Computer Graphics

Minho Kim

Dept. of Computer Science

University of Seoul

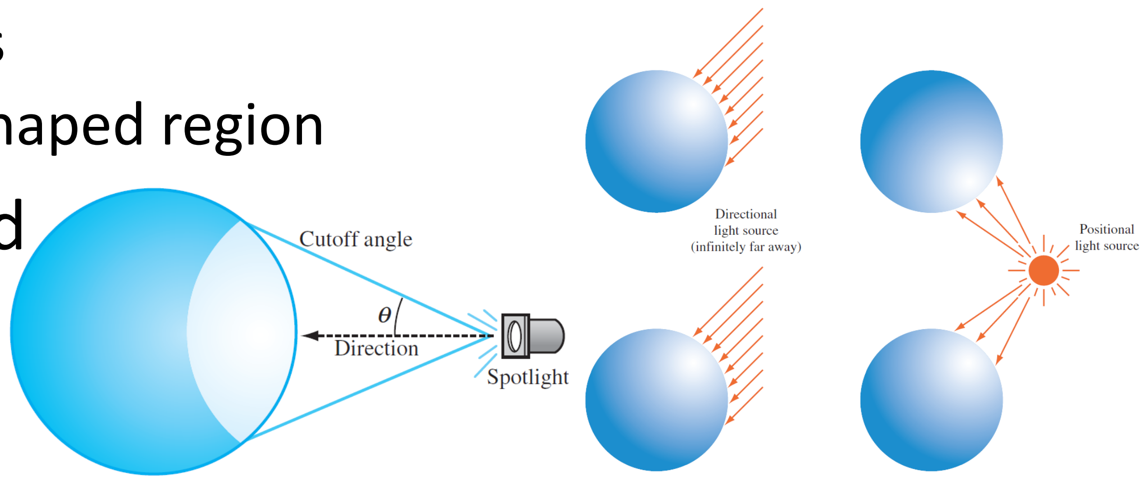
Chapter 8: Lighting Objects

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, such as the pure white cube in the previous chapter, look three-dimensional

Types of Light Sources

- No light types are defined by OpenGL
  – OpenGL 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
  - Can be set by `gl.frontFace()` with `gl.CW` or `gl.CCW`
- 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!**

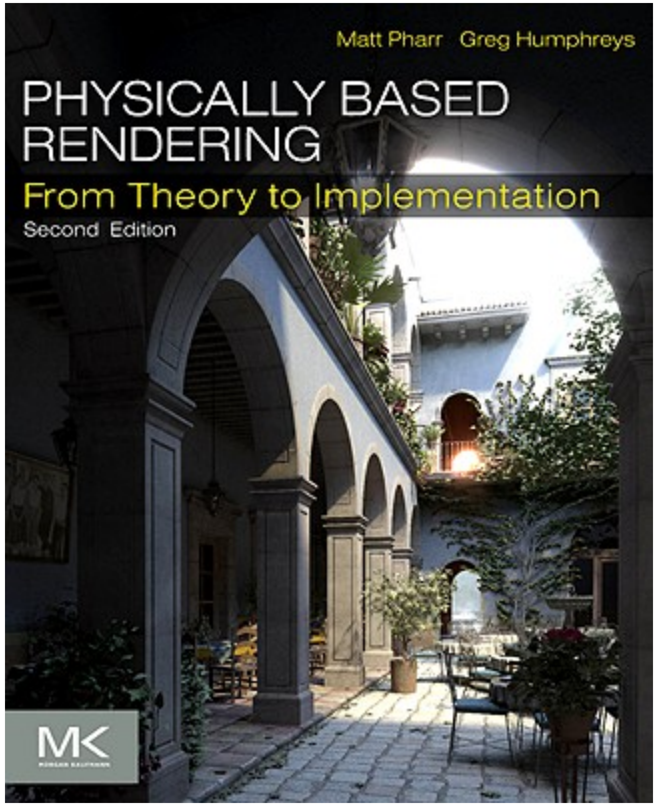
Lighting Coordinate Systems

- Typically computed in the eye 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

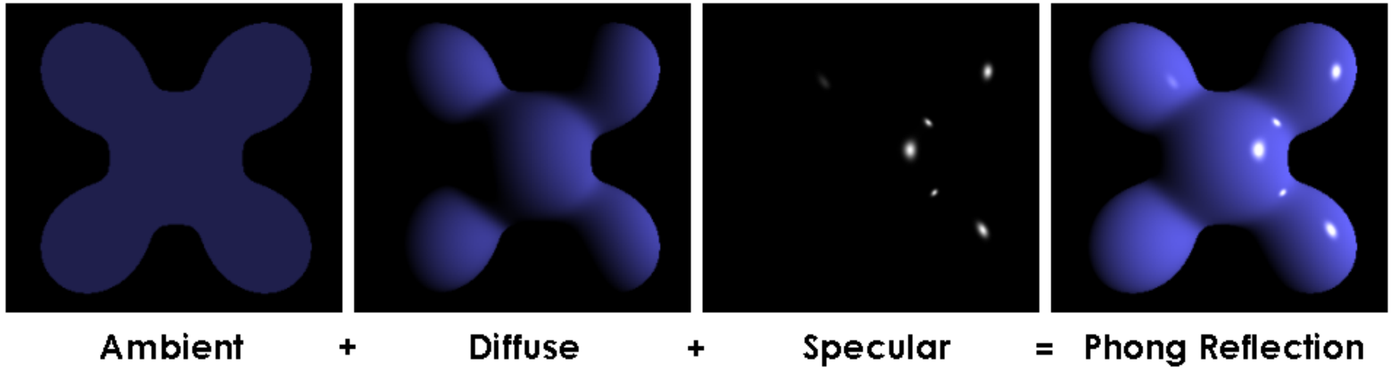
Light

- 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 OpenGL
  → You're free to implement any in the shaders. (That was the original purpose of the "shaders.")



Phong Reflection Model

- Proposed by Bui Tuong Phong (1973)
- 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

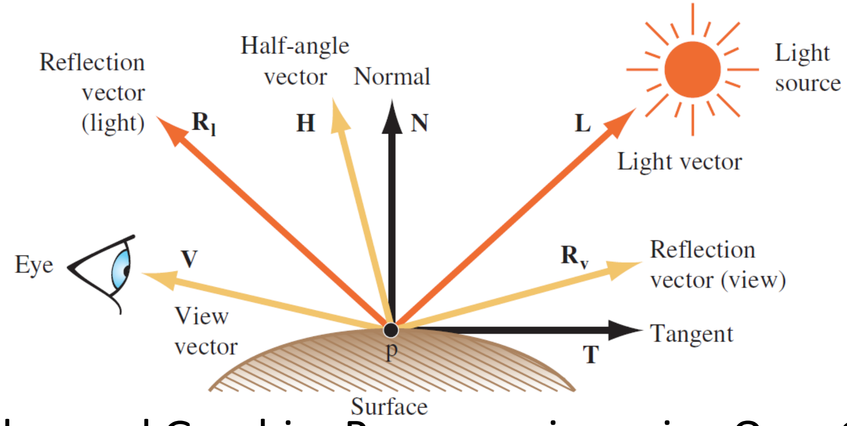


(Wikipedia)

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

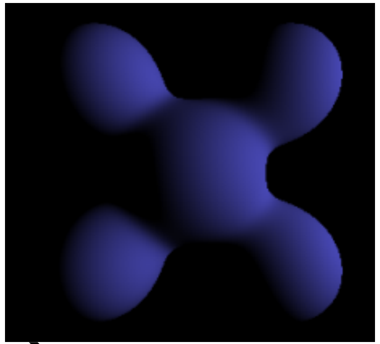
Ambient Reflection

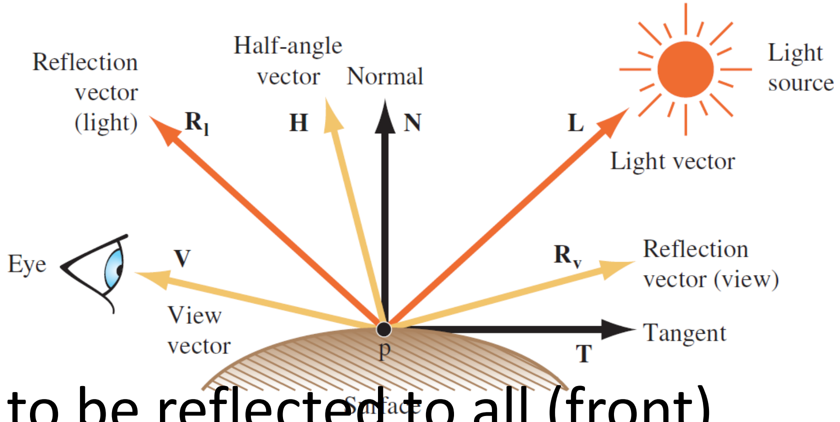


("Advanced Graphics Programming using OpenGL")

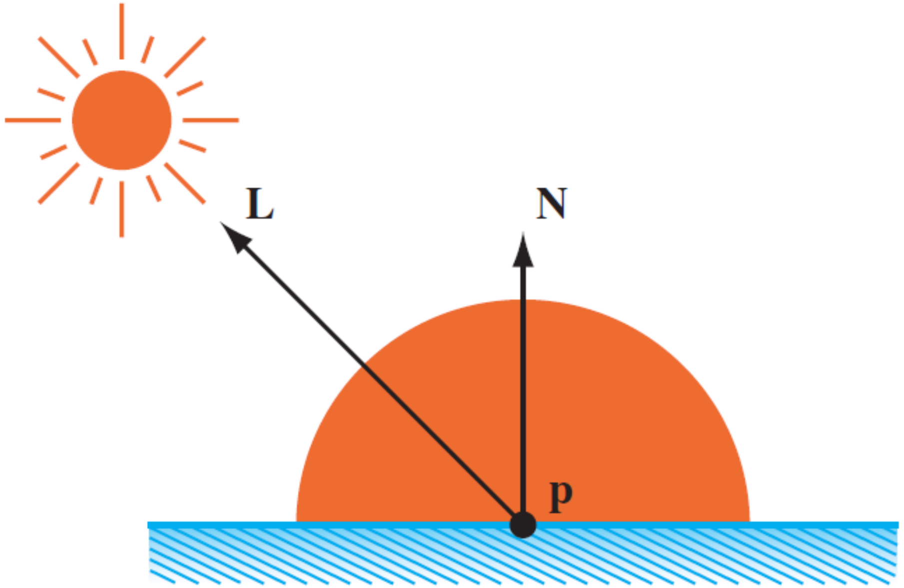
- Ambient light
  - Weak incoming light after (infinitely) large number of scattering in the scene
  - Approximated as **constant light** (1) incoming from all directions, (2) with the same intensity, and (2) distributed evenly in the whole scene
- Approximated as **diffusive** reflection
- Coming equally 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)

Diffusive Reflection



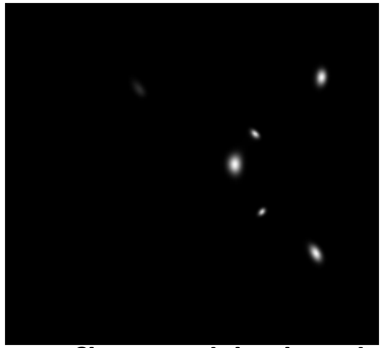


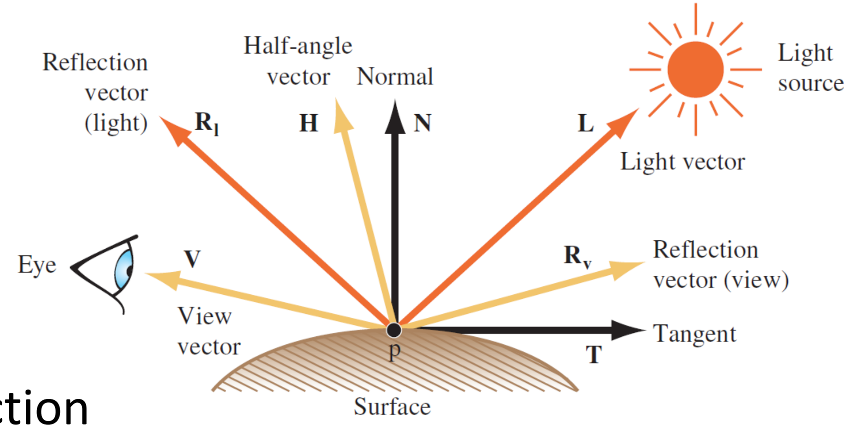
- Light reflected to many directions → approximated to be reflected to all (front) directions with equal amount
- Incoming light intensity is dependent on the incident angle ([Lambertian reflection]())
  - cosine can be replaced by the dot product for unit vectors.
  - Vectors can be normalized using the `normalize()` function in the shaders.
- Light-position-dependent & viewer-position-independent
- https://en.wikipedia.org/wiki/Diffuse_reflection
- 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



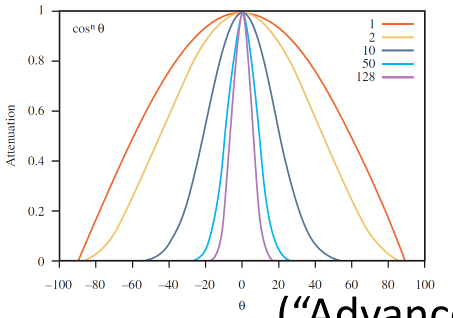
("Advanced Graphics Programming using OpenGL")

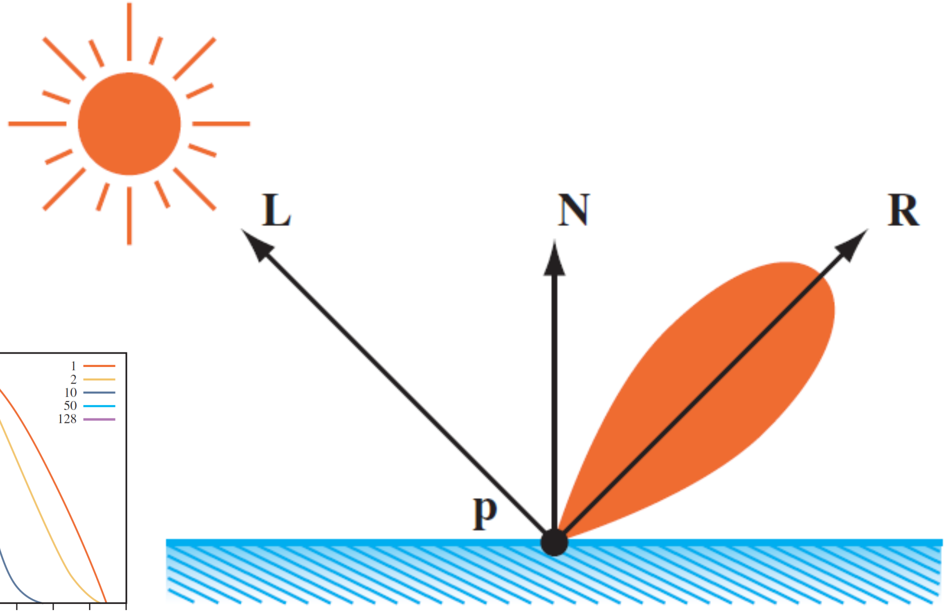
Specular Reflection





- 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”
- https://en.wikipedia.org/wiki/Specular_reflection
- 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





(“Advanced Graphics Programming using OpenGL”)

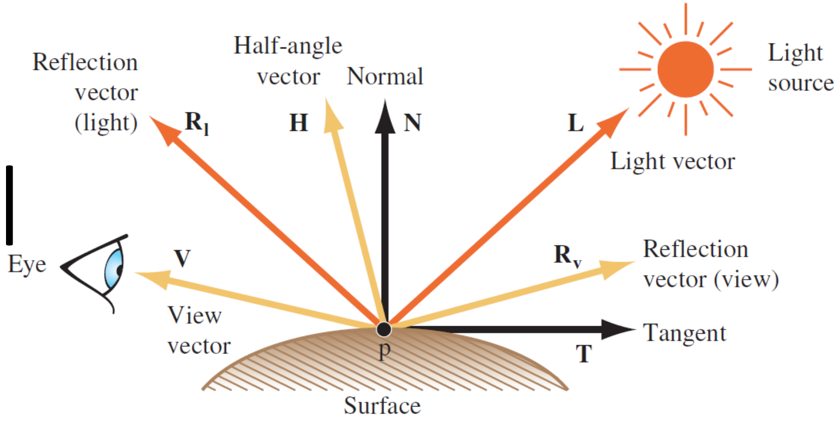
Phong Reflection Model: Wrap-Up

- Final formula

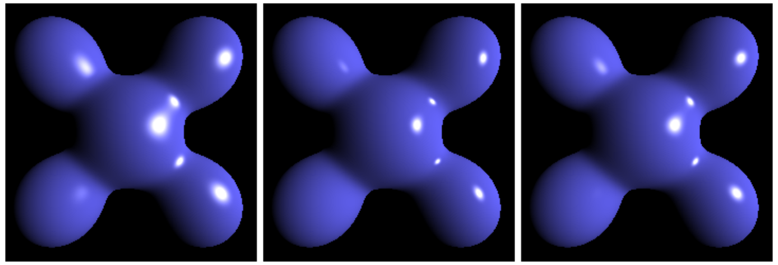
$$I(p) = k_a i_a + \sum_{m \in \text{lights}} \left(k_d (L^m \cdot N) i_d^m + k_s (R_l^m \cdot V)^\alpha i_s^m\right)$$

- 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



- Proposed by Jim Blinn (1977)
- Faster computation if both L and V are at infinity
- 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 ($H := {}^{L+V}/_{\|L+V\|}$ is the half-way vector), but L and V are constant if assumed to be located at infinity
- Difference can be minimized by using different α
→ It's an heuristic model anyway…
- https://en.wikipedia.org/wiki/Blinn-Phong_shading_model



Blinn–Phong **Phong** **Blinn–Phong** (higher exponent)

(Wikipedia)

Interpolation Models

- Gouraud interpolation (Gouraud shading)
  - 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 (Phong shading)
  - Per-fragment shading
  - Per-vertex normal is interpolated and each fragment is shaded
  - Slower but higher quality

Two-Sided Lighting

- Different materials can be assigned to front & back faces respectively
- `gl_FrontFacing` can be used to determine the orientation

Example #1: `LightedCube`

Example #1: `LightedCube`

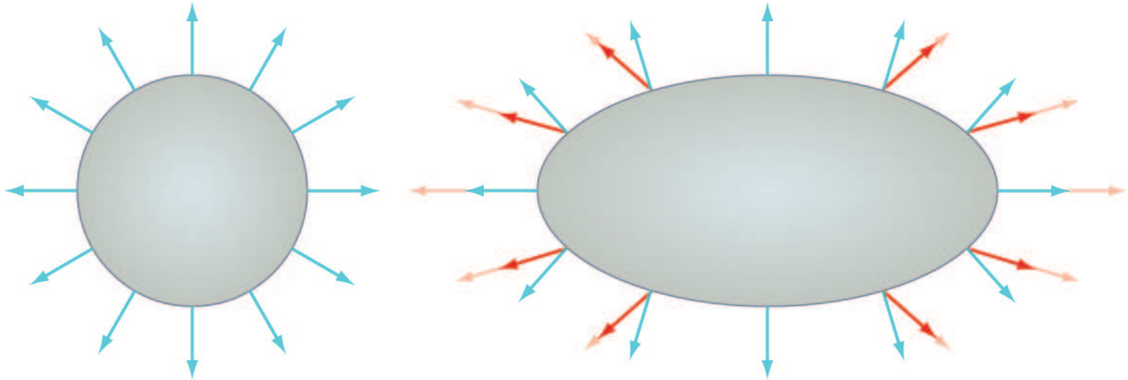
- http://rodger.global-linguist.com/webgl/ch08/LightedCube.html
- What to learn
  - Diffusive reflection + Gouraud interpolation
  - Directional light source
  - Three normals for each vertex to make the edges “look sharp”
    → Each vertex can only be shared among the triangles in the same face.
- Only vectors (not positions) are used for computation
- World coordinate system with no model transformation applied
- Problem? The area not lighted directly is too dark.
  - http://rodger.global-linguist.com/webgl/ch08/LightedCube_animation.html
  - “Ambient reflection” needs to be added.

Example #2:
`LightedCube_ambient`

Example #2: `LightedCube_ambient`

- http://rodger.global-linguist.com/webgl/ch08/LightedCube_ambient.html
- What to learn
  - Diffusive & ambient reflections + Gouraud interpolation
- Directional light source
- Only vectors (not positions) are used for computation
- World coordinate system with no model transformation applied

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)
  - $N = M$ if M is orthogonal.
- http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/the-normal-matrix

Example #3:

`LightedTranslatedRotatedCube`

Example #3: `LightedTranslatedRotatedCube`

- http://rodger.global-linguist.com/webgl/ch08/LightedTranslatedRotatedCube.html
- What to learn
  - How to compute the "normal matrix" using `Matrix4.setInverseOf()` and `Matrix4.transpose()`.
- Diffusive & ambient reflection + Gouraud interpolation
- Directional light source
- World coordinate system
- In this example, since the light computation is done in the "world coordinate system", the normal matrix computed as the inverse transpose of the "model matrix."

Example #4: `PointLightedCube`

Example #4: `PointLightedCube`

- http://rodger.global-linguist.com/webgl/ch08/PointLightedCube.html
- What to learn
  - How to handle a point light source
- The light position and the vertex positions need to be transformed to the same coordinate system. (world coord system in this example)
- Diffusive & ambient reflection + Gouraud interpolation
- Low quality due to Gouraud interpolation
  - http://rodger.global-linguist.com/webgl/ch08/PointLightedCube_animation.html

Example #5:

`PointLightedCube_perFragment`

Example #5: `PointLightedCube_perFragment`

- http://rodger.global-linguist.com/webgl/ch08/PointLightedCube_perFragment.html
- What to learn
  - Per-fragment lighting
- Positions and Normals need to be interpolated and passed to the fragment shader (Phong interpolation) → Needs to be defined as a varying variable
- Since the normal length may change after interpolation, it needs to be normalized in the fragment shader.

Example #6: `PointLightedSphere`

Example #6: `PointLightedSphere`

- http://rodger.global-linguist.com/webgl/ch08/PointLightedSphere.html
- What to learn
  - How to render a sphere
  - To see the artifacts of per-vertex shading (Gouraud interpolation) with a point light source
- Per-fragment shading
  - http://rodger.global-linguist.com/webgl/ch08/PointLightedSphere_perFragment.html

Example

- https://xregy.github.io/webgl/src/shading.html
  - Shading models: Blinn-Phong or Phong
  - Interpolation: Gouraud or Phong
  - Shading computed in the eye coordinate system. – If computed in the world coordinate system, it is cumbersome to compute the eye position.