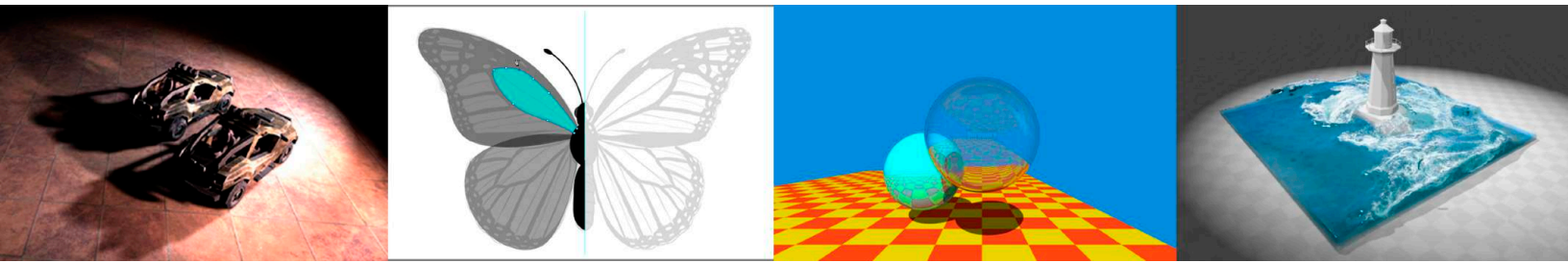
Introduction to Computer Graphics

GAMES101, Lingqi Yan, UC Santa Barbara

Lecture 8:
Shading 2
(Shading, Pipeline and Texture Mapping)

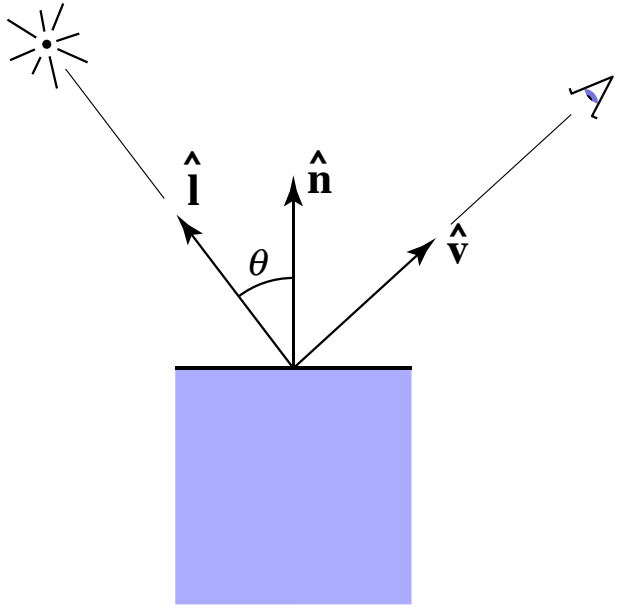


Announcements

- Homework 2
  - 45 submissions so far
  - Upside down? No problem
  - Active discussions in the BBS, pretty good
- Next homework is for shading
- Today's topics
  - Easy, but a lot

Last Lecture

- Shading 1
  - Blinn-Phong reflectance model
    - Diffuse
    - Specular
    - Ambient
  - At a **specific shading point**

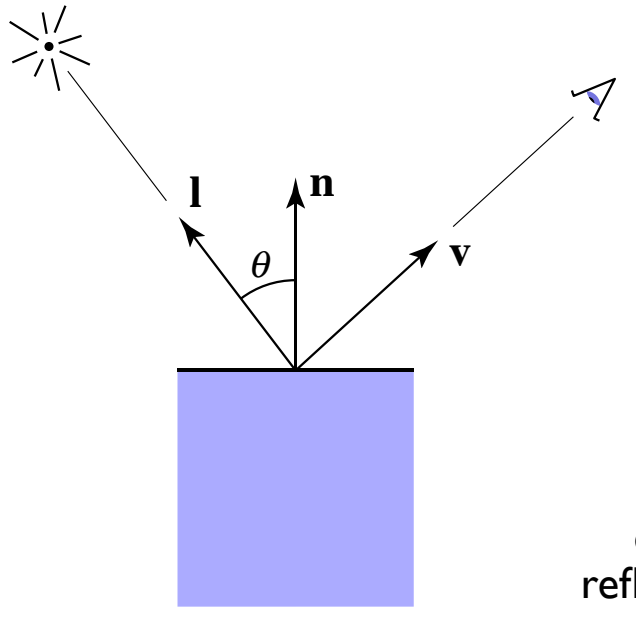


Today

- Shading 2
  - Blinn-Phong reflectance model
    - Specular and ambient terms
  - Shading frequencies
  - Graphics pipeline
  - Texture mapping
  - Barycentric coordinates

Recap: Lambertian (Diffuse) Term

Shading independent of view direction



energy arrived at the shading point

$$L_d = k_d \, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l})$$

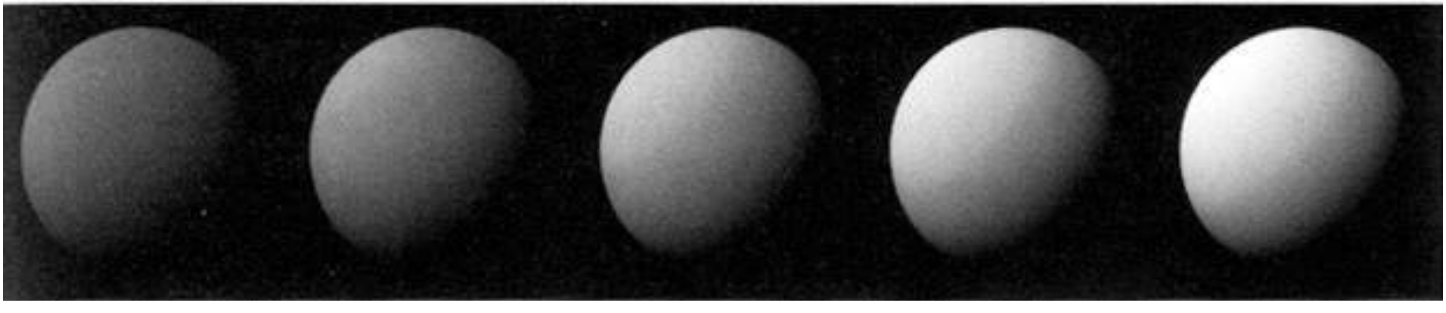
diffuse coefficient (color)

energy received by the shading point

diffusely reflected light

Recap: Lambertian (Diffuse) Term

Produces diffuse appearance



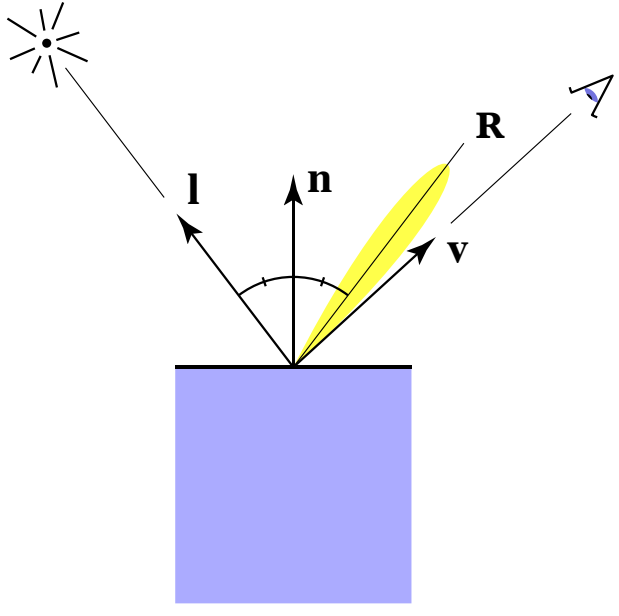
k_d ⟶

[Foley et al.]

Specular Term (Blinn-Phong)

Intensity depends on view direction

- Bright near mirror reflection direction

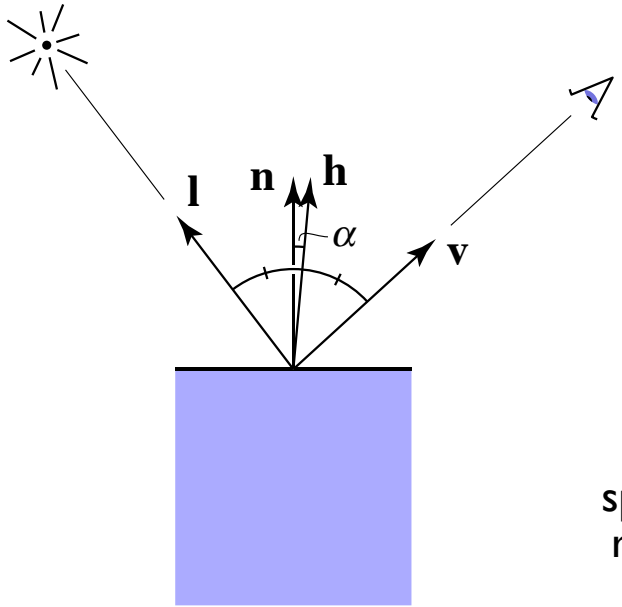


Specular Term (Blinn-Phong)

V 和 R 接近　　h 和 n 接近

V close to mirror direction ⇔ **half vector** near normal

- Measure "near" by dot product of unit vectors



$$\mathbf{h} = \text{bisector}(\mathbf{v}, \mathbf{l})$$

(半程向量)

$$= \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$

$$L_s = k_s \left(I/r^2\right) \max(0, \cos\alpha)^p$$

$$= k_s \left(I/r^2\right) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

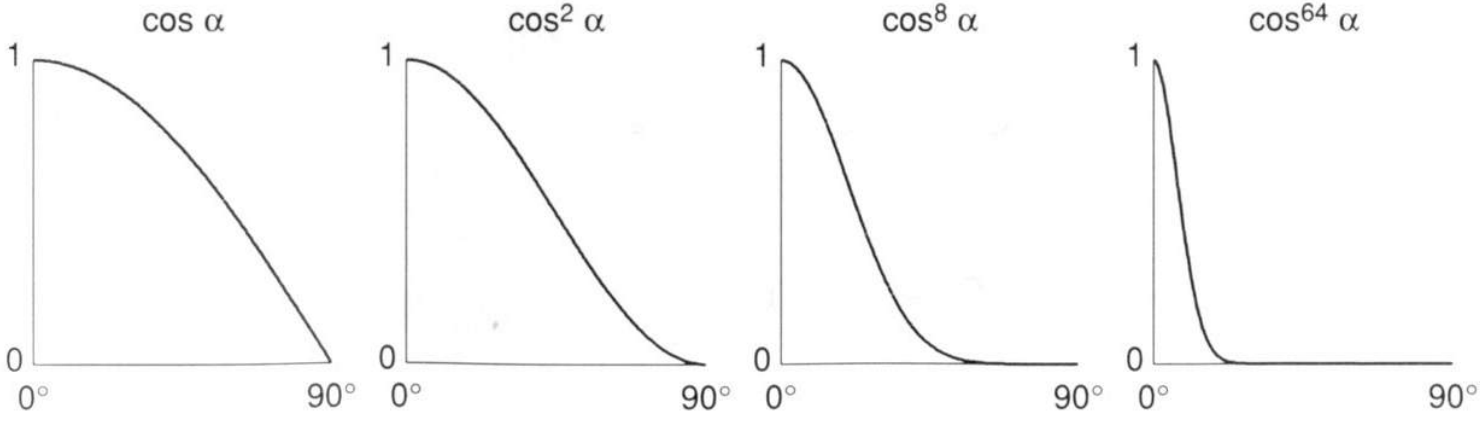
specularly reflected light

specular coefficient 镜面反射系数

Cosine Power Plots

增大p使反射波瓣变窄 (100<p<300)

Increasing p narrows the reflection lobe

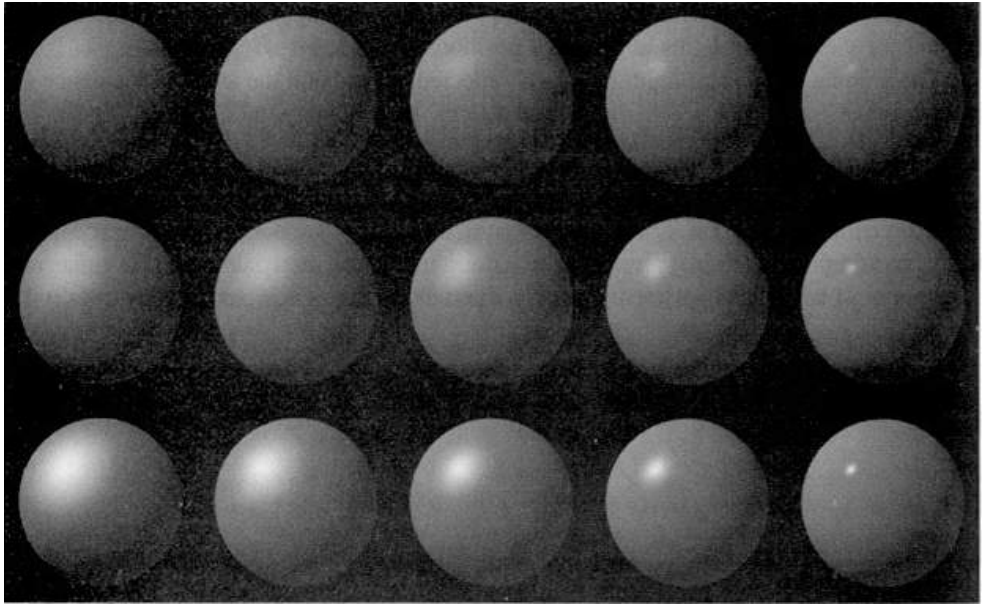


[Foley et al.]

Specular Term (Blinn-Phong)

Blinn-Phong

$$L_s = k_s \left(I/r^2\right) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$



k_s

p

Note: showing
Ld + Ls together

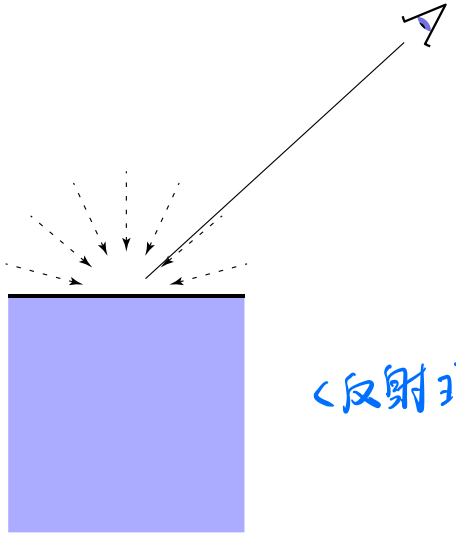
[Foley et al.]

Ambient Term 环境光照

Shading that does not depend on anything

- Add constant color to account for disregarded illumination and fill in black shadows

- This is approximate / fake!

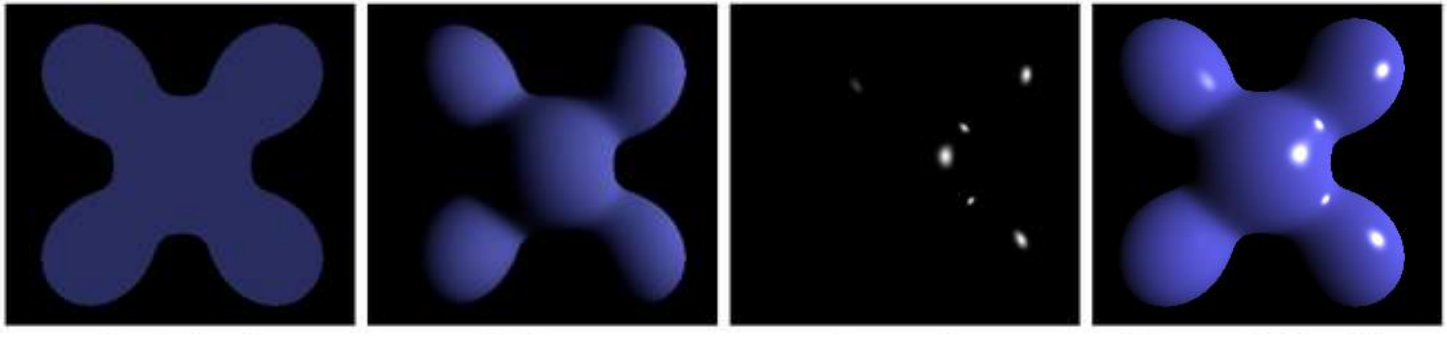


$$L_a = k_a I_a$$

ambient coefficient

reflected ambient light

Blinn-Phong Reflection Model



Ambient + Diffuse + Specular = Blinn-Phong Reflection

环境光照 漫反射 高光

$$L = L_a + L_d + L_s$$

$$= k_a\, I_a + k_d\, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s\, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

Questions?

Shading Frequencies

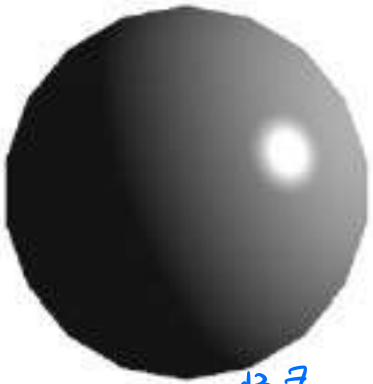
着色频率

Shading Frequencies

What caused the shading difference?

一个平面做1次 shading

每个平面有顶点.
对顶点计算shading后
内部进行插值
(应用在顶点)

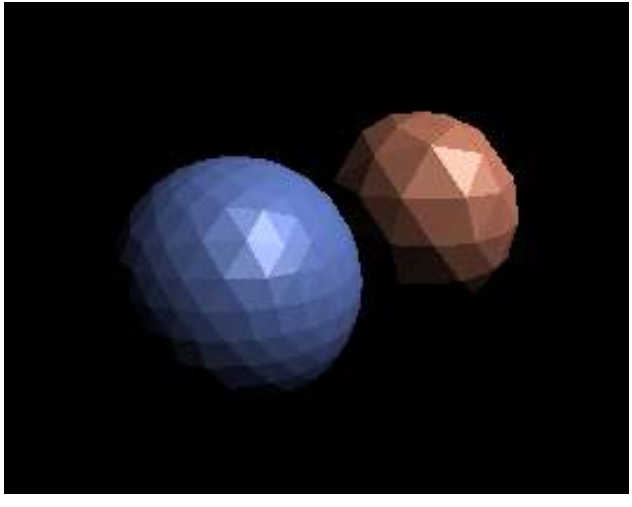


应用在每个像素.
在每个顶点求出法线方向
在三角形内部进行插值
从而每个像素都有独立法线方向.

Shade each triangle (flat shading)

Flat shading

- Triangle face is flat — one normal vector
- Not good for smooth surfaces



Shade each vertex (Gouraud shading)

Gouraud shading <顶点>

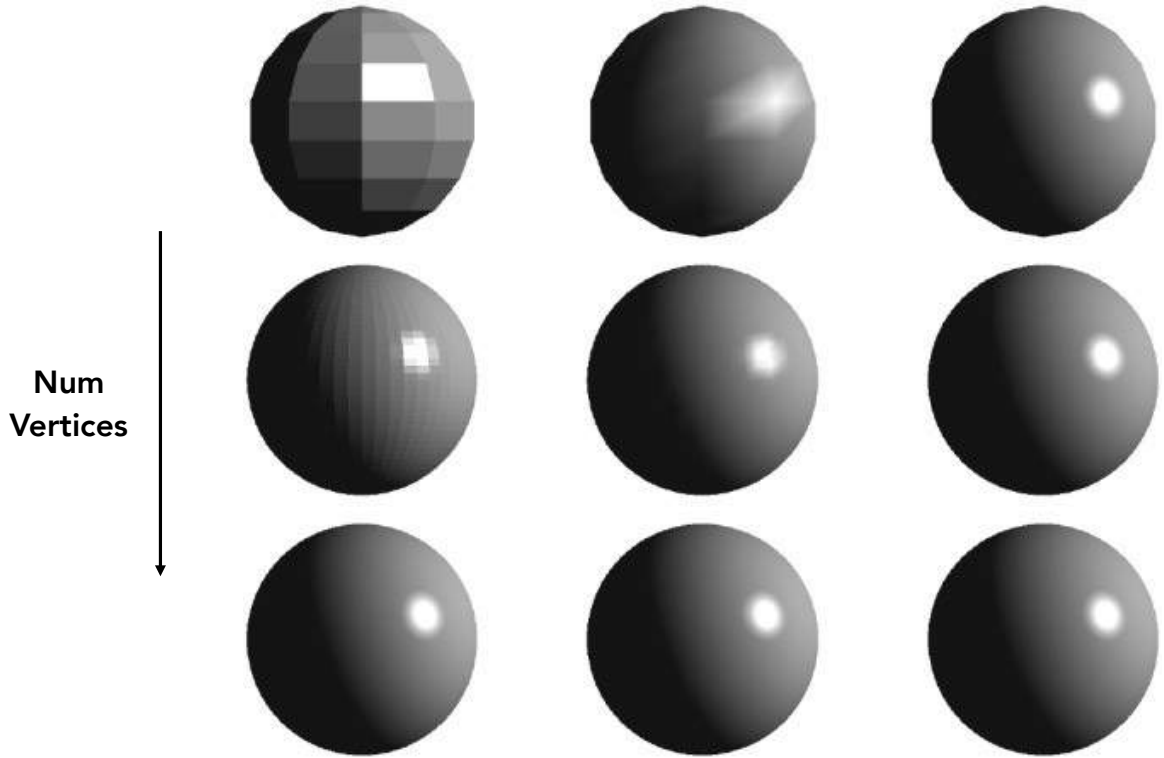
- Interpolate colors from vertices across triangle
- Each vertex has a normal vector (how?)

Shade each pixel (Phong shading)

Phong shading <像素>

- Interpolate normal vectors across each triangle
- Compute full shading model at each pixel
- Not the **Blinn-Phong Reflectance Model**

Shading Frequency: Face, Vertex or Pixel



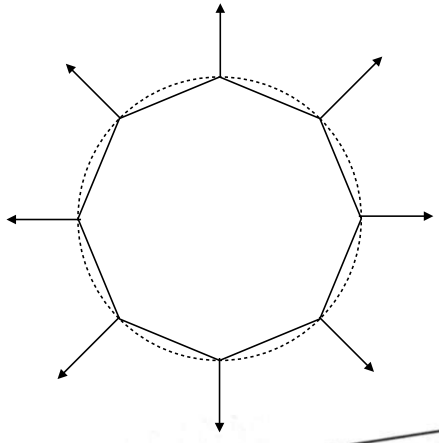
Shading freq. :	Face	Vertex	Pixel
Shading type :	Flat	Gouraud	Phong

Image credit: Happyman, http://cg2010studio.com/

Defining Per-Vertex Normal Vectors

Best to get vertex normals from the underlying geometry

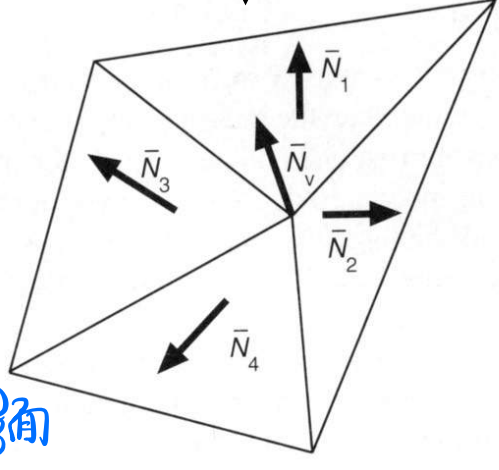
- e.g. consider a sphere



Otherwise have to infer vertex normals from triangle faces

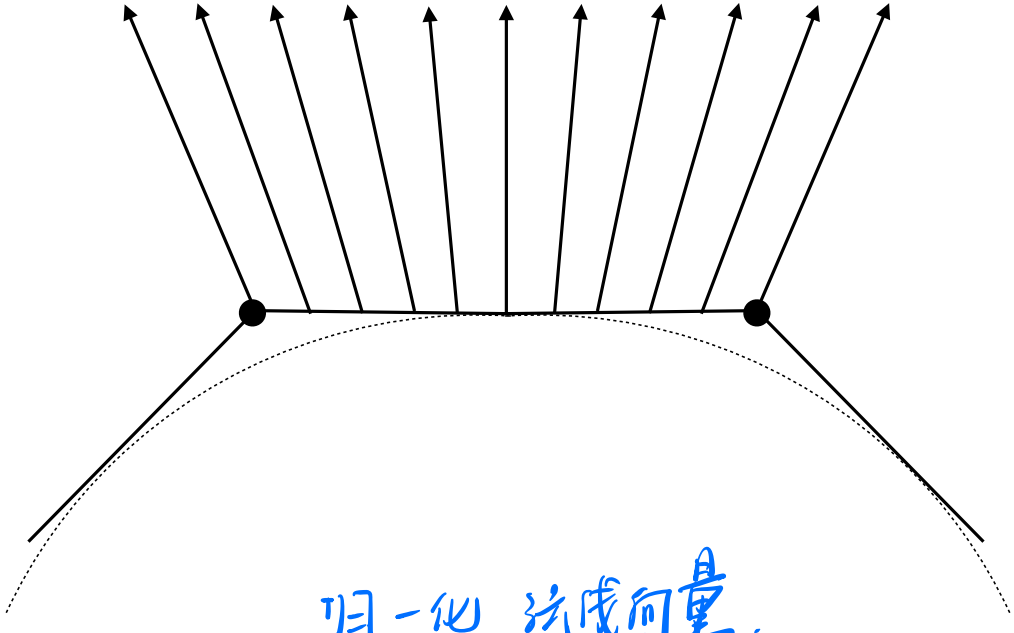
- Simple scheme: **average surrounding face normals**

$$N_v = \frac{\sum_i N_i}{\| \sum_i N_i \|}$$



Defining Per-Pixel Normal Vectors

Barycentric interpolation (introducing soon) of vertex normals

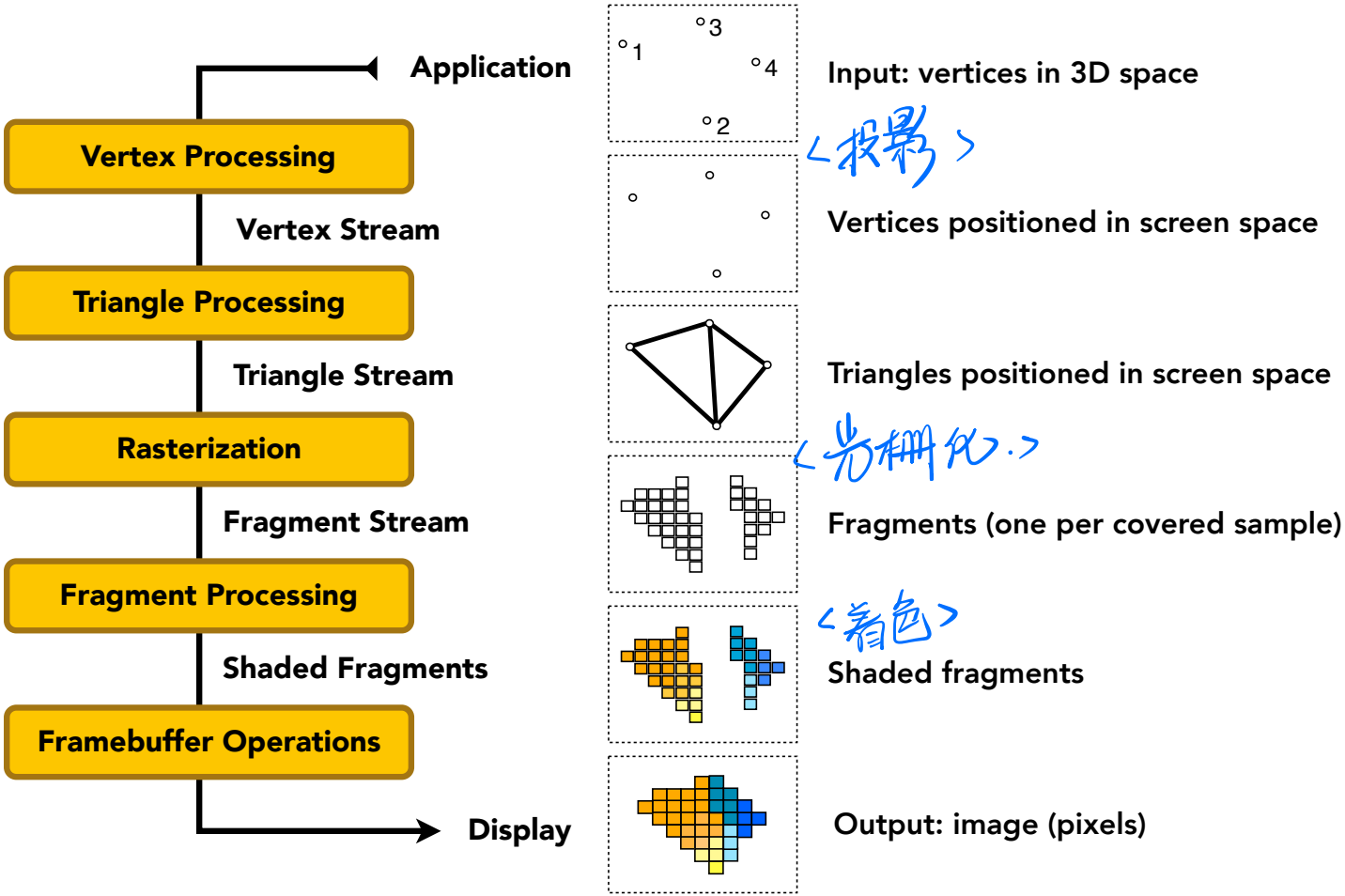


Don’t forget to normalize the interpolated directions

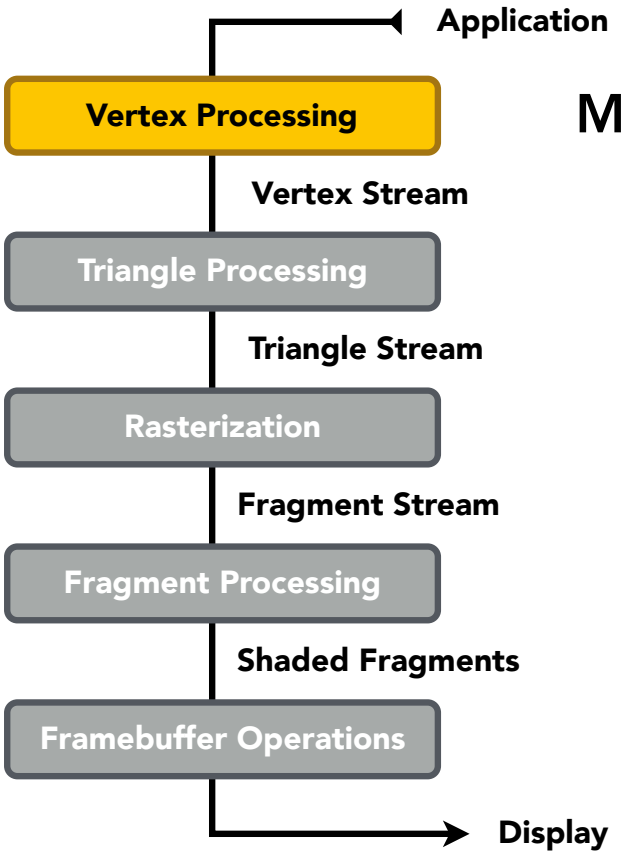
实时渲染管线

Graphics (**Real-time Rendering**) Pipeline

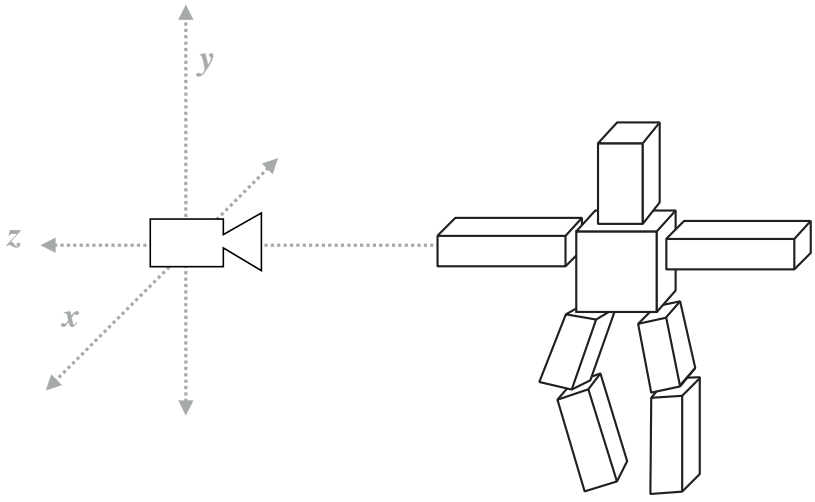
Graphics Pipeline



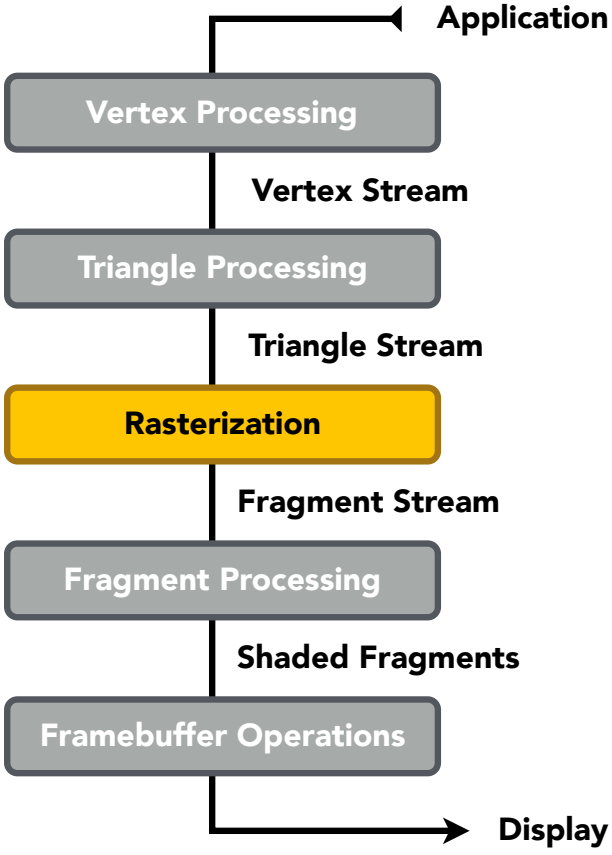
Graphics Pipeline



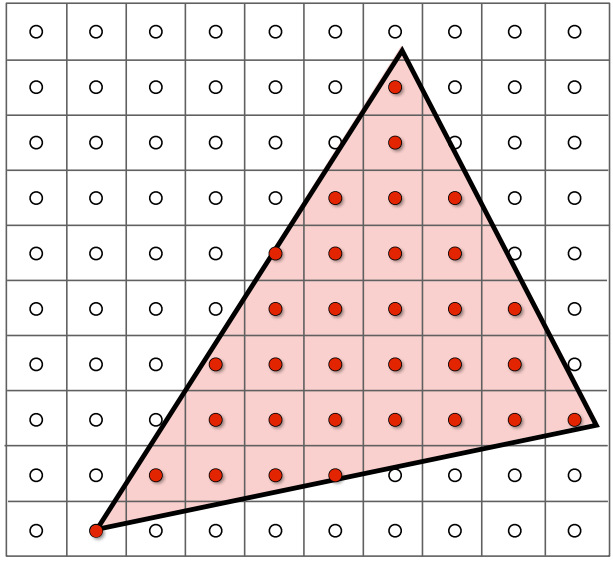
Model, View, Projection transforms



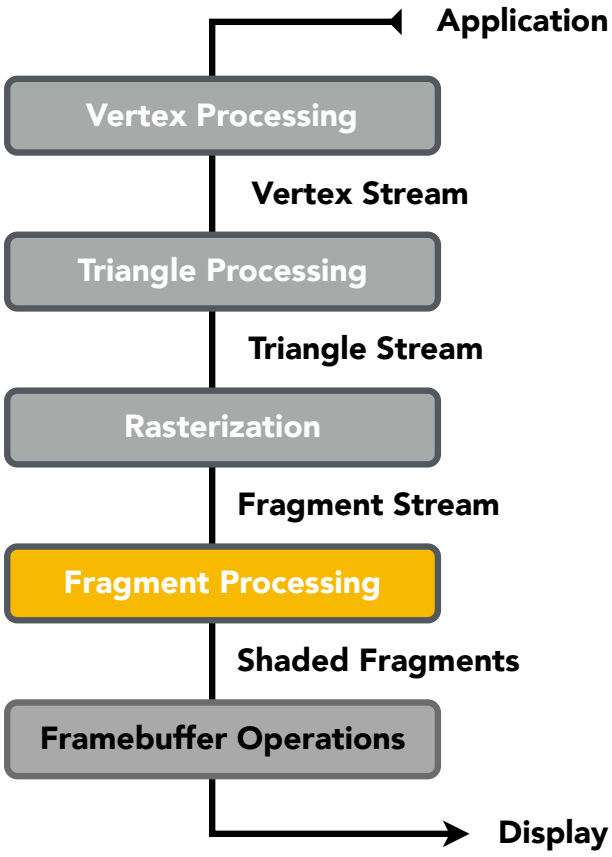
Graphics Pipeline



Sampling triangle coverage

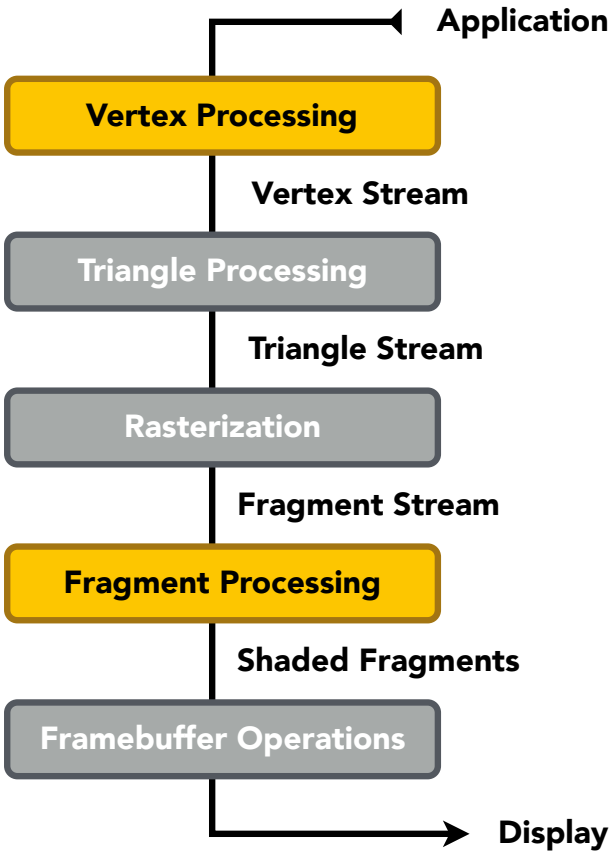


Rasterization Pipeline

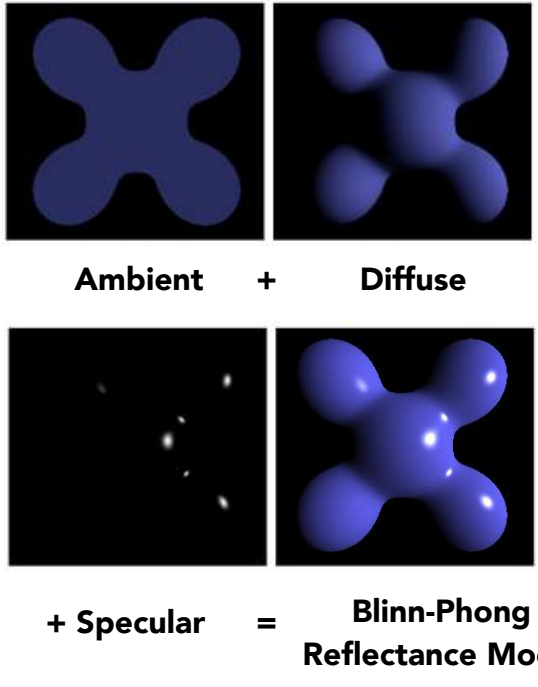


Z-Buffer Visibility Tests

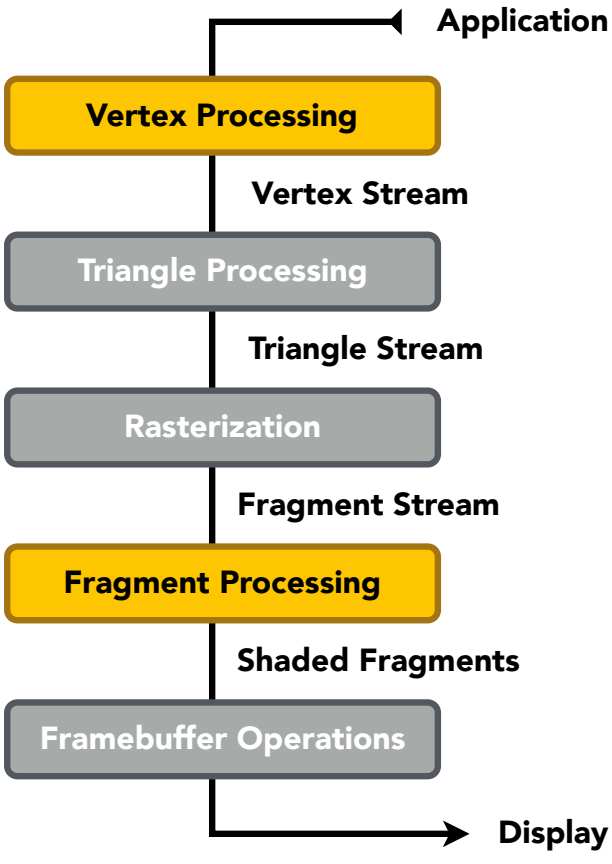
Graphics Pipeline



Shading



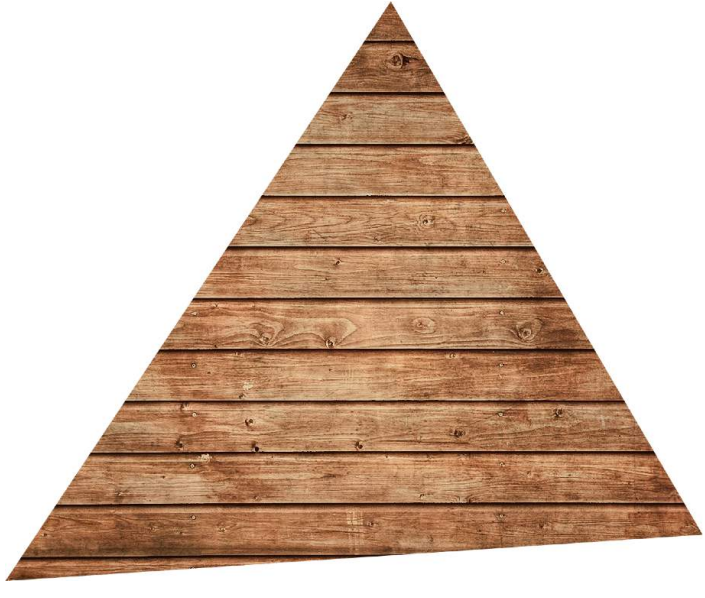
Graphics Pipeline



纹理映射

Texture mapping

(introducing soon)



Shader Programs

- Program vertex and fragment processing stages
- Describe operation on a single vertex (or fragment)

Example GLSL fragment shader program

```
uniform sampler2D myTexture;
uniform vec3 lightDir;
varying vec2 uv;
varying vec3 norm;

void diffuseShader()
{
  vec3 kd;
  kd = texture2d(myTexture, uv);
  kd *= clamp(dot(–lightDir, norm), 0.0, 1.0);
  gl_FragColor = vec4(kd, 1.0);
}
```

- **Shader function executes once per fragment.**
- **Outputs color of surface at the current fragment's screen sample position.**
- **This shader performs a texture lookup to obtain the surface's material color at this point, then performs a diffuse lighting calculation.**

Shader Programs

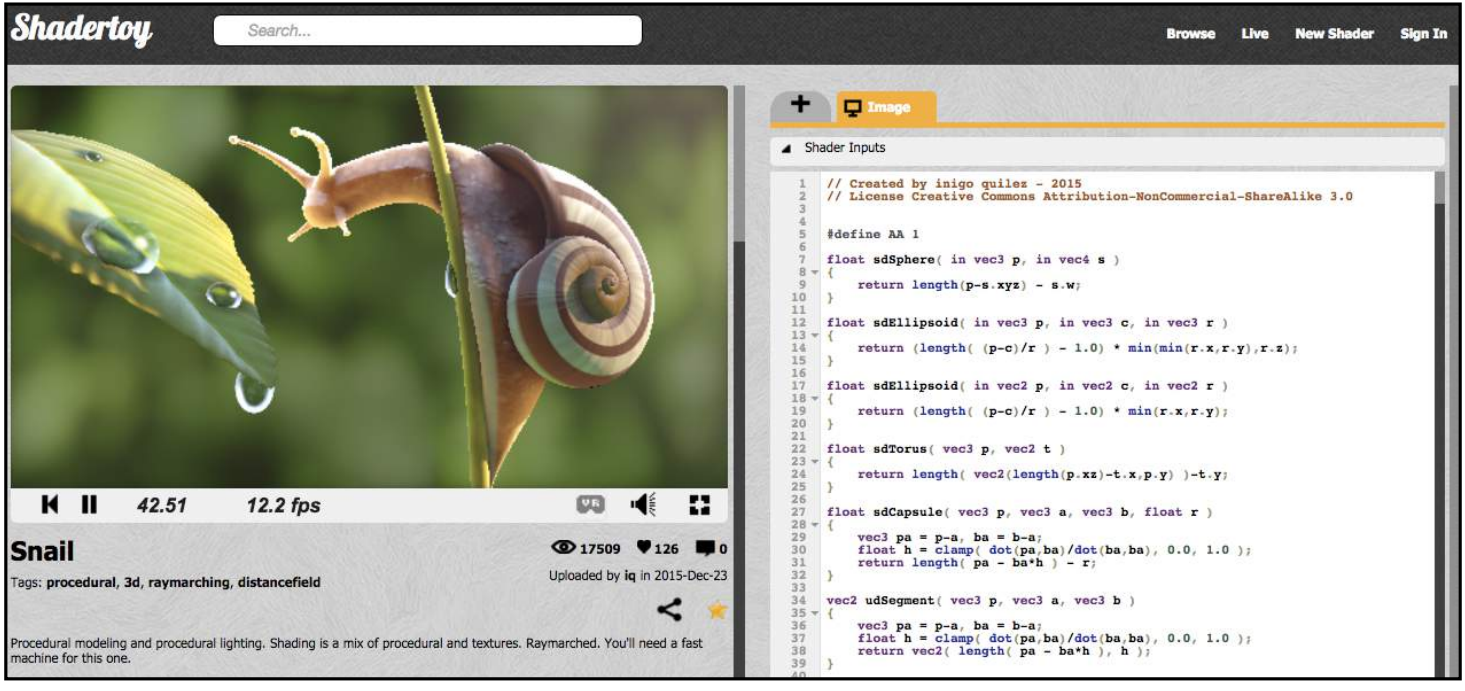
- Program vertex and fragment processing stages
- Describe operation on a single vertex (or fragment)

Example GLSL fragment shader program

```
uniform sampler2D myTexture;      // program parameter
uniform vec3 lightDir;            // program parameter
varying vec2 uv;                  // per fragment value (interp. by rasterizer)
varying vec3 norm;                // per fragment value (interp. by rasterizer)

void diffuseShader()
{
  vec3 kd;
  kd = texture2d(myTexture, uv);                  // material color from texture
  kd *= clamp(dot(–lightDir, norm), 0.0, 1.0);    // Lambertian shading model
  gl_FragColor = vec4(kd, 1.0);                   // output fragment color
}
```

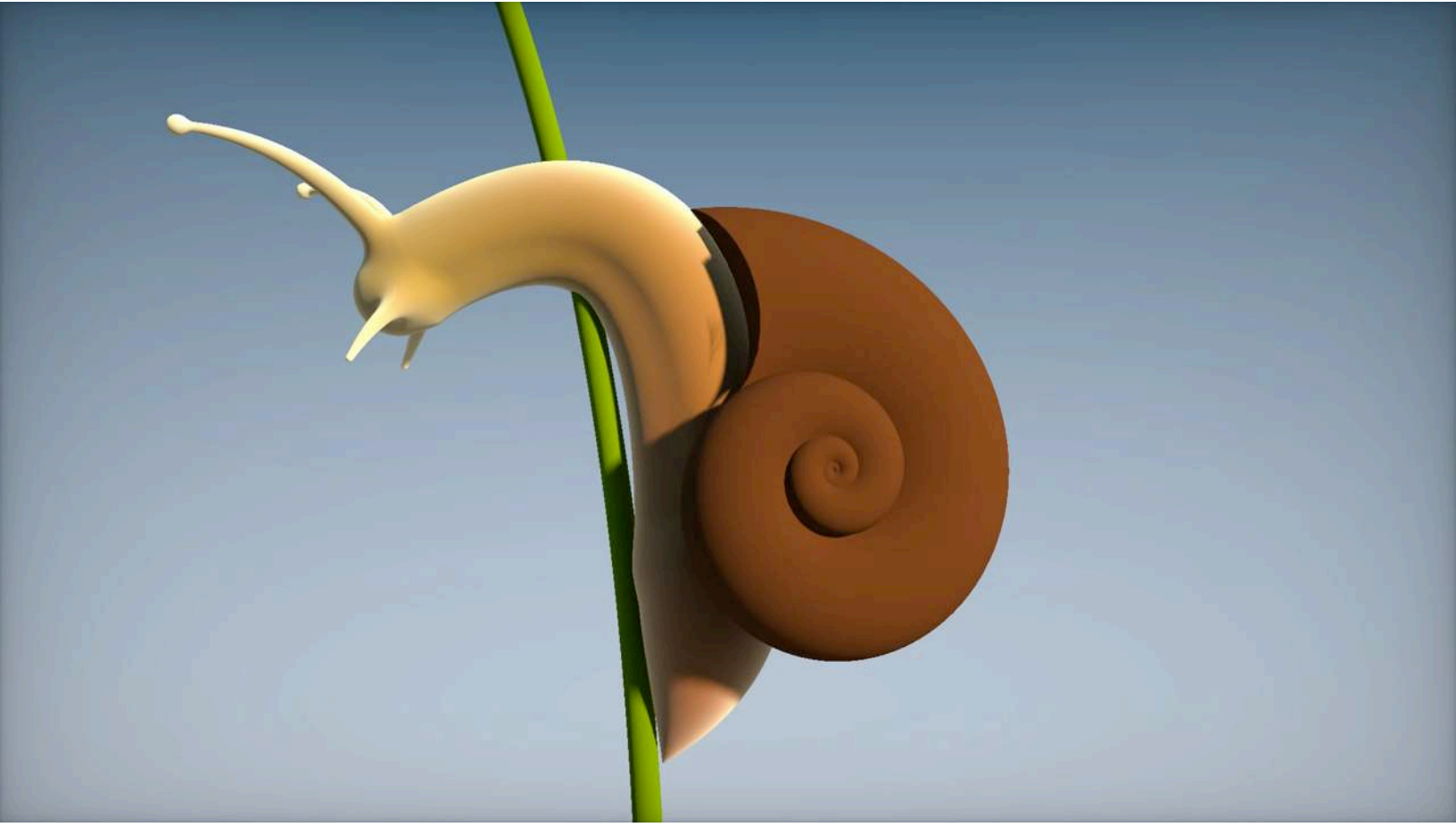
Snail Shader Program



Inigo Quilez

Procedurally modeled, 800 line shader.
http://shadertoy.com/view/ld3Gz2

Snail Shader Program



Inigo Quilez, https://youtu.be/XuSnLbB1j6E

Goal: Highly Complex 3D Scenes in Realtime

Unreal Engine Kite Demo (Epic Games 2015)

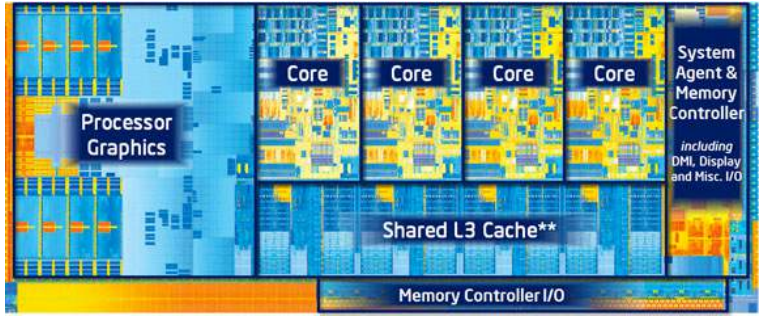
图形管道实现：GPU

Graphics Pipeline Implementation: GPUs

Specialized processors for executing graphics pipeline computations

分离式GPU卡

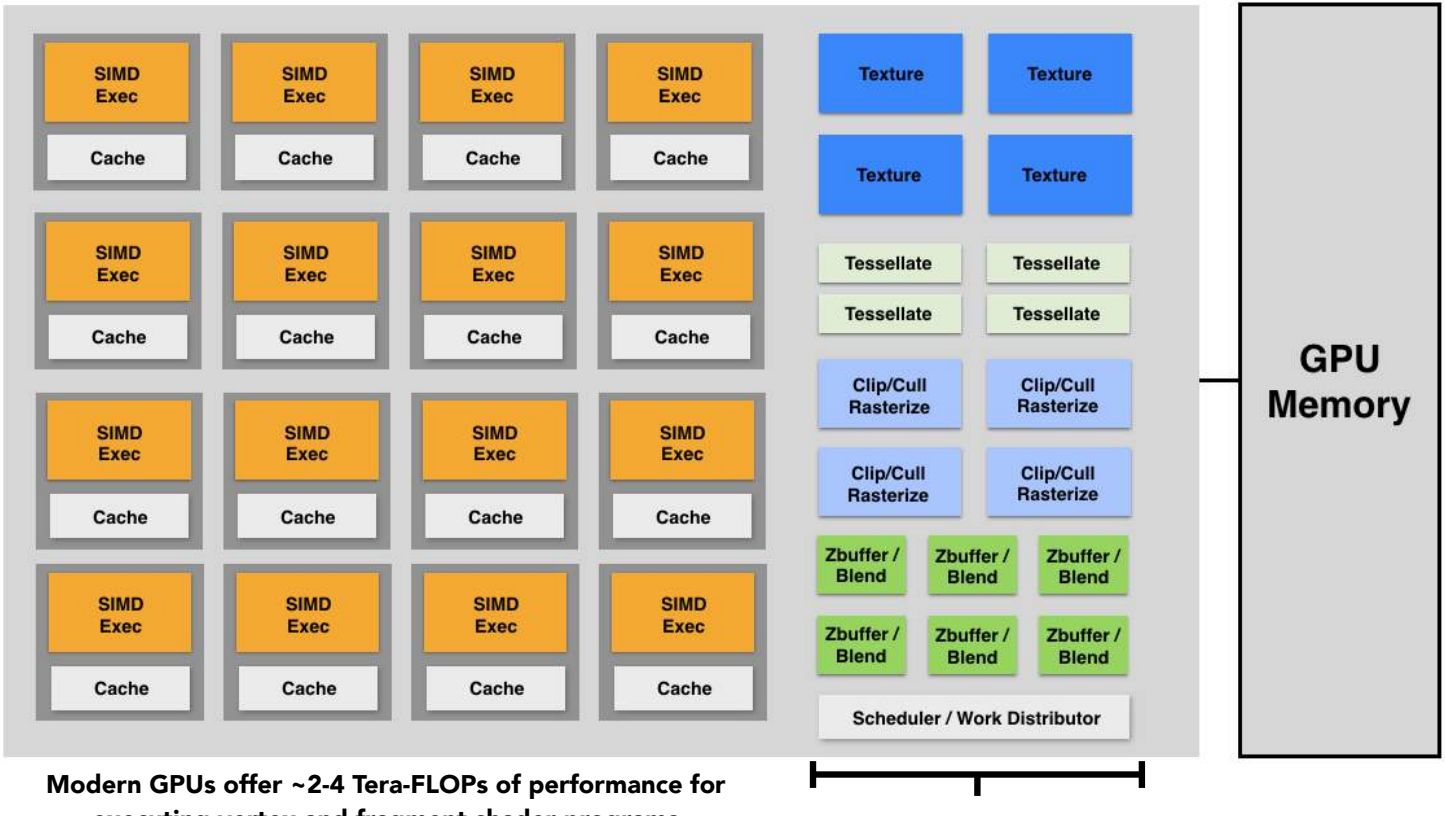
Discrete GPU Card
(NVIDIA GeForce Titan X)



集成显卡

Integrated GPU:
(Part of Intel CPU die)

GPU: Heterogeneous, Multi-Core Procesor



Modern GPUs offer ~2-4 Tera-FLOPs of performance for executing vertex and fragment shader programs

Tera-Op's of fixed-function compute capability over here

Texture Mapping

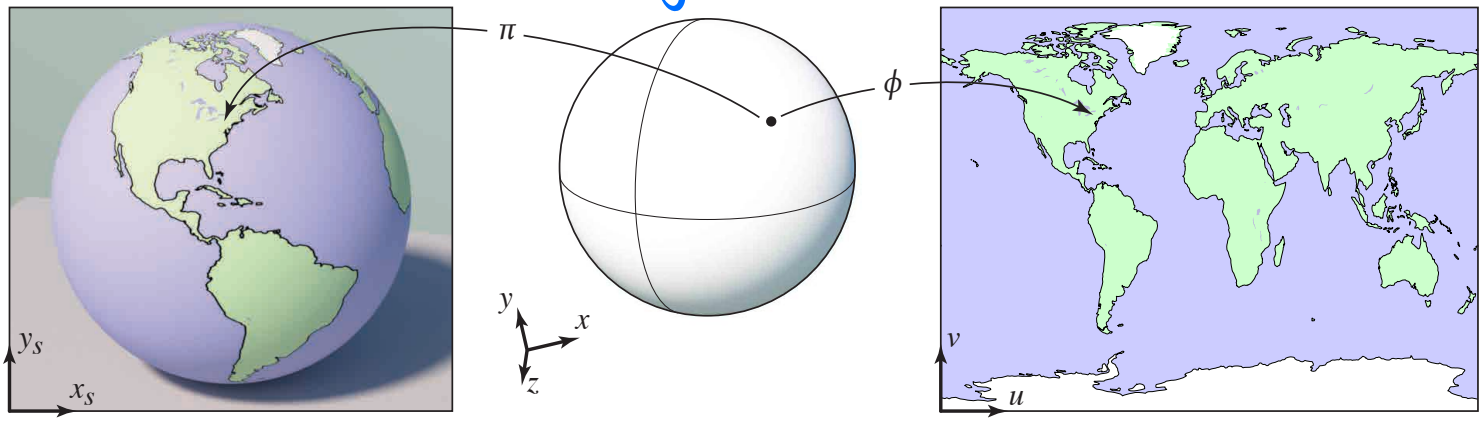
Different Colors at Different Places?

Surfaces are 2D

Surface lives in 3D world space

Every 3D surface point also has a place where it goes in the 2D image (**texture**).

"一一映射"



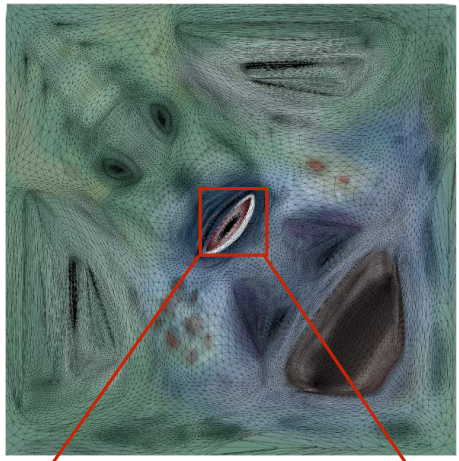
Screen space　　World space　　Texture space

Texture Applied to Surface

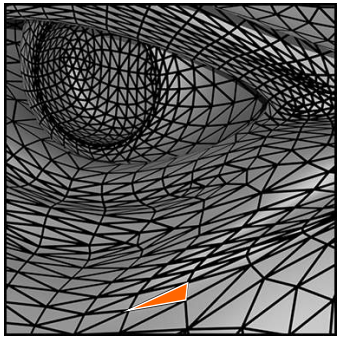
Rendering without texture

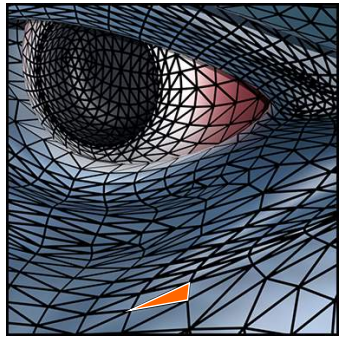
Rendering with texture

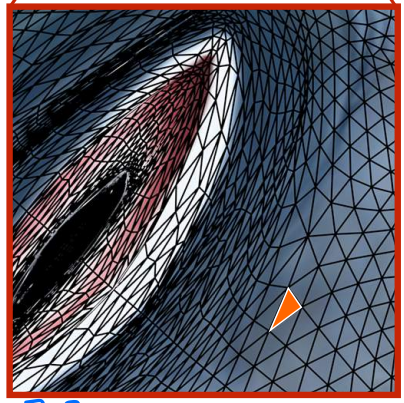
Texture



Zoom







Each triangle "copies" a piece of the texture image to the surface.

每个三角形"复制"一块纹理图像到表面.

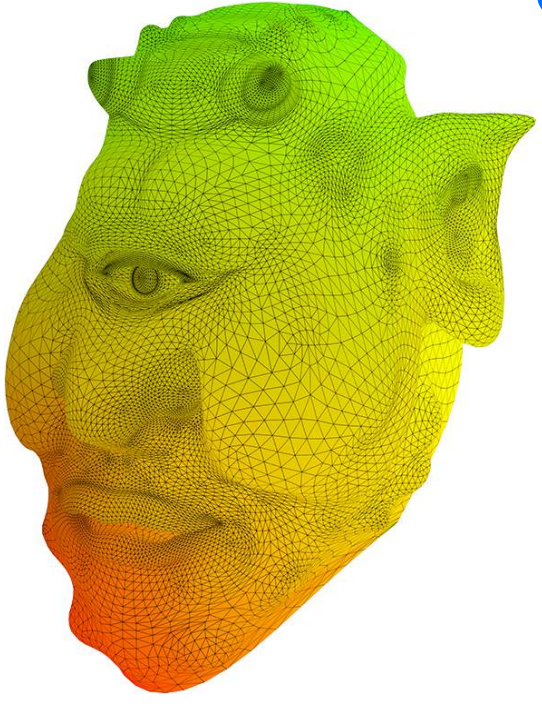
纹理生成的可视化

Visualization of Texture Coordinates

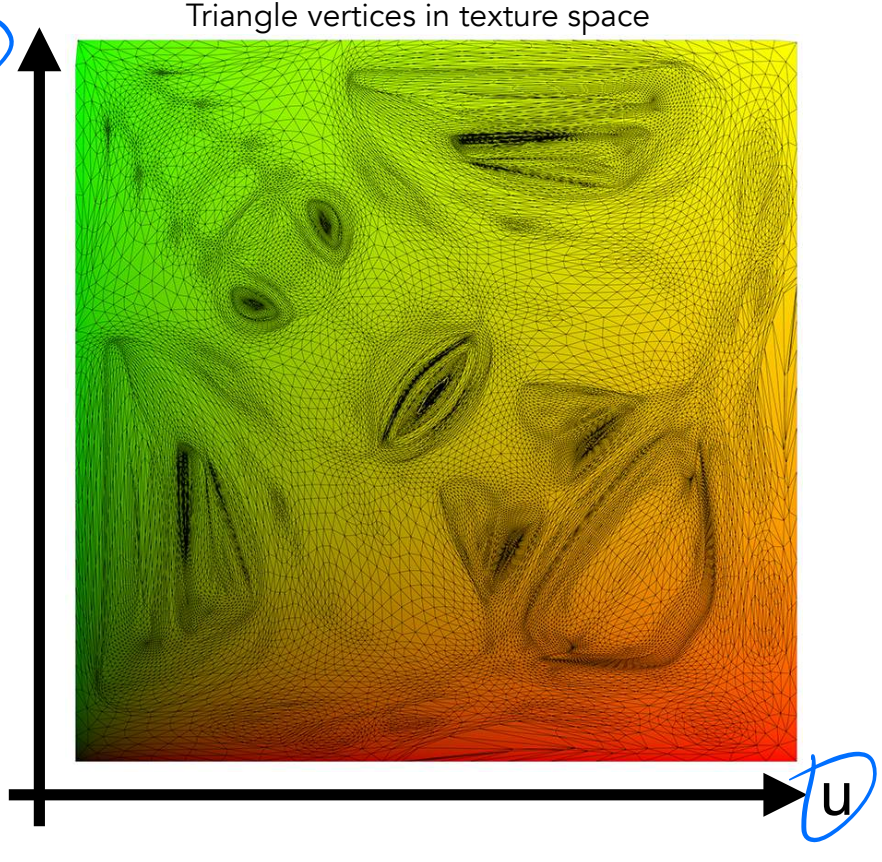
每个三角形顶点分配一个纹理坐标

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

Visualization of texture coordinates

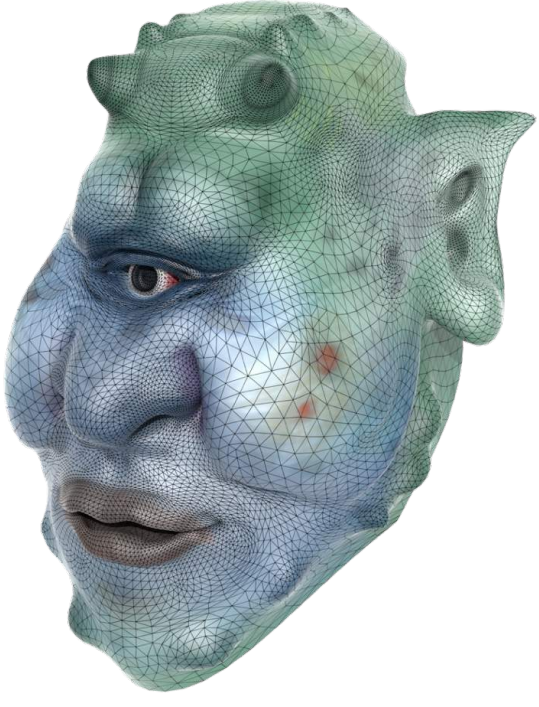


Triangle vertices in texture space

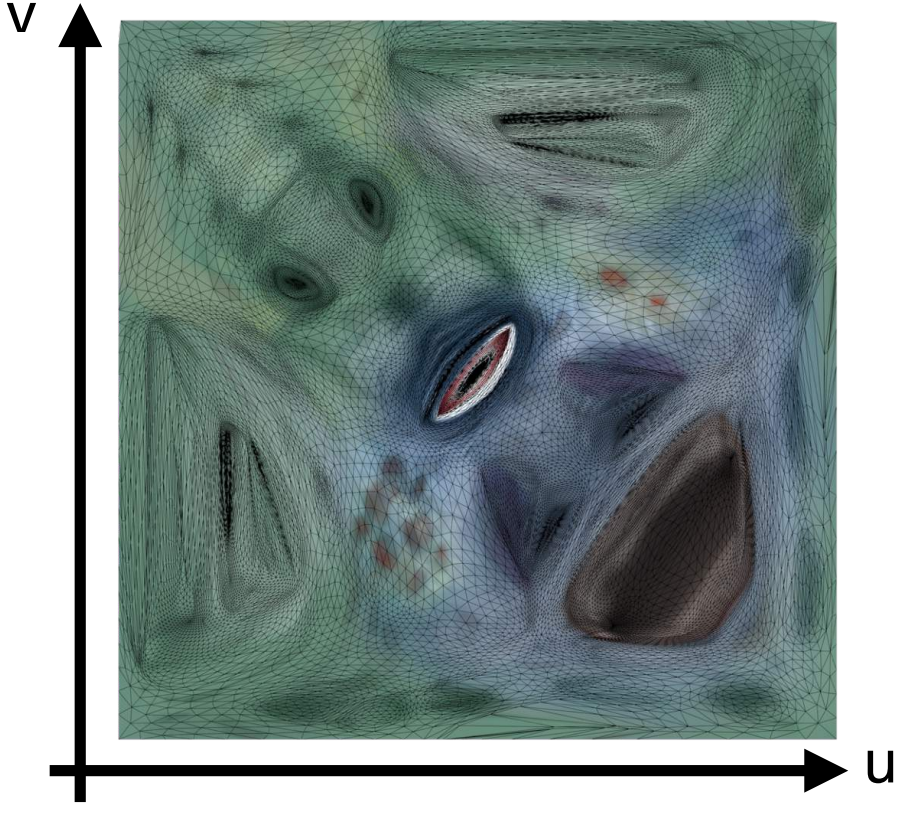


Texture Applied to Surface

Rendered result

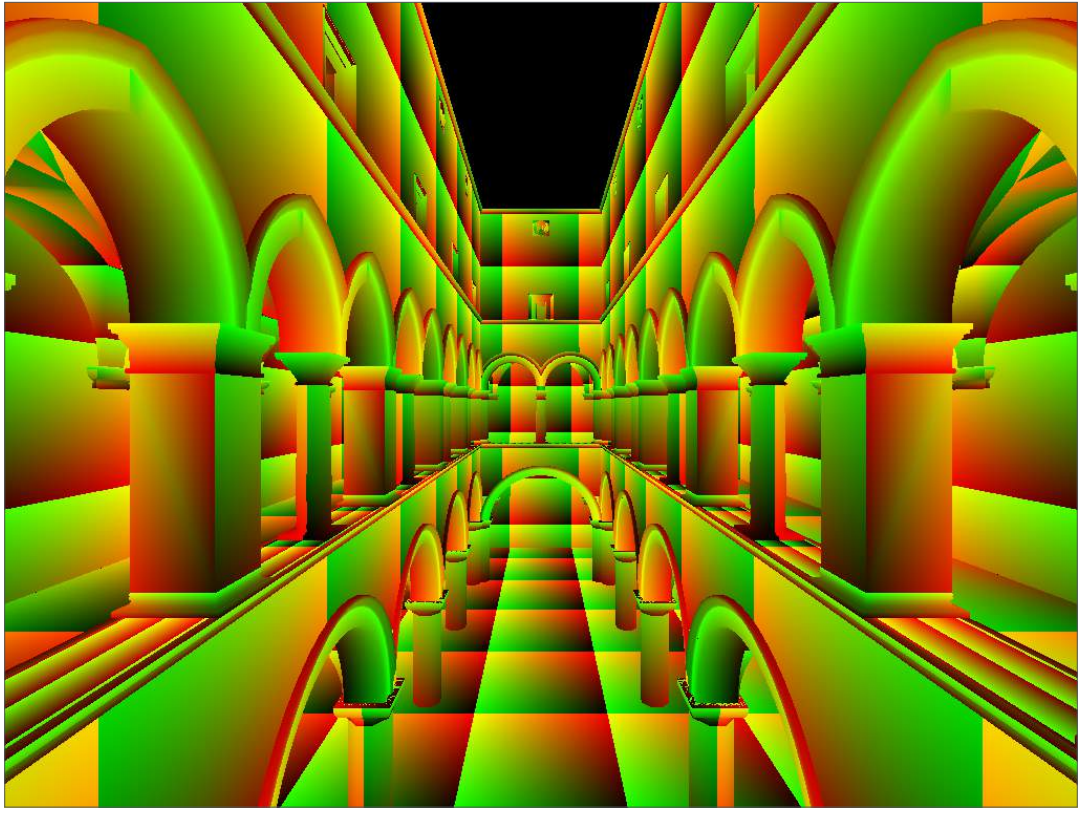


Triangle vertices in texture space



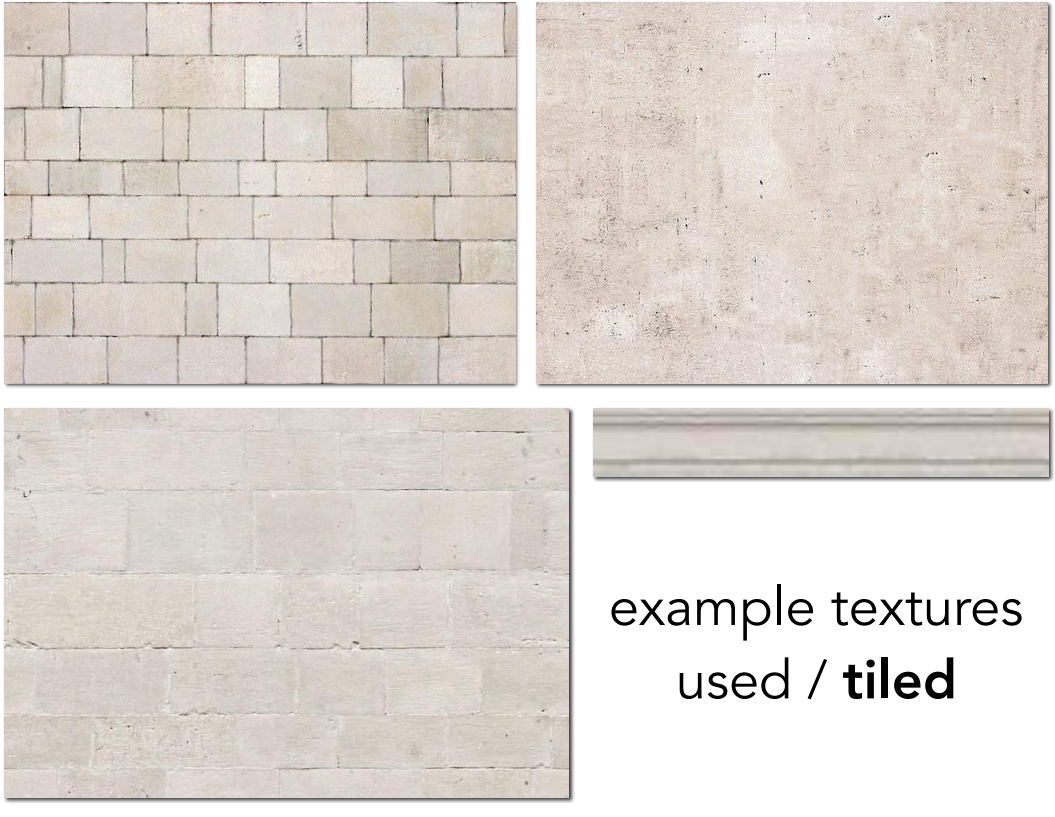
Textures applied to surfaces

Visualization of texture coordinates



Textures can be used multiple times!

纹理可以实现多次.



example textures used / **tiled**

Thank you!

(And thank Prof. Ravi Ramamoorthi and Prof. Ren Ng for many of the slides!)