2018-2019年度第二学期 00106501

计算机图形学

童 伟 华　管理科研楼1205室

E-mail: tongwh@ustc.edu.cn

中国科学技术大学　数学科学学院

http://math.ustc.edu.cn/

第六章 可编程着色器

第一节 可编程流水线

简介

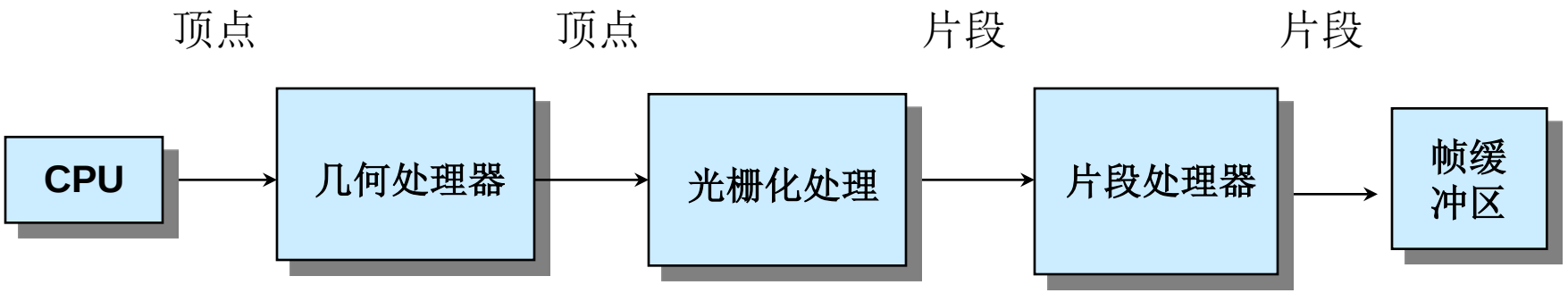
- 可编程流水线（programmable pipeline）
  - 由NVIDIA GForce 3首先引入，目前大部分显卡都支持
  - 软件支持
    - Direct X 10，11，12 (HLSL)
    - OpenGL Shading Language (GLSL)

背景知识

- OpenGL按着色器在图形流水线不同的阶段，可分为
  - 顶点着色器
  - 细分着色器
  - 几何着色器
  - 片元着色器

  以及通用的计算着色器
- 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 r和v

计算反射角

- 入射角 = 反射角

$\cos\theta_i = \cos\theta_r$ 或 $r \cdot n = l \cdot n$

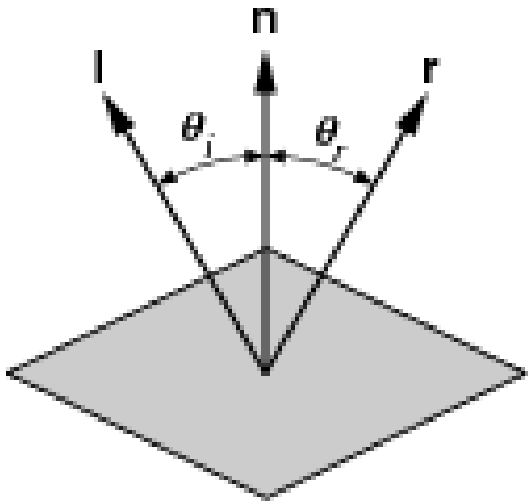
r, n, 和 l 共面

$r = \alpha l + \beta n$

规范化

$1 = r \cdot r = n \cdot n = l \cdot l$

得到: $r = 2(l \cdot n)n - l$

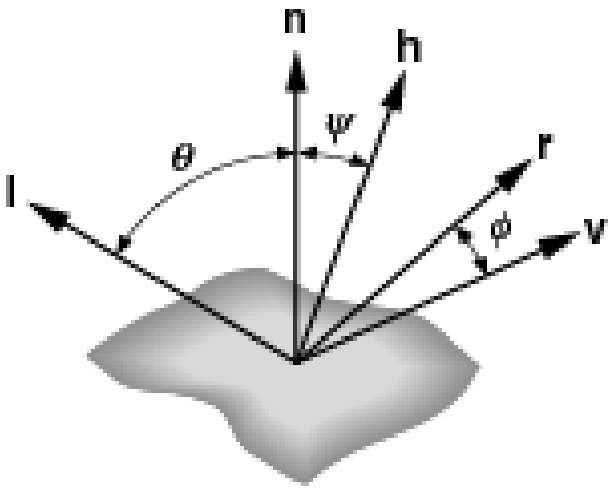


OpenGL的光照

- 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n·h代替v·r，其中
  $h = (l+v)/|l+v|$
  $(l+v)/2$ 是l和v的平分线
  如果n, l, v 共面，
  $\psi = \phi/2$
  那么就必须调整指数，
  使得$(n\cdot h)^{e'} \approx (r.v)^{e}$



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
  - 视景体$x=\pm w, y=\pm w, z=\pm w$
  - 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
  - 法向量的变换和规一化
  - 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

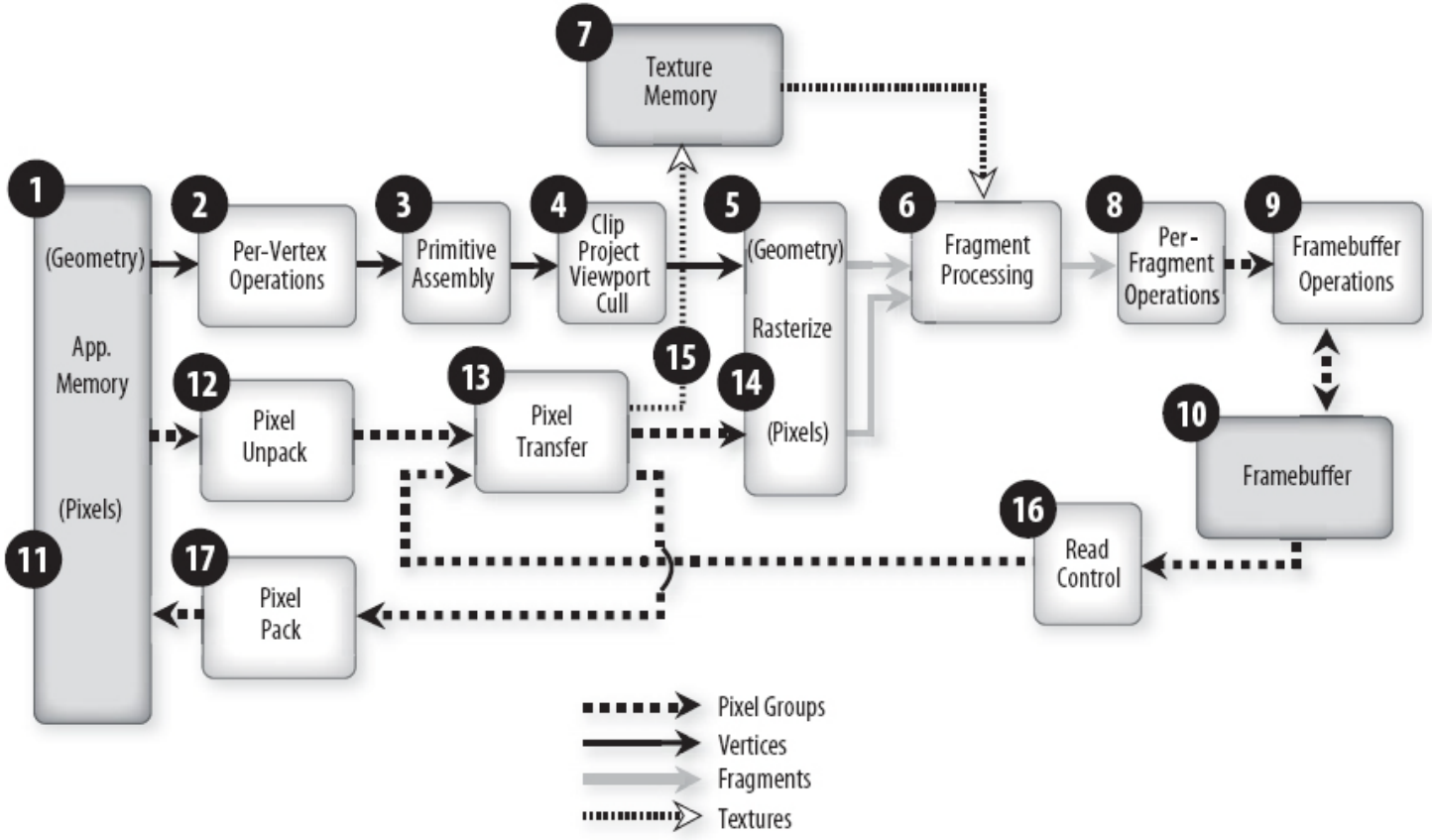


Figure 1.1 Overview of OpenGL operation

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

可编程流水线

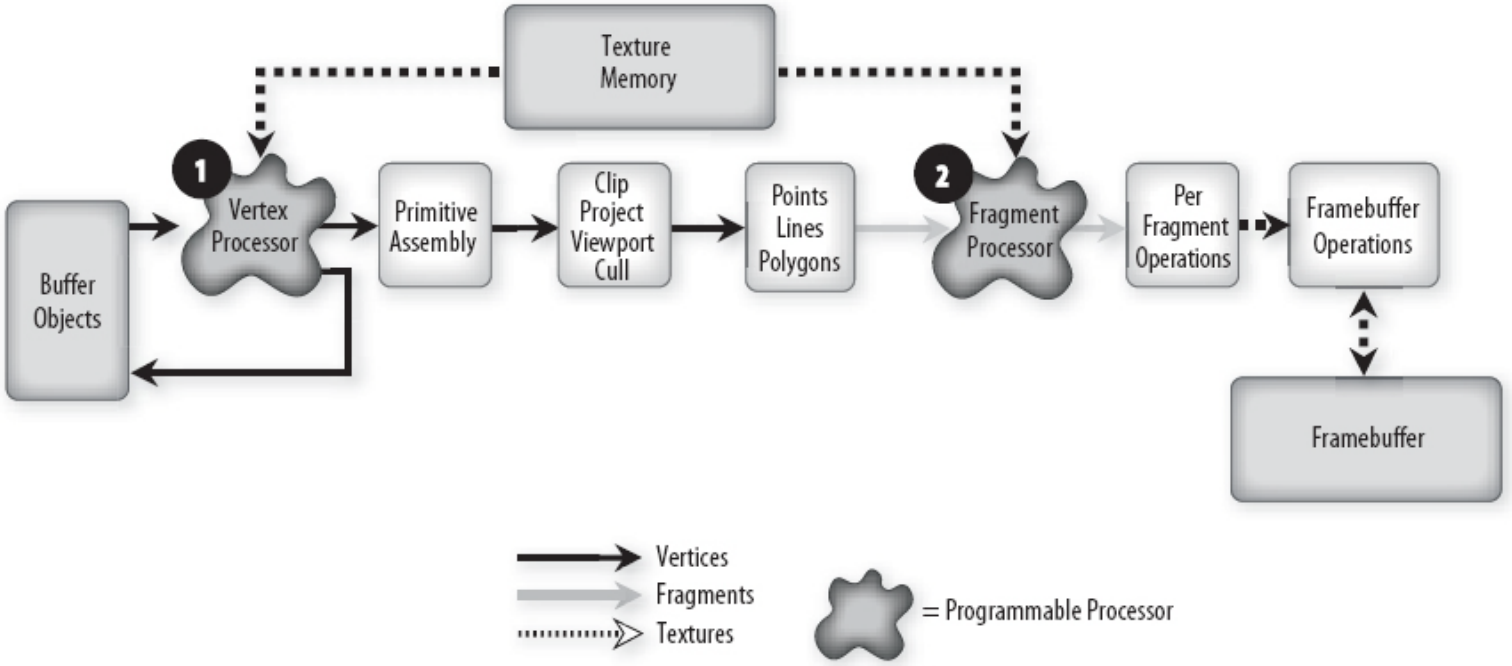


Figure 2.1 OpenGL logical diagram showing programmable processors for vertex and fragment shaders rather than fixed functionality

可编程着色器

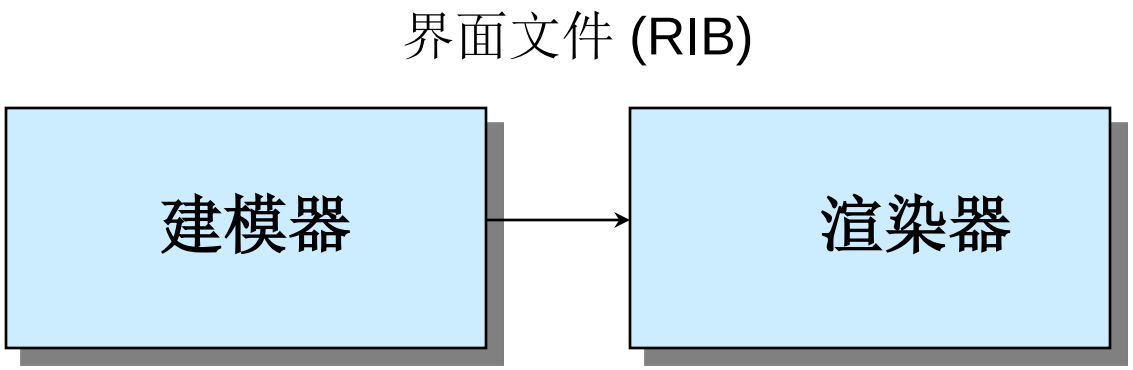
- 把顶点和片段处理中那些固定的功能用可编程处理器代替
- 可能取代两者之一或者同时取代
- 如果采用了可编程程着色器，那么就必须做固定功能处理器的所有事情，或者对其进行了相应的修改

发展

- RenderMan的着色语言
  - 离线渲染
- 硬件着色语言
  - OpenGL Shading Language
  - Microsoft DirectX HLSL

RenderMan

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



建模 VS 渲染

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

着色树

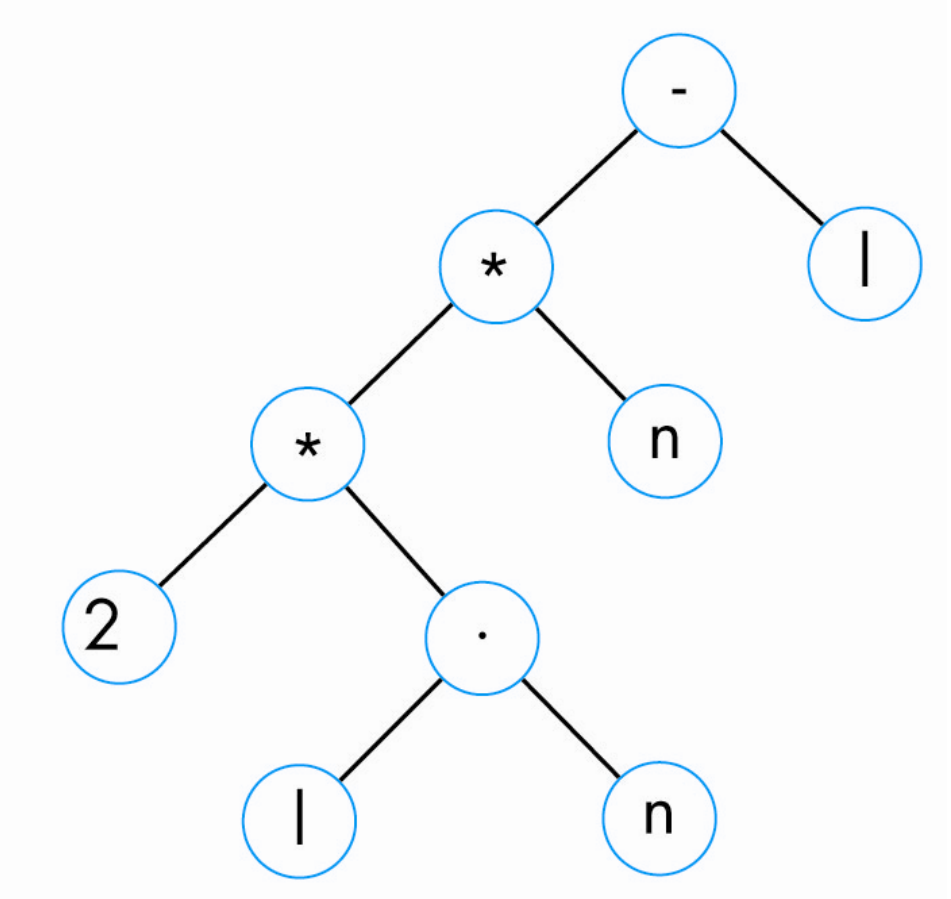
- 基于Phong模型的着色器可以用下述代数表达式表示：

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

- 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

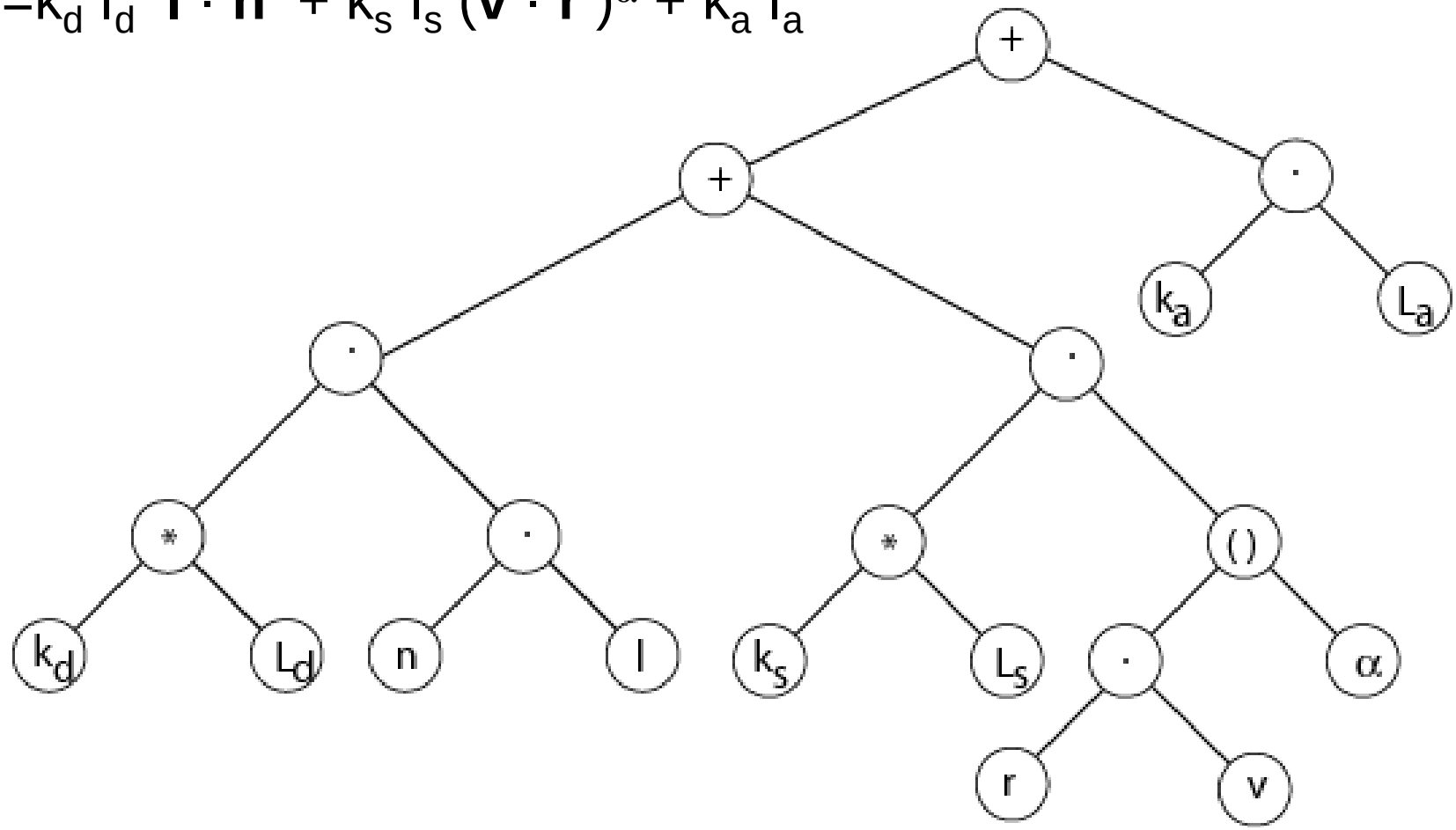
反射向量

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



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 4.6 (GLSL 4.6)
- 教材基于版本OpenGL 2.1 (GLSL 1.2)
- Microsoft visual studio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

GLEW

- OpenGL Extension Wrangler Library（GLEW）是开源的跨平台OpenGL扩展加载库，当前版本2.1.0（支持OpenGL 4.6）
  - http://glew.sourceforge.net/
- 类似的OpenGL扩展加载库：GL3W, Glad, Glee等
  - https://www.khronos.org/opengl/wiki/OpenGL_Loading_Library

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");

// Initialize GLEW library
GLenum err = glewInit();
if (err != GLEW_OK)
{
    printf("GLEW initialize error!\n“);
    exit(-1);
}
```

GLEW的使用

- 显示OpenGL版本信息

```
// Show the version of OpenGL
int GLVersion[3];
glGetIntegerv(GL_MAJOR_VERSION, &GLVersion[0]);
glGetIntegerv(GL_MINOR_VERSION, &GLVersion[1]);
GLVersion[2] = GLVersion[0]*10 + GLVersion[1];
Printf(" The OpenGL version is %d.%d\n“, GLVersion[0],
GLVersion[1]);
```

- 检查扩展， GLEW_{extension_name}

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

- 检查OpenGL版本，GLEW_VERSION_{version}

Thanks for your attention!