WebGPU

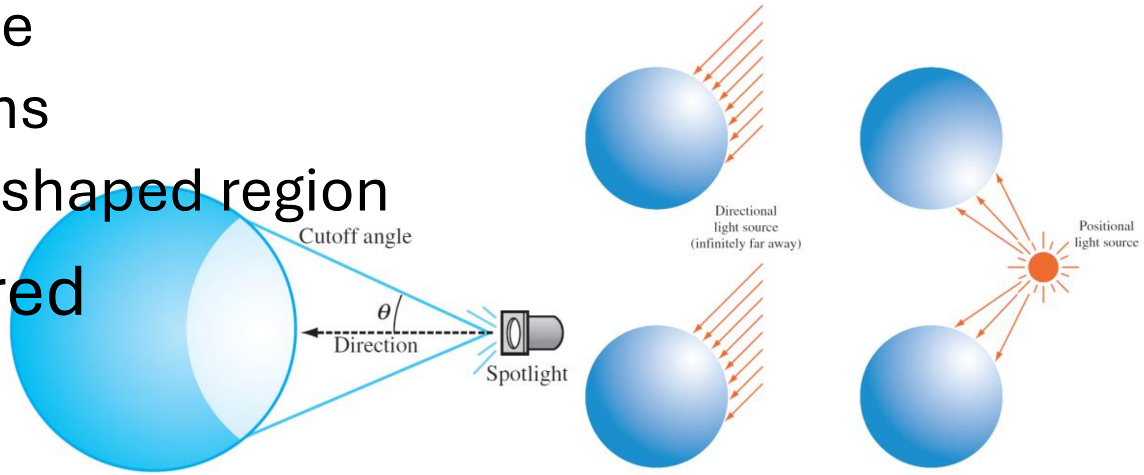
03. Lighting

What to Learn

- Shading, shadows, and different types of light sources including point, directional, and ambient
- Reflection of light in the 3D scene and the two main types: diffuse and ambient
- The details of shading and how to implement the effect of light to make objects look three-dimensional

Types of Light Sources

- No light types are defined by WebGPU
  – WebGPU doesn’t even know anything about lighting
- Classic light types
  - Ambient light – a good approximation to the scattered light present in a scene
  - Directional light – light source located infinitely far away
    → constant direction, easier to compute
  - Point light – light radiated to all directions
  - Spot light – light radiated within a cone-shaped region
- Attenuation may need to be considered



(“Advanced Graphics Programming using OpenGL”)

Orientation of Surfaces

- In lighting (shading) computation, “the orientation of the surface” plays a crucial role. → How to define it?
- Surface orientation
  - Default: count clock-wise order defines the front face
  - set by [`primitive.frontFace`]() of [`GPUDevice:createRenderPipeline()`]().
- In shading computation, the orientation of the surface is defined by the **normal vectors** defined at each vertex.
  - Is it defined once the orientation of the triangle defined?
  - The normal is different for each triangle sharing the same vertex. Can we still share the normal as the vertex attribute?
- **In fact, we can assign the normal to each vertex regardless of the shapes of the surrounding triangles!**

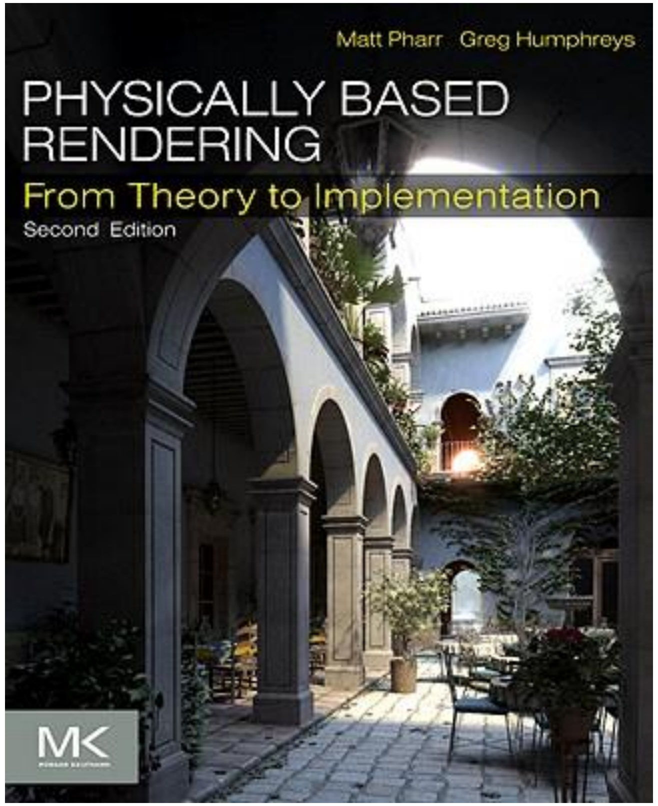
Lighting Coordinate Systems

- Typically computed in the eye/camera space
  - The eye is located at $(0,0,0)$ looking $-z$ direction
- If we assume that the viewer is located at infinity (V is constant), and if a directional light source is used (L is constant), the light computation gets even simpler (especially for Blinn-Phong model)

Shading Models

Light

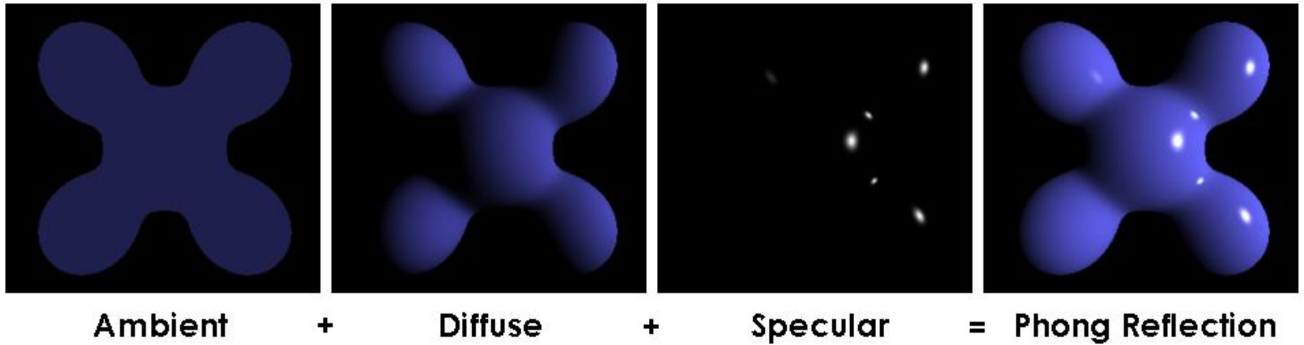
- Behaviors – reflection, refraction, transmission, absorption, etc.
- Too complicated to simulate all, especially in real-time
- Simplified “models” are proposed for real-time performance with reasonable quality
- No specific lighting (shading) model in WebGPU → You’re free to implement any in the shaders. (That was the original purpose of the “shaders.”)



Phong Reflection Model

(https://alchetron.com)

- Proposed by Bui Tuong Phong
  - "Illumination for computer-generated images" (1973, Ph.D dissertation)
  - "Illumination for computer-generated pictures" (1975 paper)
- The most popular real-time shading model
- Three types of reflections – ambient, diffusive, specular
- Local illumination – No interaction with other objects
  → suitable for parallel processing
- https://en.wikipedia.org/wiki/Phong_reflection_model

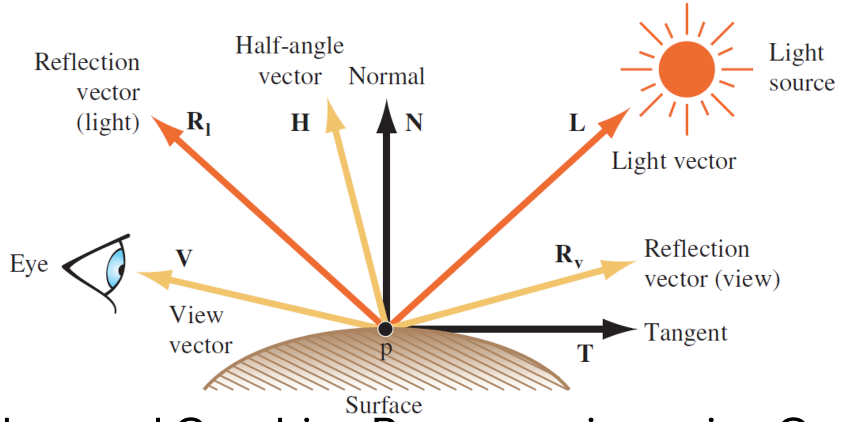


(Wikipedia)

Phong Reflection Model (cont’d)

- For simplification, light intensity is decomposed into three types
  - **ambient, diffusive, specular**
- Material property denotes the **reflected ratio of incoming light intensity** for each type
- Three types of reflections are computed **independently**
- Illumination for each channel (color) is computed **independently**
- At which stage do we compute? (quality vs. performance)
  - Per-vertex → Output color is interpolated for each fragment (Gouraud shading) → lower quality, better performance
  - Per-fragment → Interpolated normal are used in fragment (Phong shading) → Better quality, lower performance

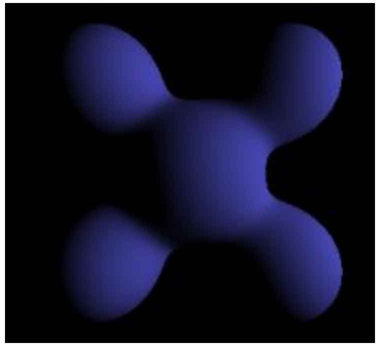
Ambient Reflection

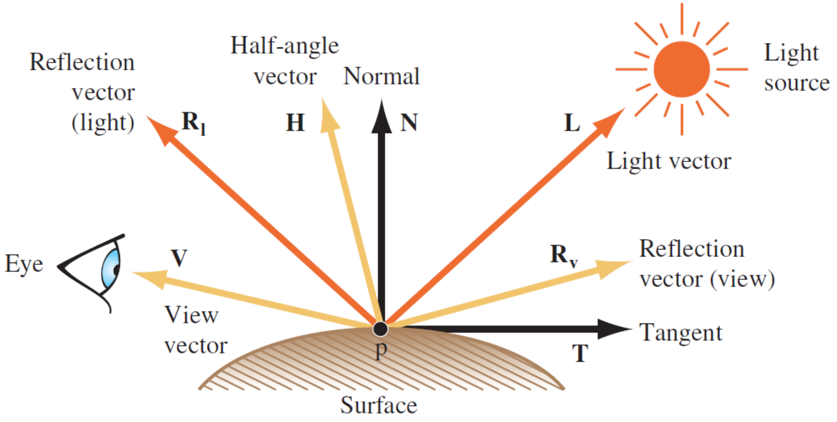


("Advanced Graphics Programming using OpenGL")

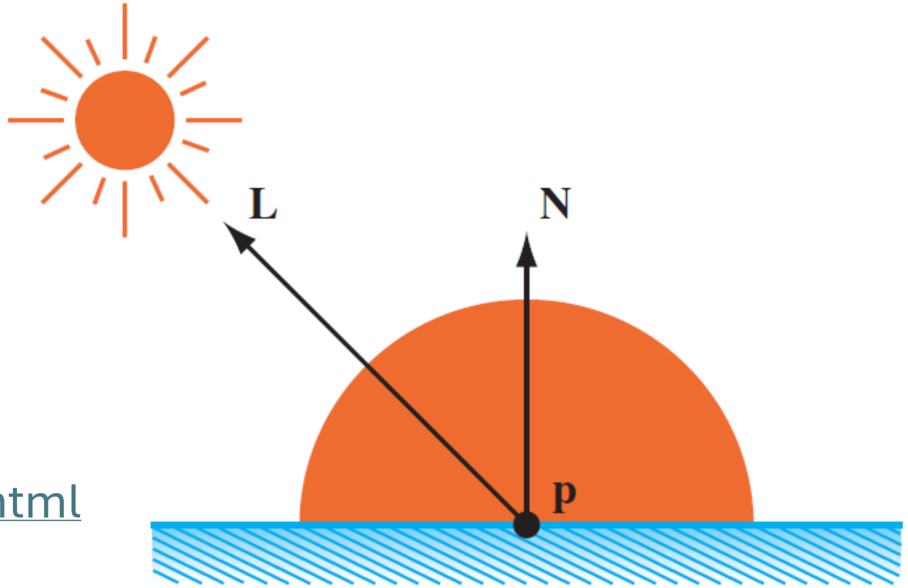
- Ambient light
  - Weak incoming light after (infinitely) large number of scattering in the scene
  - Approximated as **constant light** (1) incoming from all directions, (2) with the same intensity, and (2) distributed evenly in the whole scene
- Approximated as **diffusive** reflection
- Coming equally from all directions, distributed evenly → independent of the light position
- Reflected equally to all directions (diffusive) → independent of the viewer position
- Keeps the parts not lighted directly from being completely black
- Formula: $I_a = k_a \, i_a$
  - I_a: ambient illumination
  - k_a: ambient reflection constant of the material (**material** property, reflected ratio of the incoming light intensity)
  - i_a: incoming ambient light intensity (**light** property)
- Reference
  - https://en.wikipedia.org/wiki/Shading#Ambient_lighting
  - https://paroj.github.io/gltut/Illumination/Tut09%20Global%20Illumination.html

Diffusive Reflection



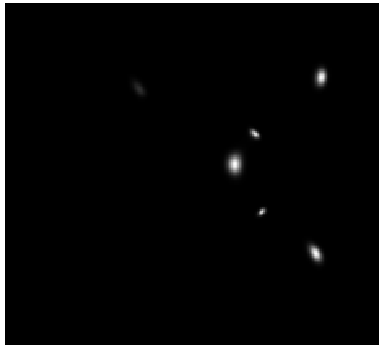


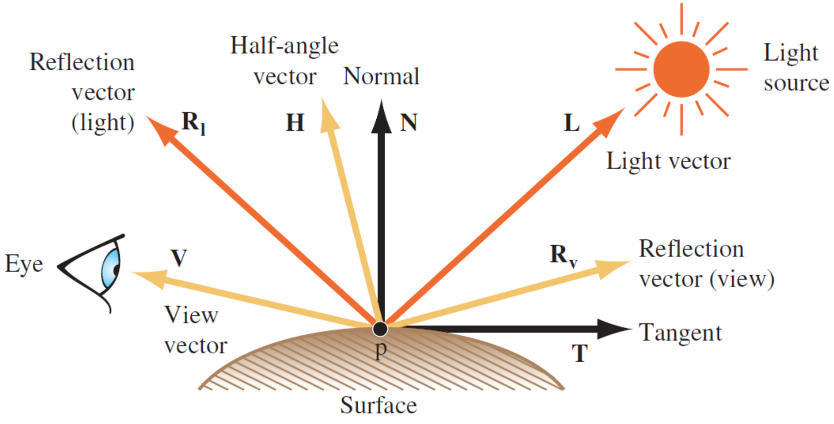
- Light reflected to many directions
  → approximated to be reflected to all (front) directions with equal amount
- Incoming light intensity is dependent on the incident angle (Lambertian reflection)
  - cosine can be replaced by the dot product for unit vectors.
  - Vectors can be normalized using the `normalize()` function in the shaders.
- Light-position-dependent & viewer-position-independent
- Formula: $I_d = k_d(L \cdot N)i_d$
  - I_d: diffusive illumination
  - k_d: diffusive reflection constant of the material
  - L: (normalized) direction to light source
  - N: (normalized) normal at the surface point p
  - i_d: incoming diffusive light intensity
- Reference
  - https://en.wikipedia.org/wiki/Diffuse_reflection
  - https://paroj.github.io/gltut/Illumination/Tutorial%2009.html



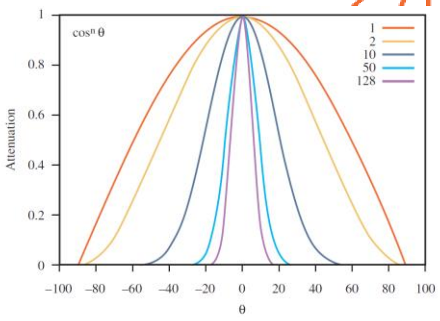
("Advanced Graphics Programming using OpenGL")

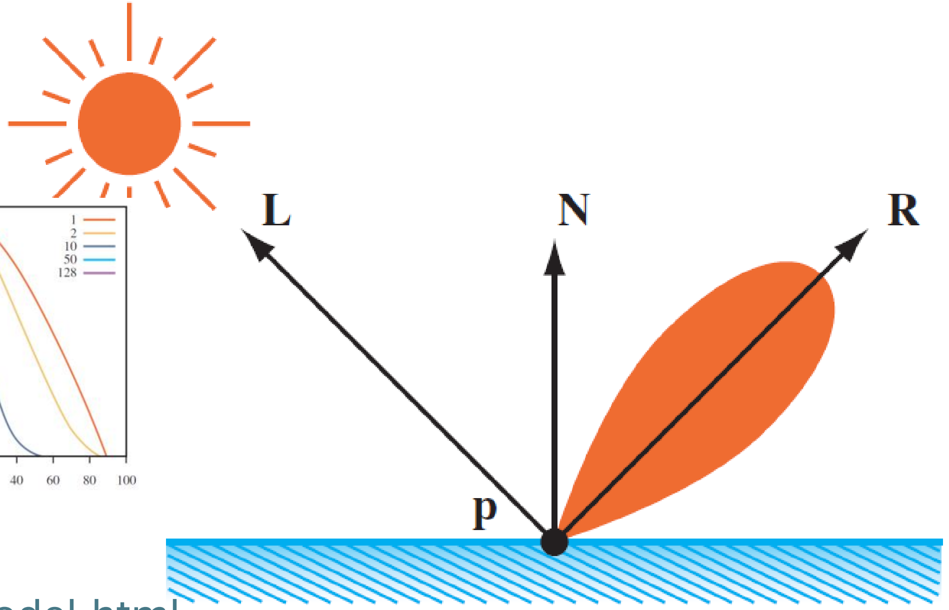
Specular Reflection





- Mirror-like reflection concentrated to the reflected light direction
- Reflected pattern is (heuristically) modeled by a cosine function, based on no physical model
- "Shininess" (α) determines how "narrowly" reflected and modeled by the power of cosine function
- Light-position-dependent & viewer-position-dependent
- Makes objects look "shiny"
- Formula: $I_s = k_s(R_l \cdot V)^{\alpha} i_s$
    - I_s: specular illumination
    - k_s: specular reflection constant of the material
    - R_l: reflected light direction ($= 2(L \cdot N)N - L$)
    - V: direction to the viewer
    - α: shininess
    - i_s: incoming specular light intensity
- Reference
    - https://en.wikipedia.org/wiki/Specular_reflection
    - https://paroj.github.io/gltut/Illumination/Tut11%20Phong%20Model.html





("Advanced Graphics Programming using OpenGL")

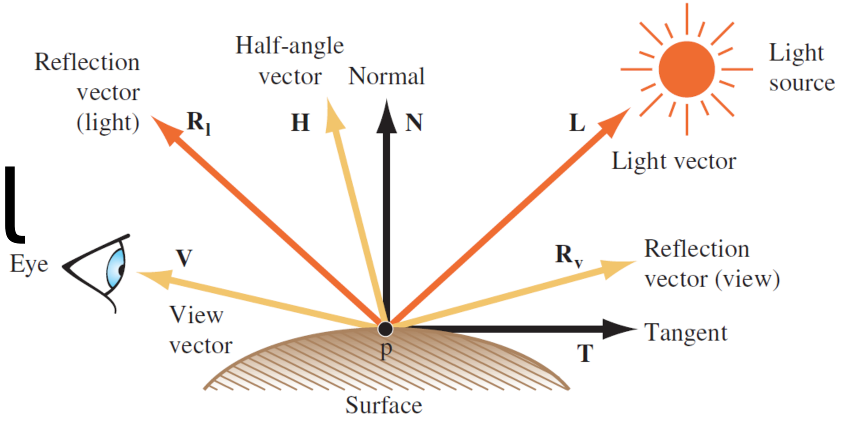
Phong Reflection Model: Wrap-Up

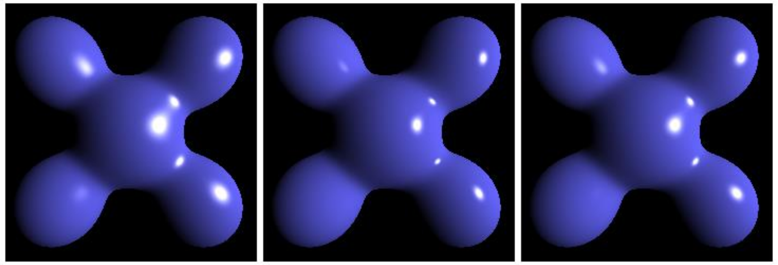
- Final formula

$$I(p) = k_a i_a + \sum_{m \in \text{lights}} (k_d (L^m \cdot N) i_d^m + k_s (R_l^m \cdot V)^\alpha i_s^m)$$

- Ambient light intensities are summed up to i_a
- Diffusive and specular illuminations are computed for each light source
- Negative dot products need to be clamped to 0 (Why?)

Blinn-Phong Reflection Model





Blinn–Phong | Phong | Blinn–Phong (higher exponent)

(Wikipedia)

- Proposed by Jim Blinn (1977)
- Faster computation if both L and V are at infinity (directional light + orthographic projection)
- Very similar result with the Phong reflection model
- Marginally more realistic than Phong reflection model
- To reduce the overhead of computing the reflected light vector (R_l) since N varies for each P:
$$R_l = 2(L \cdot N)N - L$$
- $N \cdot H$ and $R_l \cdot V$ act similarly ($H \coloneqq {}^{L+V}/_{\|L+V\|}$ is the half-way vector), but L and V thus H is constant if assumed to be located at infinity
- Difference can be minimized by using different α
→ It's a heuristic model anyway...
- Reference
    - https://en.wikipedia.org/wiki/Blinn-Phong_shading_model
    - https://paroj.github.io/gltut/Illumination/Tut11%20BlinnPhong%20Model.html

Interpolation Models

- Gouraud interpolation
  - "Continuous Shading of Curved Surfaces" (1971)
  - Per-vertex shading
  - Illumination color is interpolated and assigned to each fragment
  - Faster but lower quality
  - Phong reflection model + Gouraud interpolation was the default shading model for classic OpenGL
- Phong interpolation
  - Per-fragment shading
  - Per-vertex position & normal need to be passed from the vert shader
  - Slower but higher quality

Two-Sided Lighting

- Different materials can be assigned to front & back faces respectively
- The built-in value `front_facing` can be used in the FS to determine the orientation

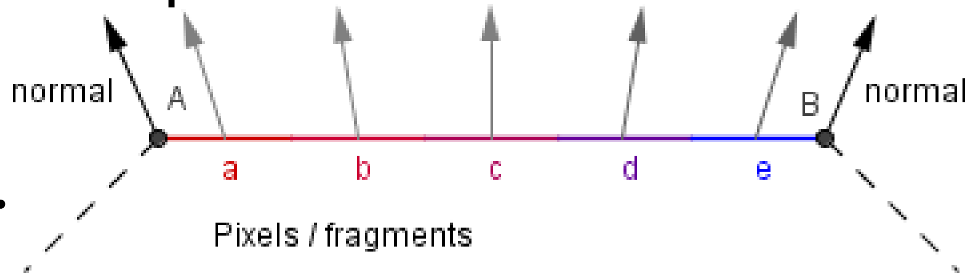
Directional Lighting

Directional Lighting

- Assumes the light is coming uniformly from one direction.
  - e.g., “sun”
- Lambert’s cosine law: “…the observed radiant intensity or luminous intensity… is directly proportional to the cosine of the angle between the observer’s line of sight and the surface normal.” (Wikipedia)

webgpu-lighting-directional

- https://webgpufundamentals.org/webgpu/webgpu-lighting-directional.html
- One directional light source
- A normal attribute is assigned to each vertex
- The normal vector (inter-stage var) should be "normalized" since its length might get shorter than 1.
- Light computation in FS → "Phong interpolation"
- Diffusive reflection only
- Normals & light direction are fixed.

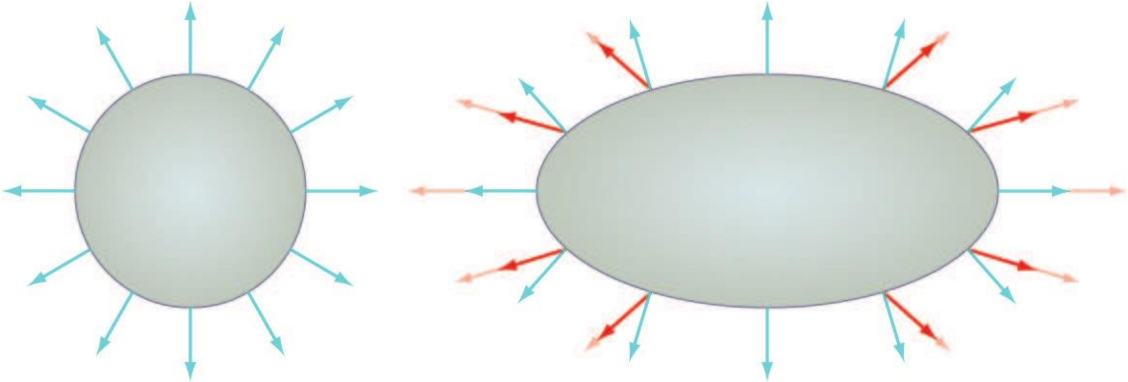


(https://cglearn.eu/pub/computer-graphics)

webgpu-lighting-directional-world

- https://webgpufundamentals.org/webgpu/webgpu-lighting-directional-world.html
- Light computation in world space.
  - The model matrix (“world matrix”), which transforms from the model space to the world space, is passed as a uniform to transform the normal vector.
  - The “w” component of the normal vector should be “0”.
    → not affected by any translation

Normal Matrix



- If a model is transformed, How can we transform its normals? Can we apply the same transformation?
- “Normal matrix” needs to be applied to normals
- The normal matrix (N) is the same as the MV matrix (M) for “orthogonal transformation”, but they are not the same otherwise. (e.g., scaling)
- $N = M^{-T}$ (inverse transpose of the submatrix of M)
  - for world coord system ($N = (MV)^{-T}$ for eye coord system)
  - $N = M$ if M is orthogonal.
- References
  - https://webgpufundamentals.org/webgpu/lessons/resources/normals-scaled.html
  - http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/the-normal-matrix
- Example: https://xregy.github.io/webgl/src/normal_matrix.html

webgpu-lighting-directional-worldinversetranspose

- https://webgpufundamentals.org/webgpu/webgpu-lighting-directional-worldinversetranspose.html
- A 3x3 “normal matrix” is passed to transform normals.

Point Lighting

Point Lighting

- Light radiated to all directions
- https://webgpufundamentals.org/webgpu/lessons/resources/point-lighting.html

webgpu-lighting-point

- https://webgpufundamentals.org/webgpu/webgpu-lighting-point.html
- vertex-to-light direction is computed in the VS
  → needs to be normalized in the FS

Specular Highlighting

- https://webgpufundamentals.org/webgpu/lessons/resources/specular-lighting.html
- https://webgpufundamentals.org/webgpu/lessons/resources/specular-lighting.html

webgpu-lighting-point-w-specular

- https://webgpufundamentals.org/webgpu/webgpu-lighting-point-w-specular.html
- Blinn-Phong reflection model

webgpu-lighting-point-w-specular-power

- https://webgpufundamentals.org/webgpu/webgpu-lighting-point-w-specular-power.html
- “shininess” to control the range of highlight

Spot Lighting

Spot Lighting

- https://webgpufundamentals.org/webgpu/lessons/resources/spot-lighting.html

webgpu-lighting-spot

- https://webgpufundamentals.org/webgpu/lessons/webgpu-lighting-spot.html

webgpu-lighting-spot-w-linear-falloff

- https://webgpufundamentals.org/webgpu/webgpu-lighting-spot-w-linear-falloff.html
- inner & outer limits

webgpu-lighting-spot-w-smoothstep-falloff

- https://webgpufundamentals.org/webgpu/webgpu-lighting-spot-w-smoothstep-falloff.html