照，能更好地处理高光和阴影。

   缺点：计算量大，实时渲染效率较低。

结合的策略

在实际应用中，许多案例都采用混合着色策略。在渲染平滑表面和大面积颜色过渡时，使用高洛德着色。

处理高光、阴影和细节丰富的区域时，使用冯氏着色。比如上面提到的电影和游戏的处理，也是使用这种方式进行的。

基于物理的渲染（PBR）中的应用

PBR 是通过物理原理模拟光线与物体表面相互作用的渲染技术。其使用物理上准确的模型和算法，使渲染的图像更加逼真和一致[5]。

高洛德着色在 PBR 中的应用

PBR 是一种倾向于复杂光照和材质模型的技术，高洛德着色作为高效顶点着色技术，可以在 PBR 框架中发挥作用，特别是在需要权衡计算效率和视觉质量的场景中，例如：

1. **基础光照计算**：高洛德着色通过在顶点进行简单的光照计算，然后在多边形表面进行插值，提供快速的初始光照效果。
2. **初始材质处理**：高洛德着色可以在 PBR 中计算物体表面的漫反射和环境光效果。可以在性能有限的环境中提供合理的视觉效果。
3. **混合使用**：在一些需要高精度光照和材质处理的场景（如高光和阴影处理），可以使用 PBR 模型，在其他部分使用高洛德着色，以平衡性能和质量。

第六章：未来发展趋势

实时渲染技术的发展

实时渲染技术的是一种在用户交互的过程中生成和显示高质量的图像的技术[7]。随着硬件性能的提升和渲染算法的进步，实时渲染技术取得了显著的发展。基于 GPU 的并行计算大大提高了渲染速度，使得复杂的光照和材质处理能够在毫秒级时间内完成。

AI 在图形学中的应用

如表 6-1，清晰的展示了 AI 技术的应用领域，对应的技术以及详细描述：

应用领域	技术	详细描述
图像生成和增强	生成对抗网络（GANs）	使用深度学习技术生成高质量的图像和动画。AI 算法能够学习和模仿艺术风格，生成与训练数据相似的图像，用于电影特效和游戏角色设计。
实时渲染优化	AI 优化算法	AI 用于优化实时渲染的性能，例如通过预测和简化复杂的计算任务，提高渲染效率。AI 算法还可以通过学习场景和光照条件，自动调整渲染参数，实现高效的光照和阴影处理。
自动化建模和纹理映射	AI 建模算法	AI 可以自动生成 3D 模型和纹理映射，从而减少人工建模的工作量。通过分析和学习大量的 3D 数据，AI 算法能够创建复杂且高质量的模型，广泛应用于虚拟现实和游戏开发。

表 6-1

云渲染技术利用云计算资源进行图形渲染，表 6-2 详细的解释了其应用领域和对应技术：

应用领域	技术	详细描述
高性能计算	云渲染平台	云渲染平台可以提供海量的计算资源，支持复杂的图形渲染任务，例如电影特效和高端游戏的渲染。用户可以根据需求动态分配资源，无需投资昂贵的硬件设备。
按需扩展	云渲染	云渲染允许按需扩展计算资源，适应不同规模的渲染任务。从个人项目到大型企业级项目，云渲染平台能够灵活调整计算资源，确保高效完成任务。
协同工作	云渲染	云渲染支持全球范围内的协同工作，团队成员可以实时访问和共享渲染结果，提高工作效率和项目协作能力。这对于跨国公司和远程工作团队尤为重要。

表 6-2

结论

本文详细探讨了高洛德着色技术的原理、实现方法和应用、优势和局限性。高洛德着色通过在顶点计算光照效果并在多边形表面插值颜色，提供平滑的颜色过渡和较高的计算效率，适合实时渲染。然而，其在处理高光和阴影方面存在不足，需要结合其他着色技术。

高洛德着色在计算机图形学中具有重要地位。其在实时渲染应用中广泛使用，不仅在保证视觉质量的同时提供高效的渲染性能，还在基础光照计算和初始材质处理方面发挥重要作用。

参考文献

[1]. 陈宏欣. 计算机图形学与图形图像处理技术应用[J]. 南京铁道职业技术学院信息管理中心, 2022.

[2]. 季洁，黄少年，施游. "互联网+教育"背景下《计算机图形学》课程建设的研究及实践[J]. 湖南工商大学计算机与电子工程学院，湖南师范大学, 2020.

[3]. 汪绪彪. 基于明暗处理算法的虚拟三维场景漫游设计[D]. 电子科技大学, 2020.

[4]. 邹耀斌. Gouraud 明暗处理的探索[J]. 电子科技大学, 2022.

[5]. 叶博生，张艳君，曾立展. 三维表面重建的着色算法研究与应用[J]. 华中科技大学数控系统国家工程研究中心, 2022.

[6]. 潘基斌，陈炳发，王静秋，钱志峰. Phong 明暗处理方法的探讨与改进[J]. 南京航空航天大学机电学院, 2001.

[7]. 郭海波. 实时渲染技术研究综述[J]. 西京学院电子信息学院, 2023.

附录

正文：3864 字