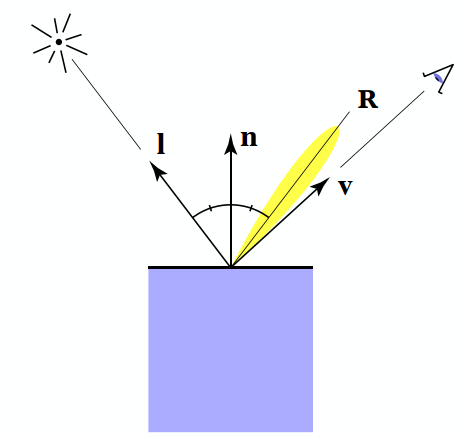
# (一)Blinn-Phong Reflection Model

## 1、Specular Term (Blinn-Phong)(高光，镜面反射)

Intensity depends on view direction

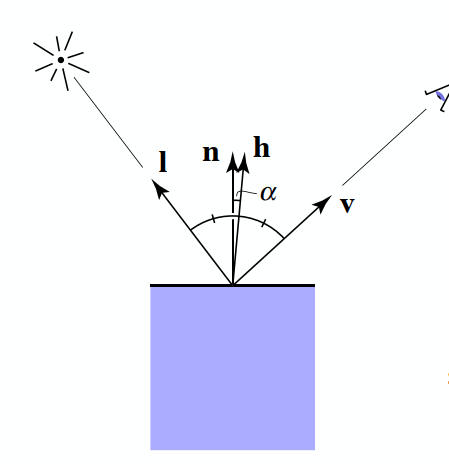
Bright near mirror reflection direction(光滑表面的反射接近镜面反射)



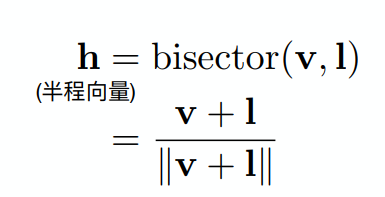
我们能看到高光的原因是镜面反射的方向**R**和观察方向**v**大致重合。也就是说当**v**和**R**足够接近的时候，就会看到高光。

**v**和**R**接近↔ 半程向量**h**与法线**n**接近。

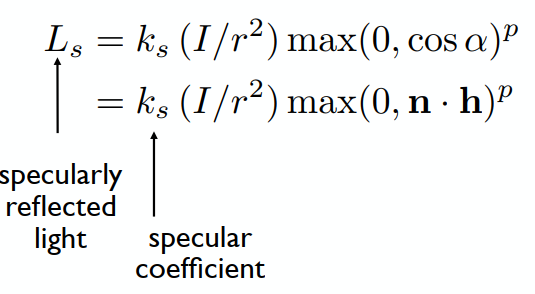
衡量两个单位向量是否接近可以用点乘。点乘结果越接近1，说明夹角余弦越接近0，两个向量越接近。



Define:半程向量（**h**）:入射方向**l**和出射方向**v**的角平分线 （**h**的方向很好算，由于**l**和**v**都是单位向量，根据平行四边形法则，**h**=**l+v**，再将**h**归一化将长度变成1即可）



（为什么要引入半程向量？因为好算，而反射向量**R**不好算）



Ls 镜面反射光

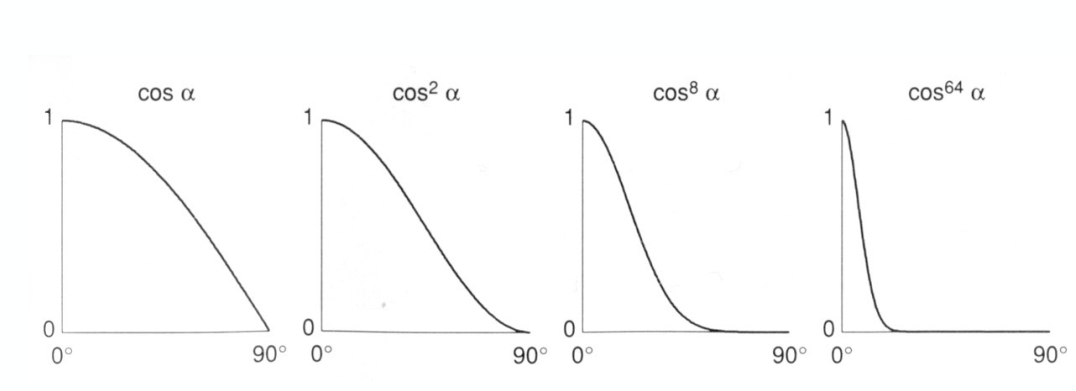
ks 镜面反射系数(一般是白的，默认为1)

I/r2 达到shading point 的光强

max(0,**n** **h**)p 考虑该高光与镜面反射有多接近，这里之所以不考虑**n**和**l**点乘(有多少光强经过)，是因为布林冯反射模型是经验性模型，这部分被省略了。若为max(0,**r v**)p则为冯反射模型

### Cosine Power Plots

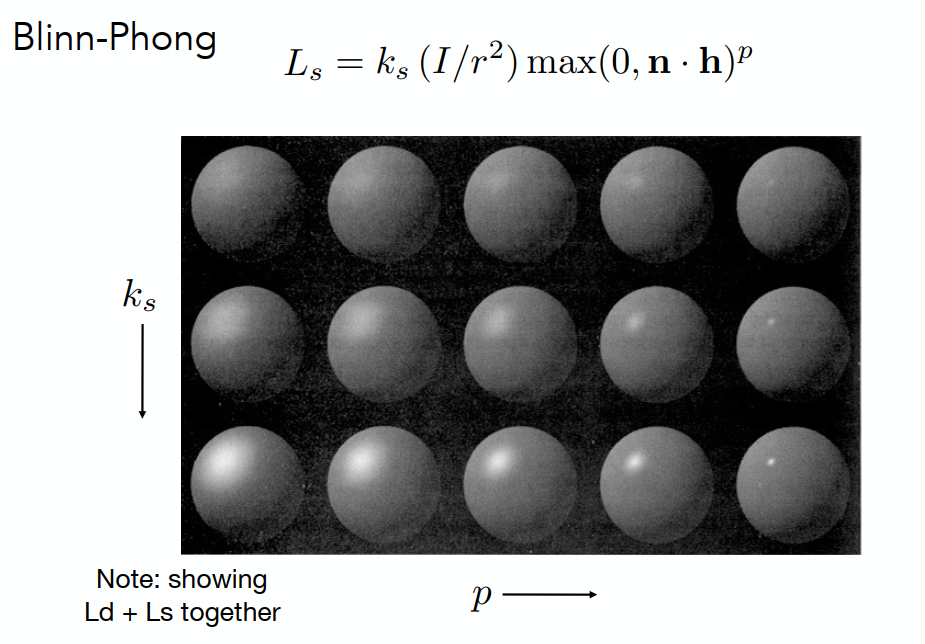
Increasing p narrows the reflection lobe(增加p以缩小容忍度)



为什么cos要引入一个指数？

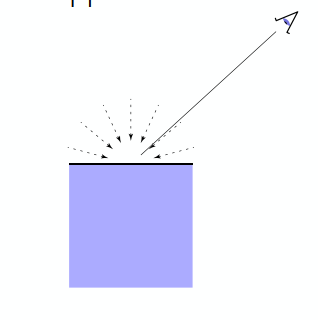
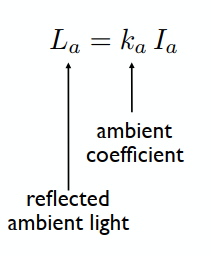
上图最左侧图我们可以看出，如果是cosα，随着夹角变大，由1变到0非常缓慢，假设当**n**和**h**夹角在45°时，感觉上这两个向量夹角已经离得很远了(说明已经不是镜面反射了)，但是在夹角余弦中，仍然有一个较大的值，此时高光还是很大，但是实际上**n**和**h**夹角这么大的时候说明不是镜面反射，高光肯定没有这么大。于是我们引入指数，可以看到随着指数的增大，n和h夹角余弦的指数倍的值减小的会越来越快，也就达到了我们的目的。正常情况下指数会用到100~200这个范围。

从这个图，每一行的球横向比较可以看出来，当ks一定时，p越大，高光范围越小。



## 2、Ambient Term(环境光)

Shading that does not depend on anything(不受观测方向及光照方向影响)

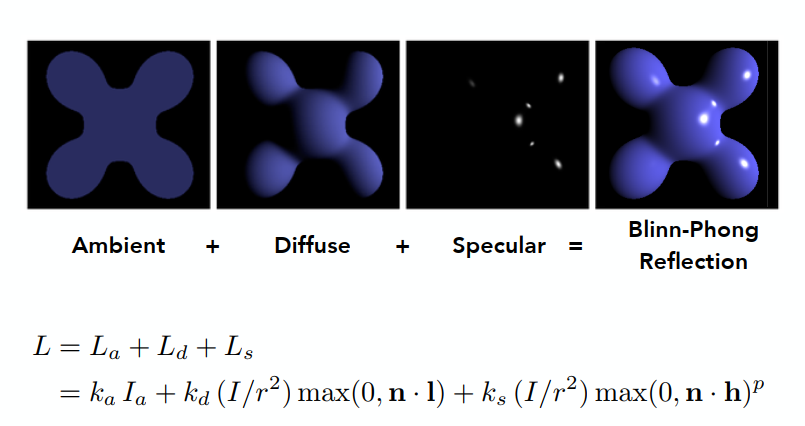
 

Add constant color to account for disregarded illumination and fill in black shadows（其实简化版的环境光就是物体表面的颜色，是个常数。他的存在就是为了避免黑影）

在此我们暂且认为环境光就是一个单一值的颜色光。如果想要精确计算，要用到全局光照的知识（以后再讲）。

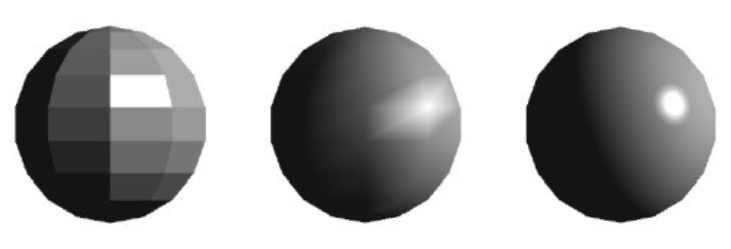
## 3、Blinn-Phong Reflection Model

当上面三部分：漫反射（Diffuse Reflection）、高光（Specular Term）、环境光（Ambient Term）三部分的式子我们都推导完成以后，将三部分的式子加起来也就是Blinn-Phong反射模型。



# (二)Shading Frequencies

What caused the shading difference?



所谓的着色频率，其实就是把着色应用在哪些“点”上，

左图把着色应用在每个面上，一个平面只做一次shading。

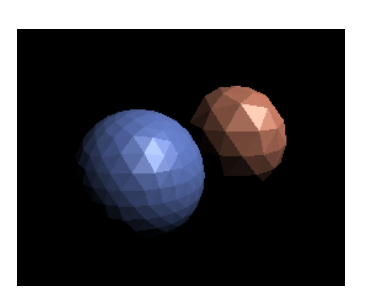
中图把着色应用在每个顶点上，一个面（假设是三角面）有三个顶点，计算面上每一个顶点对应的法线，每一个顶点做一次着色，然后通过插值计算三角形内部每个点的颜色。

右图把着色应用在每个像素上，对于每个面（还是假设是三角形），求出每个顶点的法线，然后把这些法线的方向在三角形内部进行插值，得到任何一个像素都有一个自己的法线方向，并且可以做一遍着色。

## 1、Shade each triangle (flat shading)

Triangle face is flat — one normal vector

Not good for smooth surfaces

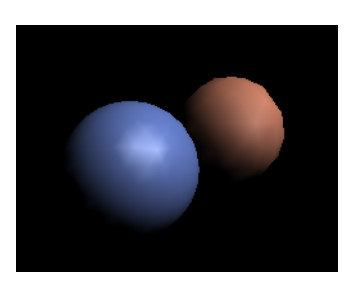


将三角形的两条边做一个叉乘，就可以得到三角形的法线，然后对这个三角形做一次着色，就是这个三角形的样子。这种着色方式对于三角形的内部不会有任何着色的变化。

## 2、Shade each vertex (Gouraud shading)

Interpolate colors from vertices across triangle

Each vertex has a normal vector (how?)



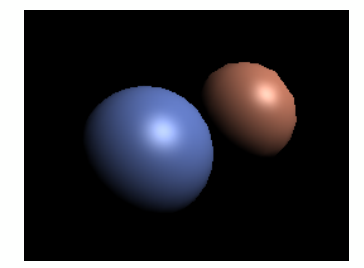
对于三角形每个顶点求出各自的法线，然后每个顶点做一次着色，然后三角形内部的颜色通过插值的方法算出来。但是问题是，当三角形大一点时，高光就不是那么明显了。

## 3、Shade each pixel (Phong shading)

Interpolate normal vectors across each triangle

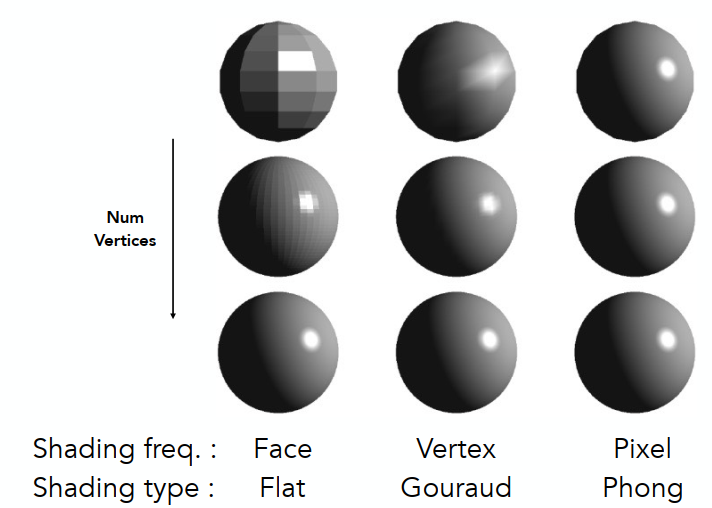
Compute full shading model at each pixel

Not the Blinn-Phong Reflectance Model



对于三角形每个顶点求出各自的法线，在三角形内部每一个像素上都可以插值出一个单独的法线方向，对每一个像素进行一次着色。

## 4、Shading Frequency: Face, Vertex or Pixel



每种着色方式各有利弊。上图中每一行的模型都是一样的，行与行之间的模型面数增加，几何形体本身定义的更加密集（光滑）。

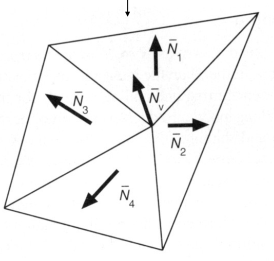
我们可以看出，当几何的形体足够复杂（细分曲面的面数足够多）时，其实就可以用一些相对简单的着色模型，效果与利用复杂的着色模型的效果几乎没有差别。

因此，着色频率取决于面（或者顶线、像素）出现的频率。当这些因素出现的频率很高时，就不再需要用很高的着色频率去着色了。（着色频率越高计算量越大，所以需要跟面（或者顶线、像素）之间做一个权衡，两者结合达到一个合理的开销）

## 5、Defining Per-Vertex Normal Vectors(如何计算2中的顶点法线)

Infer vertex normals from triangle faces(如何从三角形面计算顶点法线)？

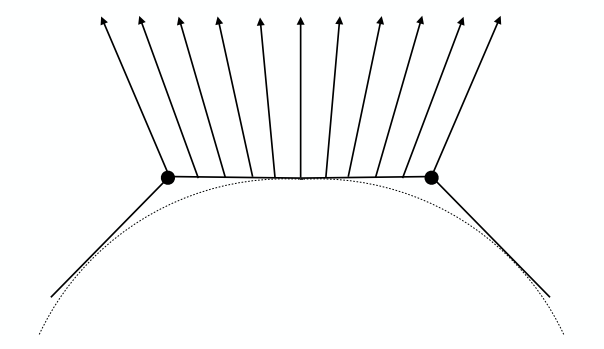
Simple scheme: average surrounding face normals(周围三角形面法线的平均)



将顶点所关联的周围的多面形面的法线求一个平均（无论是简单平均还是加权平均（根据顶点周围的多边形面的各自面积比例的加权），作为顶点的法线。

## 6、Defining Per-Pixel Normal Vectors(如何计算3中的像素法线)

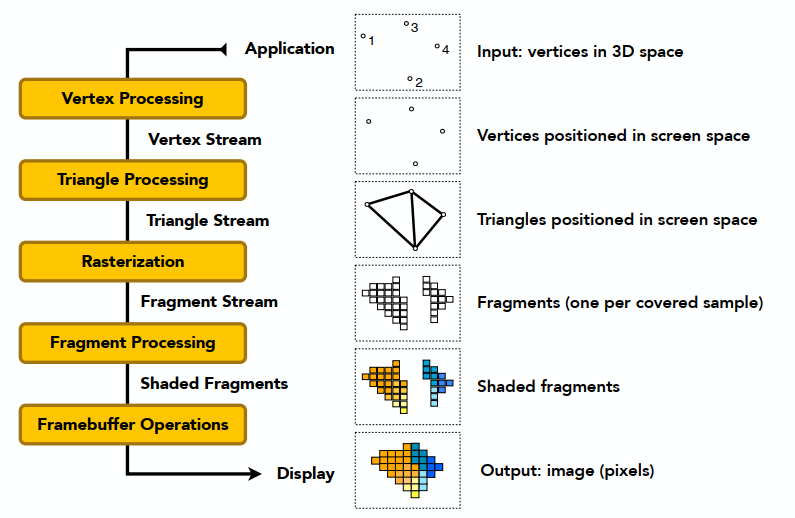
运用顶点法线的重心坐标(一会讲到)插值



Don’t forget to normalize the interpolated directions(归一化插值向量)

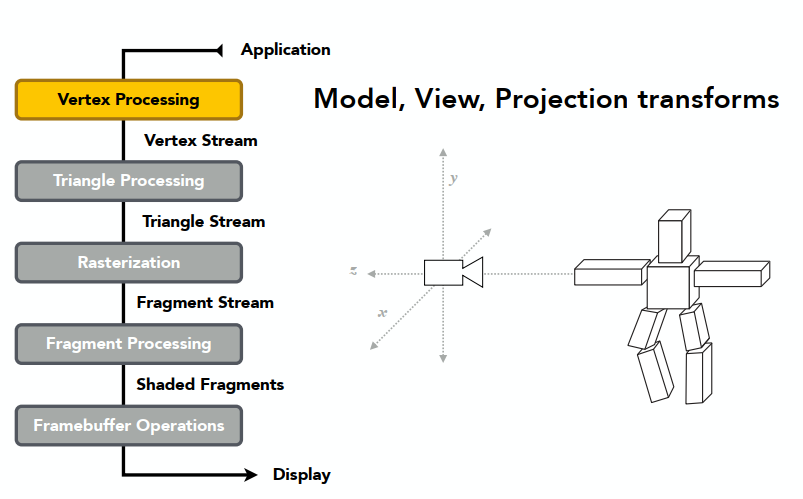
# (三)、Graphics (Real-time Rendering) Pipeline

整个渲染管线流程：

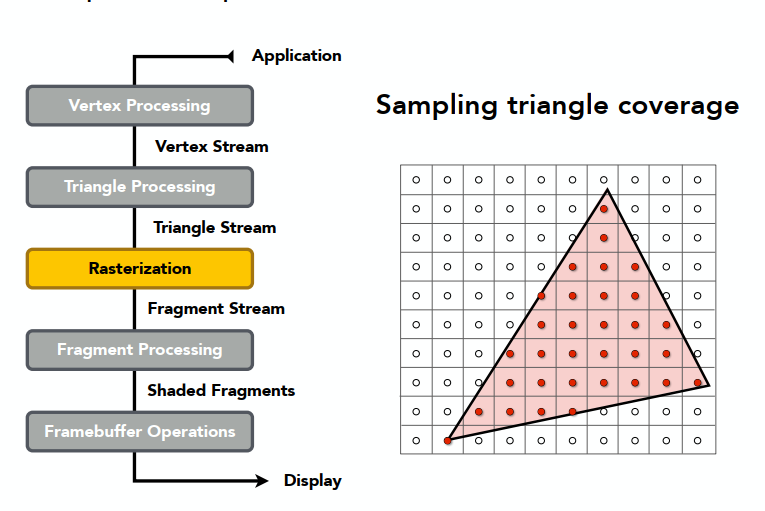


回顾目前学习的模块：

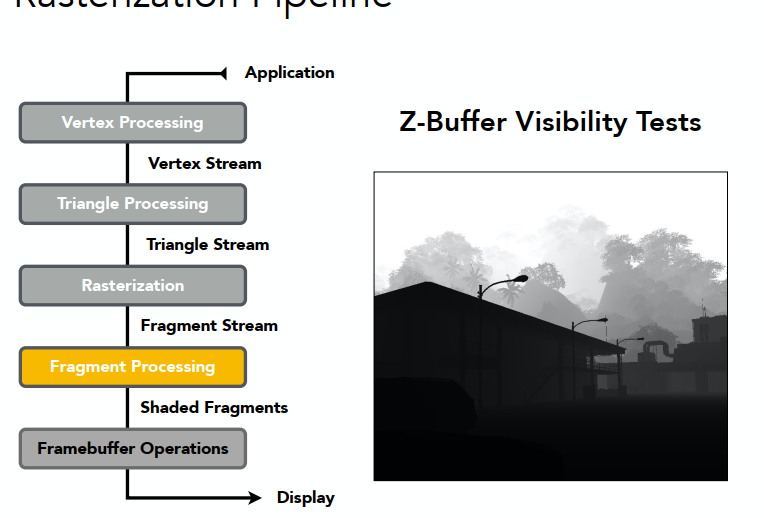
顶点操作:MVP变换



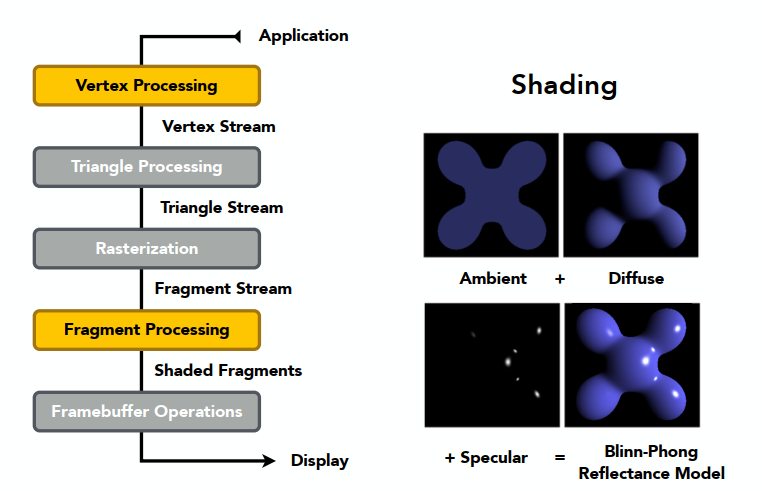
光栅化



深度缓存



着色



## Shader Programs

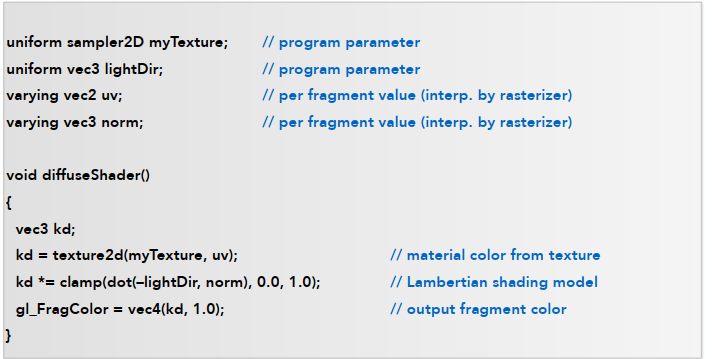
程序顶点和片段处理阶段

描述单个顶点（或片段）上的操作

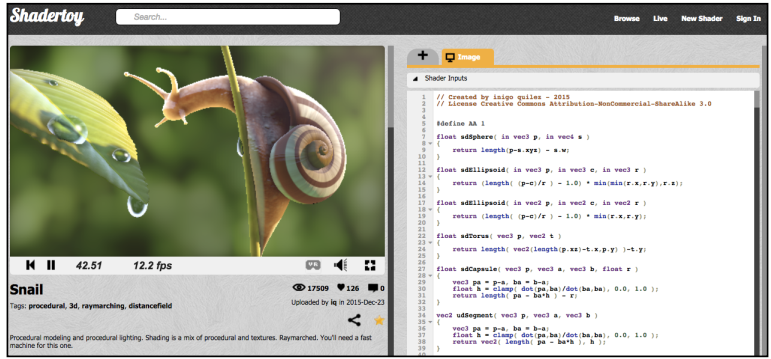
现在的GPU允许自己去编程来实现对顶点或者像素的着色

Example GLSL fragment shader program

本代码讲解[参考视频](https://www.bilibili.com/video/BV1X7411F744?p=8&spm_id_from=pageDriver&vd_source=669b556fb7f994f37ee00750260684bf)第42分钟

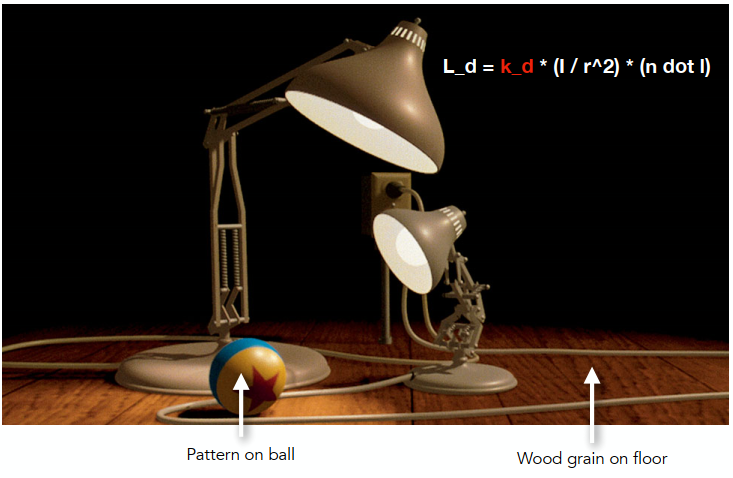


在线shading网站[Shader - Shadertoy BETA](https://www.shadertoy.com/view/ld3Gz2)



# (四) Texture Mapping(纹理映射)

## Different Colors at Different Places?

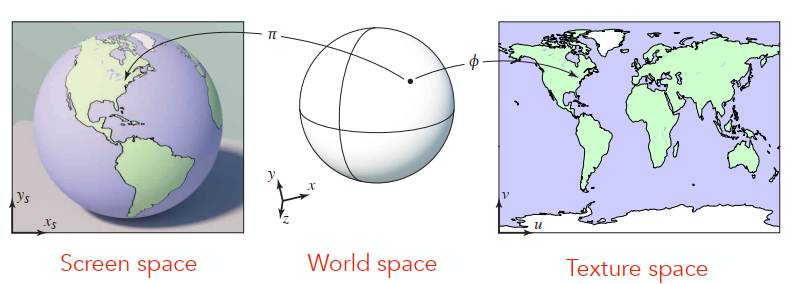


假设有两个光源照到一个球上，这个球上面有不同的颜色，也就是说kd的值不同，我们希望能用一种方式，定义这个球上每一个顶点的属性。因此引入纹理映射。

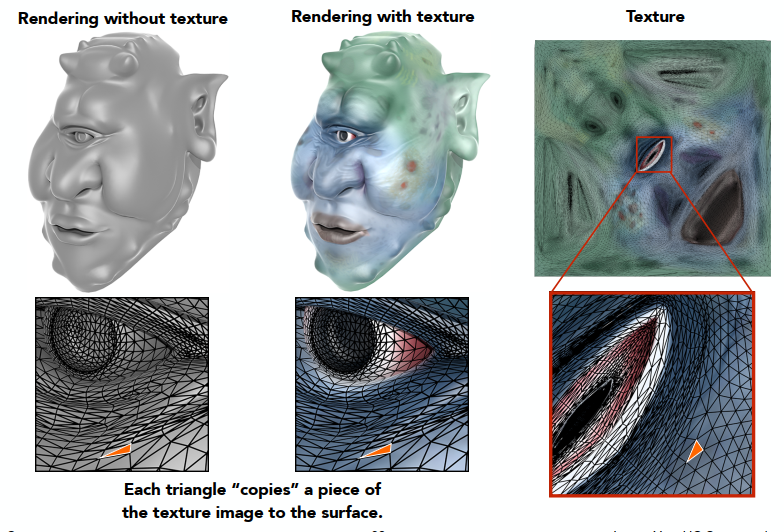
## Surfaces are 2D

任何一个三维物体的表面其实都是二维的，如上图对地球仪的拆解即可以解释这个现象。

通过对二维纹理的拉伸、变换等一系列操作，将其能蒙到三维物体上，我们称这个过程即为纹理映射。纹理映射使物体表面任何一个点和纹理上一个点产生一一对应关系。



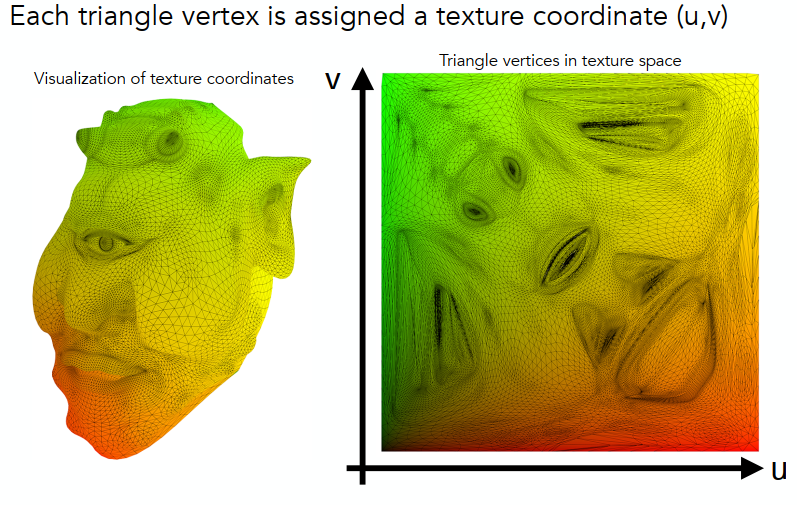
## Texture Applied to Surface



3D物体上三角形的顶点已经规定了纹理图上的对应坐标。(人工标注或者自动化)

## Visualization of Texture Coordinates

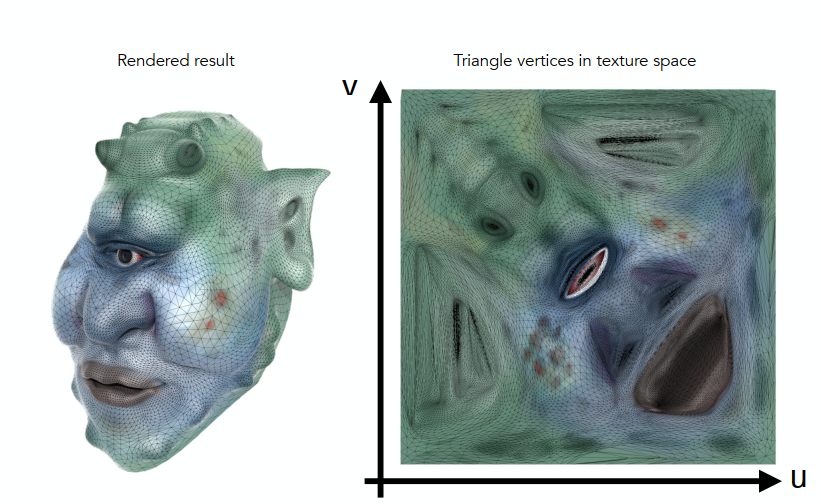
Each triangle vertex is assigned a texture coordinate (u,v)



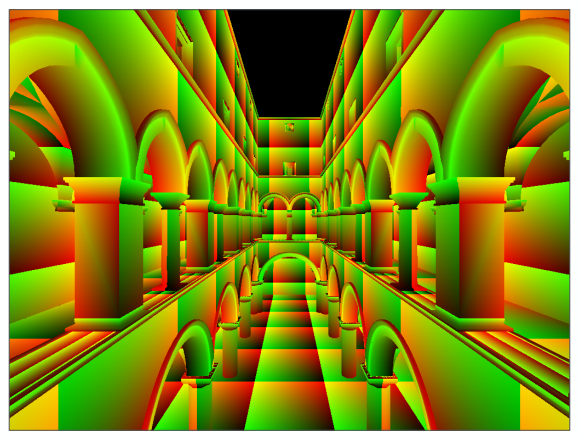
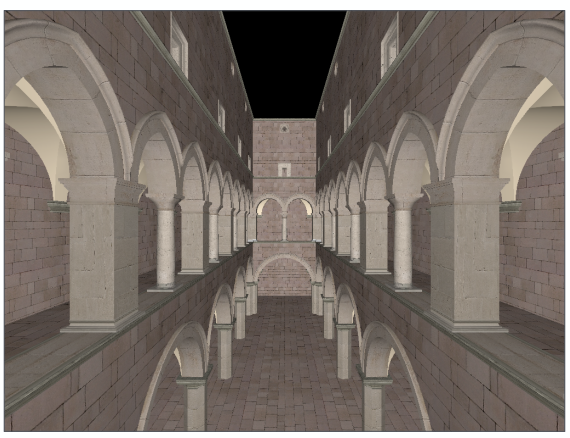
我们通常用（u，v）坐标来表示纹理上任意一个点，一般u和v的范围都从0到1。

## Texture Applied to Surface

三角形三个顶点，每个顶点都对应一个uv。



纹理可以重复，也可以做到无缝衔接



Textures can be used multiple times!

