

高级计算机图形学

中国科学技术大学计算机学院

黄章进

zhuang@ustc.edu.cn

第十章

可编程着色器

主要内容

- 介绍可编程流水线
- OpenGL着色语言(GLSL)
 - 顶点着色器 (vertex shader)
 - 片段着色器 (fragment shader)

第十章之第一节

可编程流水线

主要内容

- 固定功能流水线回顾
- 着色语言简介
- OpenGL扩展

简介

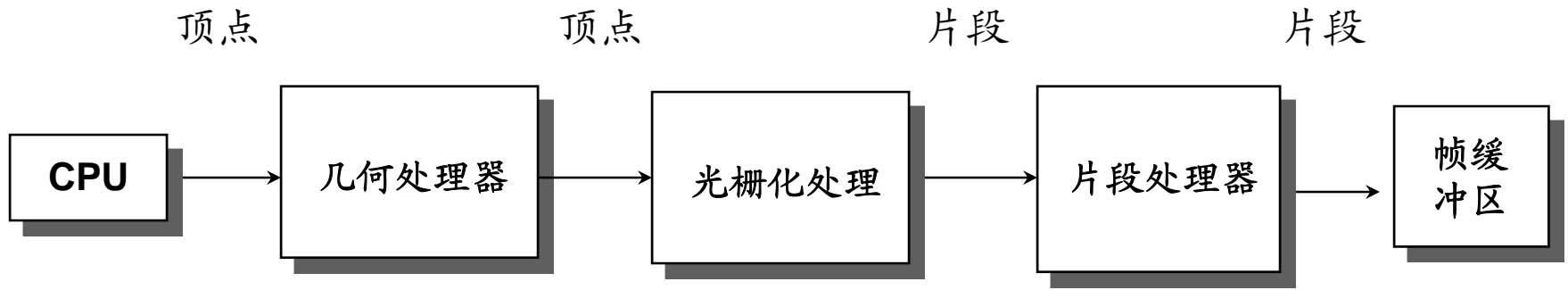


- 近几年在实时图形学方面的主要进展是可编程流水线（programmable pipeline）
 - 由NVIDIA GeForce 3首先引入
 - 高端商业显卡支持
 - NVIDIA, ATI, 3D Labs
 - 软件支持
 - Direct X 8 , 9, 10 (HLSL)
 - OpenGL Extensions
 - OpenGL Shading Language (GLSL)
 - Cg

背景知识

- 由两个成分组成
 - 顶点程序 (着色器)
 - 片段程序 (着色器)
- 需要对两个看起来矛盾方法的深入理解
 - OpenGL流水线
 - 实时
 - RenderMan的想法
 - 离线

黑盒子观点



几何计算

- 几何数据: 顶点集合 + 类型
 - 可以来自于程序、求值器或者显示列表
 - 类型: 点、线段、多边形
 - 顶点数据可以是
 - 用顶点的(x,y,z,w)坐标指定(glVertex)
 - 法向量
 - 纹理坐标
 - RGBA 颜色
 - 其它数据: 颜色索引、边标志
 - 在GLSL中用户定义的数据

逐顶点操作

- 顶点位置由模型视图矩阵变换到视点坐标
- 法向量相应处理
 - 可能需要重新单位化
- 如果激活了纹理自动生成功能，纹理坐标被生成，并且应用可能的纹理矩阵

光照计算

- 进行如下的逐顶点基础上的 Phong光照模型

$$I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^\alpha + k_a I_a$$

- Phong模型需要在每个顶点处计算 \mathbf{r} 和 \mathbf{v}

计算反射角

入射角 = 反射角

$$\cos \theta_i = \cos \theta_r \quad \text{或} \quad \mathbf{r} \cdot \mathbf{n} = \mathbf{l} \cdot \mathbf{n}$$

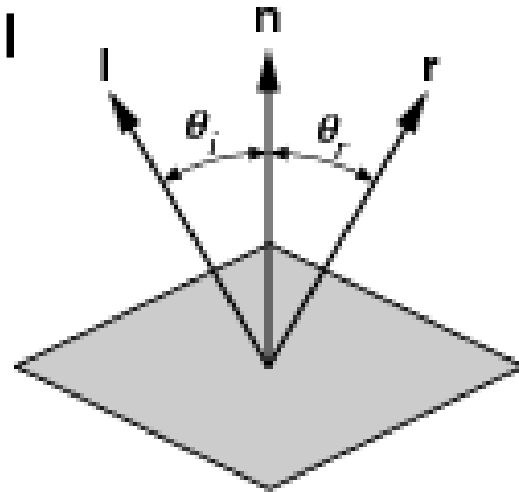
\mathbf{r} , \mathbf{n} , 和 \mathbf{l} 共面

$$\mathbf{r} = \alpha \mathbf{l} + \beta \mathbf{n}$$

规范化

$$1 = \mathbf{r} \cdot \mathbf{r} = \mathbf{n} \cdot \mathbf{n} = \mathbf{l} \cdot \mathbf{l}$$

$$\text{得到: } \mathbf{r} = 2(\mathbf{l} \cdot \mathbf{n})\mathbf{n} - \mathbf{l}$$



OpenGL的光照

- 对Phong模型进行了修改
 - 采用中值 (Halfway) 向量
 - 全局环境光项
- 细节的标准中有详细定义
- 硬件支持

中值向量

Blinn建议用 $\mathbf{n} \cdot \mathbf{h}$ 代替 $\mathbf{v} \cdot \mathbf{r}$ ，其中

$$\mathbf{h} = (\mathbf{l} + \mathbf{v}) / |\mathbf{l} + \mathbf{v}|$$

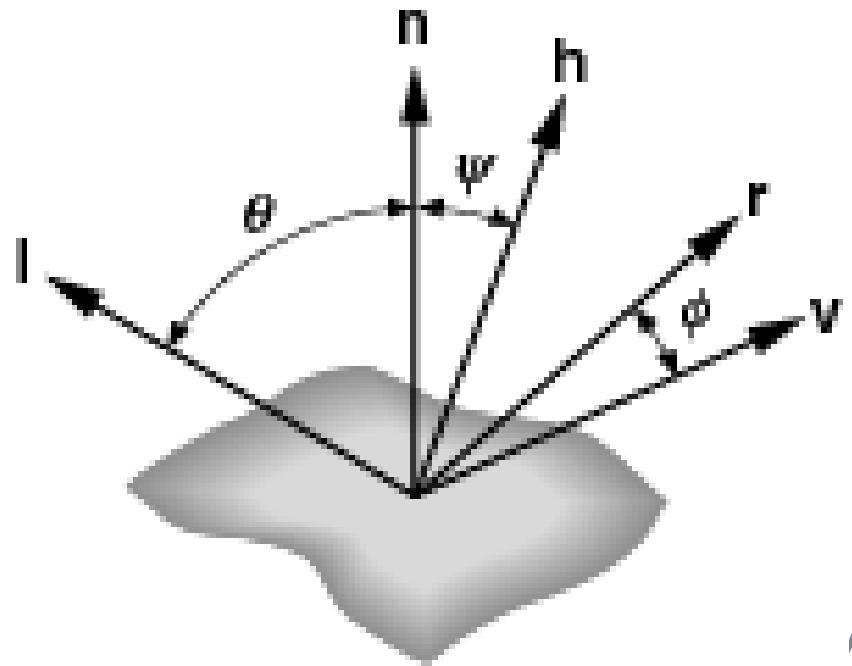
$(\mathbf{l} + \mathbf{v})/2$ 是 \mathbf{l} 和 \mathbf{v} 的平分线

如果 $\mathbf{n}, \mathbf{l}, \mathbf{v}$ 共面，

$$\psi = \phi/2$$

那么就必须调整指数，

使得 $(\mathbf{n} \cdot \mathbf{h})^{e'} \approx (\mathbf{r} \cdot \mathbf{v})^e$



图元装配

- 顶点接下来被装配为对象
 - 多边形、线段、点
- 背向面剔除
- 用投影矩阵用来变换
- 裁剪
 - 相对于用户定义的平面
 - 视景物 $x = \pm w, y = \pm w, z = \pm w$
 - 裁剪会产生新的顶点
- 透视除法
- 视口映射

光栅化

- 几何对象被光栅化成片段（**fragment**）
- 每个片段对应着整数栅格的一个格点：显示出来的像素
- 因此每个片段就是一个潜在的像素
- 每个片段具有插值顶点属性得到的
 - 颜色
 - 可能的深度值
 - 纹理坐标

片段操作

- 输入：插值得到的片段信息
- 纹理生成
- 雾化
- 输出：片段的颜色值和深度值

光栅操作

- 输入：
 - 像素位置
 - 片段的颜色和深度值
- 融合
- 反走样
- Alpha测试
- 深度测试
- 输出：帧缓冲区中的像素

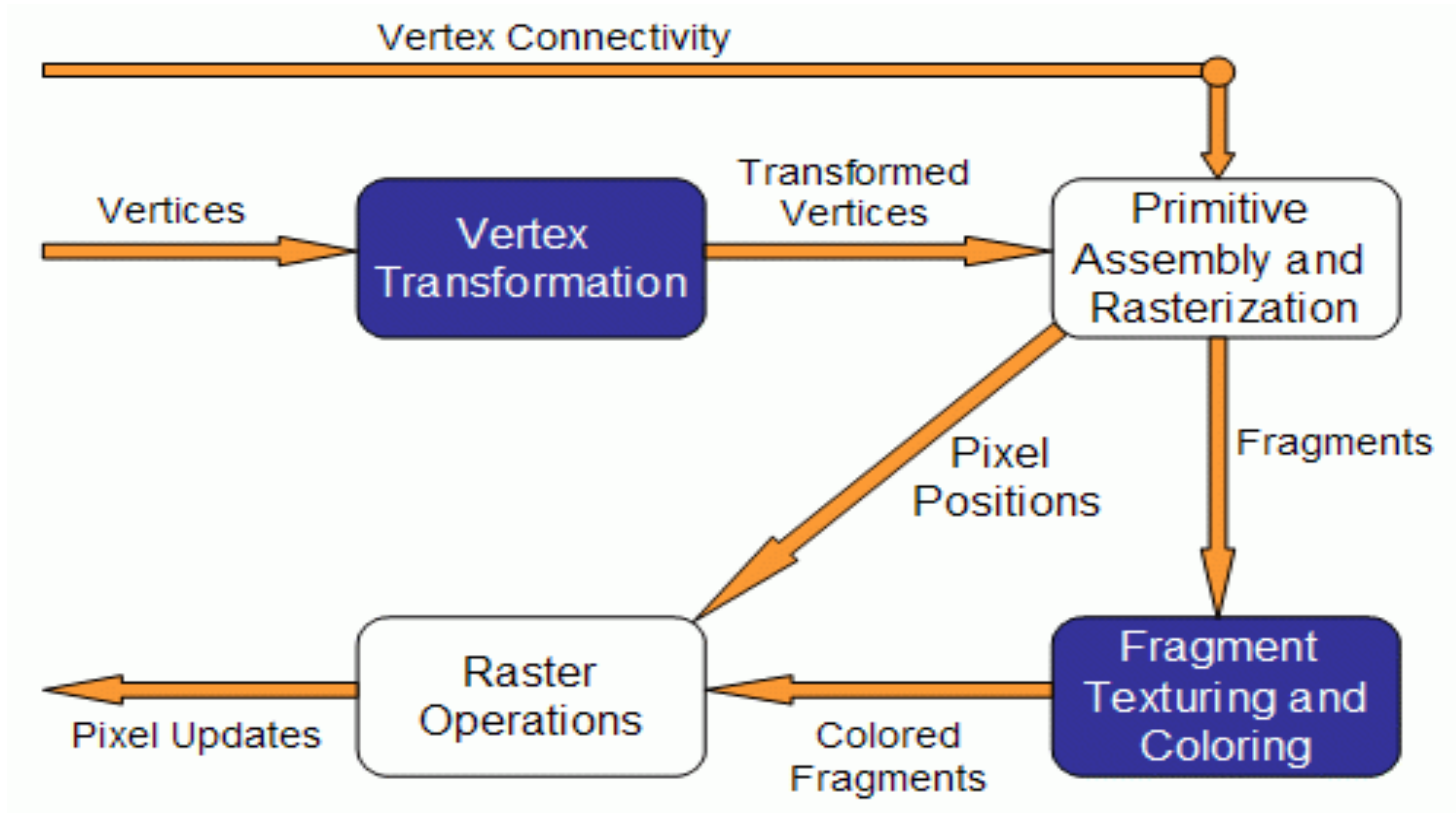
顶点处理器

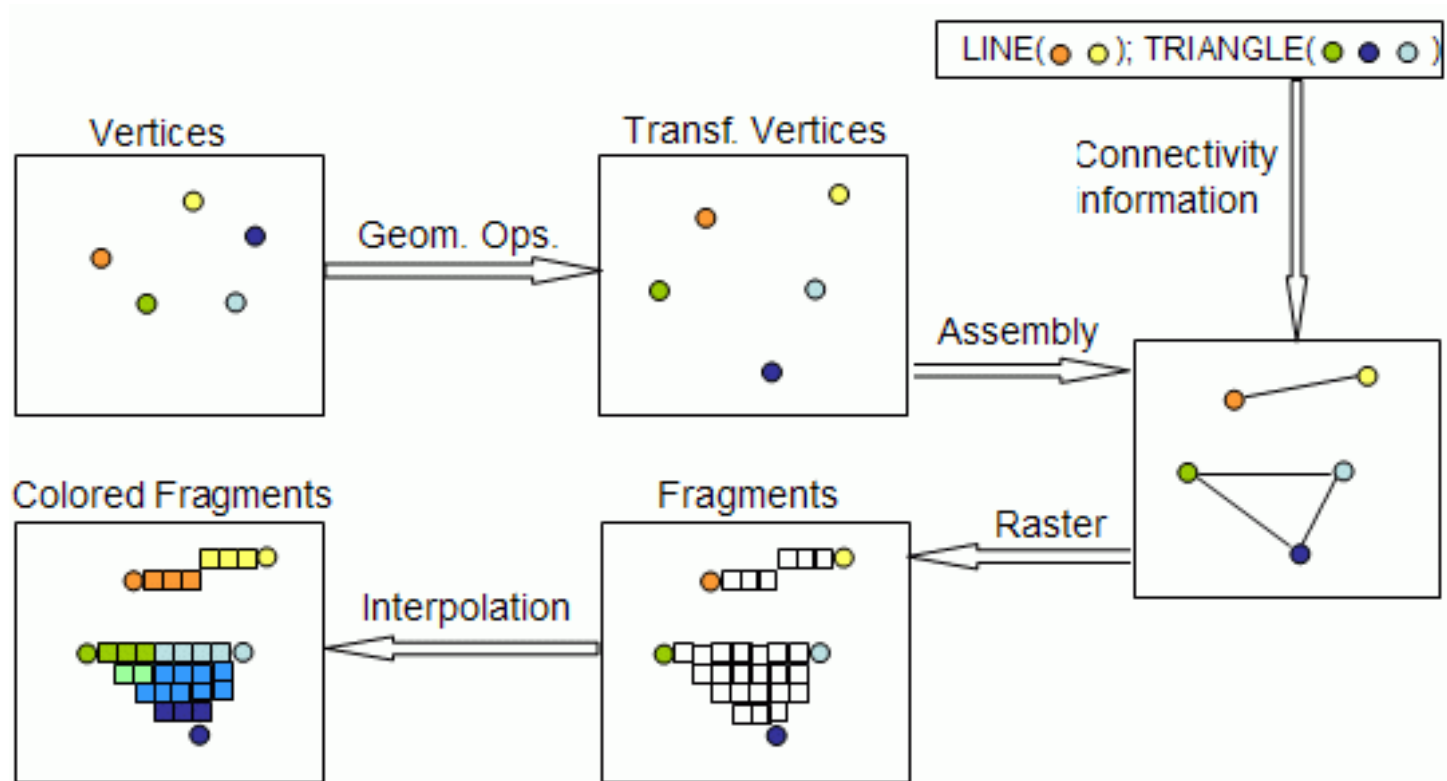
- 输入顶点数据
 - 位置属性
 - 可能的颜色
 - OpenGL状态
- 输出
 - 在裁剪坐标中的位置
 - 顶点颜色
- 执行任务
 - 用模型-视图矩阵和投影矩阵变换顶点位置
 - 法向量的变换和归一化
 - 纹理坐标生成和变换
 - 逐顶点的光照计算

片段处理器

- 输入：光栅化得到的片段
 - 插值的顶点位置
 - 插值的法向
 - 插值的颜色
- 输出
 - 片段颜色
 - 片段深度
- 执行任务
 - 逐像素的颜色、纹理坐标计算
 - 纹理应用
 - 雾计算

固定功能流水线





固定功能流水线的缺陷

- OpenGL固定功能流水线只支持改进的Phong光照模型，无法有效实现
 - 物理真实感的光照效果：折射、区域光、软阴影
 - 非真实感效果：卡通着色效果、水墨画
- 可使用离线渲染器如RenderMan来实现

可编程着色器

- 把顶点和片段处理中那些固定的功能用可编程处理器代替
- 可能取代两者之一或者同时取代
- 如果采用了可编程着色器，那么就必须要做固定功能处理器的所有事情，或者对其进行了相应的修改

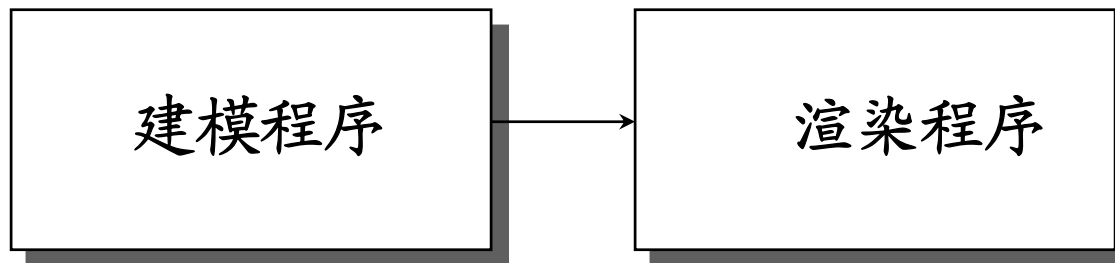
发展

- RenderMan的着色语言
 - 离线渲染
- 硬件着色语言
 - UNC, Stanford
 - NVIDIA
 - OpenGL Vertex Program Extension
 - OpenGL Shading Language
 - Cg --- OpenGL and Microsoft HLSL

RenderMan

- 由Pixar所开发
 - 书籍：S. Upstill, *The RenderMan Companion*, Addison-Wesley, 1989.
- 模型

界面文件 (RIB)



建模 VS 渲染

- 建模程序输出几何模型以及供渲染程序使用的信息
 - 照相机的参数
 - 材料
 - 光源
- 可以用不同类型的渲染程序
 - 光线跟踪
 - 辐射度方法
- 如何定义着色器呢？

着色树

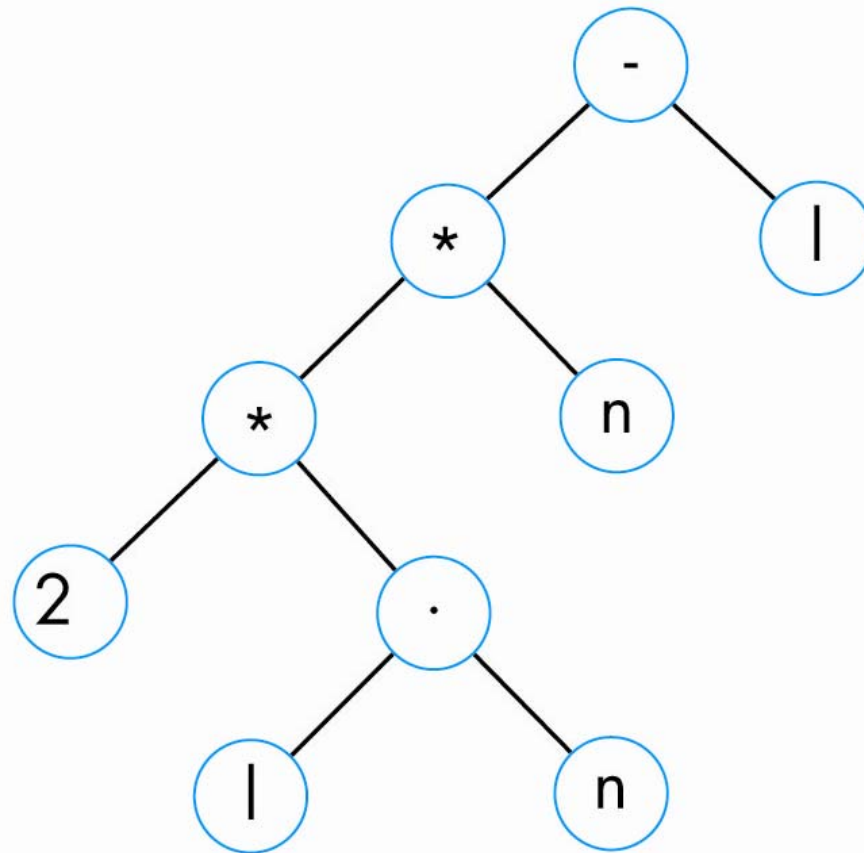
- 基于Phong模型的着色器可以用下述代数表达式表示:

$$I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^s + k_a I_a$$

- 这个表达式可以用树形结构描述
- 需要类似于点积和外积等新运算符以及类似于矩阵和向量等新数据类型
- 环境变量是状态的一部分

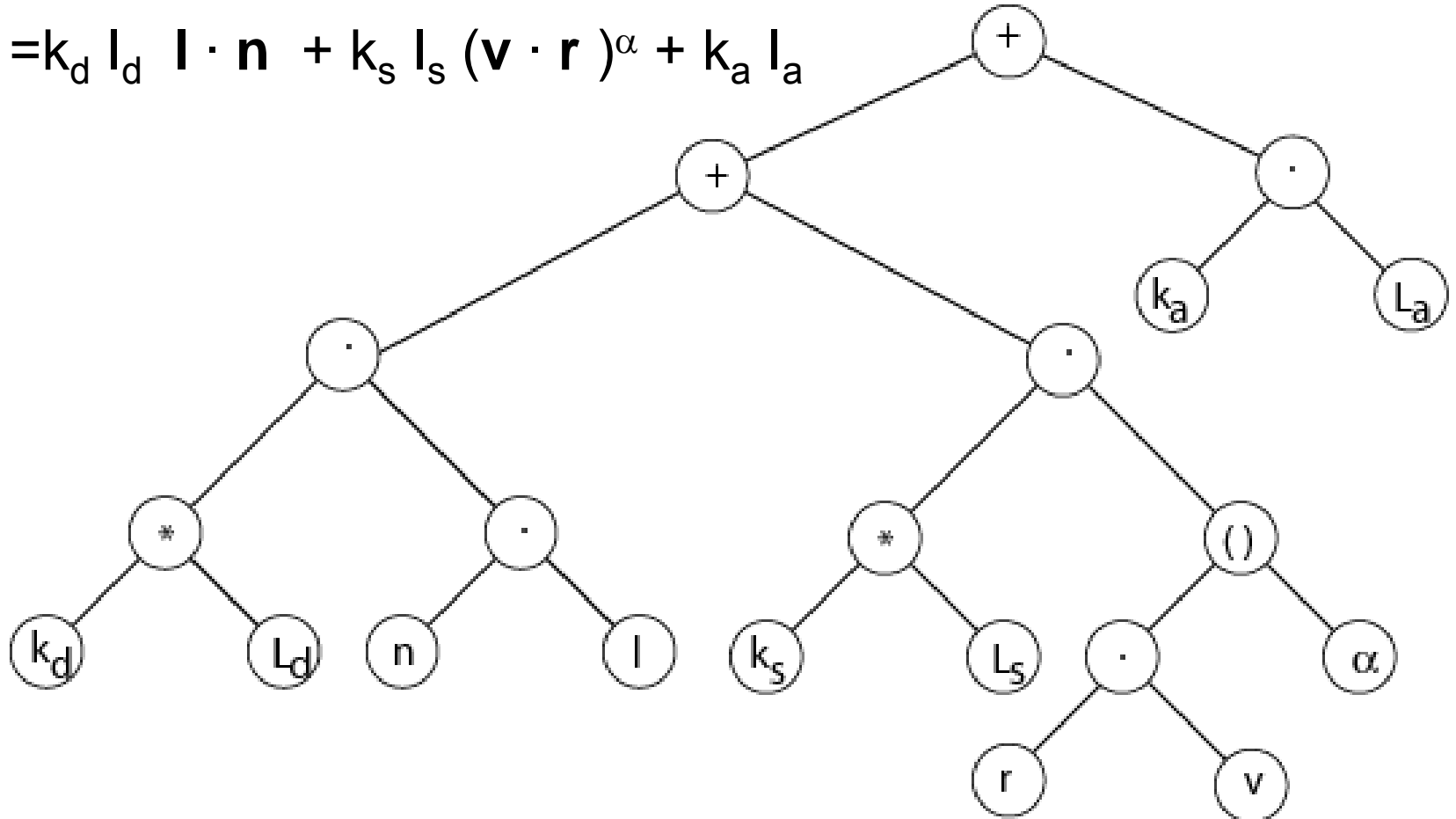
反射向量

$$r = 2(l \cdot n)n - l$$



Phong模型

$$I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^\alpha + k_a I_a$$



表达式树

- 表达式树是一棵二叉树
 - 变量和常量出现在叶节点
 - 内部节点表示运算
- 计算一个数学表达式的值等效于遍历它对应的二叉表达式树
- Renderman着色语言采用表达式树来设计新的着色器
 - 该方法得到可编程图形卡的支持

OpenGL版本

- 最新版本OpenGL 3.2 (GLSL 1.5)
- 教材基于版本OpenGL 2.1 (GLSL 1.2)
- Microsoft VS只支持OpenGL 1.1

`const GLubyte* glGetString(GLenum name);`

- 返回OpenGL实现相关的信息。name可以是
 - GL_VENDOR: 厂商信息
 - GL_RENDERER: 硬件平台
 - GL_VERSION: OpenGL版本
 - GL_EXTENSIONS: 扩展信息

标准的扩展

- 厂商扩展: NV, ATI, ...
- 多厂商扩展EXT:
 - glCommandEXT()和GL_DEFINITION_EXT
- ARB扩展: 通过OpenGL ARB的批准
 - glCommandARB()和GL_DEFINITION_ARB
- 最后ARB扩展有可能纳入OpenGL标准
 - GL_ARB_fragment_shader
 - GL_ARB_vertex_shader

glview & GLEW

- glview 可用来查看 OpenGL 实现版本和所支持的扩展，当前版本 3.15
 - <http://www.realtech-vr.com/glview/index.html>
- OpenGL Extension Wrangler Library (GLEW) 是开源的跨平台 OpenGL 扩展加载库，当前版本 1.5.1 (支持 OpenGL 3.0)
 - <http://glew.sourceforge.net/>

GLEW的安装

- 把头文件和库文件复制到相应目录
 - bin/glew32.dll 到 %SystemRoot%/system32
 - lib/glew32.lib 到 {VC Root}/Lib
 - include/GL/glew.h 到 {VC Root}/Include/GL
 - include/GL/wglew.h 到 {VC Root}/Include/GL
- 用 **GL/glew.h** 代替 **GL/gl.h**

```
#include <GL/glew.h>  
#include <GL/glut.h>  
<gl, glu, and glut functionality is available here>
```

- 链接时加入 **glew32.lib**

GLEW的使用

- 在创建OpenGL绘制场境后调用glewInit()初始化GLEW，若成功返回GLEW_OK

```
#include <GL/glew.h>
#include <GL/glut.h>

...
glutInit(&argc, argv);
glutCreateWindow("GLEW Test");
GLenum err = glewInit();
if (GLEW_OK != err)
{
    /* Problem: glewInit failed, something is seriously wrong. */
    fprintf(stderr, "Error: %s\n", glewGetErrorString(err));
    ...
}
fprintf(stdout, "Status: Using GLEW %s\n", glewGetString(GLEW_VERSION));
```

GLEW的使用

- 检查OpenGL的版本, `GLEW_VERSION_{version}`

```
if (GLEW_VERSION_1_3)
{
    /* Yay! OpenGL 1.3 is supported! */
}
```

- 检查扩展, `GLEW_{extension_name}`

```
if (GLEW_ARB_vertex_program)
{
    /* It is safe to use the ARB_vertex_program extension here.
    */
    glGenProgramsARB(...);
}
```