Lecture 22
Shaders 2

Now that we can we can write shaders...

Now we know how to write shaders...

What shaders do we write?

1. Lighting
2. Textures
3. Other tricks

Lighting

THREE implements lighting very well (nice shaders)

Generally, we don't need to implement it ourselves

- if we want to understand lighting
- if we want to do something non-standard
- if we want to combine it with some other shading

Where to implement lighting?

Vertex Shader

Compute lighting at every vertex

Interpolate colors across triangle

Per-Vertex Lighting

Gouraud Shading

Used in the old days

interpolate colors

Fragment Shader

Interpolate normals and surface colors

Compute lighting at every pixel

Per-pixel (fragment) shading

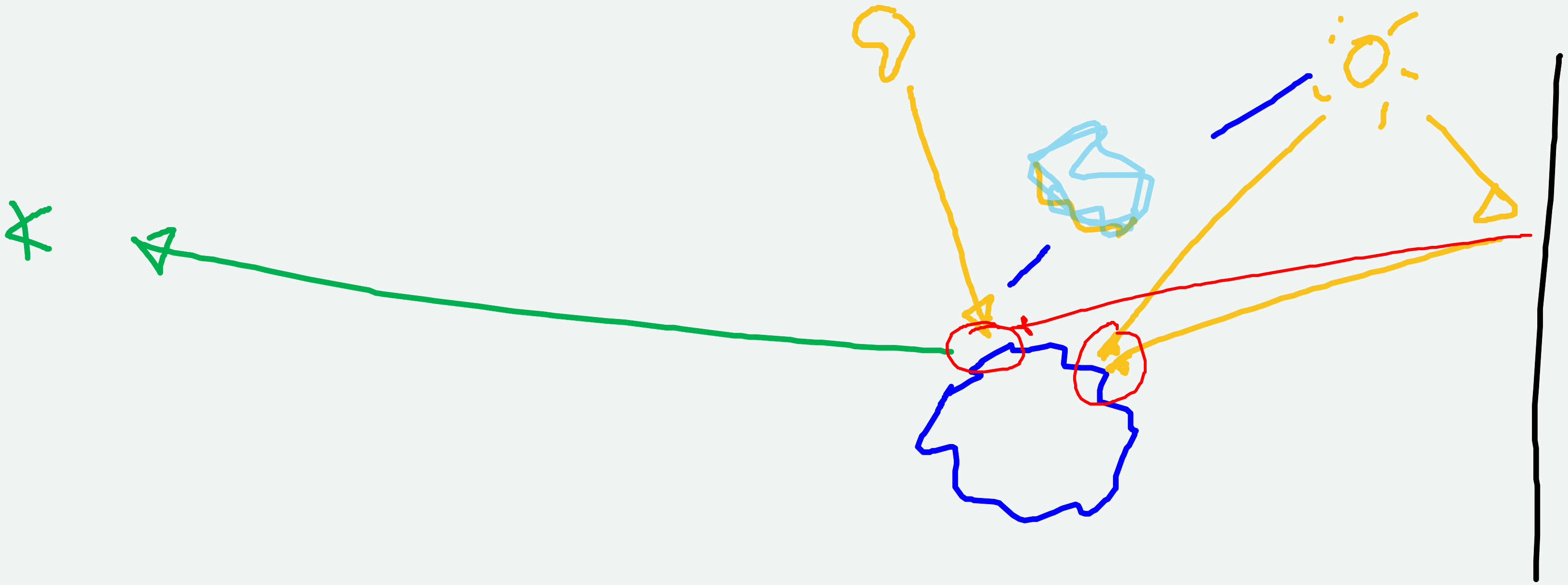
Hardware is fast enough to do it

interpolate normals surface properties

What Lighting to Implement?

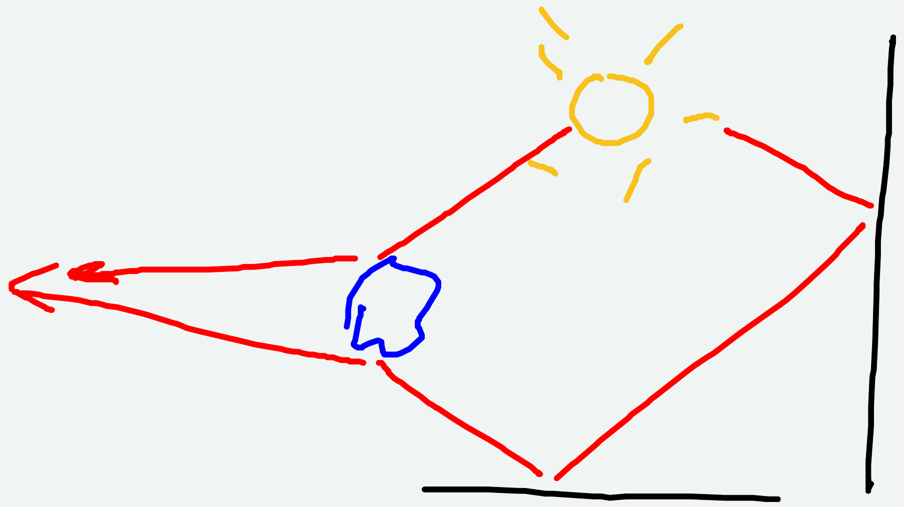
We skipped talking about lighting…

Lighting in the Real-World



Lighting: The General Idea

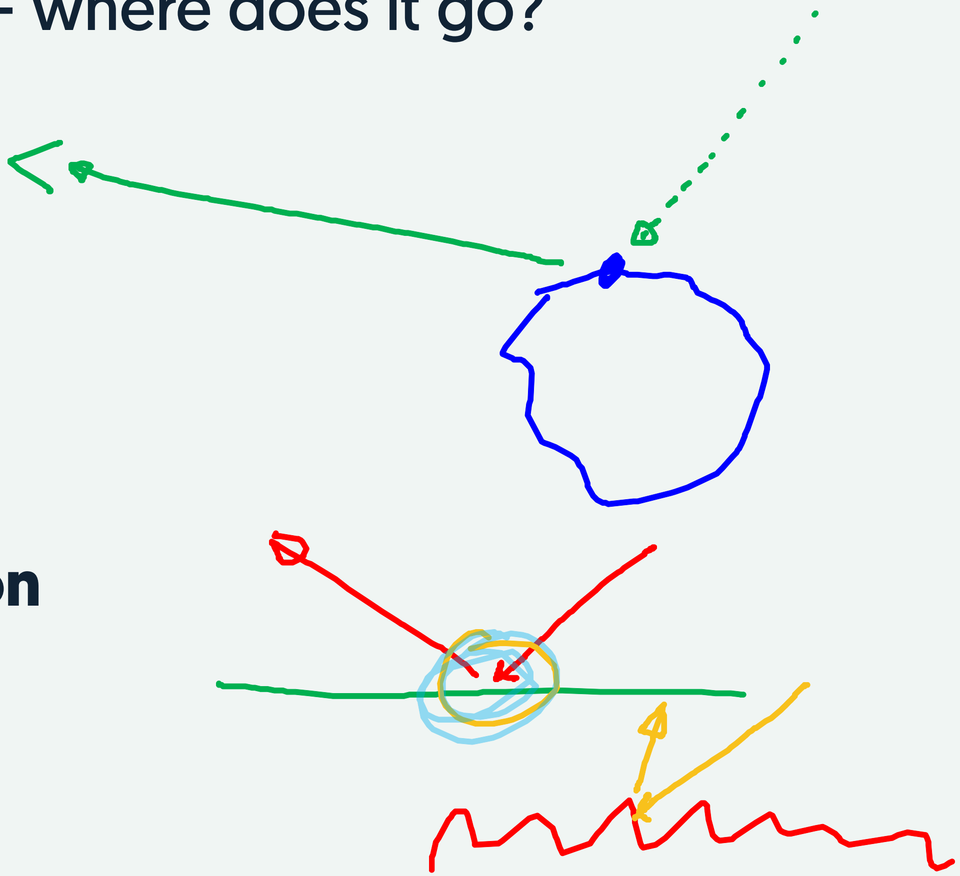
- Light starts at a source
- Interacts with objects
- Arrives at a sensor



Lighting: the simple local model

A photon arrives from some direction - where does it go?

- absorbed?
- perfect mirror bounce?
- micro-geometry
- subsurface effects
- depends on **material** and **direction**



Lighting: Local vs. Global

Local Lighting

what happens at **specific point**

each point considered individually

assume light comes from somewhere (direct from source)

Global Lighting

considers how points interact

entire scene considered together

considers how light **transports**

Local Lighting

we have to start somewhere...

1. consider each point individually
2. assume light comes from sources (not other objects)
3. only consider how material reflects light

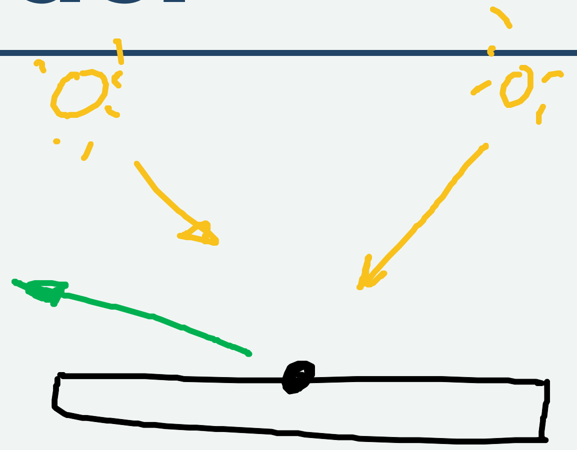
this means no global effects:

- no shadows
- no reflections
- no spill / bleed

we'll get these back later

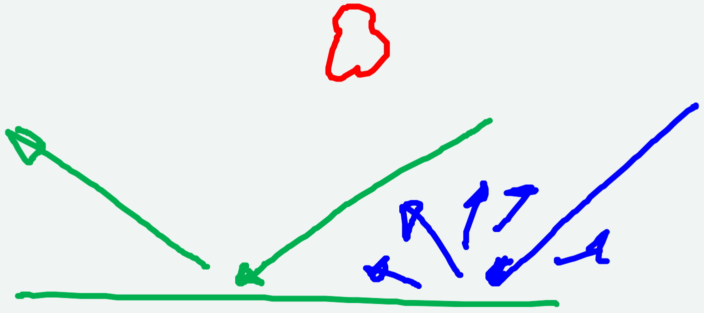
A Simple Local Lighting Model

- Assume an "ideal" material
- Make a model
- Tweak the model
- Combine with other models to mix effects



"The" Lighting Model (Historic)

- Emission (things give off light)
- Ambient (light from nowhere)
- Specular (direct reflection)
- Diffuse (rough reflection)

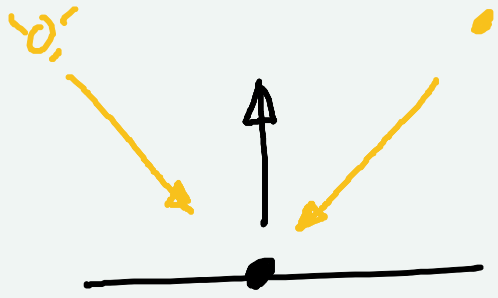


Less Historic

- ditch the hack pieces (emission, ambient)
- use less idealized models for specular and diffuse

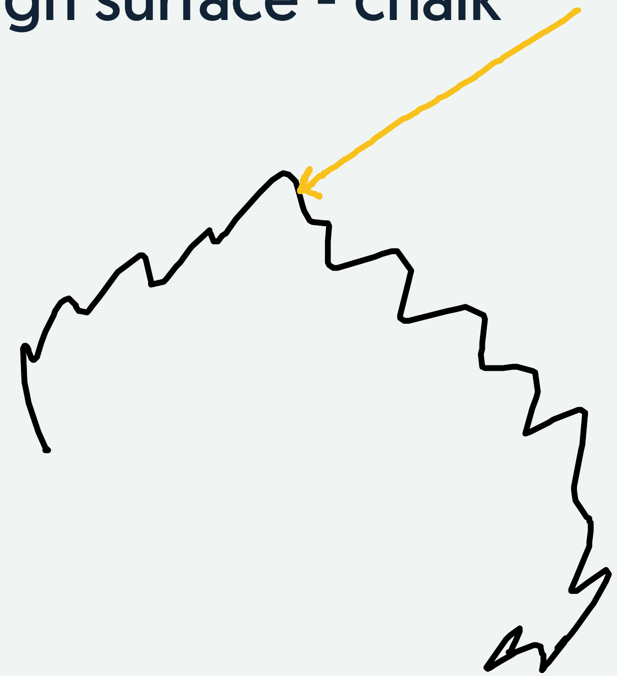
Preliminaries

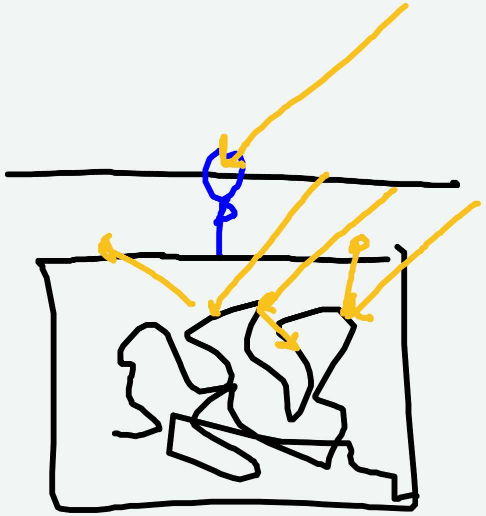
- Consider a point - with local geometry
  - surface normals
- Sum over all lights and reflectance types
  - close to how lighting really works
- Colors by doing things component-wise
  - can work in RGB or other color spaces



Diffuse Materials

Consider a very rough surface - chalk

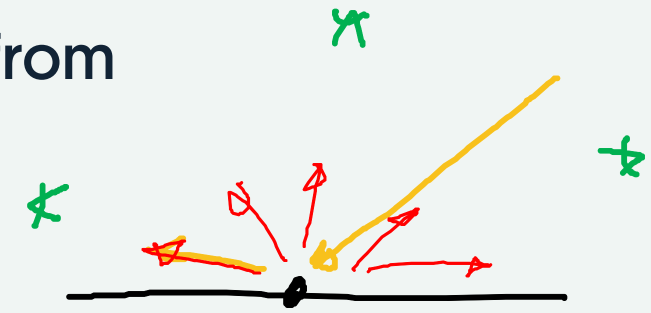




Lambertian Materials

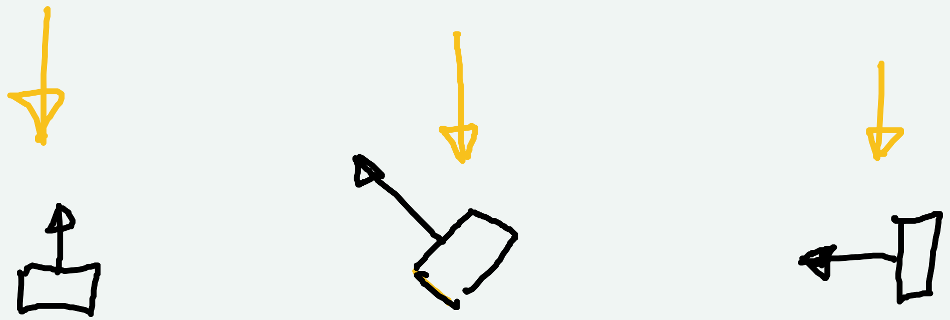
- scatters light in all directions
- doesn't matter what direction you look from

Lambert's Law



The direction of the light does matter

Consider a direction, and a small piece of surface



Diffuse Reflection

$$r_{\text{diffuse}} = \hat{\mathbf{n}} \cdot \hat{\mathbf{l}}$$

where:

- r_{diffuse} = amount of diffuse reflection
- $\hat{\mathbf{n}}$ - unit surface normal
- $\hat{\mathbf{l}}$ - unit vector to light source

using this...

$$\mathbf{color} = \underbrace{(\hat{\mathbf{n}} \cdot \hat{\mathbf{l}})} \ \underbrace{\mathbf{c}_{\text{light}}}_{\text{RGB}} \ \mathbf{c}_{\text{d}}$$

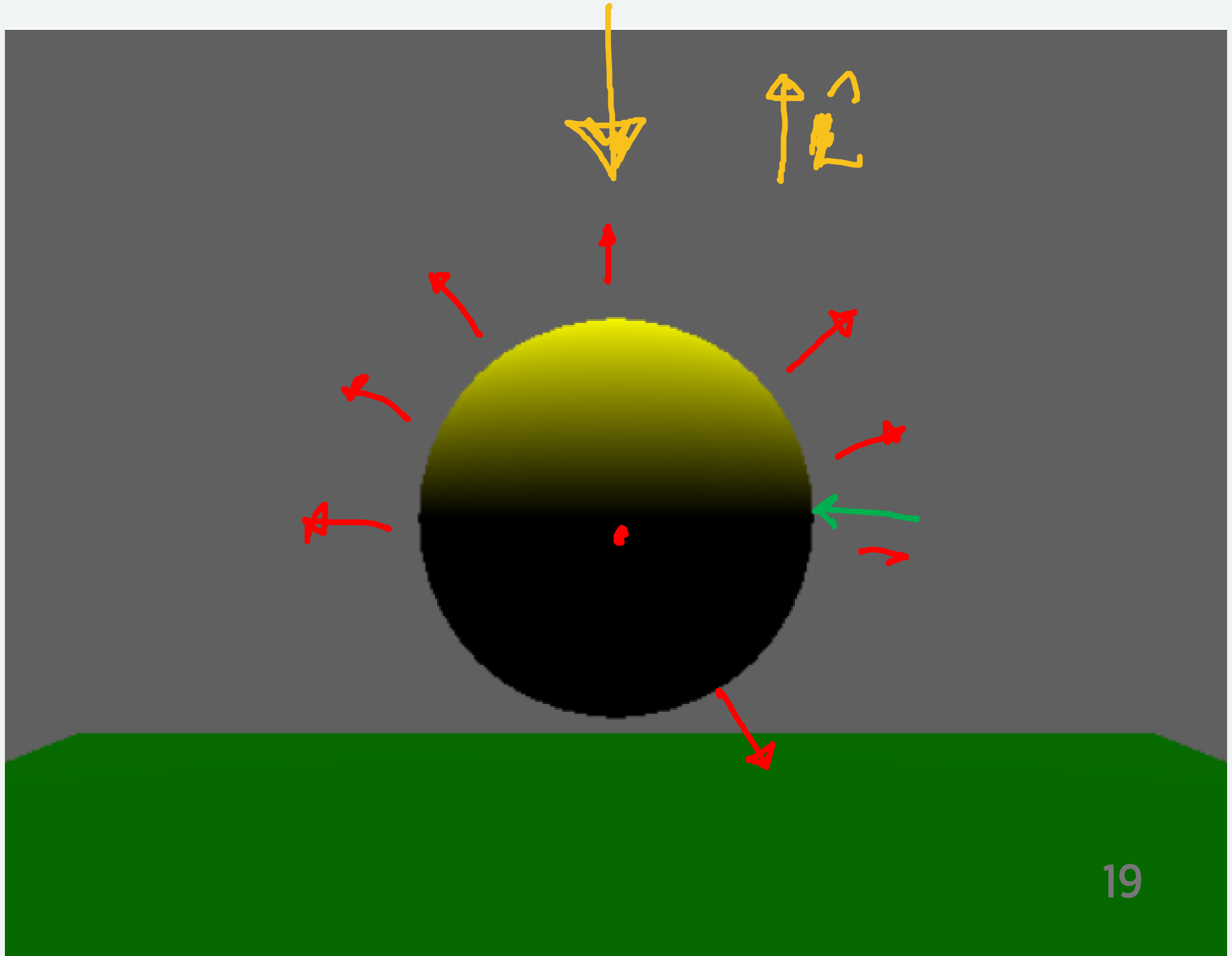
surface color
RGB

where

- $\hat{\mathbf{n}}$ - unit surface normal
- $\hat{\mathbf{l}}$ - unit vector to light source
- $\mathbf{c}_{\text{light}}$ - color/intensity of light
- $\mathbf{c}_{\text{d}}$ - color of the material (diffuse reflectance)

looking at a sphere...

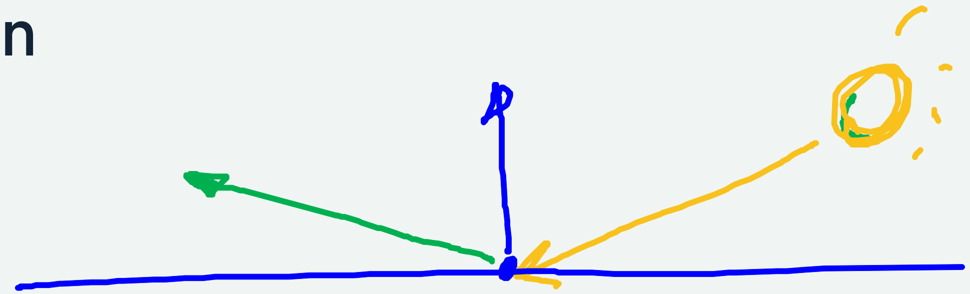
why a sphere is a good "test probe"



Shiny Things

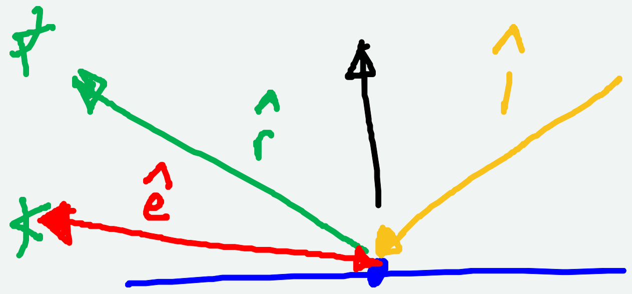
We are reflecting the lights (for now)

Specular reflection



A Perfect Mirror

- Light direction and eye position matter
- The eye needs to be in the exact correct position
- $\hat{\mathbf{e}}$ (eye vector) and $\hat{\mathbf{r}}$ (reflection vector)



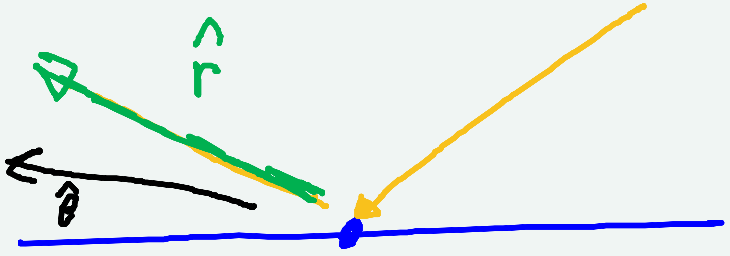
A Realistic (Imperfect) Mirror

Phong model - it's a hack!

Graduate fall off as we get away from the optimal direction

$\hat{\mathbf{e}} \cdot \hat{\mathbf{r}}$ power

keep > 0



Phong Model

Raise to the "power" of "shininess" (phong exponent)

$$r_{specular} = (\hat{\mathbf{e}} \cdot \hat{\mathbf{r}})^p$$

where

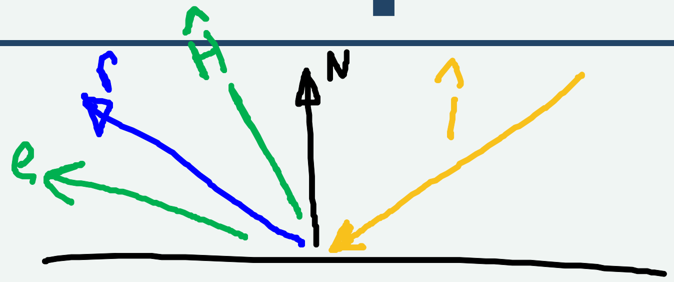
- $\hat{\mathbf{e}}$ is the eye vector
- $\hat{\mathbf{r}}$ is the reflection vector
- p is the "shininess" (material property), phone exponent
- $r_{specular}$ is the amount of specular reflection

Shinier = more like a perfect mirror

An Alternate Way to Compute

$$r_{specular} = (\hat{\mathbf{n}} \cdot \hat{\mathbf{h}})^p$$

$\hat{l}$ = light



where

- $\hat{\mathbf{n}}$ is the normal vector
- $\hat{\mathbf{h}}$ is the half-way vector (between $\hat{\mathbf{l}}$ and $\hat{\mathbf{e}}$)
- p is the "shininess" (material property), phone exponent
- $r_{specular}$ is the amount of specular reflection

Blinn - Phong

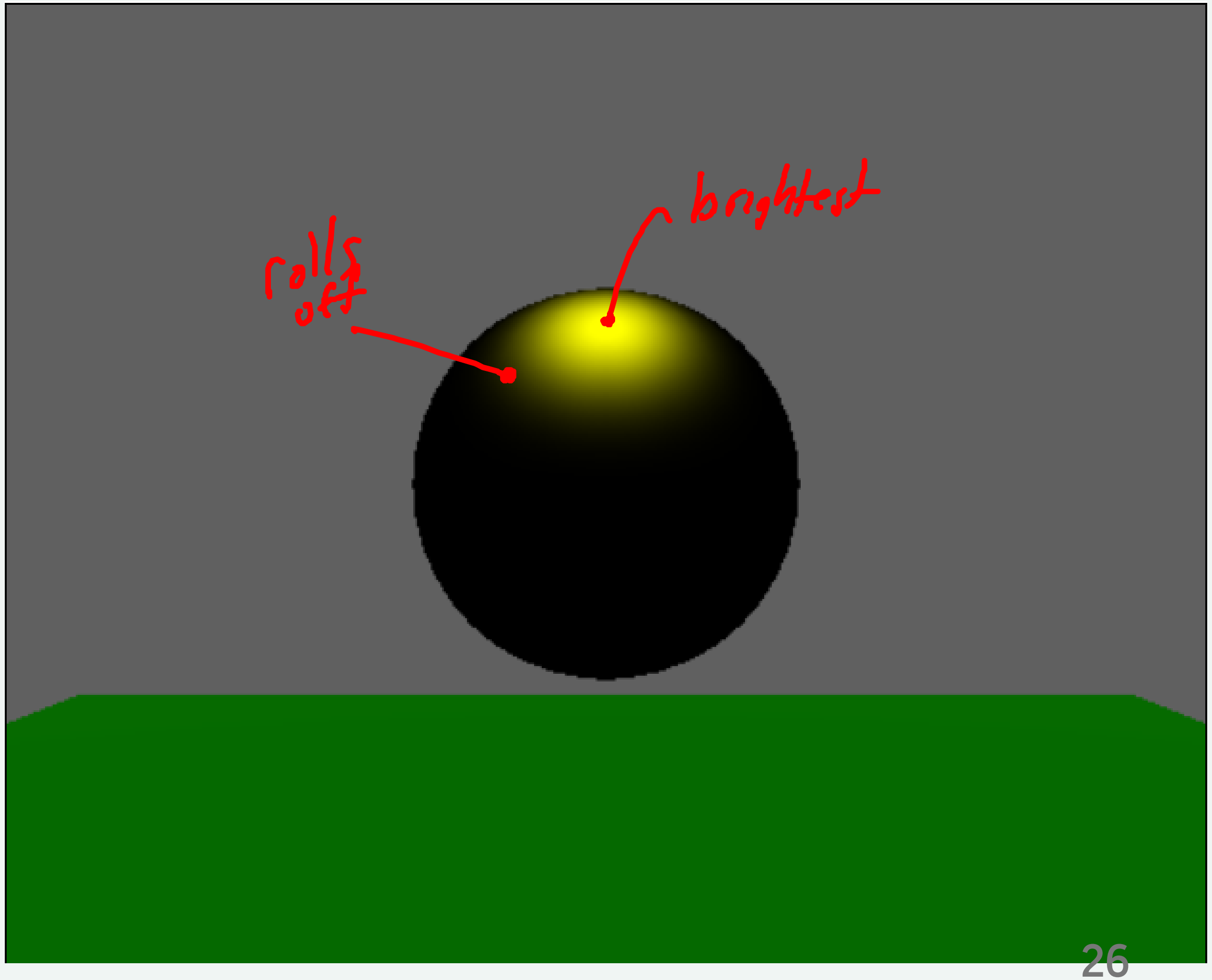
Using This

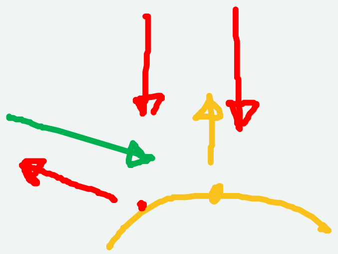
$$\mathbf{color} = \underbrace{(\hat{\mathbf{n}} \cdot \hat{\mathbf{h}})^p}\ \underbrace{\mathbf{c}_{\text{light}}}_{\text{RGB}}\ \underbrace{\mathbf{c}_{\text{s}}}_{\text{RGB}}$$

where

- $\hat{\mathbf{n}}$ - unit surface normal
- $\hat{\mathbf{l}}$ - unit vector to light source $\Rightarrow \widehat{\left(\frac{\hat{l}+\hat{e}}{2}\right)} \Rightarrow \hat{H}$
- $\mathbf{c}_{\text{light}}$ - color/intensity of light
- $\mathbf{c}_{\text{s}}$ - color of the material (specular reflectance)
- p - shininess of the material

Demo





Putting it together...

Sum of different lighting components:

- emissive (objects give off light)
- ambient (generic light from all directions)
- diffuse (add for each light source)
- specular (add for each light source)

A Material is defined by:

- an emissive color $\mathbf{c_e}$
- an ambient color $\mathbf{c_a}$
- a diffuse color $\mathbf{c_d}$
- a specular color $\mathbf{c_s}$
- a specular exponent (shininess) p

The Phong Lighting Model

$$\mathbf{color} = \mathbf{c_e} + \underbrace{\mathbf{c_a}\ \mathbf{l_a}}_{\text{ambient}} + \sum\nolimits_{l \in lights} \left((\hat{\mathbf{n}} \cdot \hat{\mathbf{l}})\ \mathbf{c_{light}}\ \mathbf{c_d} + (\hat{\mathbf{n}} \cdot \hat{\mathbf{h}})^p\ \mathbf{c_{light}}\ \mathbf{c_s} \right)$$

which uses:

- the surface geometry $\hat{\mathbf{n}}$
- the light direction $\hat{\mathbf{l}}$
- the eye direction (combined with light direction into the half-vector) ($\hat{\mathbf{h}}$)
- the light color (instensity) $\mathbf{l_a}$) and $\mathbf{c_{light}}$
- the surface properties $\mathbf{c_e}$, $\mathbf{c_a}$, $\mathbf{c_d}$, $\mathbf{c_s}$ and p

Using this in THREE.js

Use the `MeshPhongMaterial`

- specify 3 colors (no separate ambient)
  - `color` is diffuse and ambient color
  - `specular` is specular color
  - `emissive` is emissive color

Designing Materials

warning: Phong is a hack, choosing parameters that look good is an art

Amount of Shininess:

- how bright is the shininess ($\mathbf{c_s}$, relative to $\mathbf{c_d}$)
- how focused is the shininess (p)

Color of Shininess

- what color is the shininess (plastic has white highlights, metal is colored)

Alternative Workflow

Could choose a different workflow:

- **metalness** - behaves like metal or plastic
  - metal specular keeps color, plastic, specular is white
- **roughness** - dull or shiny
  - amount of specularity, size of specularity

Or...

- use a less hacky model of local lighting!

The lighting to implement (Phong)

$$\mathbf{color} = \mathbf{c_e} + \mathbf{c_a}\ \mathbf{l_a} + \sum_{l \in lights} \left(\underbrace{(\hat{\mathbf{n}} \cdot \hat{\mathbf{l}})\ \mathbf{c_{light}}\ \mathbf{c_d}}_{\text{diffuse}} + \underbrace{(\hat{\mathbf{n}} \cdot \hat{\mathbf{h}})^p\ \mathbf{c_{light}}\ \mathbf{c_s}}_{\text{specular}} \right)$$

which uses:

- the surface geometry $\hat{\mathbf{n}}$
- the light direction $\hat{\mathbf{l}}$
- the eye direction (combined with light direction into the half-vector) ($\hat{\mathbf{h}}$)
- the light color (instensity) $\mathbf{l_a}$) and $\mathbf{c_{light}}$
- the surface properties $\mathbf{c_e}$, $\mathbf{c_a}$, $\mathbf{c_d}$, $\mathbf{c_s}$ and p

A Material is defined by:

- an emissive color $\mathbf{c_e}$
- an ambient color $\mathbf{c_a}$
- a diffuse color $\mathbf{c_d}$
- a specular color $\mathbf{c_s}$
- a specular exponent (shininess) p

Get color from:

- material (as a uniform)
- the vertices (interpolate attributes)
- a texture map (later)

Diffuse

$$\mathbf{color} = (\hat{\mathbf{n}} \cdot \hat{\mathbf{l}}) \quad \mathbf{c}_{\mathbf{light}} \quad \mathbf{c}_{\mathbf{d}}$$

where

- $\hat{\mathbf{n}}$ - unit surface normal
- $\hat{\mathbf{l}}$ - unit vector to light source
- $\mathbf{c}_{\mathbf{light}}$ - color/intensity of light
- $\mathbf{c}_{\mathbf{d}}$ - color of the material (diffuse reflectance)

Diffuse and Ambient

$$\mathbf{color} = \left((\hat{\mathbf{n}} \cdot \hat{\mathbf{l}})\ \mathbf{c}_{\mathbf{light}} + \mathbf{c}_{\mathbf{ambient}} \right)\ \mathbf{c}_{\mathbf{d}}$$

where

- $\hat{\mathbf{n}}$ - unit surface normal
- $\hat{\mathbf{l}}$ - unit vector to light source
- $\mathbf{c}_{\mathbf{light}}$ - color/intensity of light
- $\mathbf{c}_{\mathbf{ambient}}$ - color/intensity of ambient light
- $\mathbf{c}_{\mathbf{d}}$ - color of the material (diffuse reflectance)

Where do those values come from?

$\hat{\mathbf{n}}$ - unit surface normal

$\hat{\mathbf{l}}$ - unit vector to light source

$\mathbf{c}_{\text{light}}$ - color/intensity of light

$\mathbf{c}_{\text{ambient}}$ - color of ambient light

$\mathbf{c}_{\text{d}}$ - **color of material**

Surface color

uniform for material?

varying (copy from attribute)

read from texture

Where do those values come from?

$\hat{\mathbf{n}}$ - unit surface normal

$\hat{\mathbf{l}}$ - unit vector to light source

$\mathbf{c}_{\text{light}}$ - **color/intensity of light**

$\mathbf{c}_{\text{ambient}}$ - **color of ambient light**

$\mathbf{c}_{\text{d}}$ - color of material

Light Properties

Constants?

Uniforms?

Get from THREE's objects

(tricky)

The Geometric Properties

$\hat{\mathbf{n}}$ - **unit surface normal**

$\hat{\mathbf{l}}$ - **unit vector to light source**

$\mathbf{c}_{\mathbf{light}}$ - color/intensity of light

$\mathbf{c}_{\mathbf{ambient}}$ - color of ambient light

$\mathbf{c}_{\mathbf{d}}$ - color of material

What coordinate system?

Transform vertices

Interpolate normal vectors

The Vertex Shader

```
attribute vec3 position;          // these are provided by THREE
attribute vec3 normal;            // you don't need to declare them
uniform mat3 normalMatrix;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

varying vec3 fNormal;
varying vec3 fPosition;

void main()
{
  gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);

  fNormal = normalize(normalMatrix * normal);
  fPosition = modelViewMatrix * vec4(position, 1.0);
}
```

Transformations of Normals

Transforming an object changes the normals

Normals are transformed by the inverse transpose (adjoint)

for a rotation

$$(M^{-1})^T \Rightarrow (M^T)^T \Rightarrow M$$

What coordinate system?

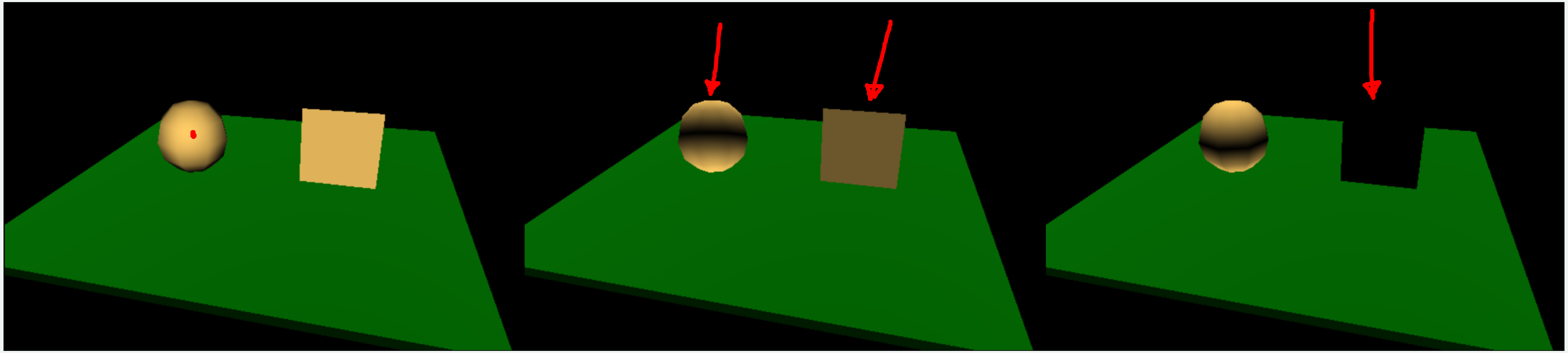
World coordinate system?

Camera coordinate system?

Typically: lighting in camera coordinates

Does it make a difference?

Z-Axis Camera   Y-Axis Camera   Y-Axis World



z

y camera

camera z

world Y

What does THREE do?

from the documentation...

```
// = inverse transpose of modelViewMatrix
uniform mat3 normalMatrix;
```

THREE's lights are transformed into view coordinates

Light Geometry: Directional Light

```
const vec3 lhat = vec3(0,1,0);
```

Y Axis in same coordinate system as the normals (view)

Light Geometry 2: Point Light

```
const vec3 lightPos = vec3(10,10,0);
varying vec3 fPosition;

main() {
    ...
    vec3 lhat = normalize(fPosition - lightPos);
```

but light position must be in camera coordinates

Light Geometry2b: Point Light

```
uniform vec3 cameraPosition;
const vec3 lightPos = vec3(10,10,0);
varying vec3 fPosition;

main() {
    ...
    vec3 lhat = normalize(wPosition - lightPos);
```

if we had `wPosition` as a varying as:

```
wPosition = modelMatrix * vec4(position, 1.0);
```

Diffuse Lighting Shader...

```
varying vec3 v_normal;
const vec3 lightDir = vec3(0,0,1);
const vec3 baseColor = vec3(1,.8,.4);
const vec3 lightColor = vec3(.5,.5,.5);
const vec3 ambientColor = vec3(.1,.1,.2);

void main()
{
    /* we need to renormalize the normal since it was interpolated */
    vec3 nhat = normalize(v_normal);
    /* deal with two sided lighting */
    float light = abs(dot(nhat, lightDir));
    // or not...
    float light = clamp(dot(nhat, lightDir),0.0,1.0);

    /* brighten the base color */
    gl_FragColor = vec4((light*lightColor + ambientColor) * baseColor,1);
}
```

Beyond Diffuse

Implement the equations of the model...

What does THREE actually do?

1. `ShaderMaterial` - whatever we tell it to do
2. `MeshPhongMaterial` - basically the model we discussed

   see `lights_phong_pars_fragment.glsl`
3. `MeshStandardMaterial` - less clear what it does

   ~~might be Phong with improved workflow~~
4. `MeshPhysicalMaterial` - a more complex model

What if we want to use THREE's lights

We need to get them from THREE

- uniforms lib

We need to loop over however many lights there are

We need to do all the different kinds of lights

This is tricky - and not that instructive

Best bet: use the code from THREE's shaders

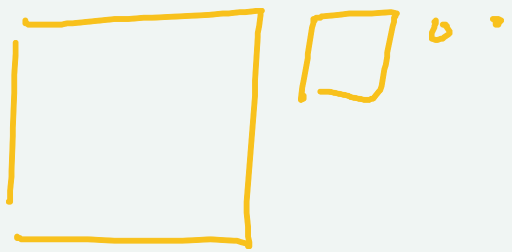
Textures

1. Image-Based Textures
2. Procedural Textures

Image-Based Textures in Shaders

Texture Maps are part of "Texture Objects"

- texure map (image)
- mip-map (images of different sizes)
- parameters (filter type, wrapping, ...)



GLSL Type `sampler2D` (for a 2D image) ← uniform

Getting the color from a texture

```
 varying vec2 v_uv;

 // get the texture from the program
 uniform sampler2D texture;

void main()
{
    gl_FragColor = texture2D(texture, v_uv);
}
```

How does it know how to filter?

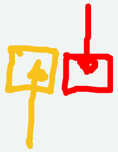
Mip-Mapping set up in JavaScript (THREE's default)

Vertex Shader

It doesn't know how to sample - uses the texture itself

Fragment Shader

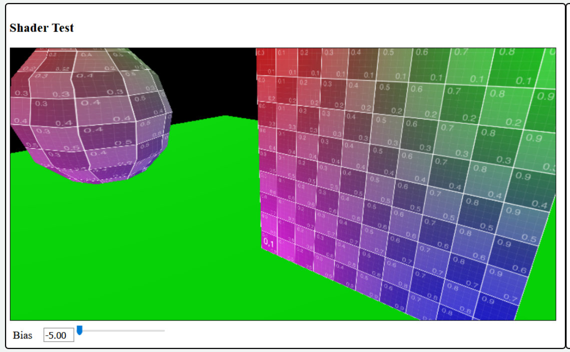
It knows how far the next pixel is and uses that to estimate

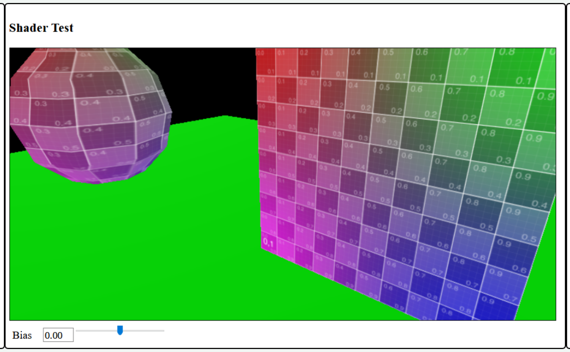


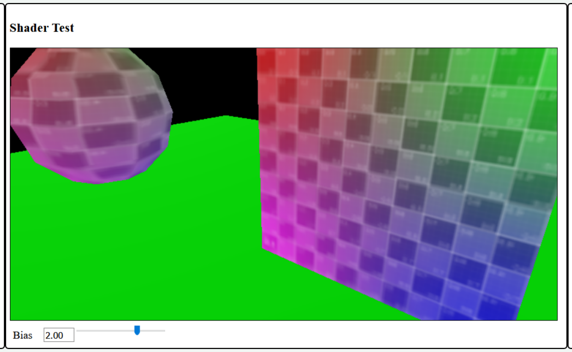
Adjusting Filtering

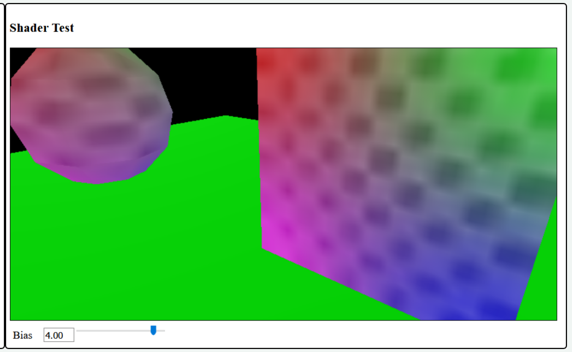
we can **bias** the mip-map level (amount of blur)

```
gl_FragColor = texture2D(texture, v_uv, bias);
```





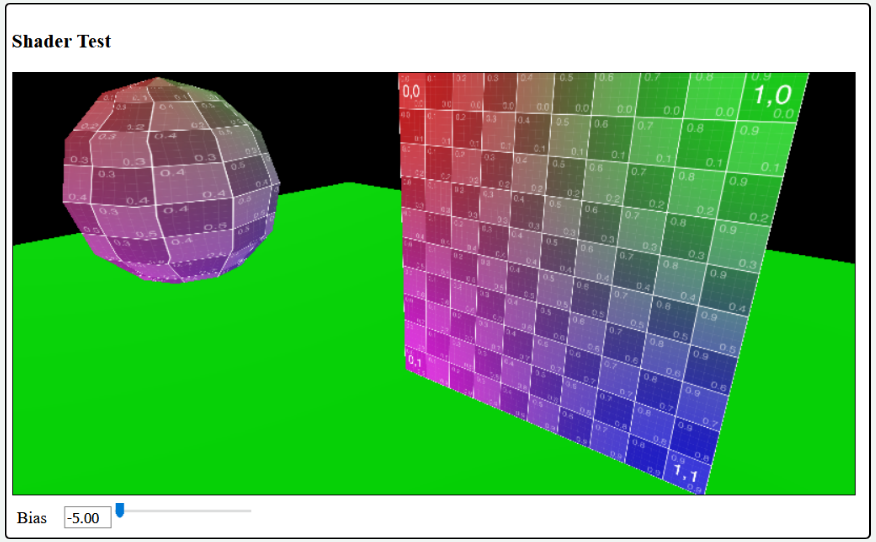


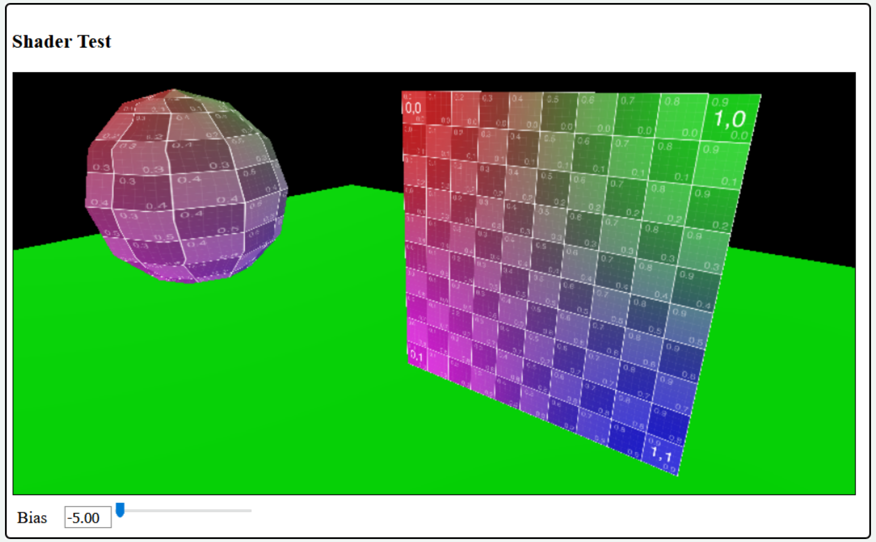


-5 Bias 0 Bias 2 Bias 4

Correct

no filtering





filtering