Tutorial 6

Material

Computer Graphics

Summer Semester 2020

Ludwig-Maximilians-Universität München

Exam

- 3 "Online-Hausarbeiten", release in the Uni2Work
- Tasks are similar to the existing assignments. The schedule:
  - Abgabe 1 (Programming tasks, 50p) 06.07.-10.07.20 (5 days)
  - Abgabe 2 (Non-programming tasks, 50p) 13.07.-18.07.20 (6 days)
  - Abgabe 3 (Programming tasks, 100p) 20.07.-31.07.20 (12 days)
- You need 100 points to pass the exam and 190 points to get 1.0
- 10% Bonus are given in the Online-Hausarbeiten
- Please register yourself via Uni2Work

Agenda

- Texturing
  - Texture Mapping
  - Barycentric Interpolation
  - Texture Sampling
  - Map Applications
- Shading and Shadowing
  - The Phong and Blinn-Phong Reflection Model
  - Shading Frequency
  - Shadow Map
- Bidirectional Reflectance Distribution Function (BRDF)
  - Radiometry
  - The Rendering Equation

Tutorial 6: Materials

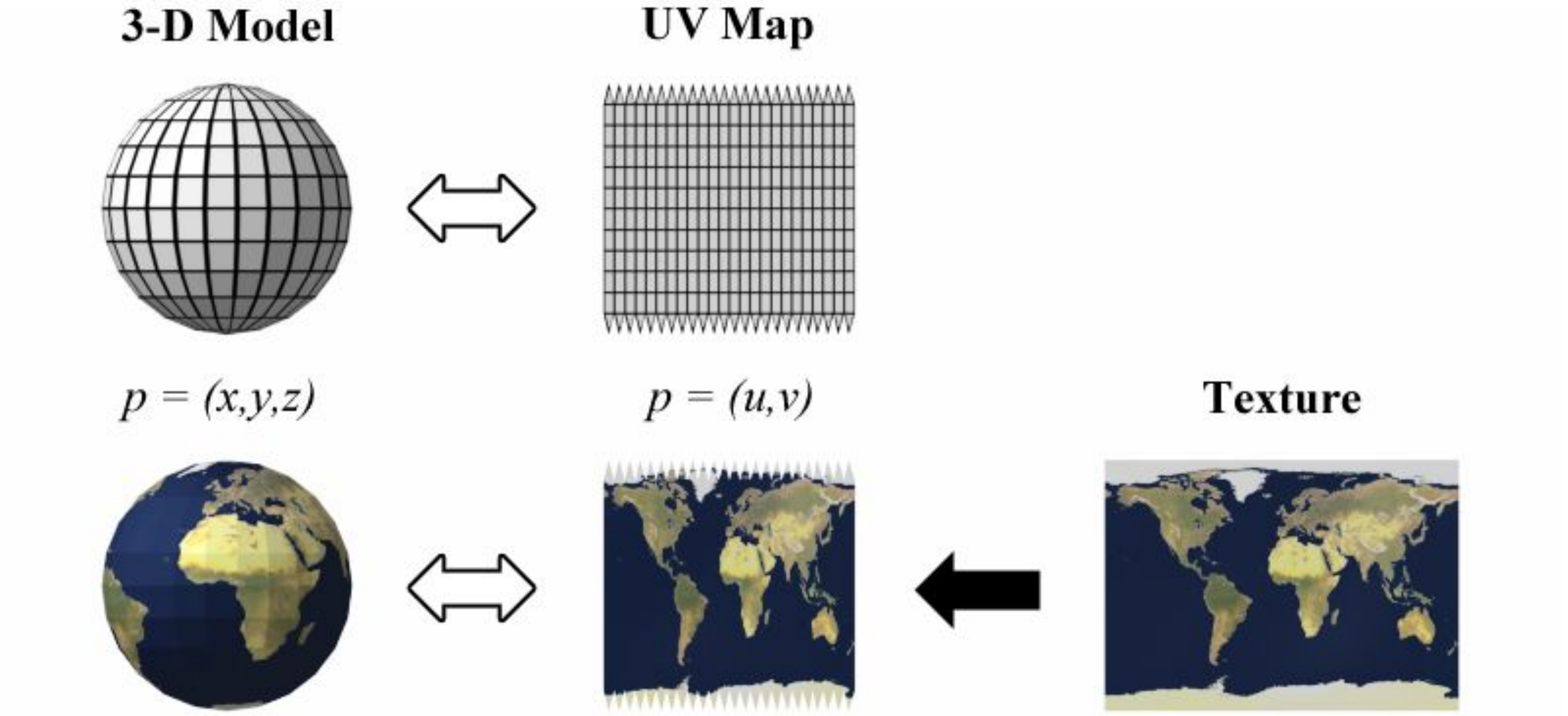
- Texturing
  - Texture Mapping
  - Barycentric Interpolation
  - Texture Sampling
  - Map Applications
- Shading and Shadowing
  - The Phong and Blinn-Phong Reflection Model
  - Shading Frequency
  - Shadow Map
- Bidirectional Reflectance Distribution Function (BRDF)
  - Radiometry
  - The Rendering Equation

Texture Coordinates

Texture coordinates define a mapping from surface coordinates to a texture domain

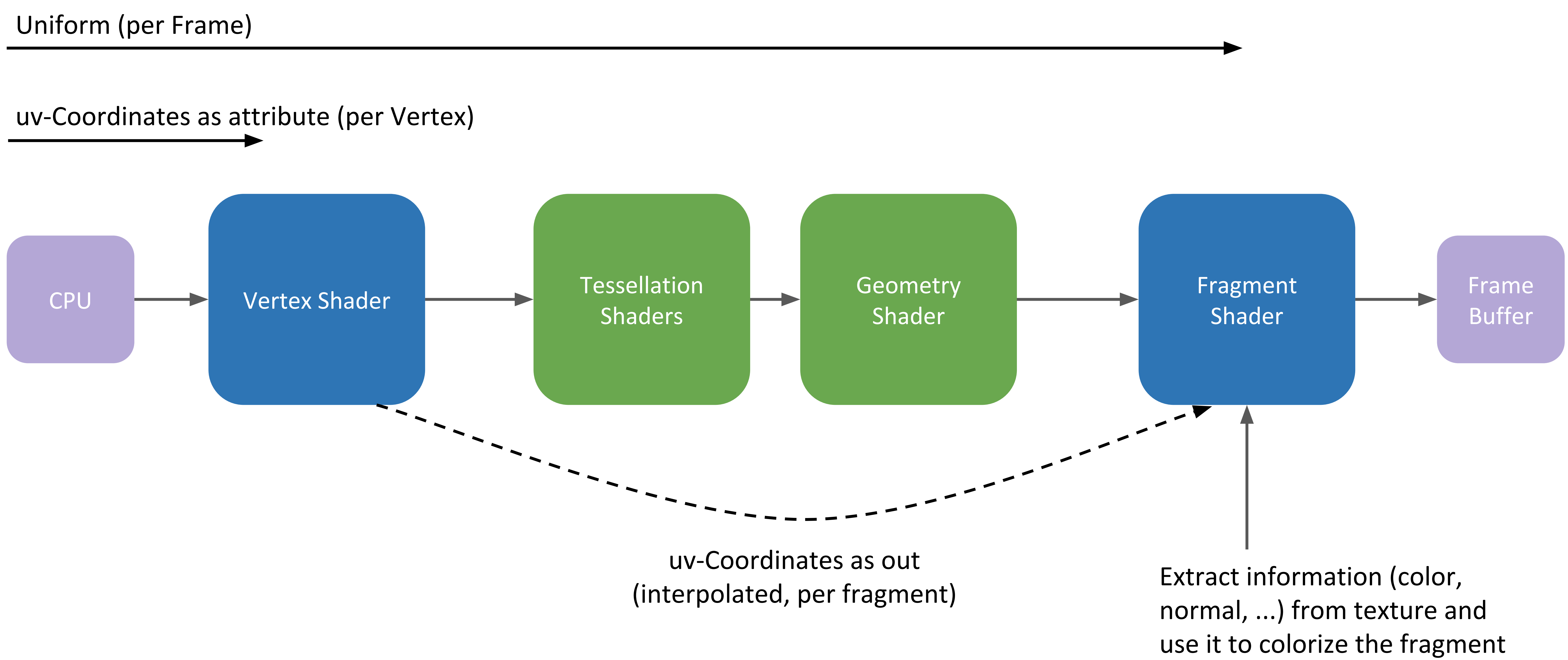
Basic idea:

```
triangle.project().pixels.forEach((x, y) => {
   [u, v]  = getTextureCoord(x, y)
   color = sampleTexture(u, v)
   draw(x, y, color)
})
```



https://en.wikipedia.org/wiki/UV_mapping#/media/File:UVMapping.png

Graphics Pipeline (Revisited)



Task 1 a) Spherical UV Coordinates

Basic idea:

- u-coordinate: Longitude
- v-coordinate: Latitude

Assume r = 1, we have:

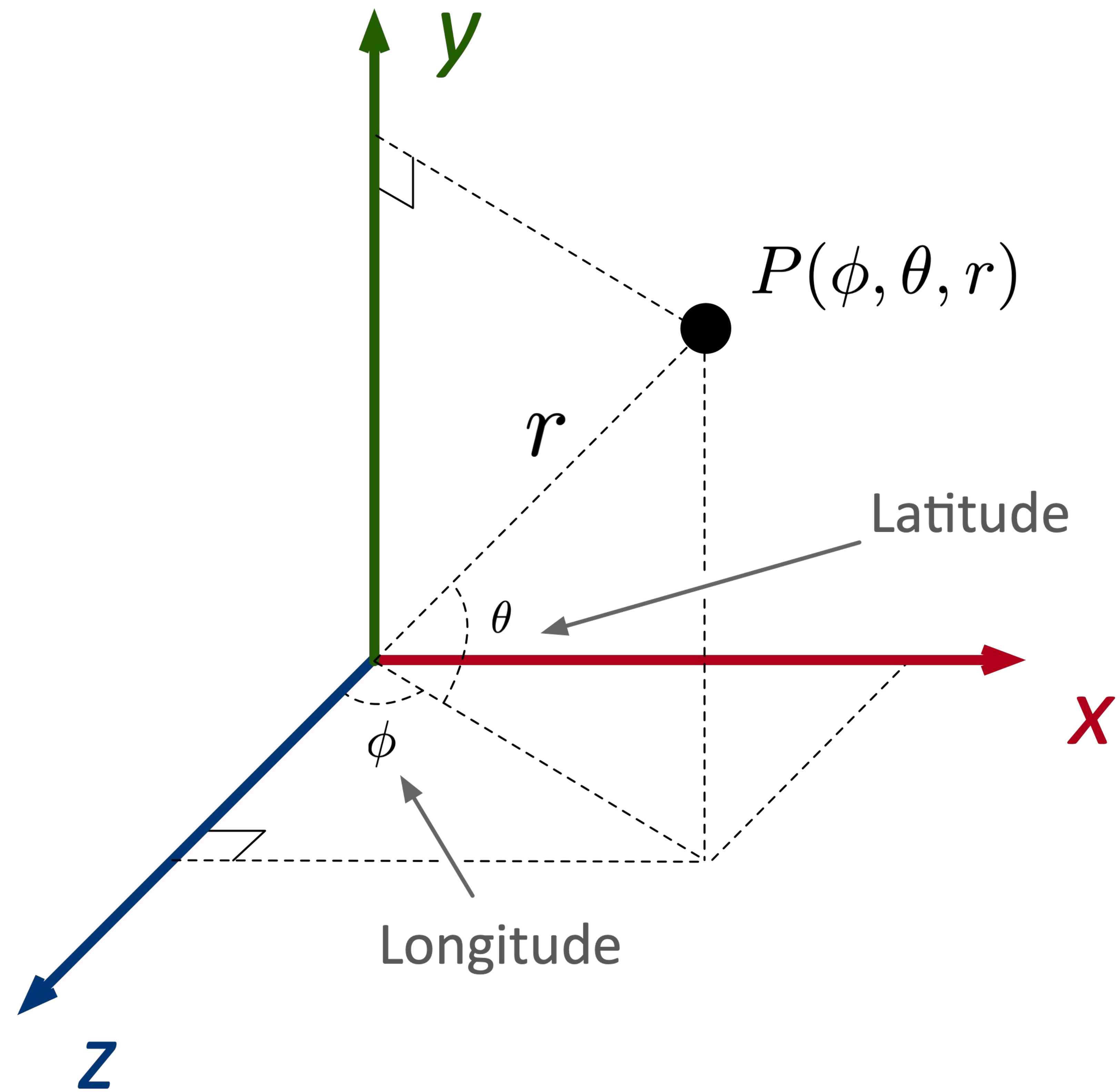
$$\phi = \arctan \frac{x}{z} \in [0, \pi]$$

$$\theta = \arcsin y \in [-\frac{\pi}{2}, \frac{\pi}{2}]$$

Then:

$$u = \frac{\phi}{2\pi} + \frac{1}{2} \in [0, 1]$$

$$v = \frac{\theta}{\pi} + \frac{1}{2} \in [0, 1]$$



Task 1 b)

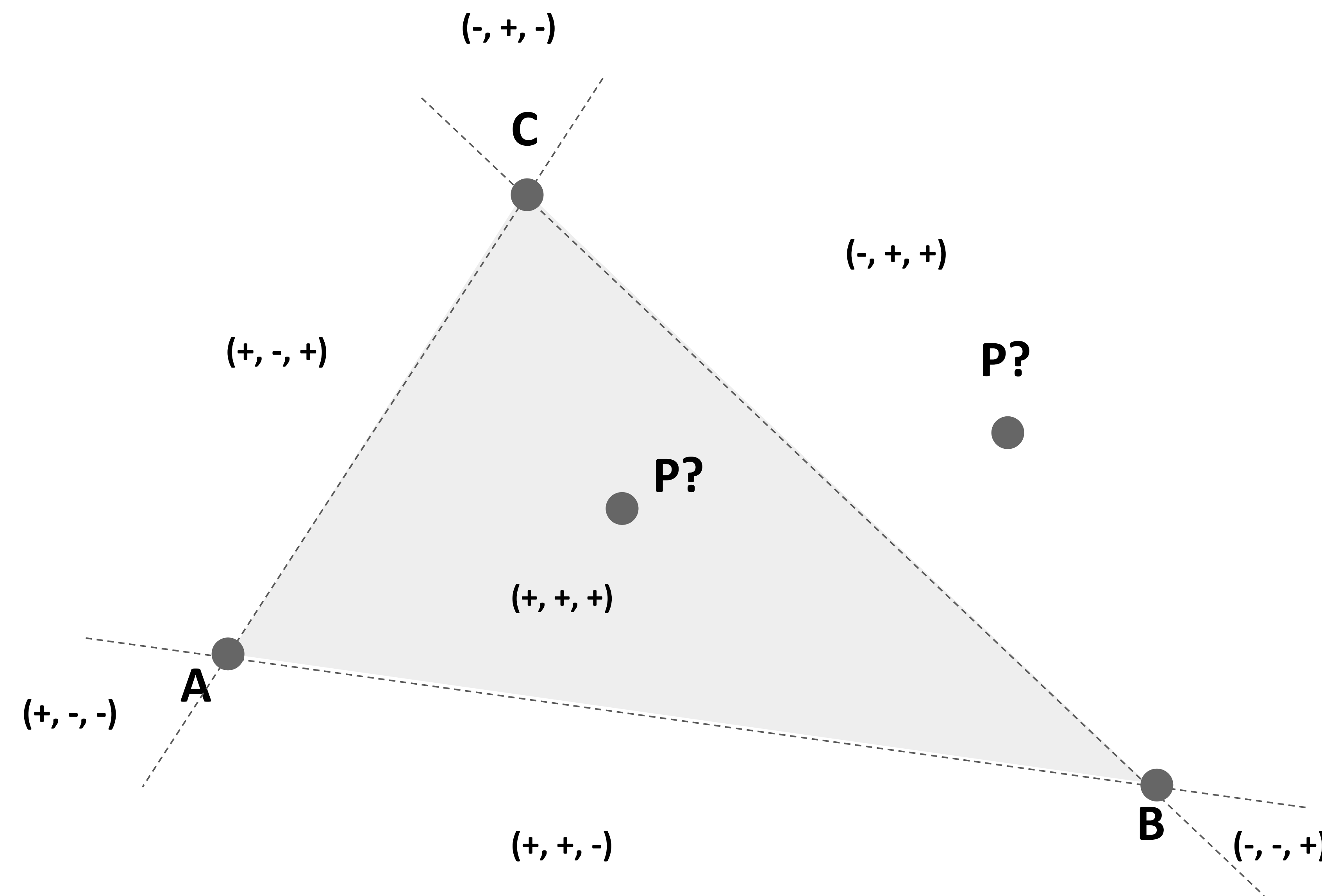
Try different values:

If $w_1 = 1, w_2 = w_3 = 0 \Rightarrow$ P = A

If $w_2 = 1, w_1 = w_3 = 0 \Rightarrow$ P = B

If $w_3 = 1, w_1 = w_2 = 0 \Rightarrow$ P = C

… just try more possibilities :)



Conclusion:

If $\forall w_i \in [0, 1]$, P is inside the triangle ABC

If $\exists w_i < 0$, P is outside the triangle ABC

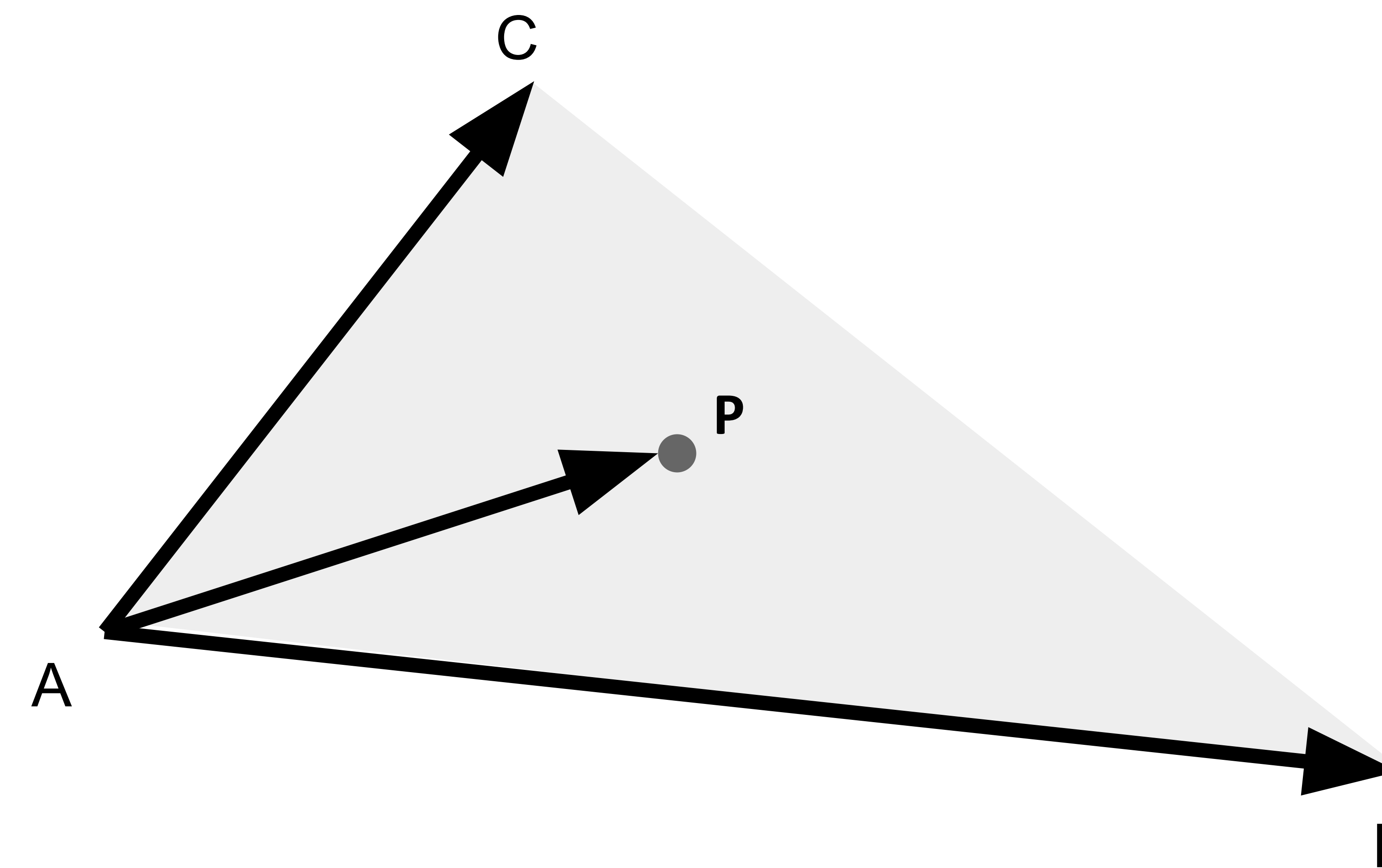
Barycentric Interpolation

If P is inside the triangle, geometrically:

$$\vec{AP} = w_2\vec{AB} + w_3\vec{AC}, w_2, w_3 \in [0,1]$$

$$\implies P - A = w_2(B - A) + w_3(C - A)$$

$$\implies P = (1 - w_2 - w_3)A + w_2B + w_3C$$



Let $w_1 = 1 - w_2 - w_3 \in [0,1]$

We have: $P = w_1A + w_2B + w_3C$

This is how we interpolate the color for P given the color of A, B, and C:

$$\text{color}(P) = w_1\text{color}(A) + w_2\text{color}(B) + w_3\text{color}(C)$$

But what are w_1, w_2, w_3?

Task 1 c)

Because:

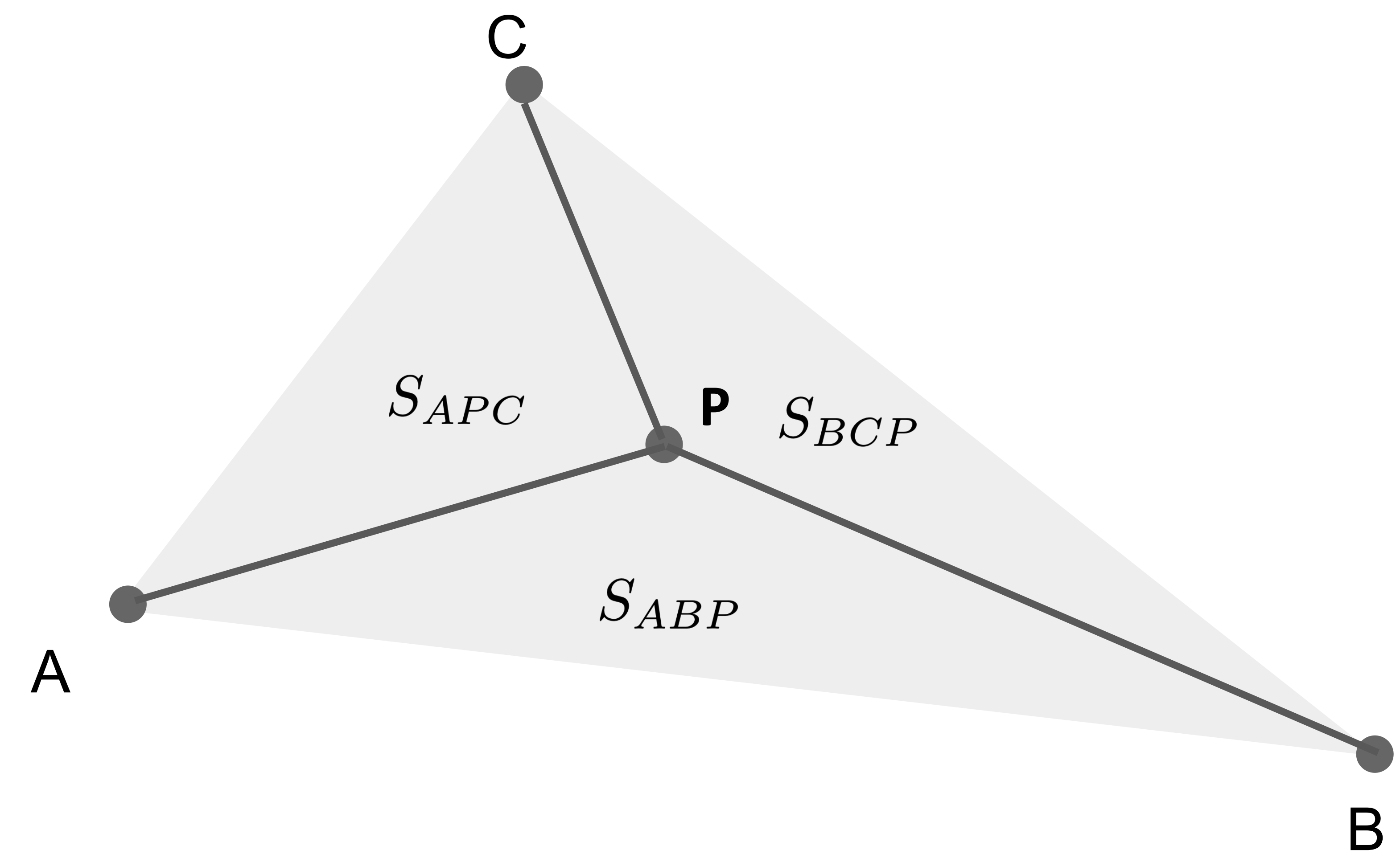
$$\vec{AP} = w_2\vec{AB} + w_3\vec{AC}, w_2, w_3 \in [0, 1]$$

We can write this linear equations:

$$\vec{AP}_x = w_2\vec{AB}_x + w_3\vec{AC}_x$$

$$\vec{AP}_y = w_2\vec{AB}_y + w_3\vec{AC}_y$$

$$w_1 + w_2 + w_3 = 1$$

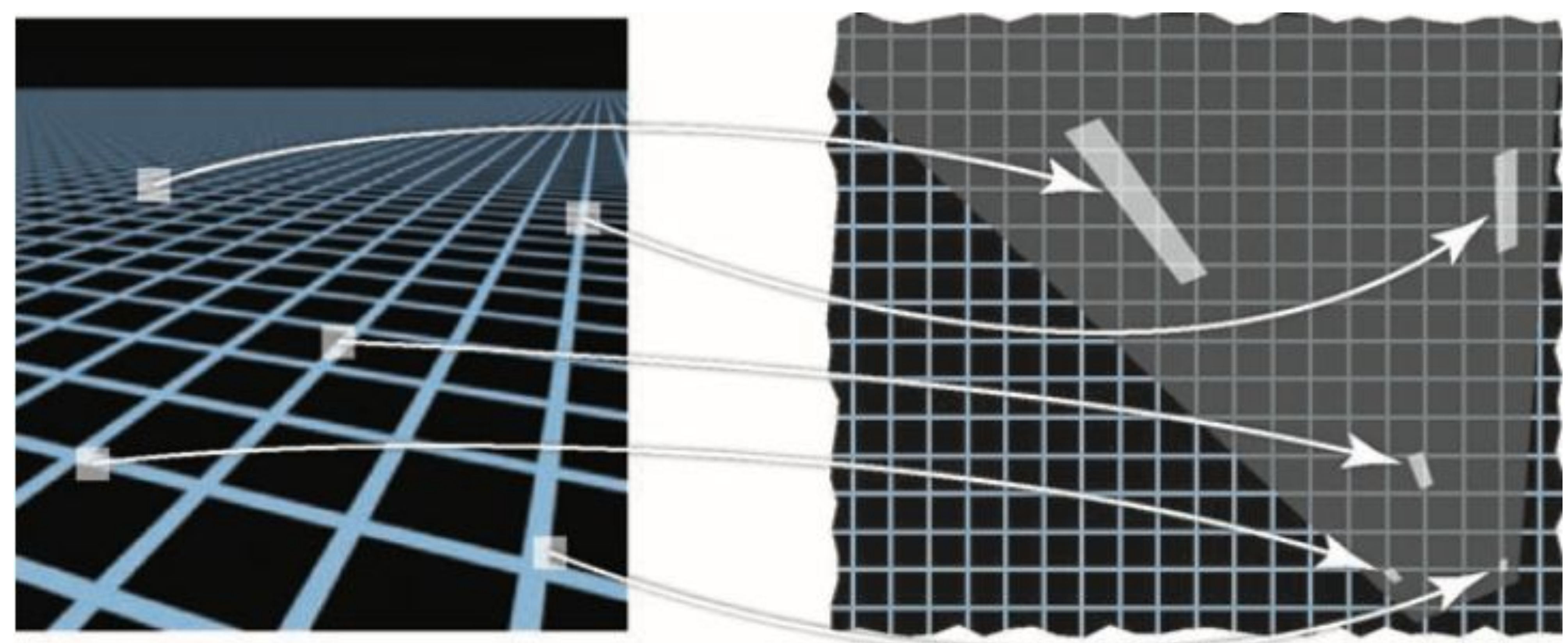


$$\Rightarrow w_3 = \frac{\vec{AP}_x\vec{AB}_y - \vec{AP}_y\vec{AB}_x}{\vec{AC}_x\vec{AB}_y - \vec{AC}_y\vec{AB}_x} = \frac{\vec{AP} \times \vec{AB}}{\vec{AC} \times \vec{AB}} = \frac{S_{ABP}}{S_{ABC}} \quad w_2 = \frac{S_{APC}}{S_{ABC}} \quad w_1 = \frac{S_{BCP}}{S_{ABC}}$$

This is how you compute the barycentric coordinates.

Texture *Sampling*

- Magnification (Upsampling): Texture resolution is too low, we want an interpolated color of a given pixel ⇒ Interpolation
  - e.g. Linear interpolation (recall interpolation in Perlin noise)
- Minification (Downsampling): Texture resolution is too high, we want the average color of an area ⇒ Range query
  - e.g. Mipmap



Texture sampling for a pixel can be quite different

Mipmap

- Fast approximate a range query
- Basic idea: Pre-compute a texture version for the "LOD". Find the correct level (or levels in between) and get the color directly
  - Level 0: 1024x1024
  - Level 1: 512x512
  - Level 2: 256x256
  - … until you get 1x1 pixel
  - **each one-fourth of the total area of the previous one**

https://en.wikipedia.org/wiki/Mipmap#/media/File:MipMap_Example_STS101.jpg

- How do we know which level to choose? Determine the level by the choosing the maximum norm of a gradient of ***u*** and ***v*** on d***x*** or d***y***: $L = \log_2 \max\left(\sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2}\right)$

Paul S. Heckbert, ''Texture Mapping Polygons in Perspective'', Technical Memo No. 13, NYIT. Computer Graphics Lab, April 1983.
Lance Williams. 1983. Pyramidal parametrics. In Proceedings of the 10th annual conference on Computer graphics and interactive techniques (SIGGRAPH ’83). Association for Computing Machinery, New York, NY, USA, 1–11. DOI:https://doi.org/10.1145/800059.801126

Task 1 d) Mipmap Storage Overhead

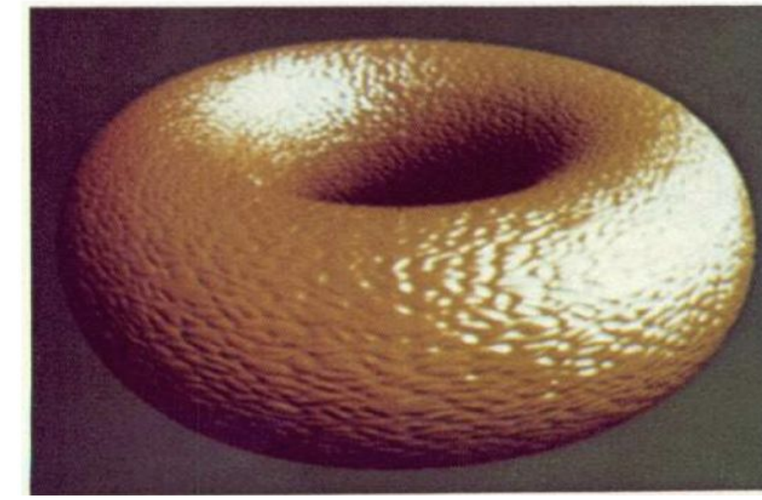
Texture size: *d x d*

$$
\begin{aligned}
& d^2 \left(1 + \frac{1}{4} + \left(\frac{1}{4}\right)^2 + \left(\frac{1}{4}\right)^3 + \ldots\right) \\
& = d^2 \lim_{n\to\infty} \sum_{i=1}^{n} \frac{1}{4^i} \\
& = d^2 \lim_{n\to\infty} \frac{1 - \frac{1}{4^n}}{1 - \frac{1}{4}} \\
& = d^2 \frac{1 - 0}{1 - \frac{1}{4}} \\
& = \frac{4}{3} d^2
\end{aligned}
$$

Storage overhead: ⅓ more storage

Bump Map

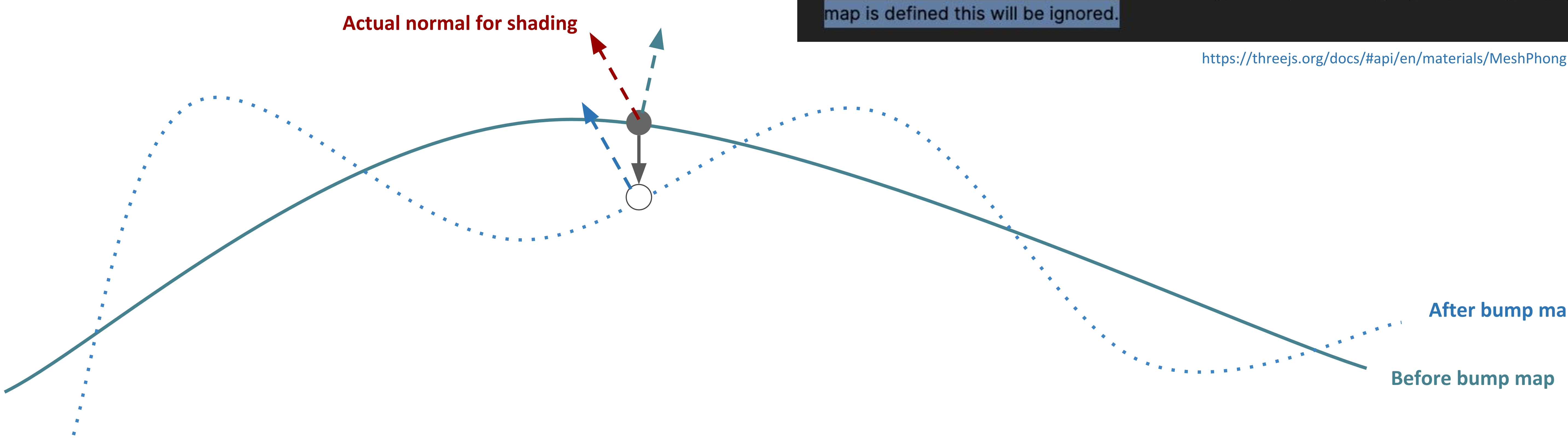
- Often referred as "normal map", although they are different
- A normal map primarily affects the normals of a surface
  - It can add surface detail without adding more triangles
  - Perturb the surface normals per pixel (for shading)
  - The object’s geometry doesn't change



.bumpMap : Texture

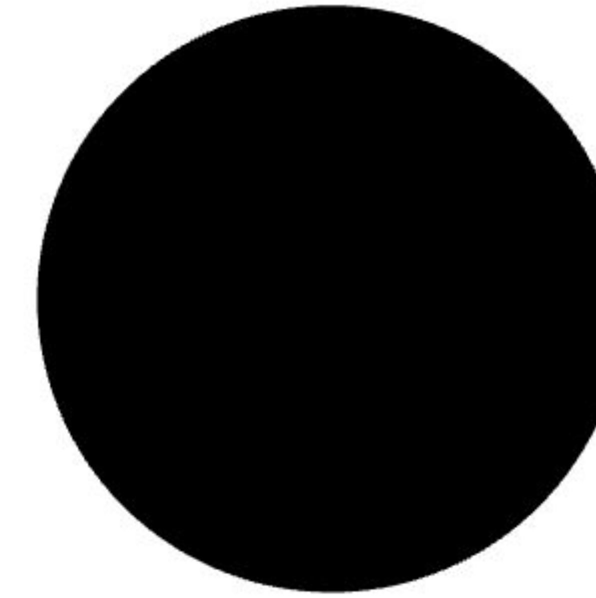
The texture to create a bump map. The black and white values map to the perceived depth in relation to the lights. Bump doesn't actually affect the geometry of the object, only the lighting. If a normal map is defined this will be ignored.

https://threejs.org/docs/#api/en/materials/MeshPhongMaterial.bumpMap



James F. Blinn. 1978. Simulation of wrinkled surfaces. In Proceedings of the 5th annual conference on Computer graphics and interactive techniques (SIGGRAPH ’78). Association for Computing Machinery, New York, NY, USA, 286–292. DOI:https://doi.org/10.1145/800248.507101

Task 1 e)



```
setup() {
  ...
  // TODO: create a directional light and an ambient light
  // using params in above




  // TODO: create a phong material that uses loaded earth texture,
  // normal map, displacement map, and specular map.
  const material = new MeshPhongMaterial({ map: this.assets.earth.texture })
  // TODO: create the earth using SphereBufferGeometry and
  // the created material, then add the earth to the scene
  this.earth = new Mesh(new SphereBufferGeometry(2, 1000, 1000), material)
  this.scene.add(this.earth)
  ...
}
```

Task 1 e) Light up the Earth

```
setup() {
  ...
  // TODO: create a directional light and an ambient light
  // using params in above
  const l = new DirectionalLight(params.light.color)
  l.position.copy(params.light.position)
  this.scene.add(l)
  this.scene.add(new AmbientLight(params.ambient.color, params.ambient.intensity))
  // TODO: create a phong material using the loaded earth texture,
  // normal map, displacement map, and specular map.
  const material = new MeshPhongMaterial({ map: this.assets.earth.texture })
  // TODO: create the earth using SphereBufferGeometry and
  // the created material, then add the earth to the scene
  this.earth = new Mesh(new SphereBufferGeometry(2, 1000, 1000), material)
  this.scene.add(this.earth)
  ...
}
```

Task 1 e) Add Earth Texture

```
setup() {
  ...
  // TODO: create a phong material using the loaded earth texture,
  // normal map, displacement map, and specular map.
  const material = new MeshPhongMaterial({
    map: this.assets.earth.texture,




  })
  ...
}
```

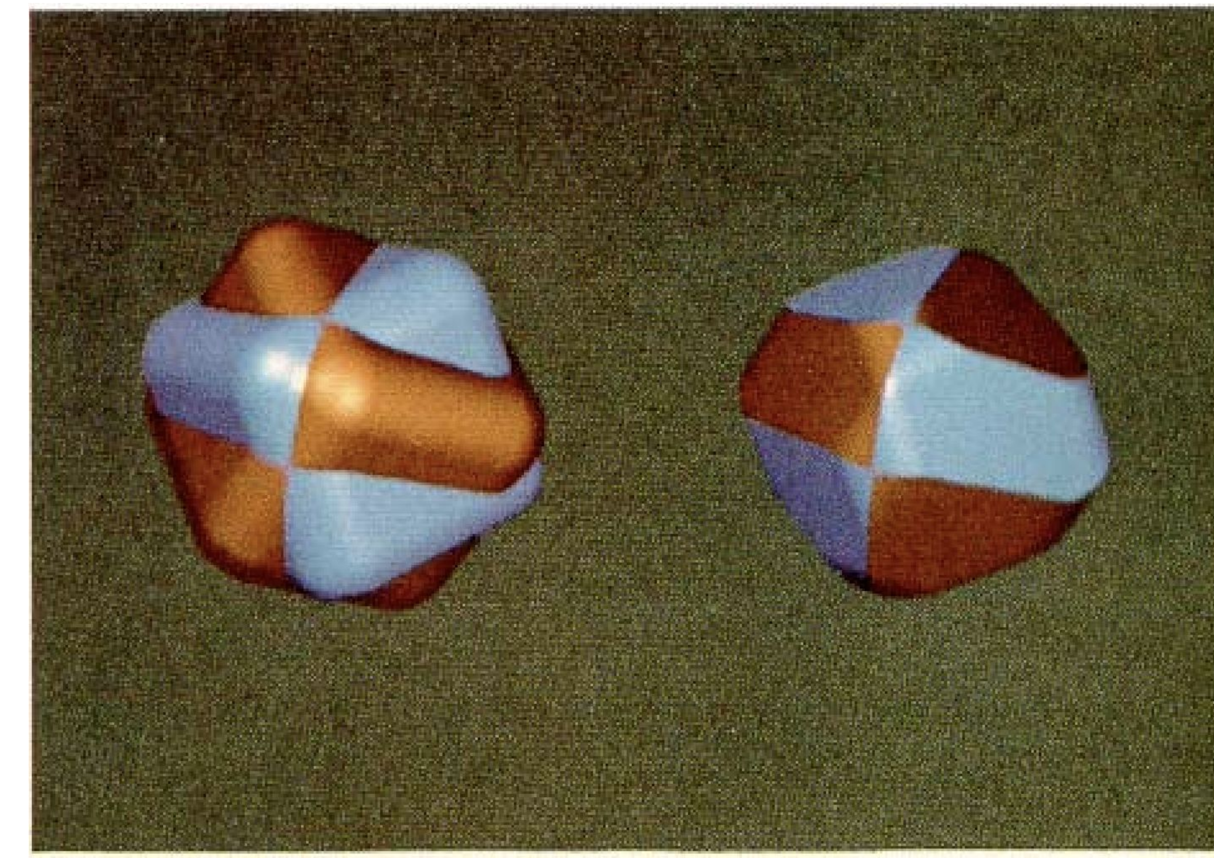
Task 1 e) Add Bump/Normal Map

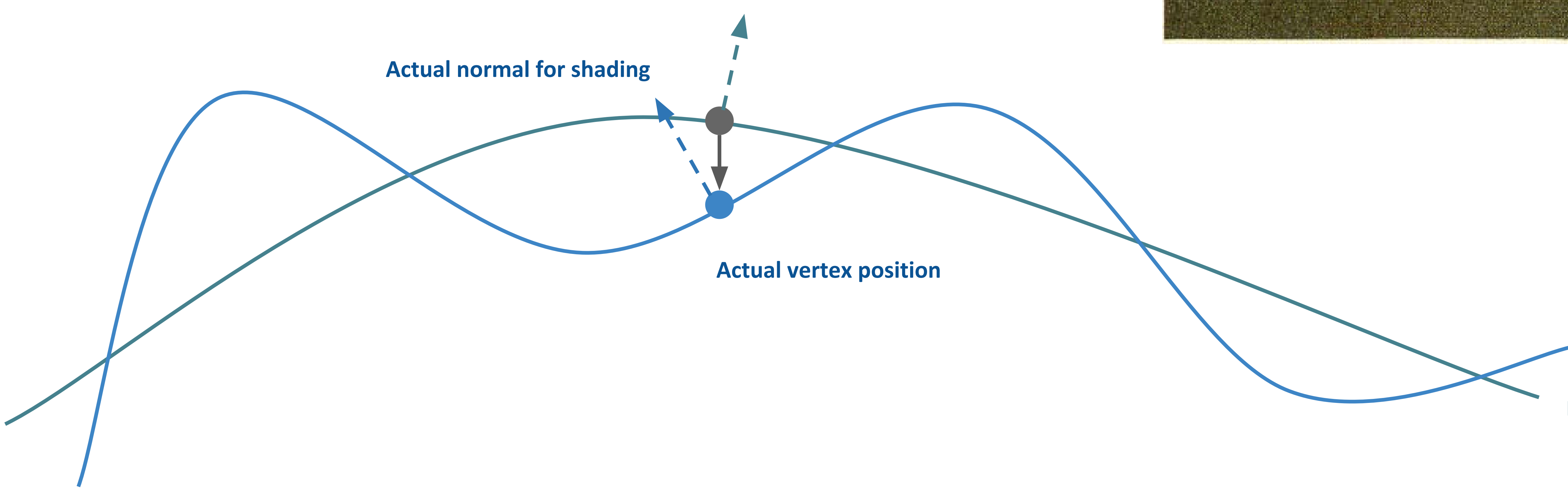
```
setup() {
  ...
  // TODO: create a phong material using the loaded earth texture,
  // normal map, displacement map, and specular map.
  const material = new MeshPhongMaterial({
    map: this.assets.earth.texture,
    normalMap: this.assets.earth.normal,



  })
  ...
}
```

Q: Can you tell the difference?

Displacement Map

- A more advanced method
- Changes the geometry (moves vertices)





Robert L. Cook. 1984. Shade trees. In Proceedings of the 11th annual conference on Computer graphics and interactive techniques (SIGGRAPH '84). Association for Computing Machinery, New York, NY, USA, 223–231. DOI:https://doi.org/10.1145/800031.808602

Task 1 e) Add Displacement Map

```
setup() {
  ...
  // TODO: create a phong material using the loaded earth texture,
  // normal map, displacement map, and specular map.
  const material = new MeshPhongMaterial({
    map: this.assets.earth.texture,
    normalMap: this.assets.earth.normal,
    displacementMap: this.assets.earth.displacement,

  })
  material.displacementScale = 0.1
  ...
}
```

Q: Can you tell the difference?

Specular Map

Yet another map for surface shininess and color highlights

https://willterry.me/2017/05/05/specular-maps/

N. Greene, "Environment Mapping and Other Applications of World Projections," in *IEEE Computer Graphics and Applications*, vol. 6, no. 11, pp. 21-29, Nov. 1986, doi: 10.1109/MCG.1986.276658.

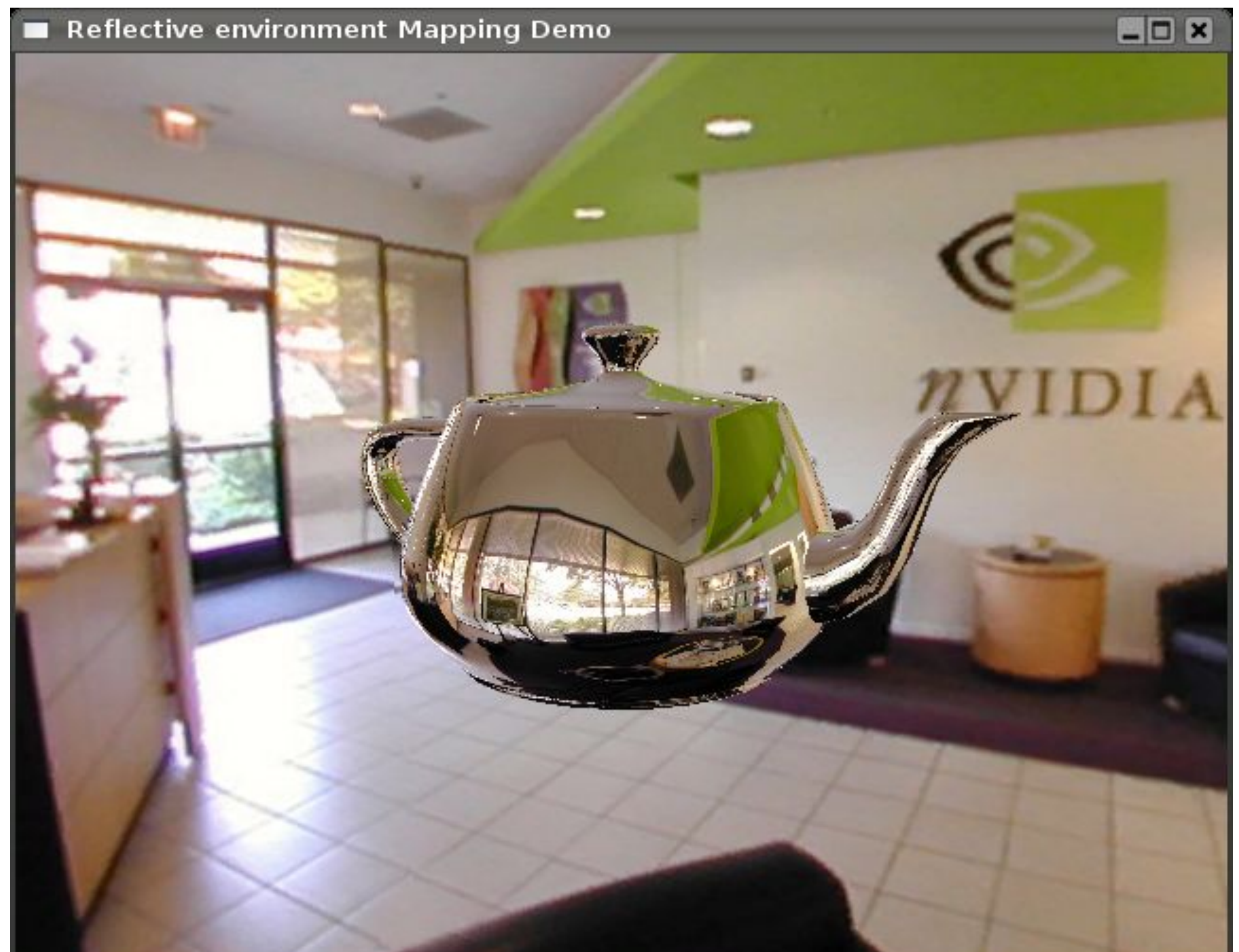
Task 1 e) Add Specular Map

```
setup() {
  ...
  // TODO: create a phong material using the loaded earth texture,
  // normal map, displacement map, and specular map.
  const material = new MeshPhongMaterial({
    map: this.assets.earth.texture,
    normalMap: this.assets.earth.normal,
    displacementMap: this.assets.earth.displacement,
    specularMap: this.assets.earth.spec,
  })
  material.displacementScale = 0.1
  ...
}
```

Q: Can you tell the difference?

Environment Map

An efficient image-based lighting technique for approximating the appearance of a reflective surface by means of a precomputed texture image.



James F. Blinn and Martin E. Newell. 1976. Texture and reflection in computer generated images. Commun. ACM 19, 10 (Oct. 1976), 542–547. DOI:https://doi.org/10.1145/360349.360353

Task 1 e) Add Environment Map

```
setup() {
  ...
  const material = new MeshPhongMaterial({
    ...
    envMap: this.assets.env.texture,
  })
  material.reflectivity = 0.8
  ...
  // TODO: add a sphere that is big enough to fake the sky,
  // and map a environment texture to the inside of the sphere
  this.scene.add(new Mesh(new SphereBufferGeometry(50, 32, 32),
    new MeshBasicMaterial({ map: this.assets.env.texture, side: BackSide })
  ))
  ...
}
update() {
  // TODO: animate the rotation of the earth
  this.earth.rotation.y += 0.01
}
```

If you zoom out...

Now you know the secret of the universe...🤫

Task 1 f) Limitations

Bump map (normal map)

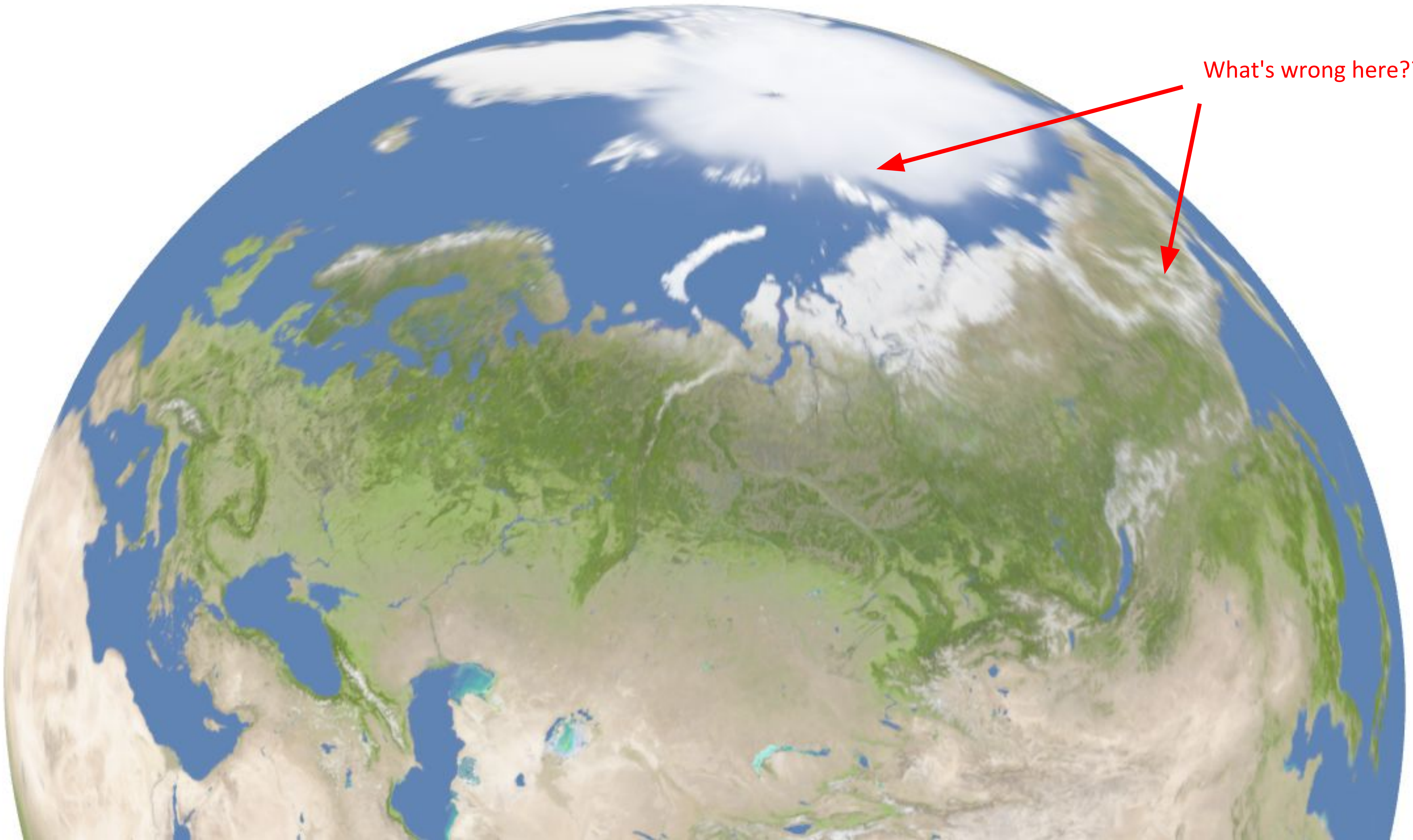
- No actual changes to the geometry
- No actual changes to the casted shadow
- …

Environment map

- No self reflections
- Some geometric objects cannot be correctly mapped to a sphere
- …

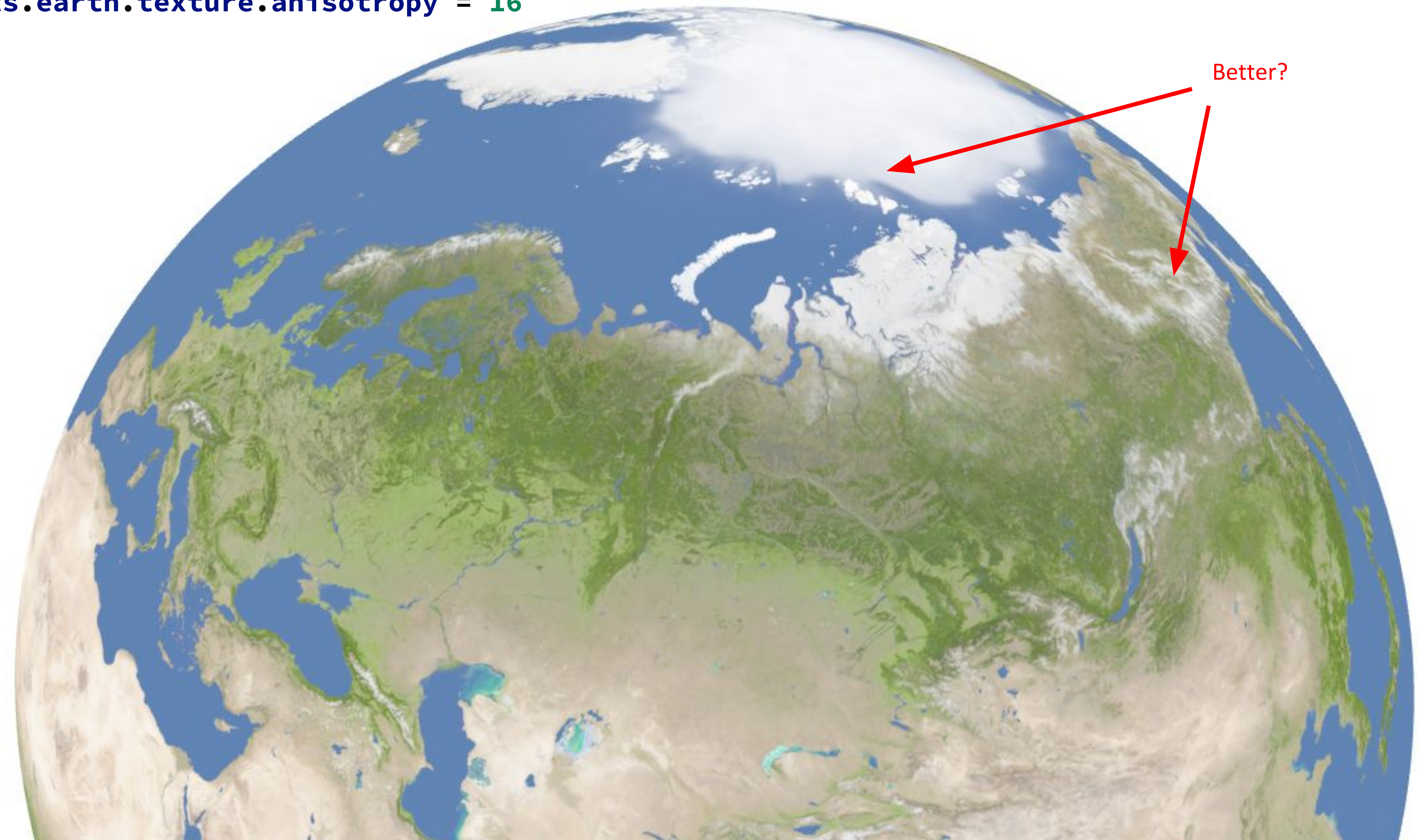
Task 1 g) Mipmap Limitation

The texture is blurred!



Solution: Use Anisotropic Filtering

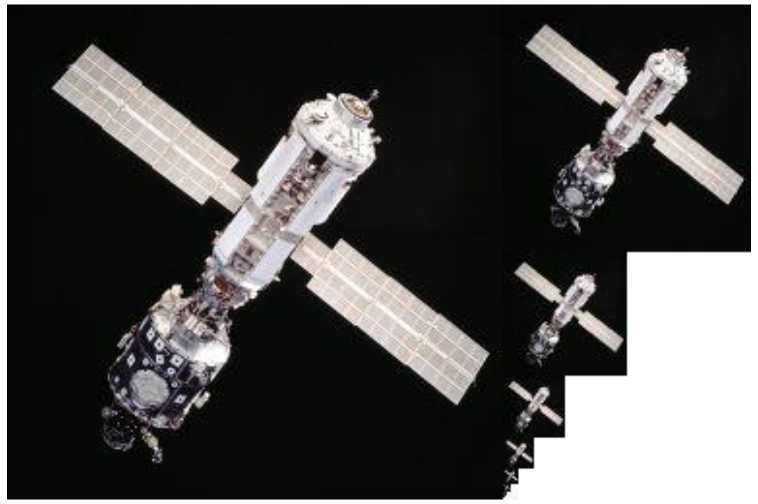
```
setup() {
    ...
    this.assets.earth.texture.anisotropy = 16
    ...
}
```



Anisotropic Filtering

- look up axis-aligned rectangular zones
- Diagonal range query is still an issue

Isotropic *Mipmap*



https://en.wikipedia.org/wiki/Mipmap#/media/File:MipMap_Example_STS101.jpg

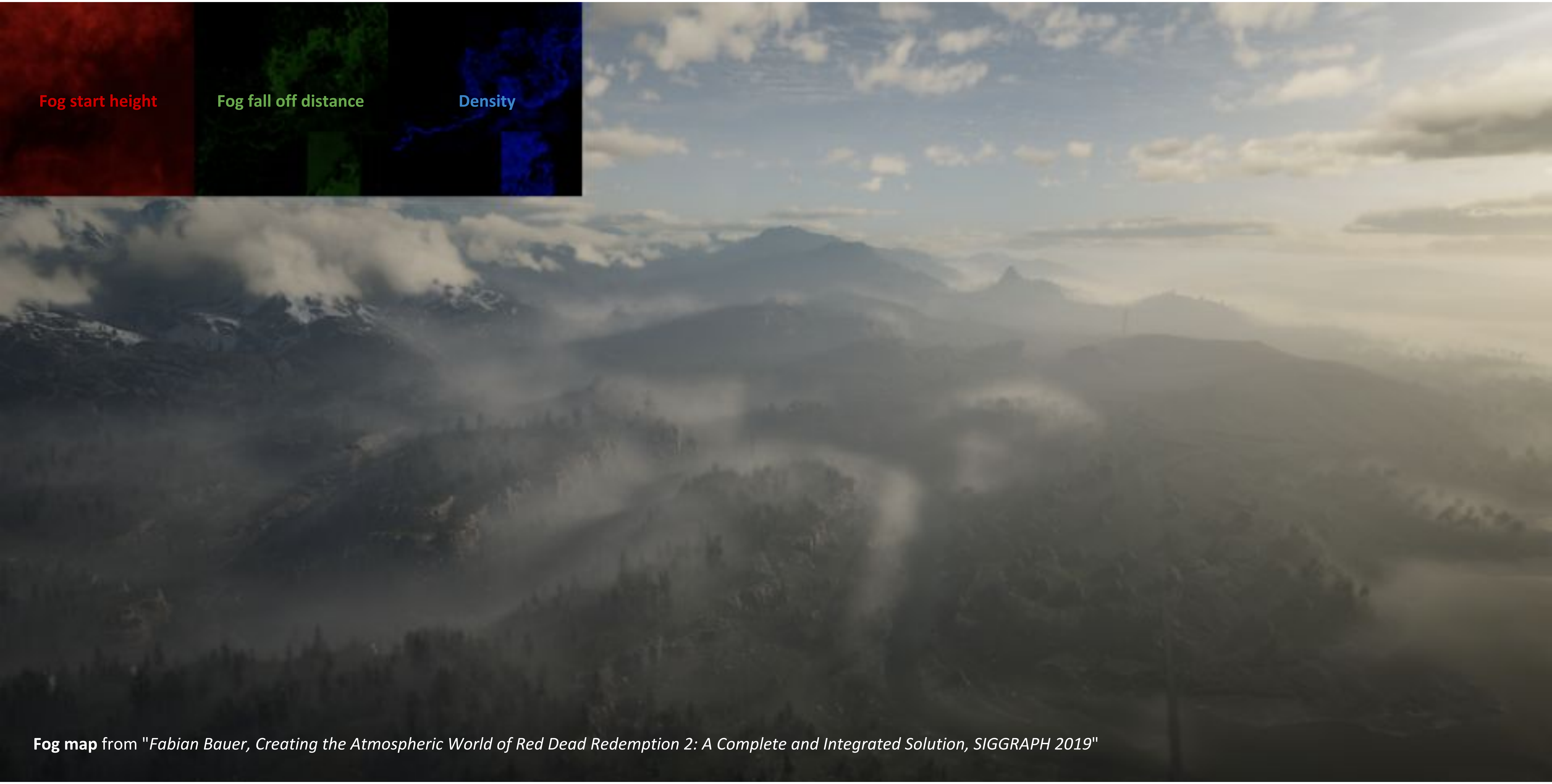
Anisotropic *(Filtering)* *Mipmap*

https://en.wikipedia.org/wiki/Anisotropic_filtering#/media/File:MipMap_Example_STS101_Anisotropic.png

Task 1 Final

Live Demo: https://www.medien.ifi.lmu.de/lehre/ss20/cg1/demo/6-material/earth/index.html

Texture Mapping is Powerful!



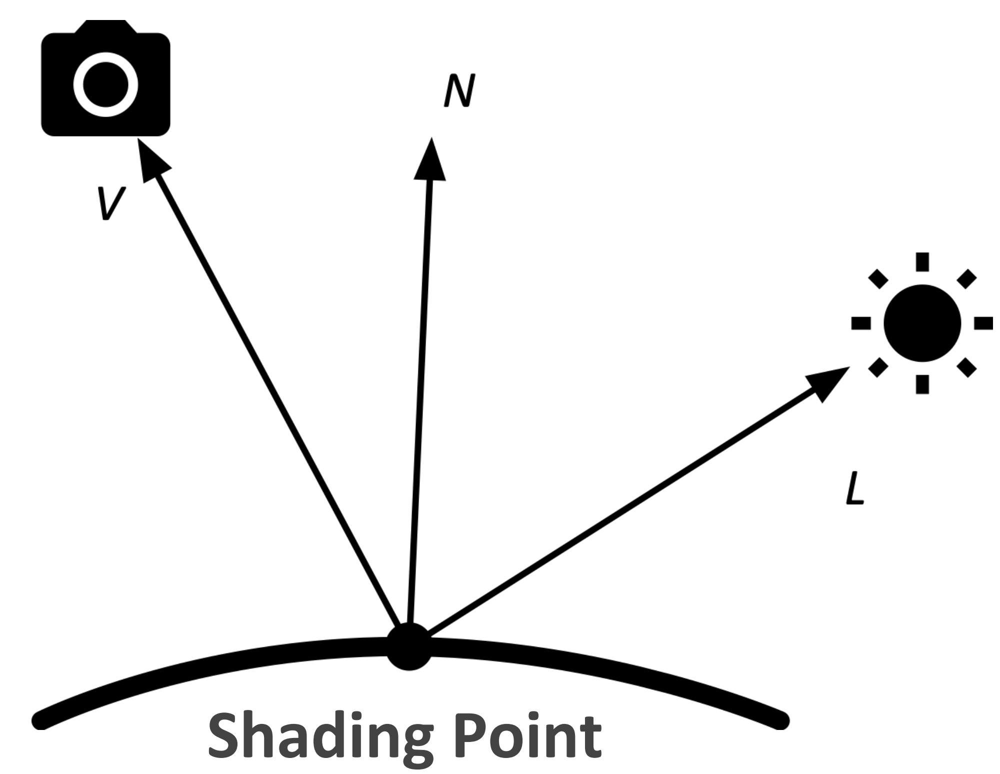
Fog map from "*Fabian Bauer, Creating the Atmospheric World of Red Dead Redemption 2: A Complete and Integrated Solution, SIGGRAPH 2019*"

Tutorial 6: Materials

- Texturing
  - Texture Mapping
  - Barycentric Interpolation
  - Texture Sampling
  - Map Applications
- Shading and Shadowing
  - The Phong and Blinn-Phong Reflection Model
  - Shading Frequency
  - Shadow Map
- Bidirectional Reflectance Distribution Function (BRDF)
  - Radiometry
  - The Rendering Equation

Shading

- Shading is *local* by definition
- The purpose of shading is to compute the color of a shading point
  - In Assignment 5, we use the color directly from three.js without any further computation
- Computation take many factors into consideration:
  - Camera (view) direction, V
  - Surface normal, N
  - Light direction, L
  - Material parameters: color, schininess, …
  - ...



The Phong Reflection Model

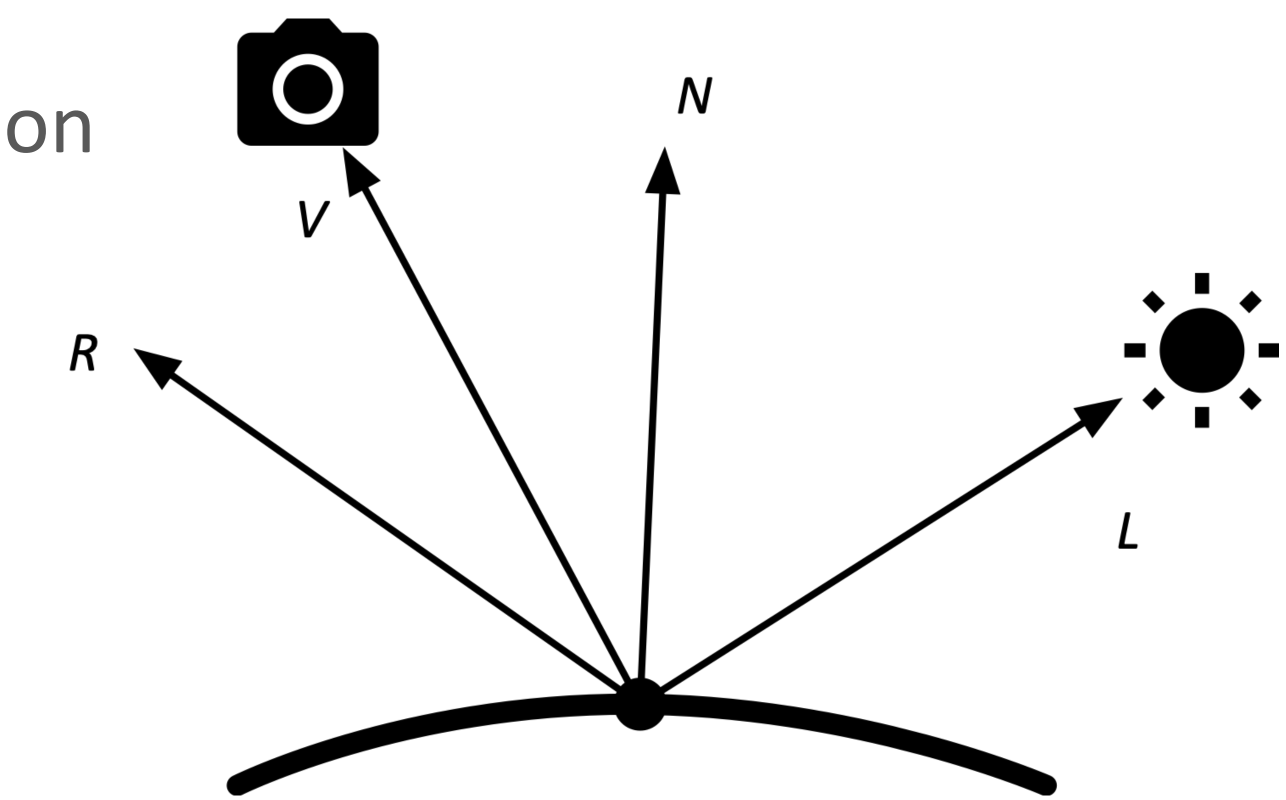
$$L_{\text{Phong}} = L_a + L_d + L_s = k_a I_a + k_d I_d \max(0, \mathbf{N} \cdot \mathbf{L}) + k_s I_s \max(0, \mathbf{R} \cdot \mathbf{V})^p$$

Ambient Term | **Diffuse/*Lambertian* Term** | **Specular/*Phong* Term**

The Phong Reflection Model takes these into account:

- Ambient: a constant intensity
- Diffuse: surface normal and light direction
- Specular: reflected beam direction and camera direction

The specular term is defined as Phong's Term



The Blinn-Phong Reflection Model

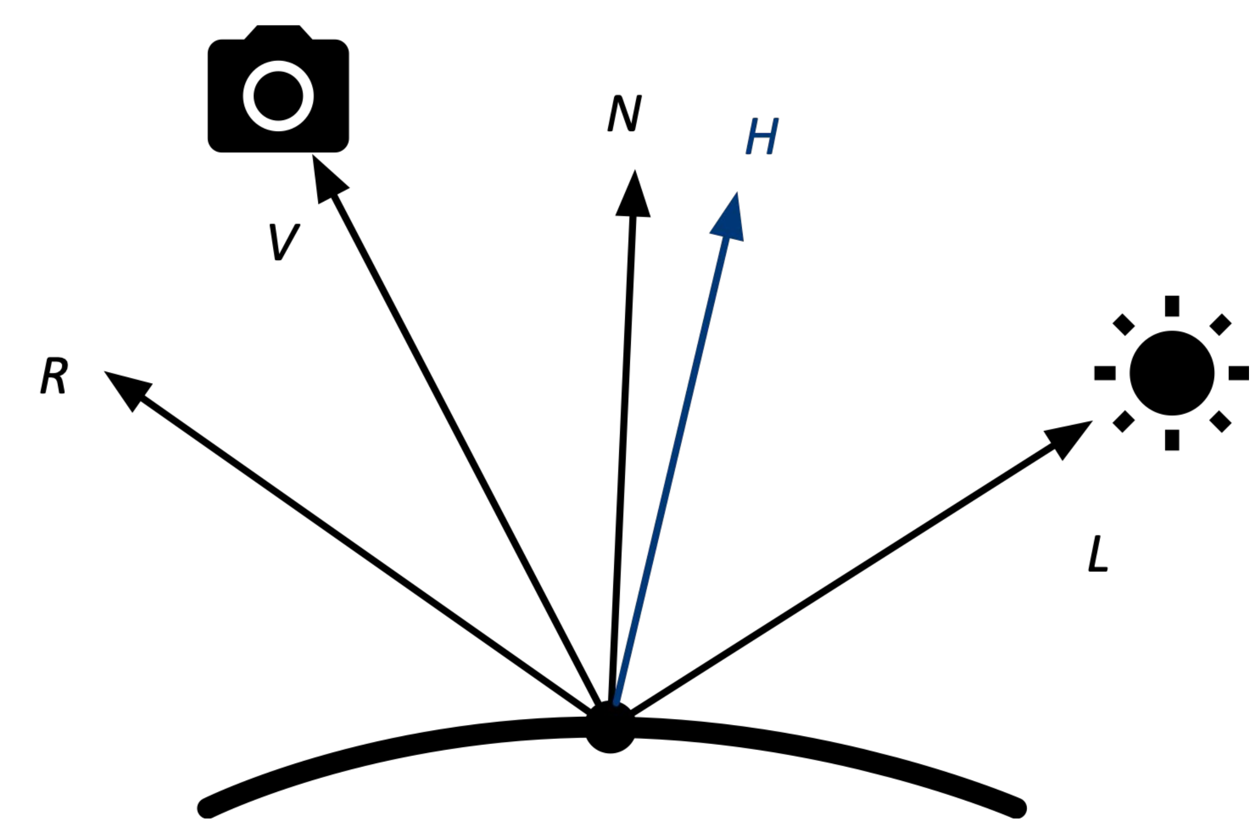
$$L_{\text{Blinn-Phong}} = L_a + L_d + L_s' = k_a I_a + k_d I_d \max(0, \mathbf{N} \cdot \mathbf{L}) + k_s I_s \max(0, \mathbf{N} \cdot \mathbf{H})^p$$

Ambient Term | **Diffuse/*Lambertian* Term** | **Specular/*Blinn-Phong* Term**

The Blinn-Phong Reflection Model takes these into account:

- Ambient: a constant intensity
- Diffuse: surface normal and light direction
- **Specular: surface normal and half vector**
  - **No reflected direction needed ⇒ Fast computation**

The specular term is defined as Blinn-Phong's Term



Task 2 a) Ambient Term

The ambient term doesn't depend on anything

$\Rightarrow$

Assumption: The intensity is equal for all directions

Task 2 b) Phong's Model with Multiple Light Sources

Because of the assumption of the ambient term, the ambient term is not influenced by the number of light sources:

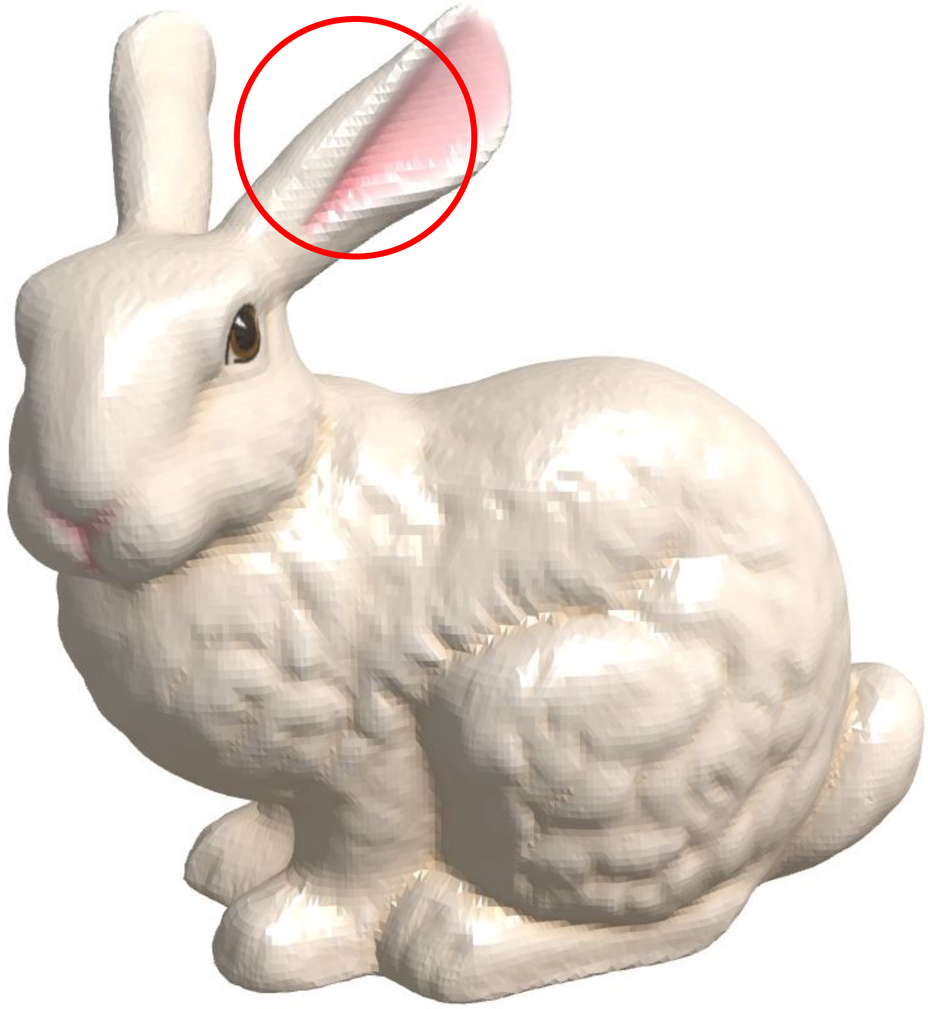
Depends on the material

$$L_{\text{Phong}} = L_a + \sum_{i=1}^{m} (L_{d,i} + L_{s,i}) = k_a I_a + k_d \sum_{i=1}^{m} I_{d,i} \max(0, \mathbf{N} \cdot \mathbf{L}_i) + k_s \sum_{i=1}^{m} I_{s,i} \max(0, \mathbf{R}_i \cdot \mathbf{V})^p$$

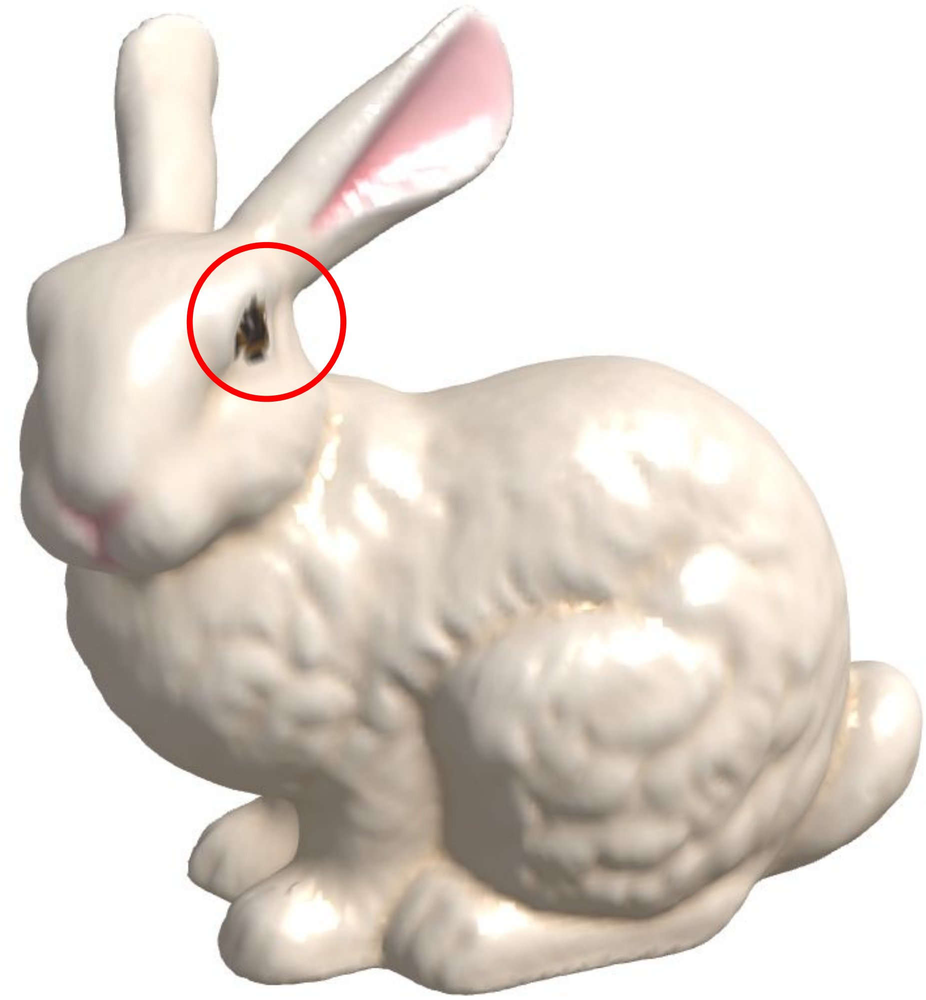
the i-th incoming light

the i-th reflected light

Shading Frequency



Flat Shading (per face)



Gouraud Shading (per vertex)

Phong Shading (per fragment)

Task 2 d)

Flat shading: shading triangles with a single color

Gouraud shading: shading of polygons by interpolating colors that are computed at vertices

Phong shading: normals are interpolated between the vertices and the lighting is evaluated per-pixel

Task 2 c) Values We Need for Uniforms

```
setupBunny() { // src/main.js
  ...
  for (let name in shaders) {
    const m = new Mesh(
      this.assets.bunny.geometry,
      new ShaderMaterial({
        vertexShader: shaders[name].vert, fragmentShader: shaders[name].frag, vertexColors: true,
        uniforms: {
          // TODO: pass the bunny's texture, light position,
          // Kamb, Kdiff, Kspec, shininess to custom shaders.
          bunnyTexture: {value: this.assets.bunny.texture},
          lightPos: {value: this.params.light.position},
          ka: {value: this.params.light.Kamb},
          kd: {value: this.params.light.Kdiff},
          ks: {value: this.params.light.Kspec},
          p: {value: this.params.light.Shininess},
        }
      }),
    )
    ...
  }
}
```

Task 2 c) *Phong Shading* - Vertex Shader

```
#version 300 es
precision highp float; // src/shaders/phong/blinn-phong.vs.glsl

// TODO: receive light position from three.js
uniform vec3 lightPos;

out vec3 N; // normal
out vec4 x; // shading point
out vec3 L; // light direction
out vec2 uvCoordinates; // uv coordinates

void main() {
   gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
   // TODO: compute the normal, position of shading point, light direction
   // and uv coordinates
   N = normalize(normalMatrix*normal);
   x = modelViewMatrix*vec4(position, 1.0);
   L = normalize(vec3(modelViewMatrix*vec4(lightPos, 1.0) - x));
   uvCoordinates = uv;
}
```

Normals aren't applicable with model view transformation (why?)

- Light direction is a unit vector thus we need normalize it
- Th shading point is a position thus we do not normalize it

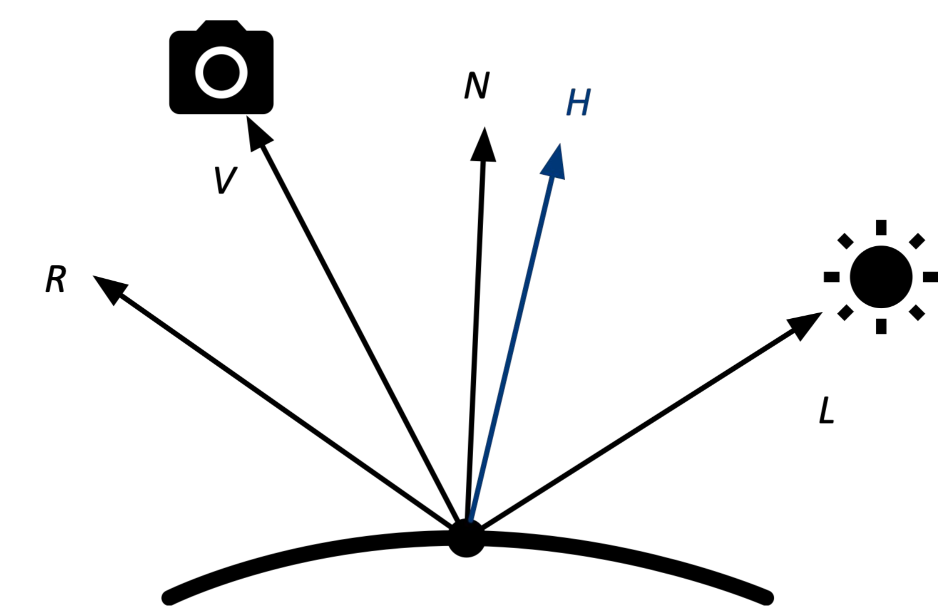
- Keep in mind you need to apply model-view transformation (why?)

Task 2 c) *Phong Shading* - Fragment Shader

```
#version 300 es
precision highp float; // src/shaders/phong/blinn-phong.fs.glsl

// TODO: receive coefficients and shininess from three.js
uniform float ka, kd, ks, p;
uniform sampler2D bunnyTexture;

in vec3 N; // normal
in vec4 x; // shading point
in vec3 L; // light direction
in vec2 uvCoordinates; // uv coordinates

out vec4 outColor;

void main() {
   // TODO: implement the Blinn-Phong reflection model
   // and compute the outColor
   vec3 V = normalize(cameraPosition-vec3(x));
   vec3 H = normalize(L + V);
   vec4 I = texture2D(bunnyTexture, uvCoordinates);

   vec4 La = vec4(ka, ka, ka, 1.0)*I;
   vec4 Ld = vec4(kd, kd, kd, 1.0)*I*max(0.0, dot(N, L));
   vec4 Ls = vec4(ks, ks, ks, 1.0)*I*pow(max(dot(N, H), 0.0), p);

   outColor = La + Ld + Ls;
}
```

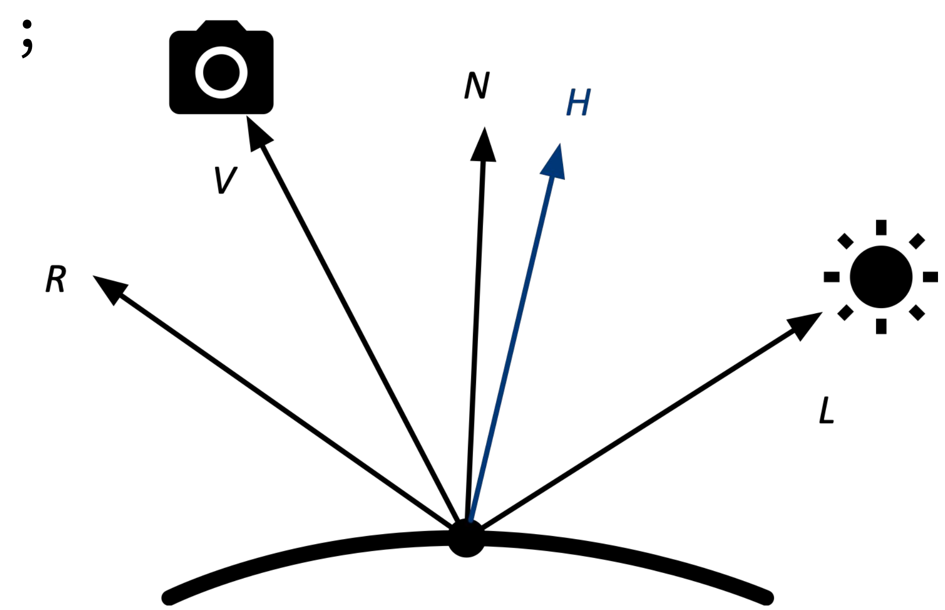
half vector

Use texture color as intensity



Task 2 c) *Gouraud Shading* - Vertex Shader

```
#version 300 es
precision highp float; // src/shaders/gouraud/blinn-phong.vs.glsl
// TODO: receive light position, bunny's texture, coefficients and
// shininess from three.js
uniform vec3 lightPos;
uniform sampler2D bunnyTexture;
uniform float ka, kd, ks, p;

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

  // TODO: implement the Blinn-Phong reflection model
  // and compute the vColor
  vec3 N = normalize(normalMatrix*normal);
  vec4 x = modelViewMatrix*vec4(position, 1.0);
  vec3 L = normalize(vec3(modelViewMatrix*vec4(lightPos, 1.0) - x));

  vec4 I = texture2D(bunnyTexture, uv);
  vec3 V = normalize(cameraPosition-vec3(x));
  vec3 H = normalize(L + V);

  vec4 La  = vec4(ka, ka, ka, 1.0)*I;
  vec4 Ld  = vec4(kd, kd, kd, 1.0)*I*max(0.0, dot(N, L));
  vec4 Ls = vec4(ks, ks, ks, 1.0)*I*pow(max(dot(N, H), 0.0), p);
  vColor = La + Ld + Ls;
}
```

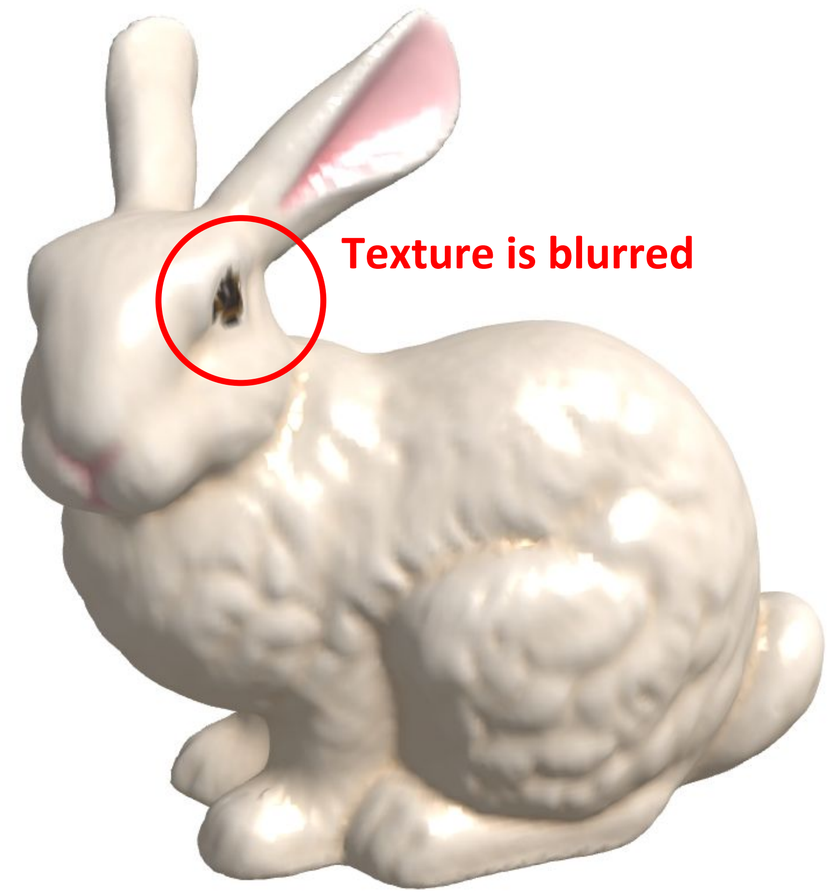


Gouraud shading computes the vertex color in vertex shader

Task 2 c) *Gouraud Shading* - Fragment Shader

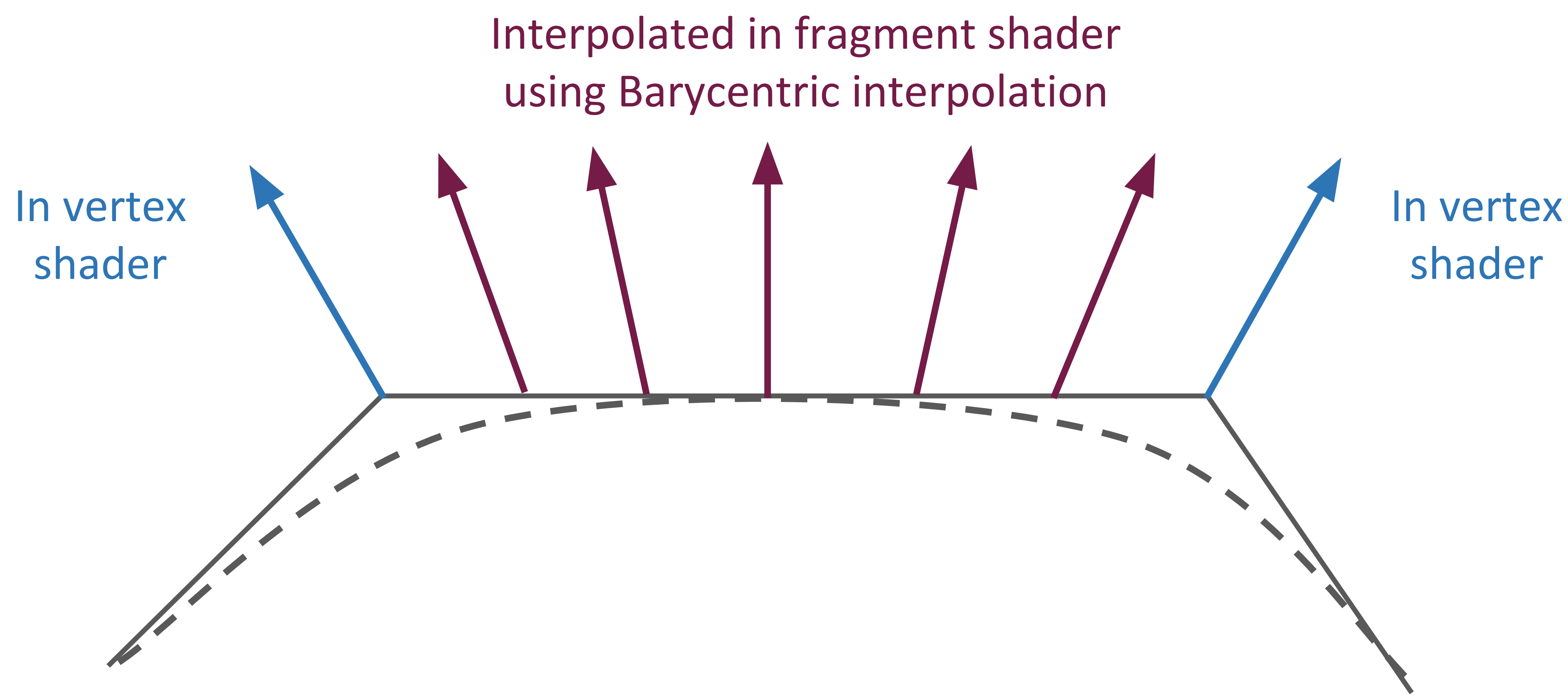
```
#version 300 es
precision highp float; // src/shaders/gouraud/blinn-phong.fs.glsl

// TODO: Receive color from vertex shader
in vec4 vColor;
out vec4 outColor;

void main(void) {
   // TODO: pass color from vertex shader to outColor
   outColor = vColor;
}
```

The received color is interpolated (recall barycentric interpolation)



Interpolation Between Vertex and Fragment Shaders

Take the vertex normal as example:



- Many attributes are interpolation between vertex and fragment shaders:
  - Colors and textures
  - UV coordinates and Normals
  - ...

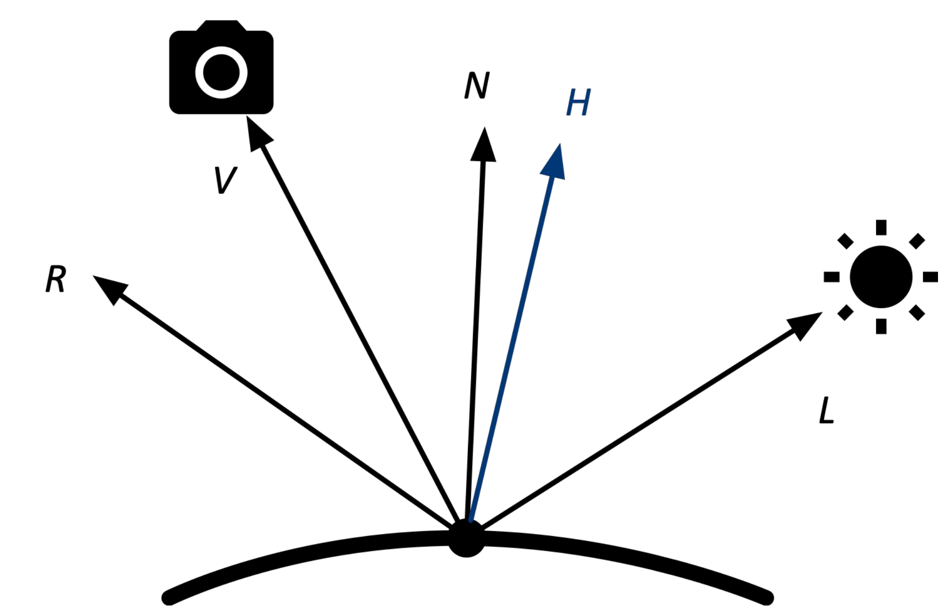
Task 2 c) *Flat Shading* - Vertex Shader

```
#version 300 es
precision highp float;  // src/shaders/flat/blinn-phong.vs.glsl

// TODO: receive light position from three.js
uniform vec3 lightPos;

out vec3 x;  // shading point
out vec3 L;  // light direction
out vec2 uvCoordinates;  // uv coordinates

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

   // TODO: compute the position of the shading point, light direction
   // and uv coordinates
   x = vec3(modelViewMatrix*vec4(position, 1.0));
   L = normalize(vec3(modelViewMatrix*vec4(lightPos, 1.0) - vec4(x, 1.0)));
   uvCoordinates = uv;
}
```

Task 2 c) *Flat Shading* - Fragment Shader

```
#version 300 es
precision highp float; // src/shaders/flat/blinn-phong.fs.glsl
// TODO: receive coefficients, shininess and
// bunny's texture uniform from three.js
uniform float ka, kd, ks, p;
uniform sampler2D bunnyTexture;

in vec3 L;
in vec3 x;
in vec2 uvCoordinates;
out vec4 outColor;
void main() {
    // TODO: implement the Blinn-Phong reflection model
    // and compute the outColor
    vec3 fN = normalize(cross(dFdx(x), dFdy(x)));
    vec3 V = normalize(cameraPosition-x);
    vec3 H = normalize(L + V);
    vec4 I = texture2D(bunnyTexture, uvCoordinates);

    vec4 La = vec4(ka, ka, ka, 1.0)*I;
    vec4 Ld = vec4(kd, kd, kd, 1.0)*I*max(0.0, dot(fN, L));
    vec4 Ls = vec4(ks, ks, ks, 1.0)*I*pow(max(dot(fN, H), 0.0), p);

    outColor = La + Ld + Ls;
}
```

$$\mathbf{N} = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}x} \times \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}y}$$



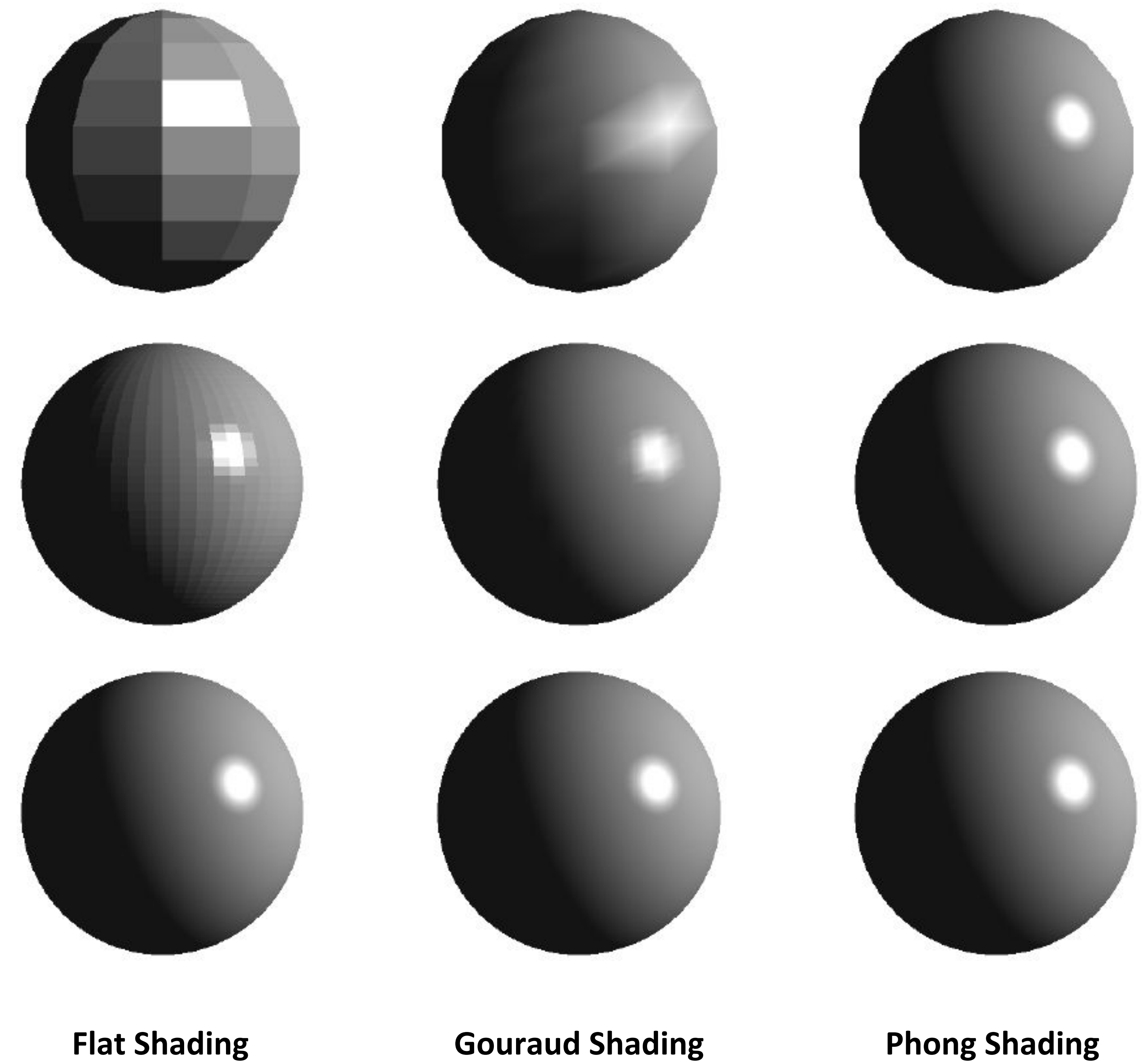
- Use the face normal to make sure the shading of the triangle gets the same color
- The face normal must be computed in fragment shader (why?)

Task 2 e)

With more faces, vertices are more close

- Flat shading
  - face becomes a pixel eventually
- Gouraud shading
  - interpolation between vertices is gone
- Phong shading
  - Works as before

That's why you cannot differentiate if you are not close enough to the object.

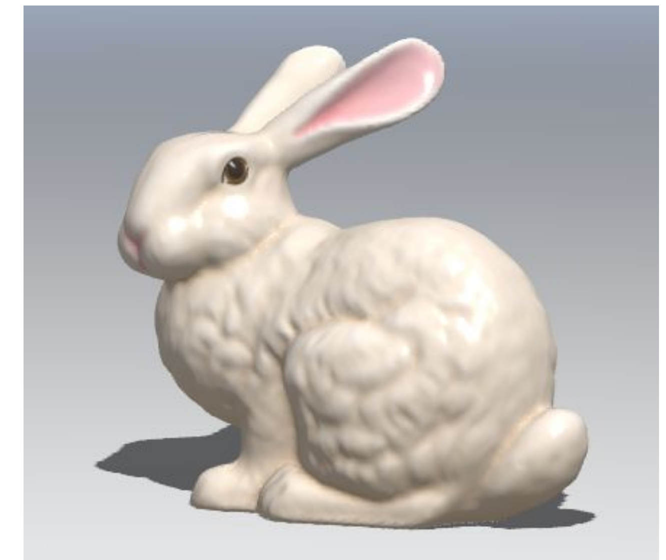


Task 2 f) Shininess

p = 1

p = 10

p = 50



p = 200

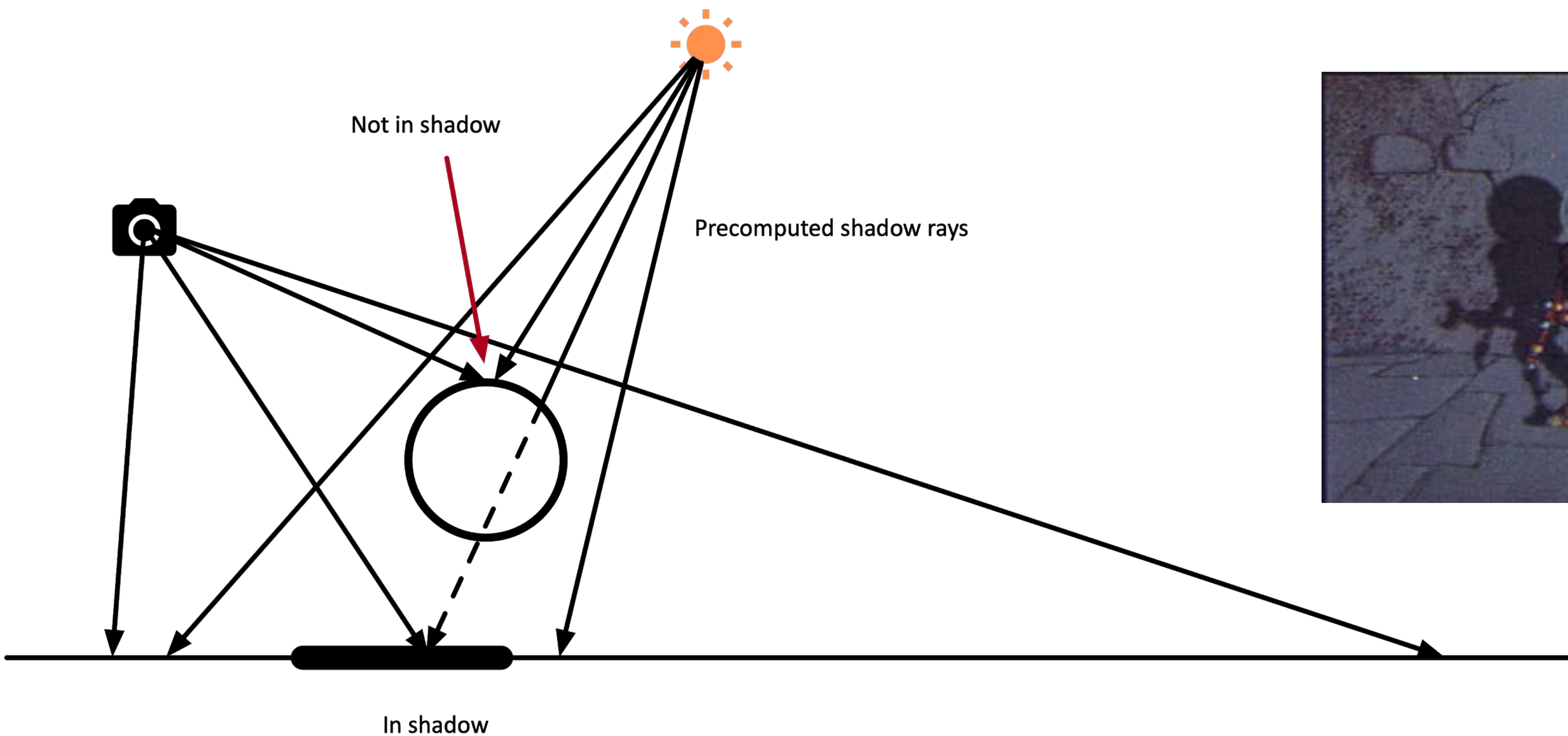
p = 1000

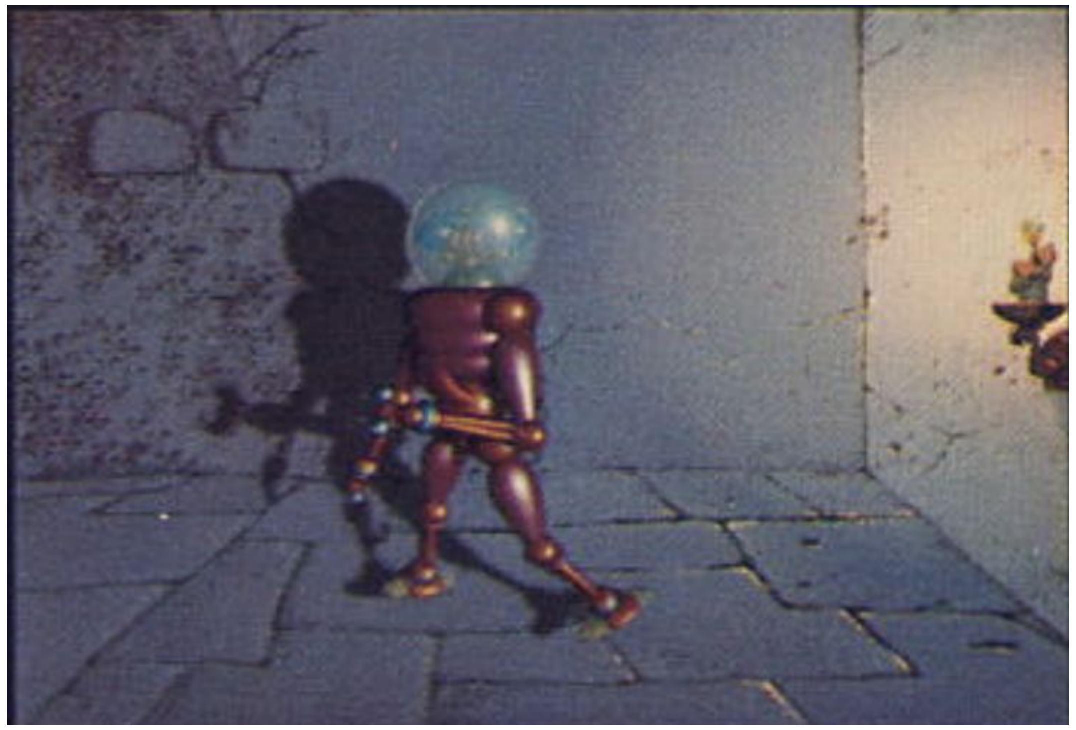
Reflection becomes more shiny when *p* increases

Shadow Map

Basic idea: A point ***not*** in shadow can be seen both by the light (camera) and view camera

The idea can be implemented by comparing depth buffer values.





Lance Williams. 1978. Casting curved shadows on curved surfaces. In Proceedings of the 5th annual conference on Computer graphics and interactive techniques (SIGGRAPH ’78). Association for Computing Machinery, New York, NY, USA, 270–274. DOI:https://doi.org/10.1145/800248.807402

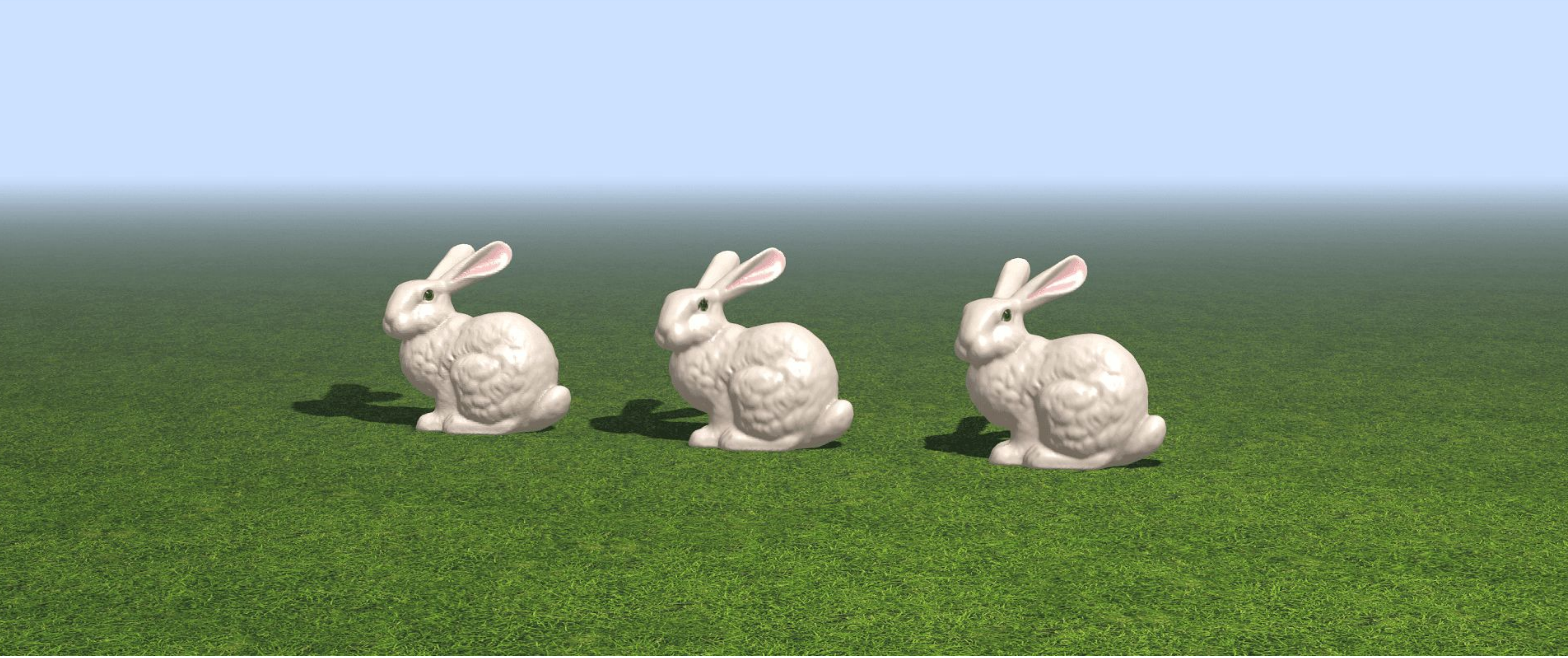
Task 2 g)

In three.js we just need to activate the shadow map...

```
setupShadow() {
  // TODO: activate shadow map
  this.renderer.shadowMap.enabled = true

  this.light.castShadow = true
  this.light.shadow.camera.far = 10000 // the light camera for creating depth buffer

  this.ground.receiveShadow = true

  this.bunnies.forEach(bunny => {
    bunny.castShadow = true
  })
}
```

Task 2 g) Final

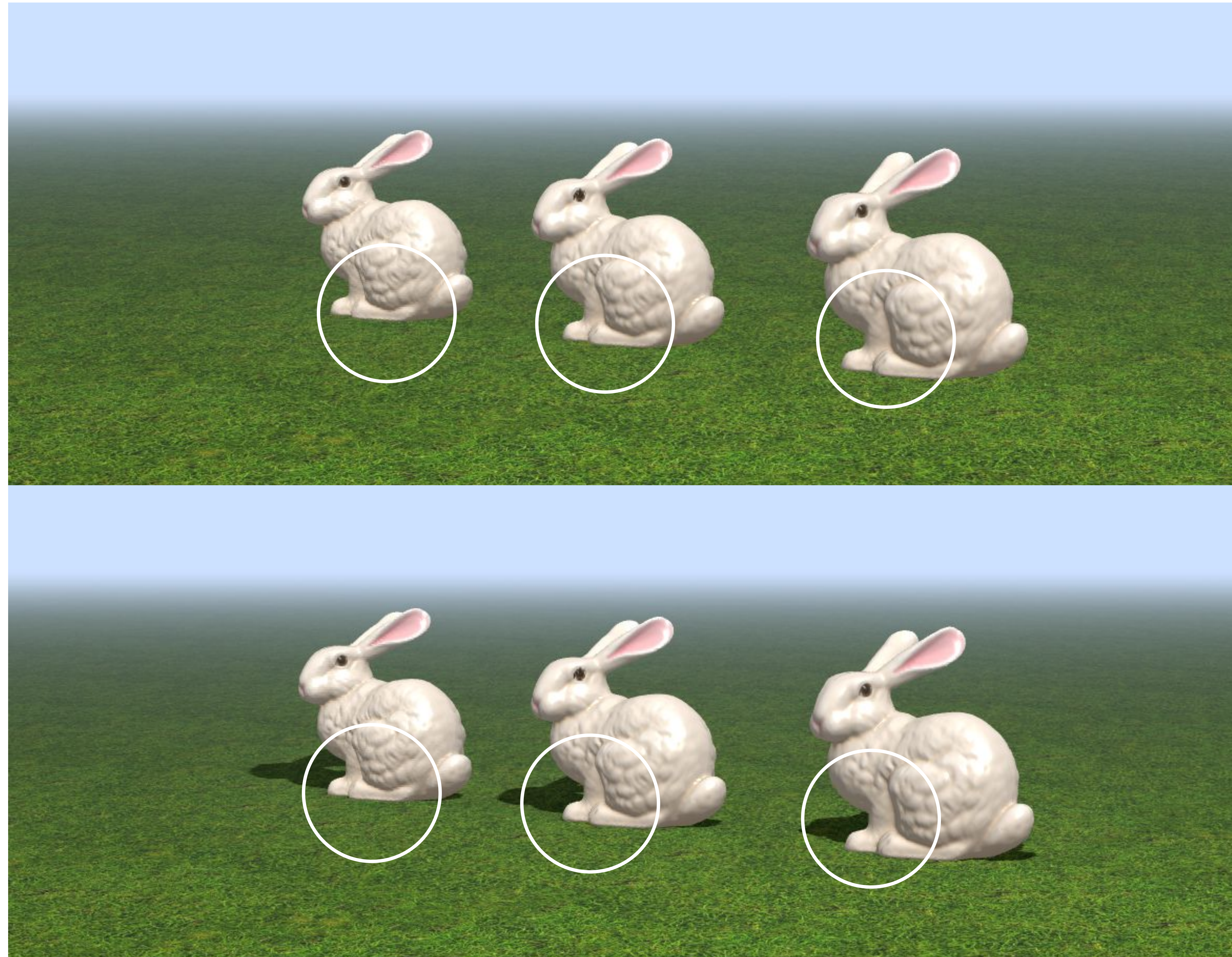
Live Demo: https://www.medien.ifi.lmu.de/lehre/ss20/cg1/demo/6-material/blinn-phong/index.html



Task 2 h)

Without shadows, the bunnies look like they are floating above the ground

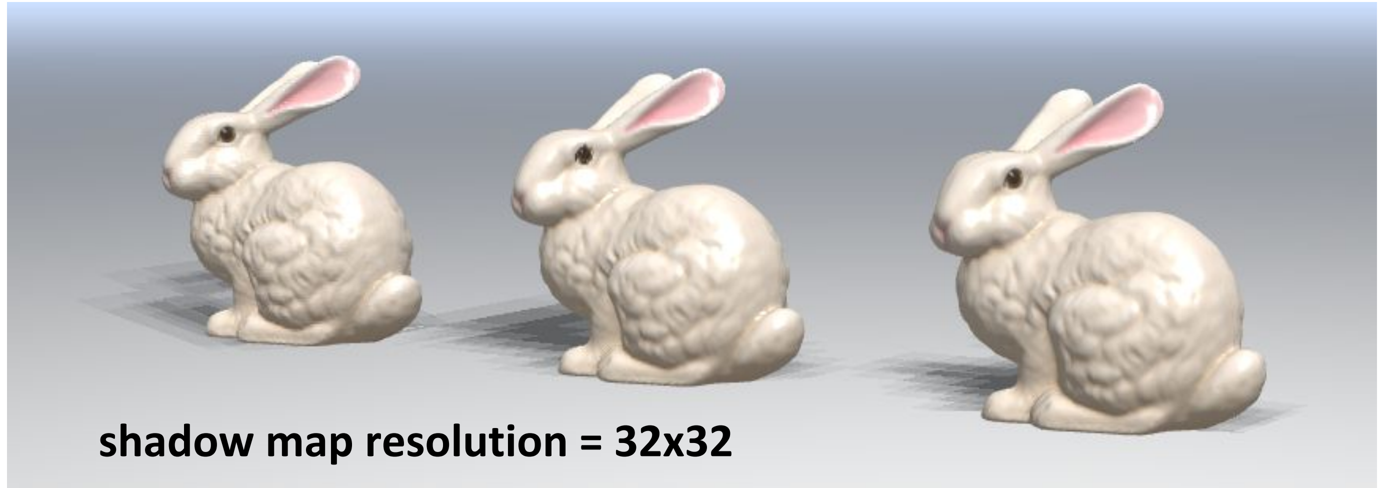
⇒ Shadow plays an important role for spatial vision (object contact)

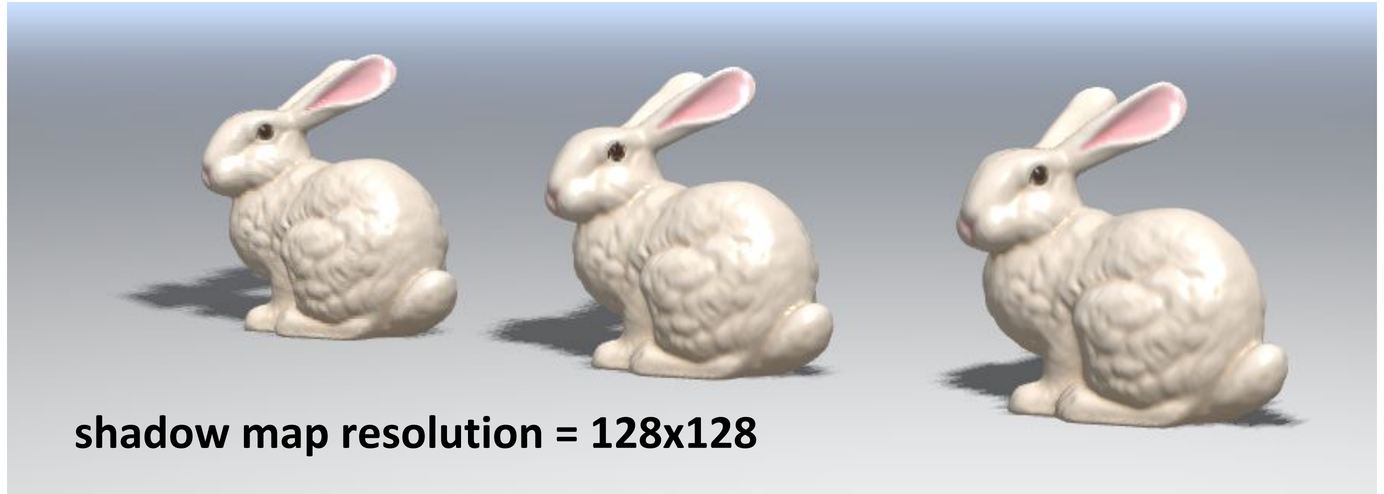


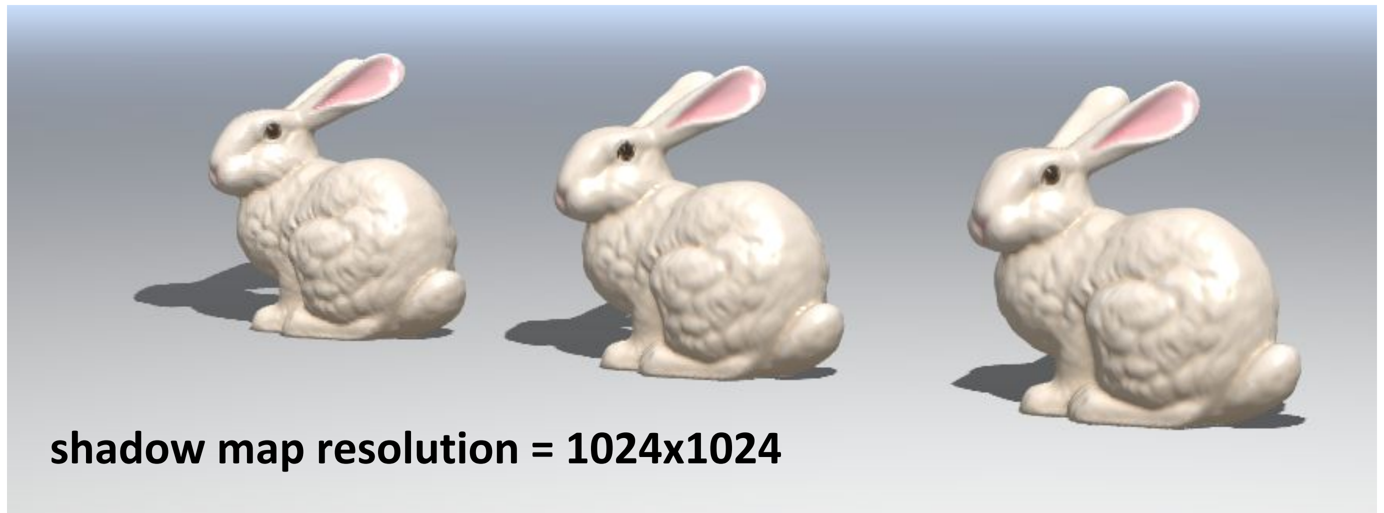
Task 2 i) Problem with Shadow Maps

- Hard shadows
- Quality depends on shadow map resolution
- Involves equality comparison of floating point depth values => issue of scale, bias, tolerance

Try different values: `this.light.shadow.mapSize`







Soft Shadows?

Not now.

Image source: The Last of Us: Part 2

Tutorial 6: Materials

- Texturing
  - Texture Mapping
  - Barycentric Interpolation
  - Texture Sampling
  - Map Applications
- Shading and Shadowing
  - The Blinn-Phong Reflection Model
  - Shading Frequency
  - Shadow Map
- Bidirectional Reflectance Distribution Function (BRDF)
  - Radiometry
  - The Rendering Equation

The Rendering Equation

$$L_o(\mathbf{x}, w_o) = L_e(\mathbf{x}, w_o) + L_r(\mathbf{x}, w_o) = L_e(\mathbf{x}, w_o) + \int_\Omega f_r(\mathbf{x}, w_i, w_o) L_i(\mathbf{x}, w_i) \cos\theta_i \mathrm{d}w_i$$

???

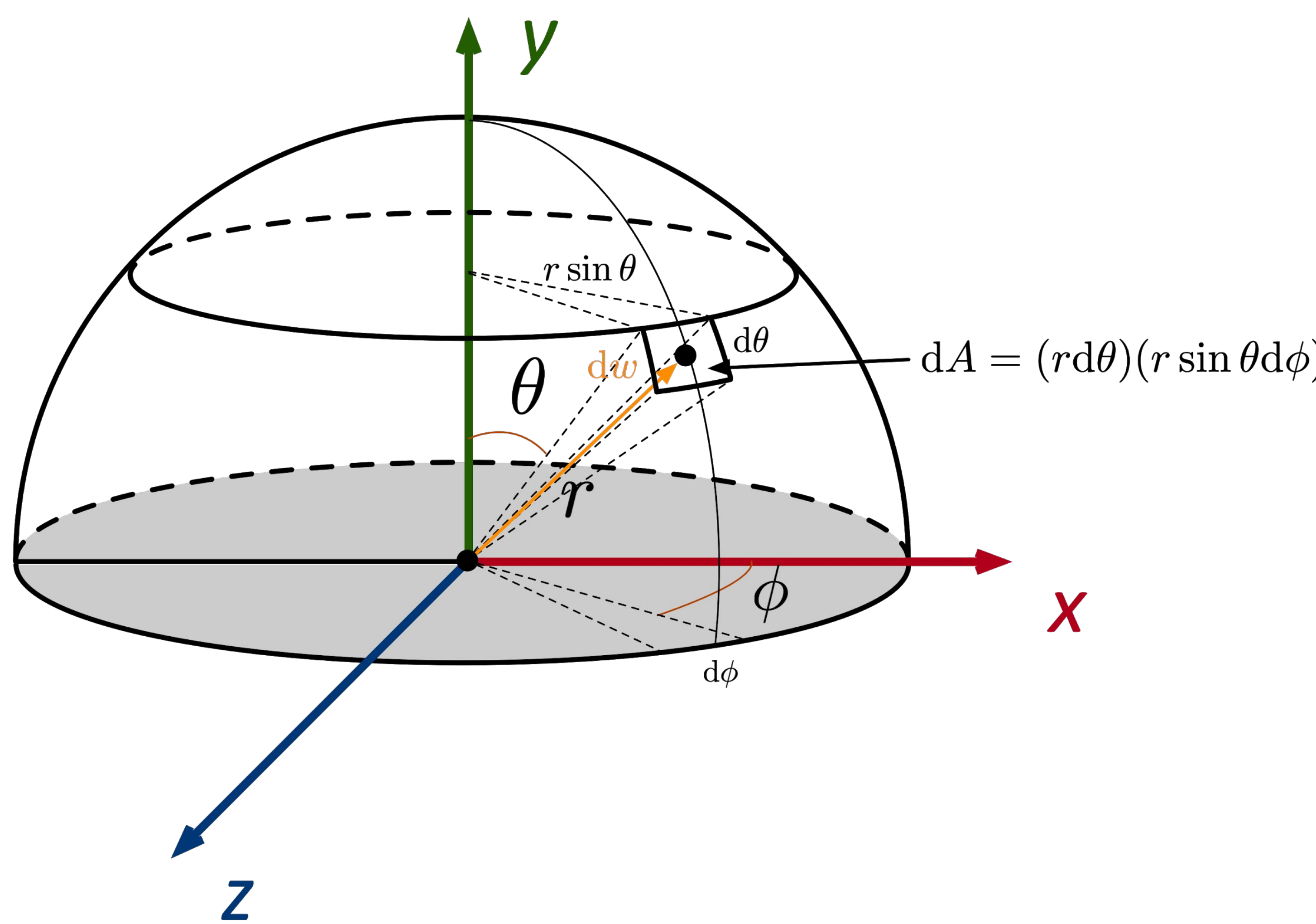
The rendering equation is based on radiometry. To understand it deeper, let's review it start from the beginning….

Solid Angle

A solid angle is a ratio of a subtended area of a sphere to the radius squared: $\frac{A}{r^2}$

A differential solid angle is

$$\mathrm{d}w = \frac{\mathrm{d}A}{r^2} = \frac{r\theta r \sin\theta \mathrm{d}\phi}{r^2} = \sin\theta \mathrm{d}\theta \mathrm{d}\phi$$



Irradiance

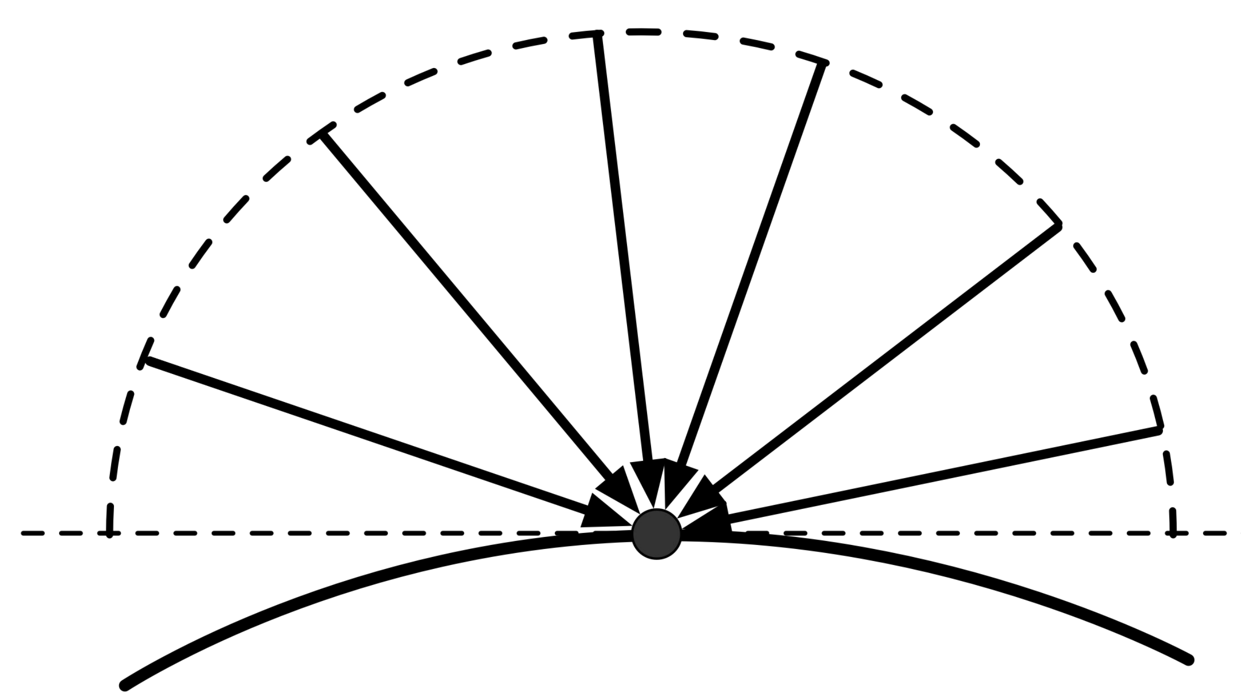
Radiant energy (electromagnetic radiation): Q

Radiant power (flux) is the radiant energy per unit time: $\Phi = \frac{\mathrm{d}Q}{\mathrm{d}t}$

Intensity is the power per solid angle: $I = \frac{\mathrm{d}\Phi}{\mathrm{d}w}$

Irradiance is the total power received by area **d*A*** on a surface point ***x***:

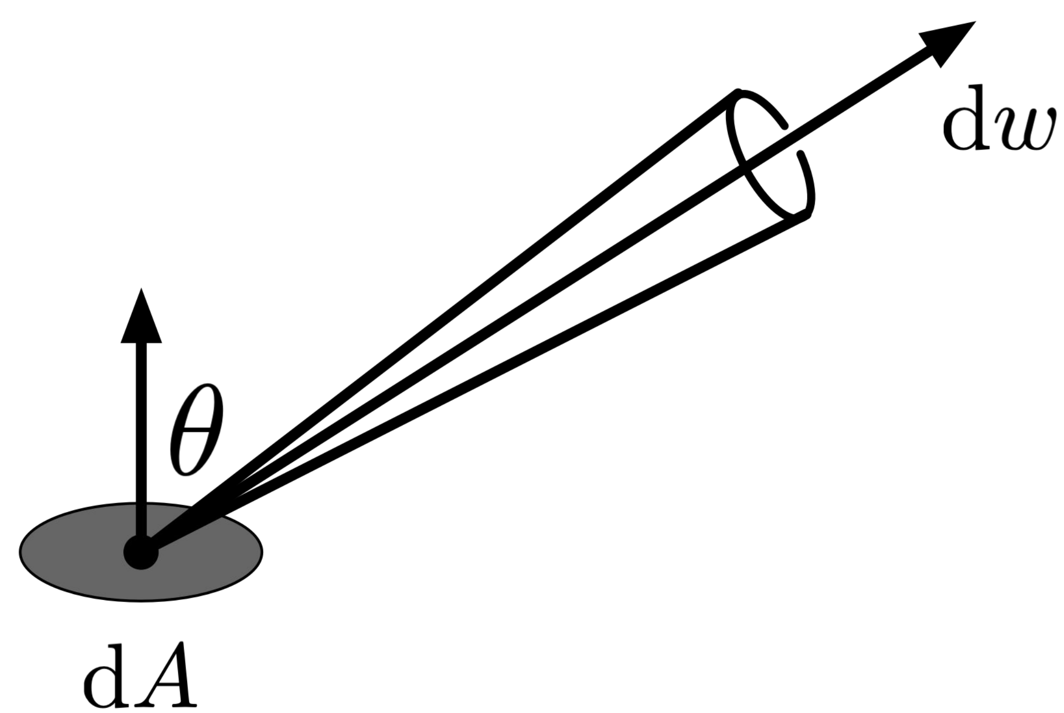
$$E(\mathbf{x}) = \frac{\mathrm{d}\Phi(\mathbf{x})}{\mathrm{d}A}$$



Radiance (Luminance)

Radiance is the power received by area **d*A*** from direction **d*w***:

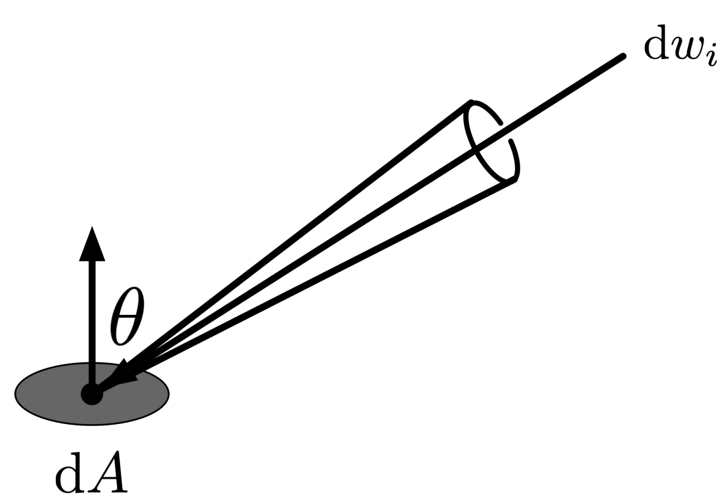
$$L(\mathbf{x}, w) = \frac{\mathrm{d}^2\Phi(\mathbf{x}, w)}{\mathrm{d}w\mathrm{d}A\cos\theta}$$



We always assume ***w*** towards from ***x*** to unit hemisphere surface even for incoming lights

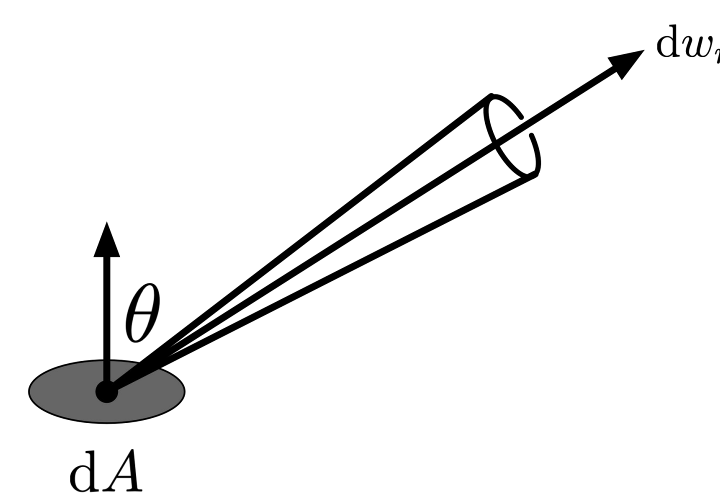
Radiometry Summary

- Irradiance: power received by ***dA***
- Intensity: power per solid angle
- Radiance: power received by area ***dA*** from direction ***dw***
  - Can be seen as: Irradiance per solid angle
  - Can be seen as: Intensity received by ***dA***



$$L(\mathbf{x}, w_i) = \frac{\mathrm{d}E(\mathbf{x})}{\cos\theta \mathrm{d}w_i}$$

Incoming radiance can be seen as irradiance per solid angle (irradiance project from ***dw*** to ***dA***)



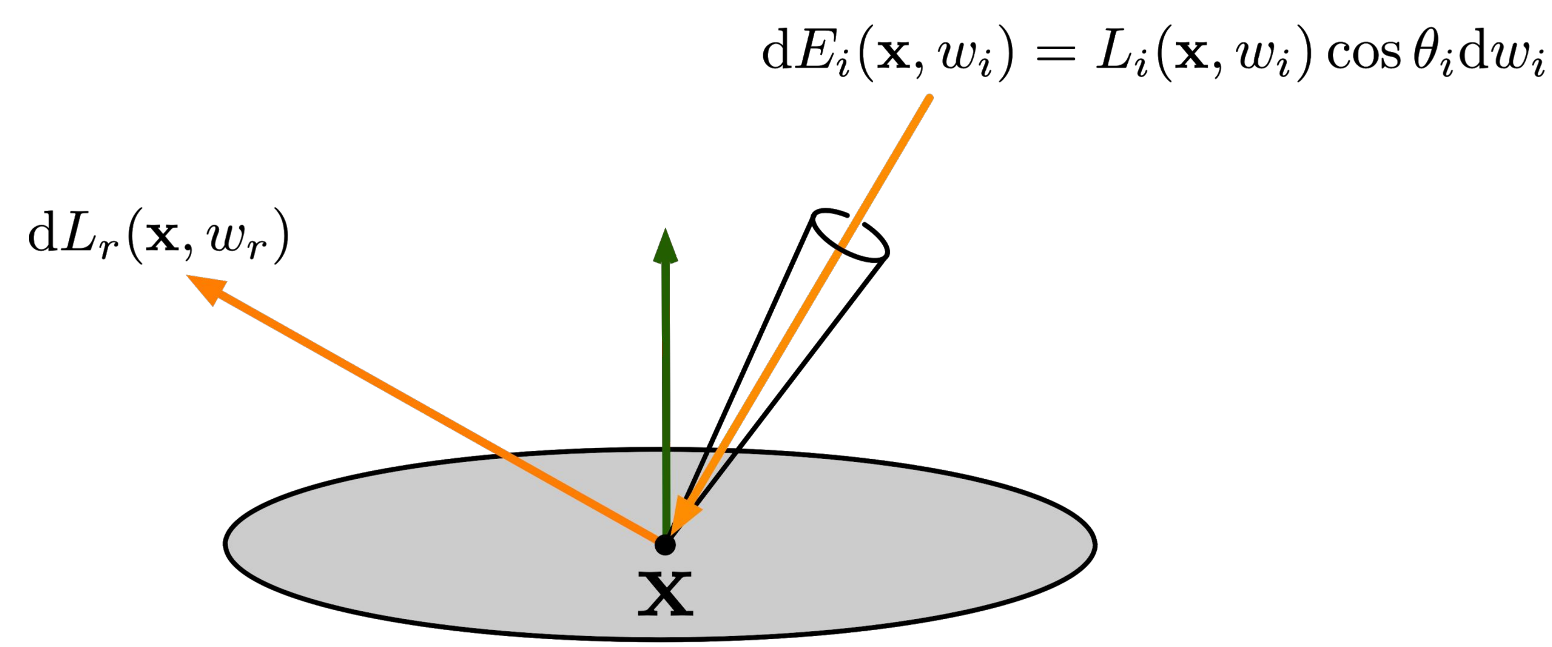
$$L(\mathbf{x}, w_r) = \frac{I(\mathbf{x}, w_r)}{\cos\theta \mathrm{d}A}$$

Outgoing radiance can be seen as intensity received by ***dA*** (intensity project from ***dA*** to ***dw***)

Bidirectional Reflectance Distribution Function (BRDF)

The BRDF indicates how much light is reflected into each outgoing direction from each incoming direction:

$$f_r(\mathbf{x}, w_i, w_o) = \frac{\mathrm{d}L_r(\mathbf{x}, w_r)}{\mathrm{d}E_i(\mathbf{x}, w_i)} = \frac{\mathrm{d}L_r(\mathbf{x}, w_r)}{L_i(\mathbf{x}, w_i)\cos\theta_i \mathrm{d}w_i}$$



A BRDF is equivalent to the physical property of a material (how light is reflected)

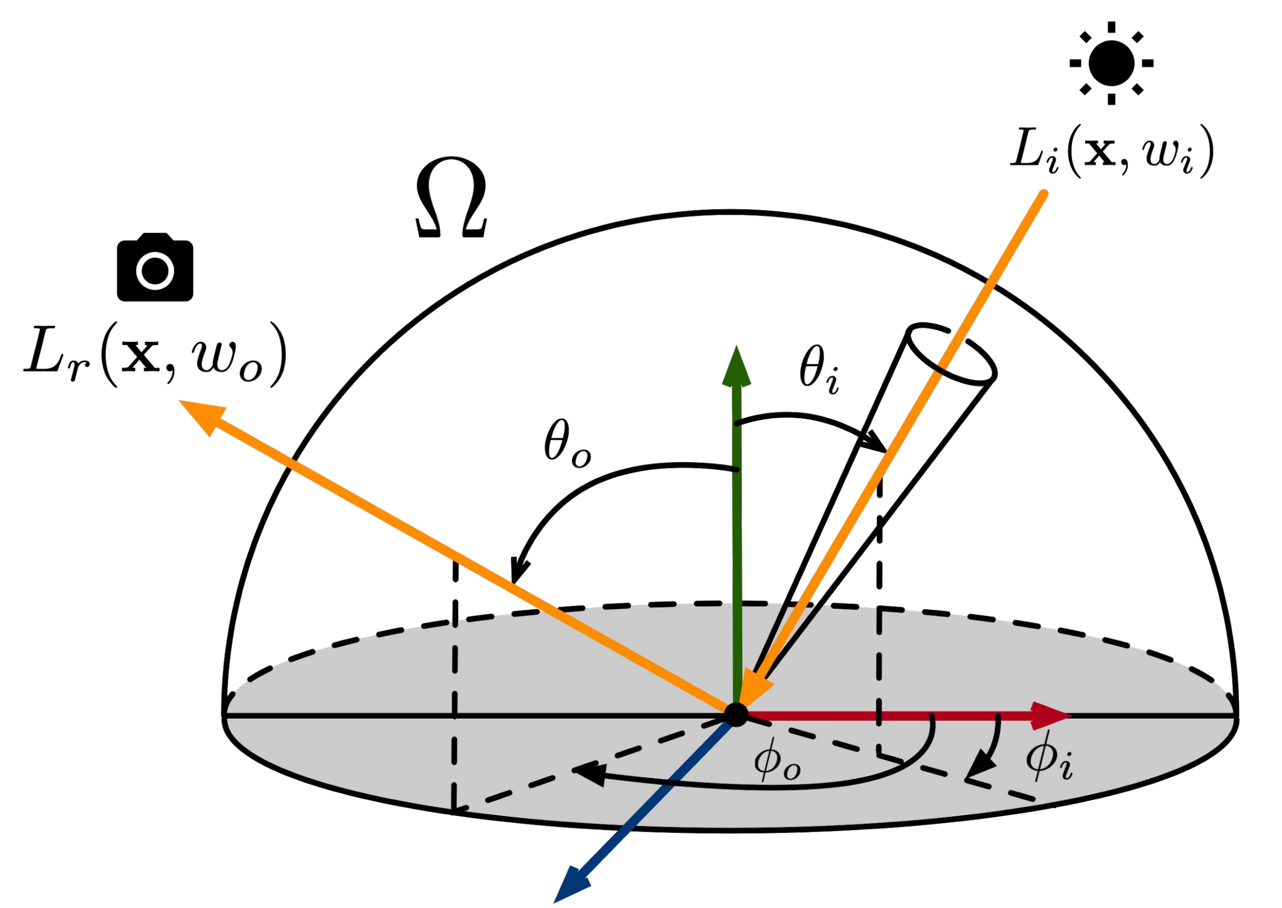
The Reflection Equation

From BRDF definition: $\mathrm{d}L_r(\mathbf{x}, w_r) = f_r(\mathbf{x}, w_i, w_o) L_i(\mathbf{x}, w_i) \cos\theta_i \mathrm{d}w_i$

We deduce *the reflection equation* by integration on the two side of the equation in above:

$$L_r(\mathbf{x}, w_o) = \int_{\Omega} f_r(\mathbf{x}, w_i, w_o) L_i(\mathbf{x}, w_i) \cos\theta_i \mathrm{d}w_i$$

Reflected Radiance *Unit Hemisphere* *BRDF* *Incoming Radiance* *Cosine of Incident Angle*



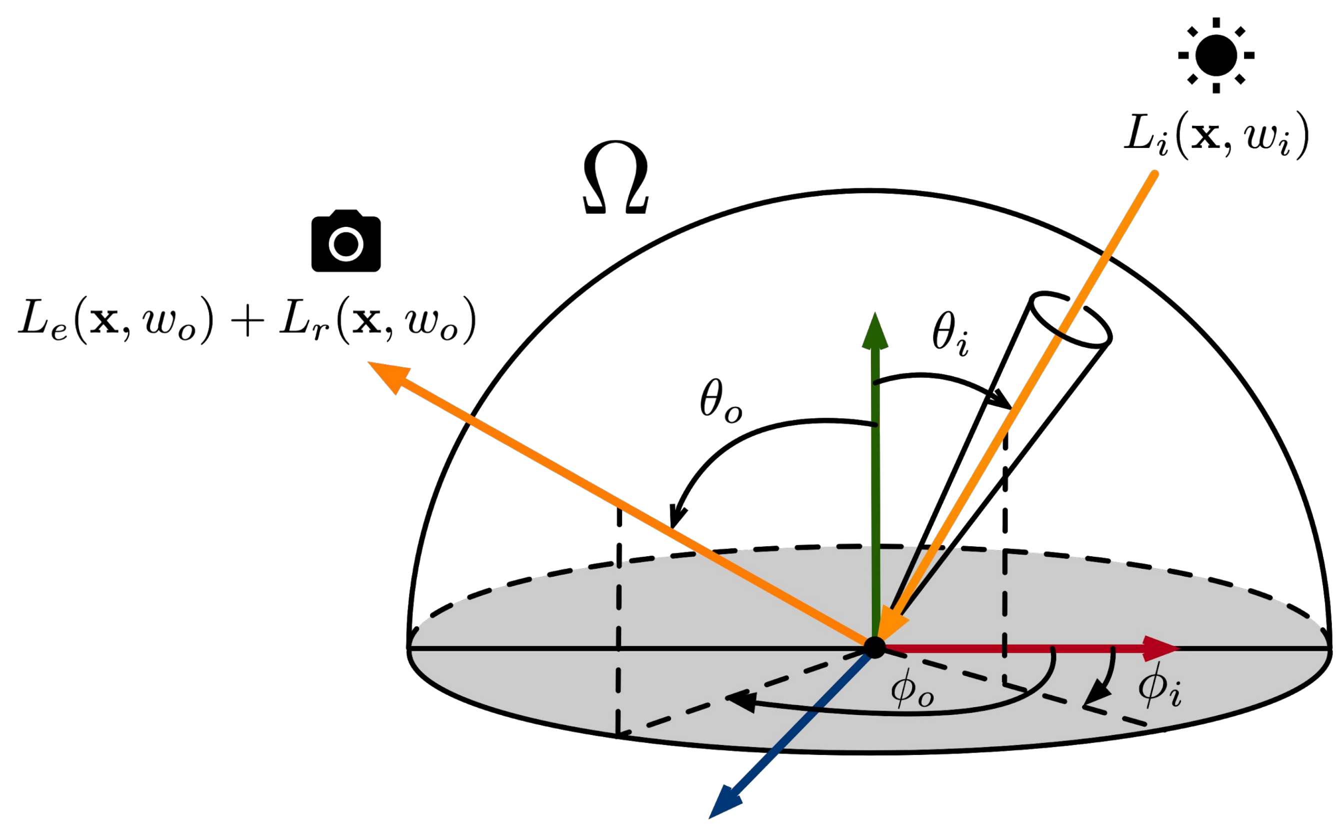
We always assume $\boldsymbol{w_i}$ towards from $\boldsymbol{x}$ to unit hemisphere surface even for incoming lights

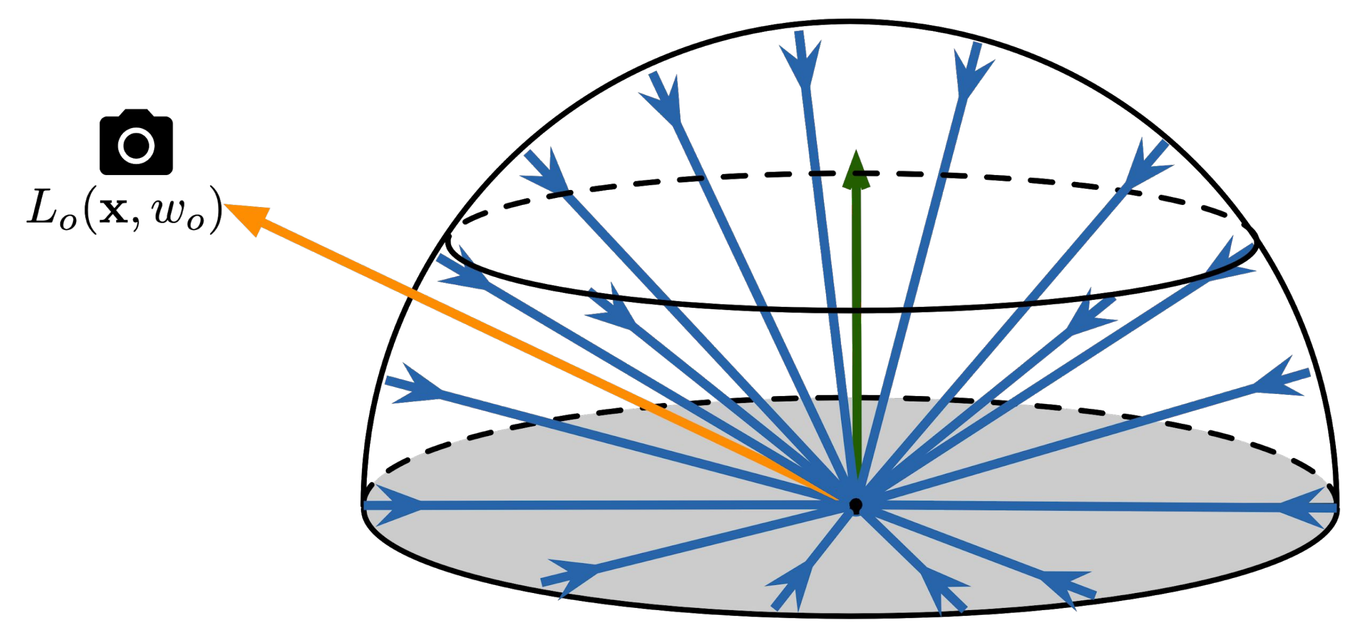
The Rendering Equation

The outgoing radiance leaving a point is given as the sum of emitted plus reflected radiance

$$L_o(\mathbf{x}, w_o) = L_e(\mathbf{x}, w_o) + L_r(\mathbf{x}, w_o) = L_e(\mathbf{x}, w_o) + \int_\Omega f_r(\mathbf{x}, w_i, w_o) L_i(\mathbf{x}, w_i) \cos\theta_i \mathrm{d}w_i$$

Outgoing Radiance | ***Emitted Radiance*** | ***Reflected Radiance*** | *Emitted Radiance* | *Unit Hemisphere* | *BRDF* | *Incoming Radiance* | *Cosine of Incident Angle*





* We always assume $\boldsymbol{w_i}$ towards from $\boldsymbol{x}$ to unit hemisphere surface even for incoming lights

James T. Kajiya. 1986. The rendering equation. In Proceedings of the 13th annual conference on Computer graphics and interactive techniques (SIGGRAPH '86). Association for Computing Machinery, New York, NY, USA, 143–150. DOI:https://doi.org/10.1145/15922.15902

How to solve the Rendering Equation?

The emitted radiance and BRDF can be defined by the material, thus the outgoing radiance depends on the incoming radiance and the incident angle:

$$L_o(\mathbf{x}, w_o) = L_e(\mathbf{x}, w_o) + L_r(\mathbf{x}, w_o) = L_e(\mathbf{x}, w_o) + \int_\Omega f_r(\mathbf{x}, w_i, w_o) L_i(\mathbf{x}, w_i) \cos\theta_i \mathrm{d}w_i$$

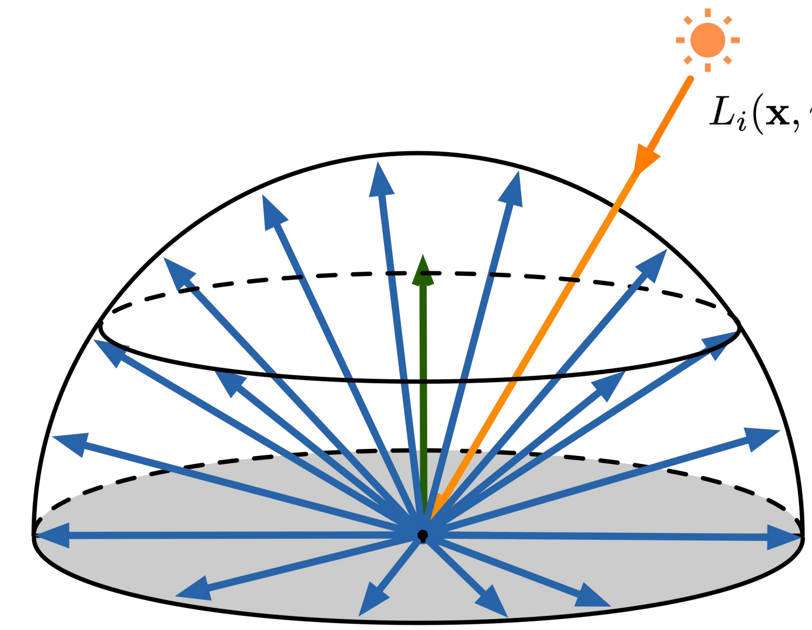
But the incoming radiance is another outgoing radiance

So the rendering equation is ***recursive***! Then how can we solve it? Not now.

Maybe before solving the equation, let's check an easy case to gain some understanding...

Task 3 a) Lambertian Material

Incoming light is equally reflected in each output direction



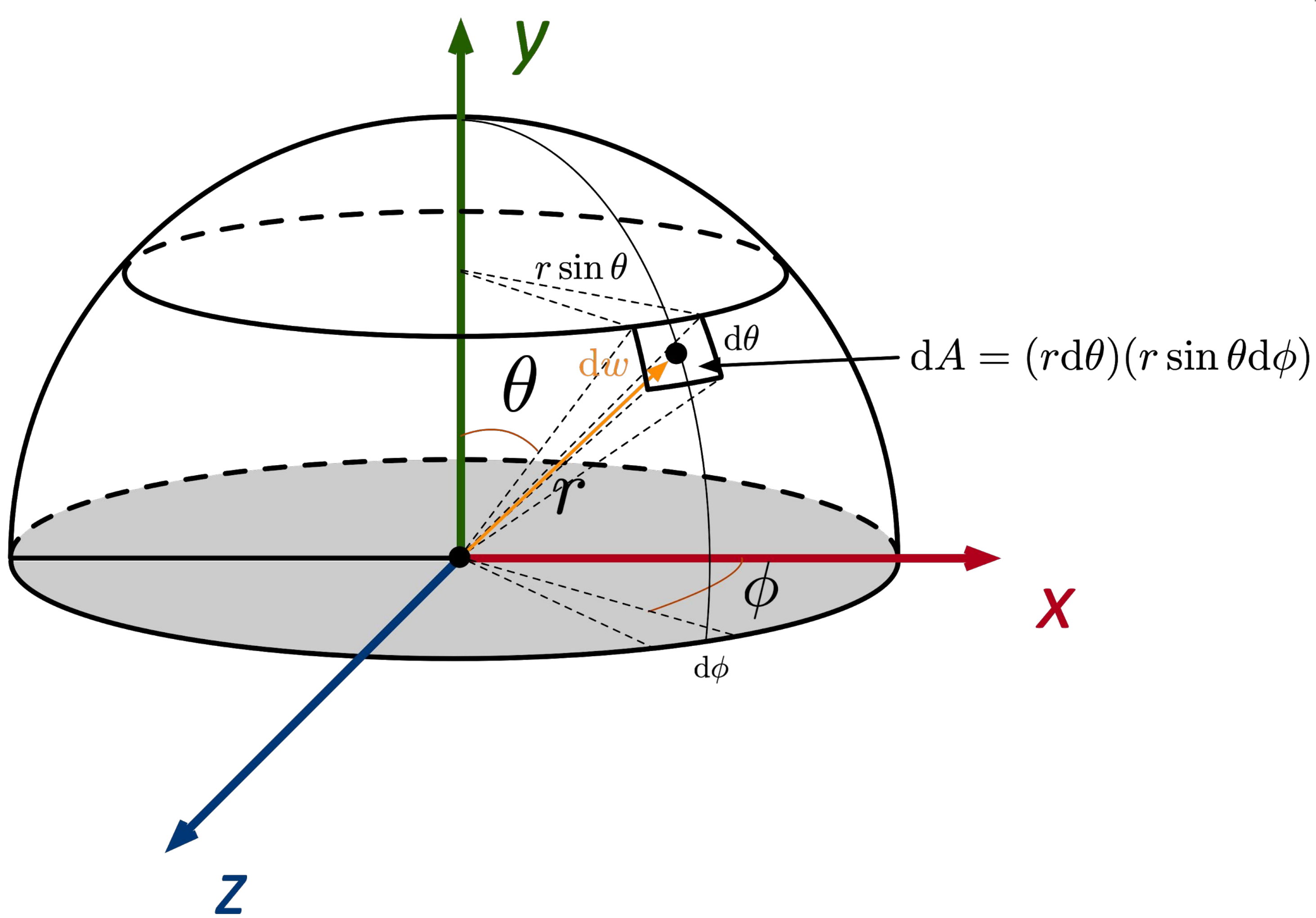
⇒ *The radiance received by the camera is irrelevant to the position*

⇒ The *BRDF must be a constant*

⇒ The incoming radiance is also a constant

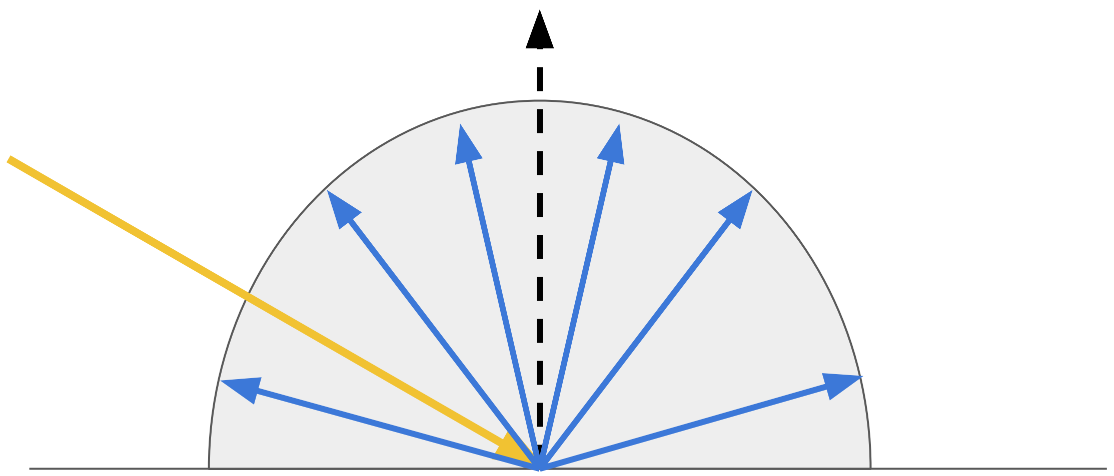
$$L_o(w_o) = \int_\Omega f_r L_i(w_i) \cos\theta_i \mathrm{d}w_i$$
$$= f_r L_i(w_i) \int_\Omega \cos\theta_i \mathrm{d}w_i$$

Task 3 a) Lambertian Material (cont.)



$$
\begin{aligned}
\int_{\Omega} \cos\theta_i \mathrm{d}w_i &= \int_{\theta_i=0}^{\pi/2} \int_{\phi=0}^{2\pi} \cos\theta_i(\mathrm{d}\theta_i)(\sin\theta_i\mathrm{d}\phi) \\
&= \int_0^{\pi/2} \sin\theta_i \cos\theta_i \mathrm{d}\theta_i \int_0^{2\pi} \mathrm{d}\phi \\
&= \int_0^{\pi/2} \frac{\sin 2\theta_i}{2} \mathrm{d}\theta_i \int_0^{2\pi} \mathrm{d}\phi \\
&= \frac{1}{4} \int_0^{\pi} \sin 2\theta_i \mathrm{d}(2\theta_i) \int_0^{2\pi} \mathrm{d}\phi \\
&= \frac{1}{4}(-\cos(2\pi) - (-\cos(0)))(2\pi - 0) \\
&= \frac{1}{4}(1+1)(2\pi) = \pi
\end{aligned}
$$

Task 3 a) Lambertian Material (cont. 2)



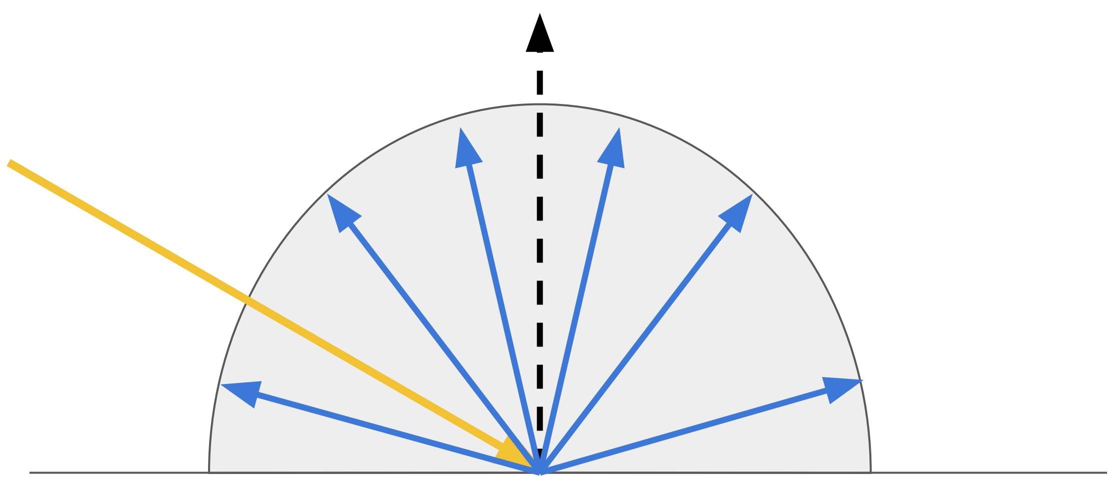
$$\int_\Omega \cos\theta_i \mathrm{d}w_i = \pi$$

$$\implies L_o(w_o) = f_r L_i(w_i)\pi$$

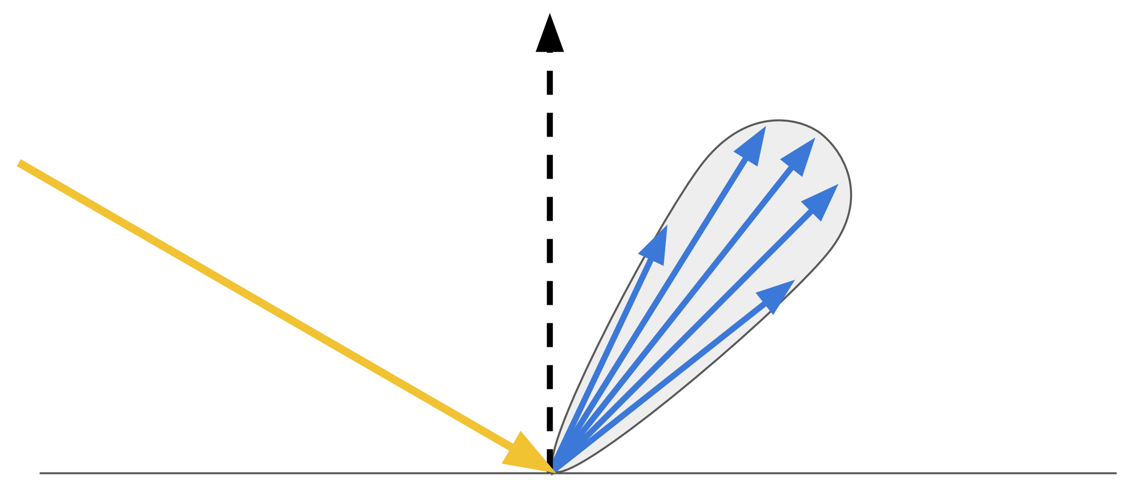
Due to the *conversation of energy*, incoming radiance is equal to the outgoing radiance, thus:

$$f_r = \frac{1}{\pi}$$

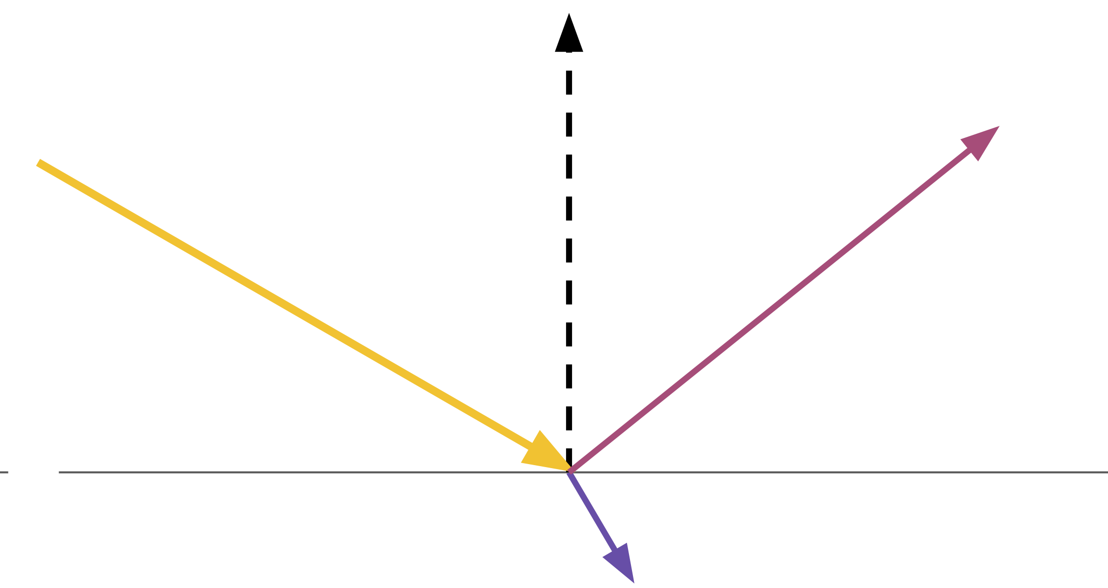
More Materials



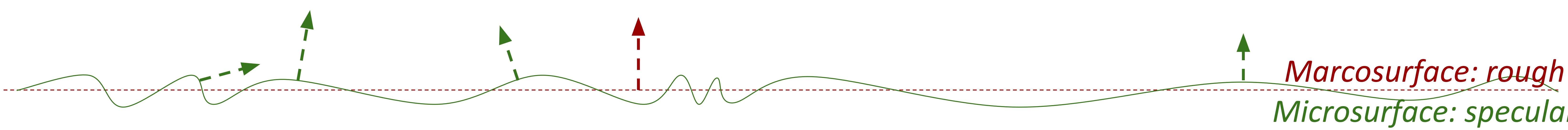
Lambertian /
***Diffuse** Material*



***Glossy** Material*



***Reflective / Refractive** Material*
*(**Specular**)*



Microfacet Material: distribution of normals change depending on the viewer

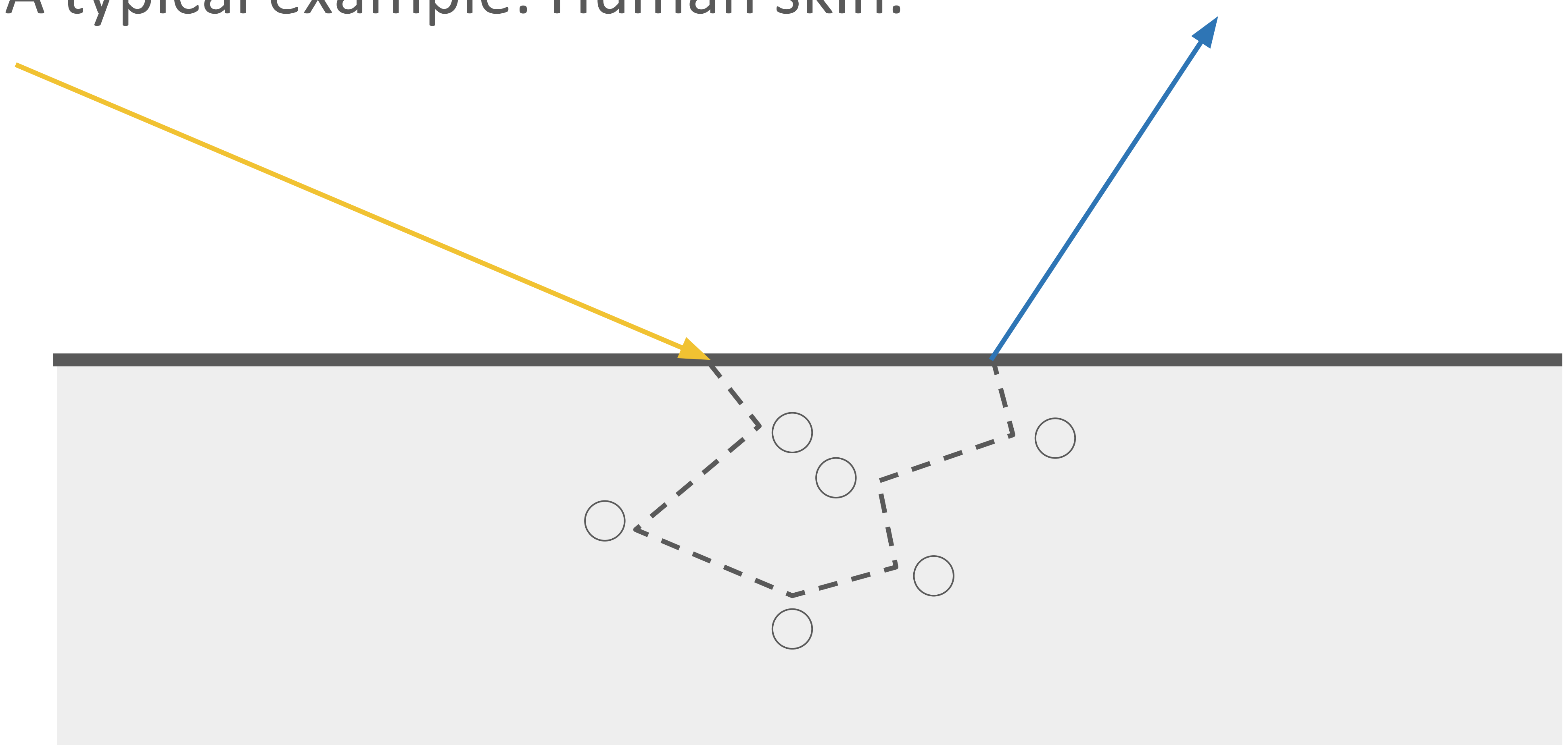
Task 3 b) and c) Subsurface Scattering

The rendering equation assumption: outgoing point doesn't change from incident point

⇒ Without the consideration of light transportation under the surface!

With the rendering equation, light won't go through an object!

A typical example: Human skin.



https://en.wikipedia.org/wiki/Subsurface_scattering#/media/File:Skin_Subsurface_Scattering.jpg

BSSRDF and A Generalized Rendering Equation

B **S**ubsurface **S**cattering **RDF** (BSSRDF): $S(x_i, w_i, x_o, w_o)$

With BSSRDF, a generalized rendering equation can be written as:

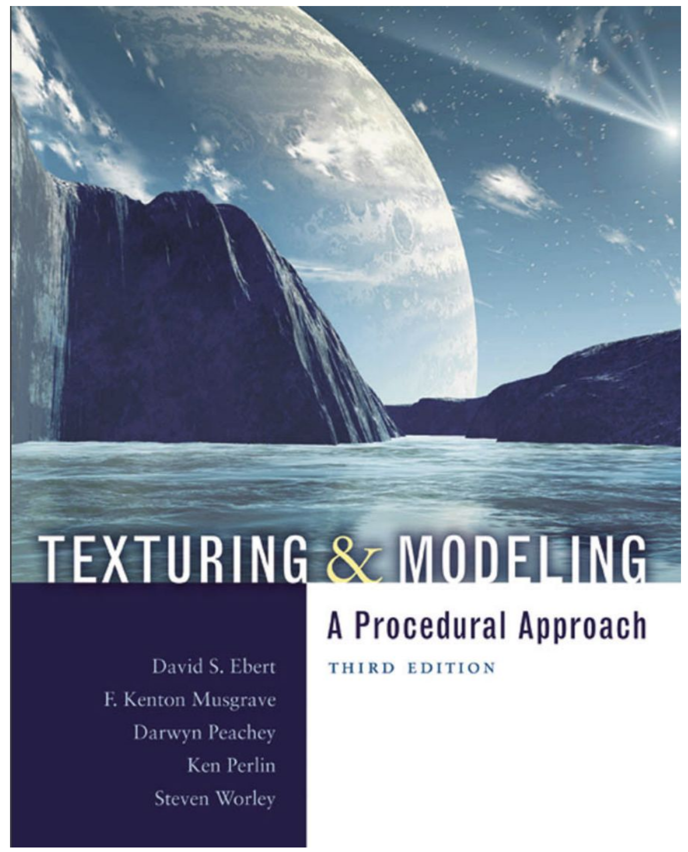
$$L(x_o, w_o) = \int_A \int_\Omega S(x_i, w_i, x_o, w_o) L_i(x_i, w_i) \cos\theta_i \mathrm{d}w_i \mathrm{d}A$$

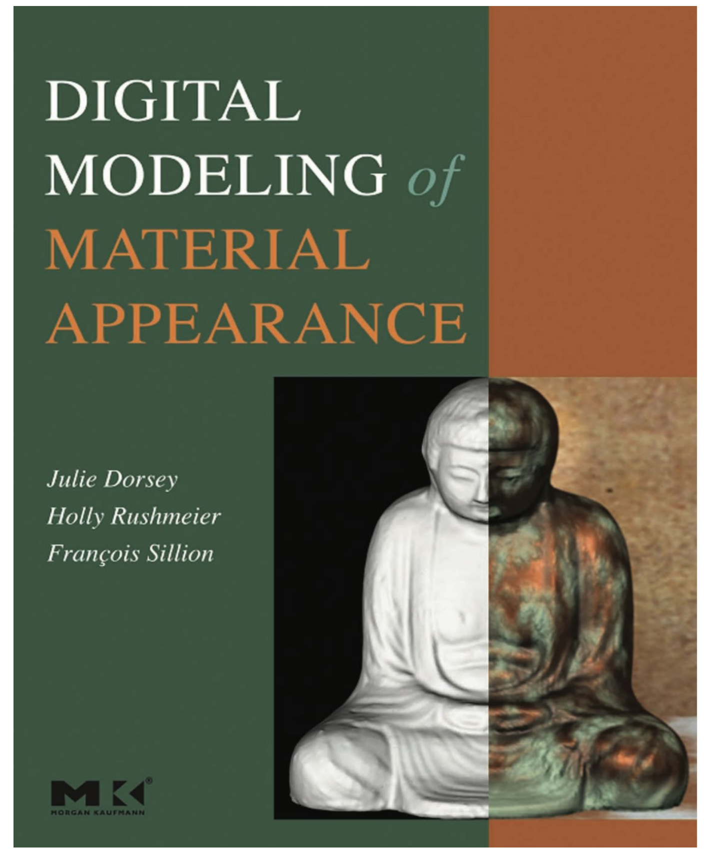
Sample Surface

Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. 2001. A practical model for subsurface light transport. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH '01). Association for Computing Machinery, New York, NY, USA, 511–518. DOI:https://doi.org/10.1145/383259.383319

Take Away

- "How to compute the 'correct' color of a given pixel" is the key question in rendering
- Interpolation and sampling play the key role in appearance modeling
- Texture mapping is old stuff but still good for faking visual appearance in real-time
- **The** *rendering equation* is the **foundation** of (modern) computer graphics rendering
- Check these books for more good old stuff and brilliant new ideas:





Thanks!
What are your questions?

Appendix

Can you tell what are the techs applied in this picture?

Image source: Red Dead Redemption 2