Shading Models

Wednesday, June 3, 2020 4:58 PM

Diffuse and per-vertex shading with a single point light source, omnidirectional scattering

From OpenGL 4 Shading Language Cookbook, Ch. 3, The Basics of GLSL Shaders

Diffuse reflection, the surface appears to scatter light in all directions equally, regardless of the angle of incidence.

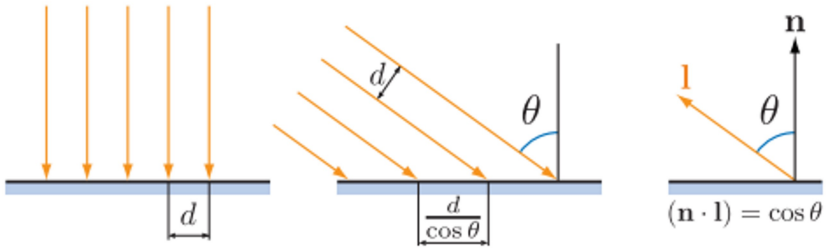
A diffuse surface has a dull look, with no shine at all.

There are 2 factors that influence the *outgoing radiance* reflected by a diffuse surface at a given point:

- The ***incoming radiance***: the radiance that arrives at a certain surface point depends on the ***angle of incidence***: it is ***maximal*** when it arrives parallel to the surface's normal and ***minimal*** when it arrives perpendicular to it.

  Here, the vector l of incoming light points away from the surface for 2 reasons: 1) this is not ray tracing, where rays point at the surface (*if they come from outside*); here l is just a vector that points in the direction of the light source; and 2) by convention, incoming radiance *points away from the surface*.

Real-time Rendering, Ch. 5, Shading Basics



- The ***absorption factor***: light interacts with the surface; if the surface is diffuse, it penetrates it microscopically, just a little, and is ***partially or fully absorbed***.

With these 2 factors in mind, the ***outgoing radiance*** is computed as follows:

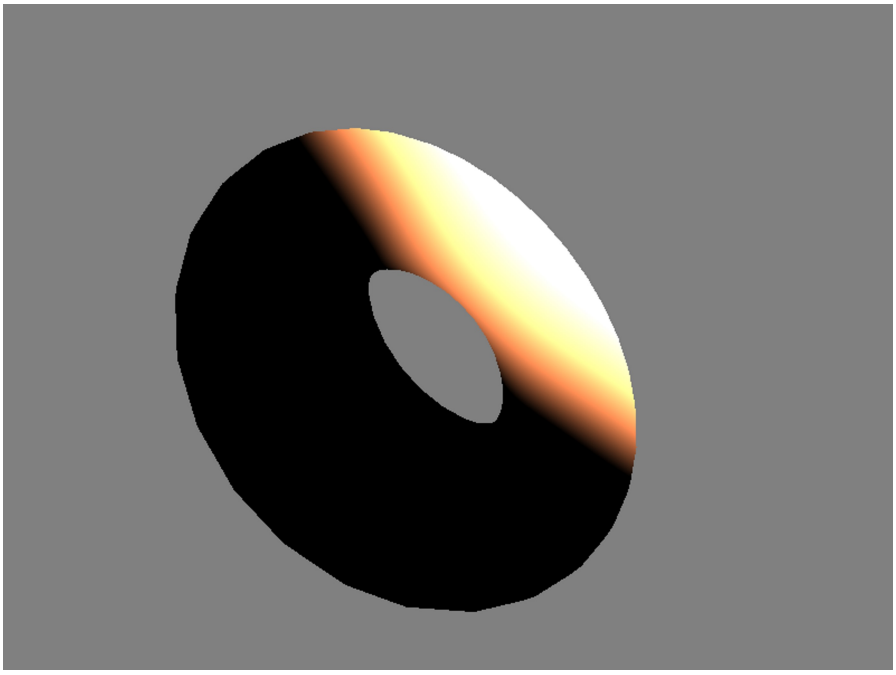
$$k_d L \cos\theta = k_d L(n \cdot l)$$

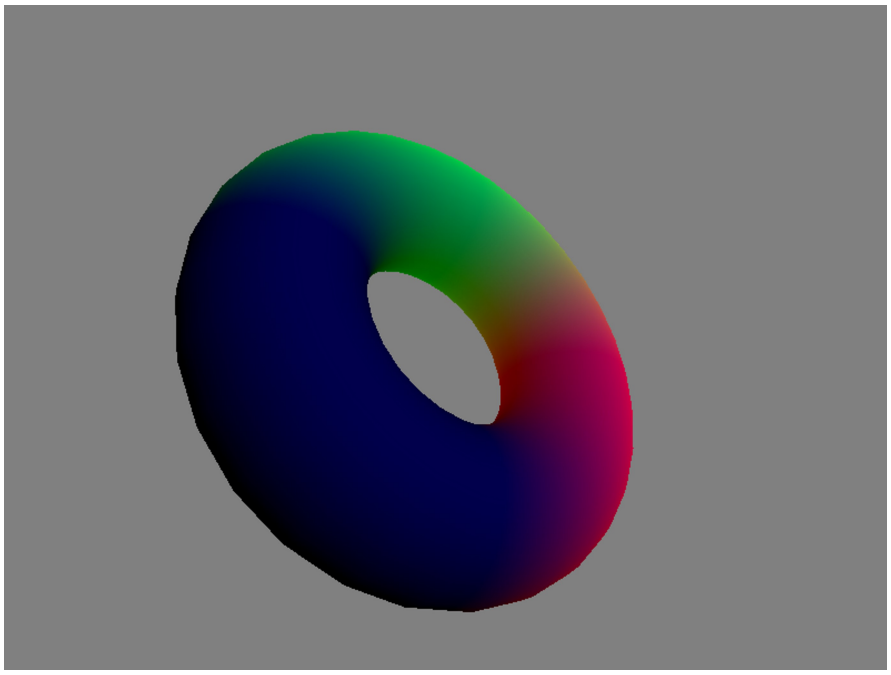
where k_d is the ***diffuse reflectivity coefficient*** (referred to in ray tracing as *absorption factor*) that models the absorption of radiance by the surface, L is the ***light intensity***, and θ is the angle between the normal and the direction of the light source l (called ***angle of incidence*** if you look at $-l$ as a ***ray of light***). Recall that $\cos\theta$ is 1 when 2 vectors are parallel and point in the same direction, and 0 when they are perpendicular. $\cos\theta$ then scales L in the desired way, conserving it fully when light arrives parallel to the normal, and decreasing it proportionally as the angle of incidence increases, to the point where it extinguishes it when it arrives perpendicular to the surface.

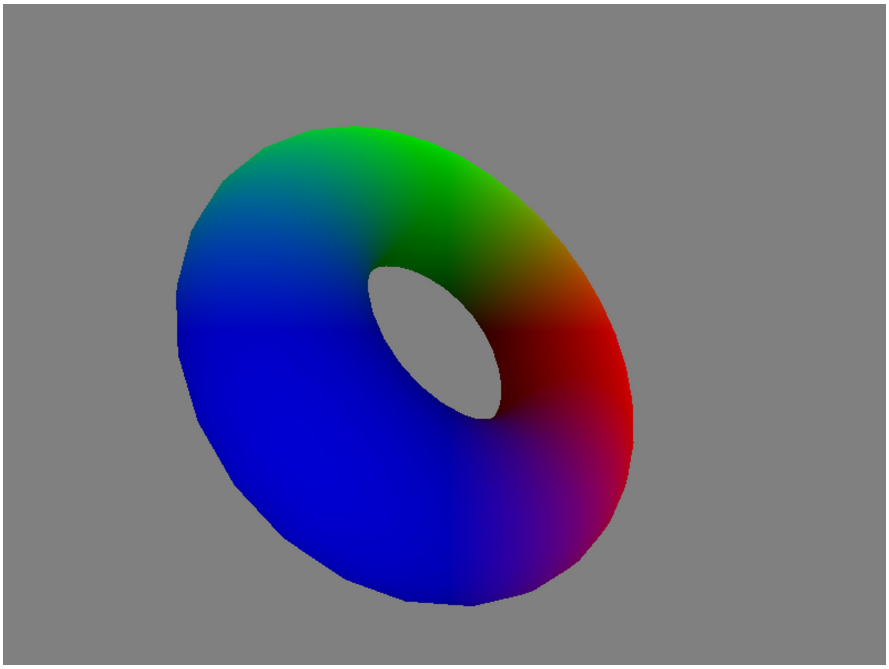
IMPORTANT: Note that this model doesn't depend on the position of the camera (there's no ***view vector*** in the equation), only on the direction of the light source relative to the surface point and the normal. This model is said to implement ***uniform or omnidirectional scattering***.

(It's interesting that the *Ray Tracing in One Weekend* series doesn't scale radiance according to the angle of incidence. It just multiplies the diffuse reflectivity coefficient by the sum of the colors of the bounces. I think it's because there's no light source in the scene per se. Besides, note that the)

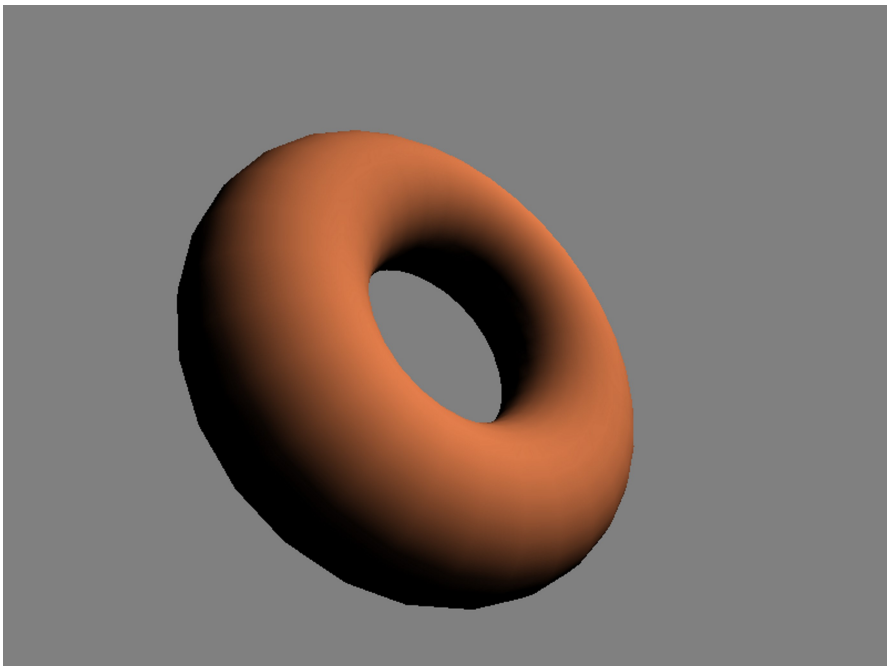
First attempt: what's wrong is that I was setting the vertex position also as the value of the vertex normal







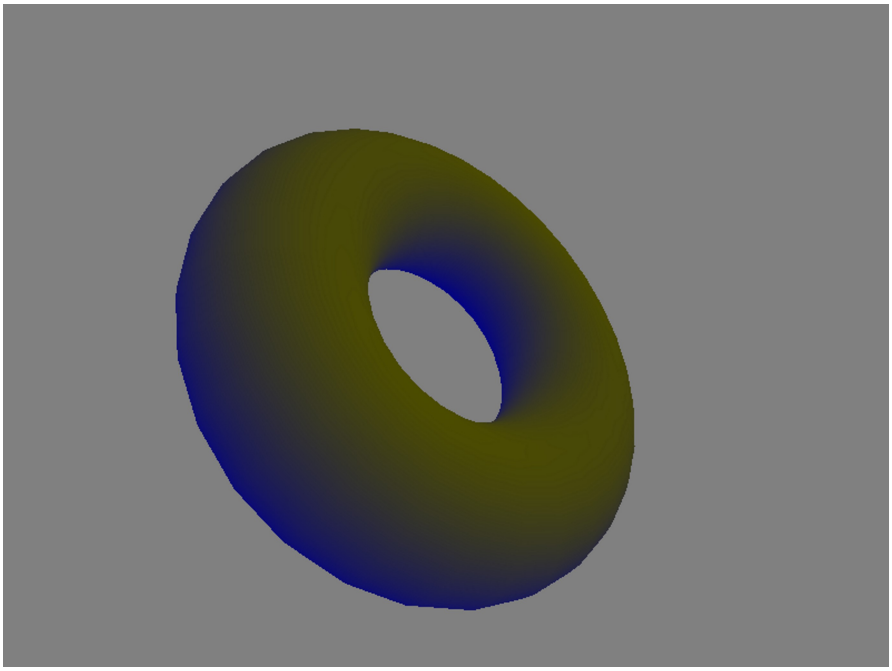
Whenever you use the ***dot product to compute the cosine***, make sure the vectors are normalized: if they aren't and they are larger than unit vectors, the dot product will go beyond cosine's range [-1, 1]. The ***dot product computes the cosine only when the vectors are normalized***.



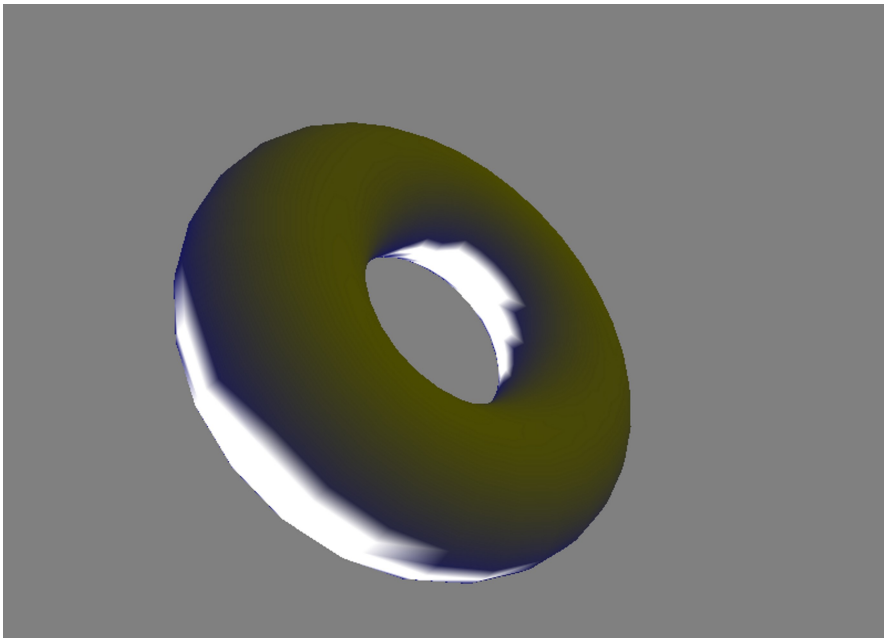
A note about the normal matrix: The normal matrix is typically the inverse transpose of the upper-left 3x3 portion of the model-view matrix. We use the inverse transpose because *normal vectors transform differently than the vertex position*. If your model-view matrix does not include any non-uniform scalings, then one can use the upper-left 3x3 of the model-view matrix in place of the normal matrix to transform your normal vectors. However, if your model-view matrix does include (uniform) scalings, you'll still need to (re)normalize your normal vectors after transforming them.

Gooch model, variation

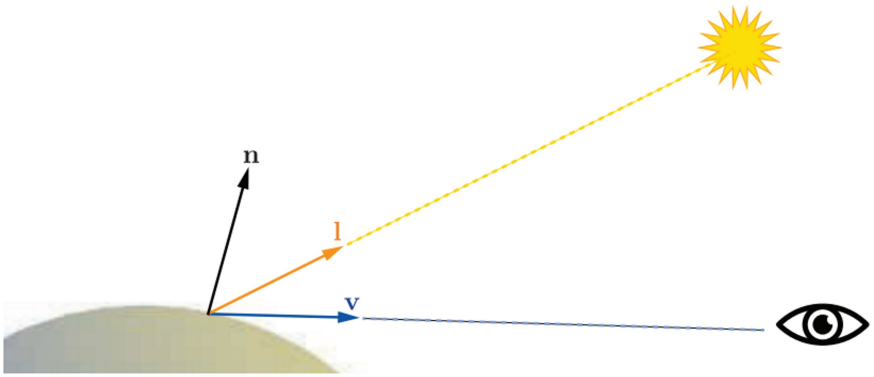
The Gooch model interpolates linearly between 2 colors. The interpolator is given by the cosine of the normal n and the direction from the vertex to the position of the light ℓ: vertices facing the light source directly are shaded with 1 color, vertices facing away from the light source completely are shaded with the other color; vertices in between have their color interpolated between the 2.



The following is the result of adding a ***highlight*** to the shading model. The highlight depends on the direction of the camera relative to the vertex. See the book for details.



The vectors used by the Gooch shading model



Phong reflection model, per-vertex ambient, diffuse, and specular Gouraud shading

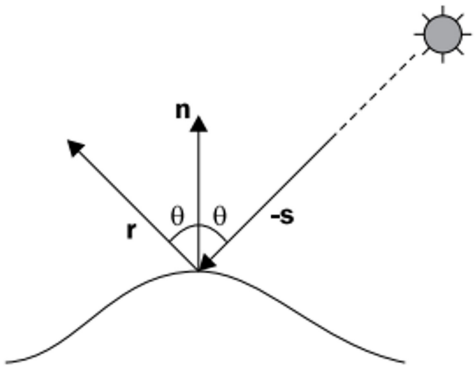
The Phong ***reflection model*** has 3 components: ***Ambient light***, ***diffuse reflection,*** and ***specular reflection***.

Ambient light models light that has been reflected so many times that it appears to emanate uniformly from all directions.

Diffuse reflection models light that reflects from a matte surface and that scatters uniformly/equally in all directions.

Specular reflection models the shininess of the surface. When the surface doesn't have imperfections and is super smooth, the ***angle of reflection*** (of the outgoing radiance) is the ***same as the angle of incidence,*** and the incoming radiance, outgoing radiance, and normal vectors $(-s, r, n)$ are ***coplanar***.

The 3 vectors are coplanar and the angles are the same



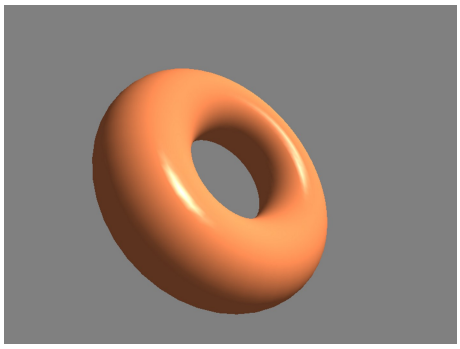
The outgoing radiance computed by this model ***just adds these 3 components together***.

$k_a L_a + k_d L_d (n \cdot l) + k_s L_s (r \cdot v)^f$

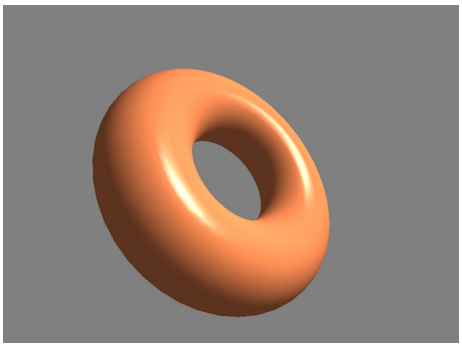
where the various ks are the surface's reflectivity coefficients of each component, the Ls are the intensities of the light of each component (the ambient light, which comes from the environment; the point light; and the shine), v is the direction towards the camera, and f determines how shiny the material is.

f causes the dot product to decrease rapidly. The faster it decreases, the smaller the shiny area will be.

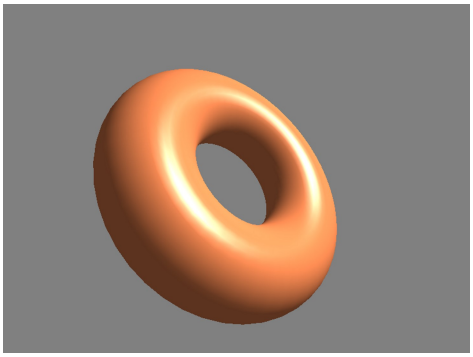
$f = 100$



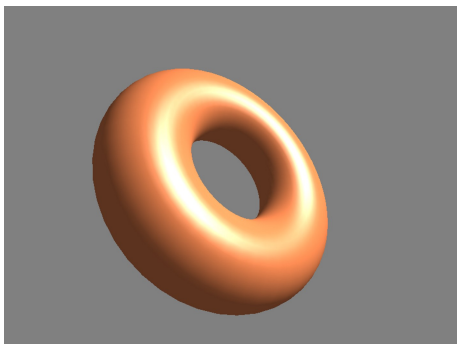
f = 50



f = 20

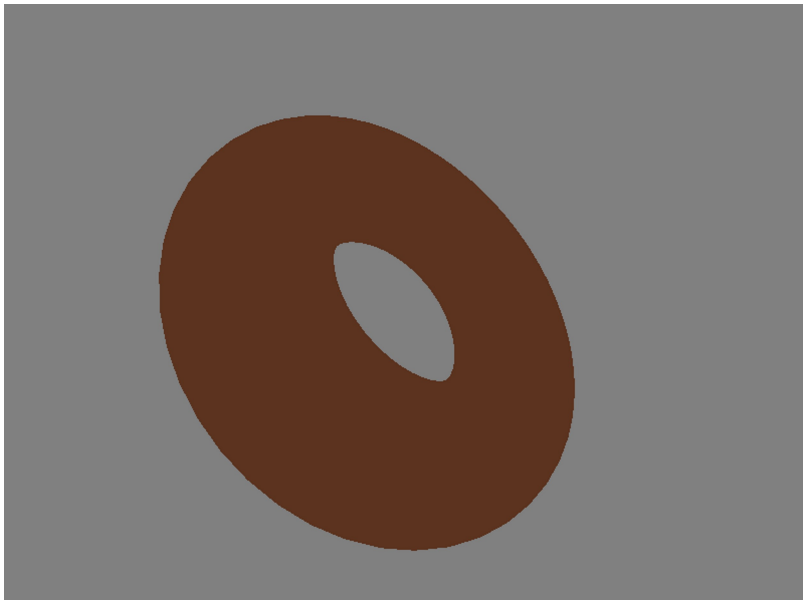


f = 10

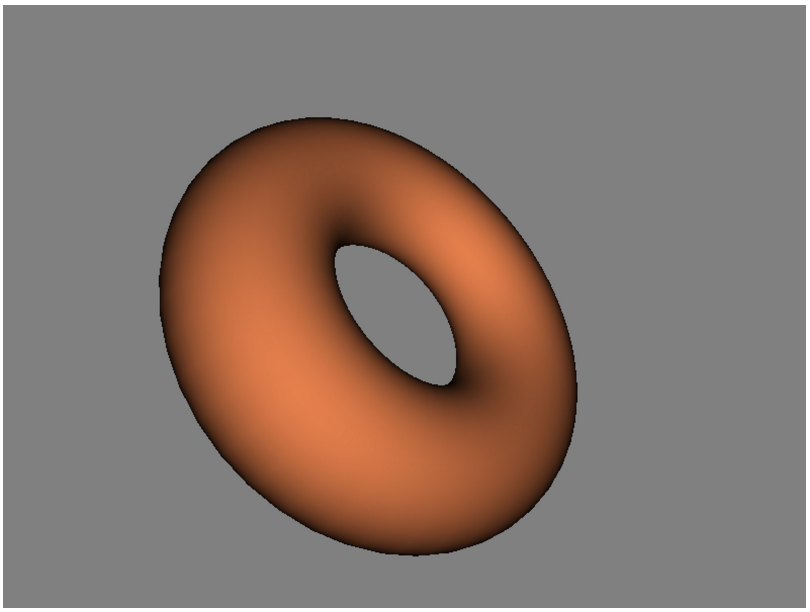


Since it's just a sum, you can see why the scene lights up when you make the ***ambient light*** whiter. Ambient light illuminates all surfaces equally.

Ambient light only

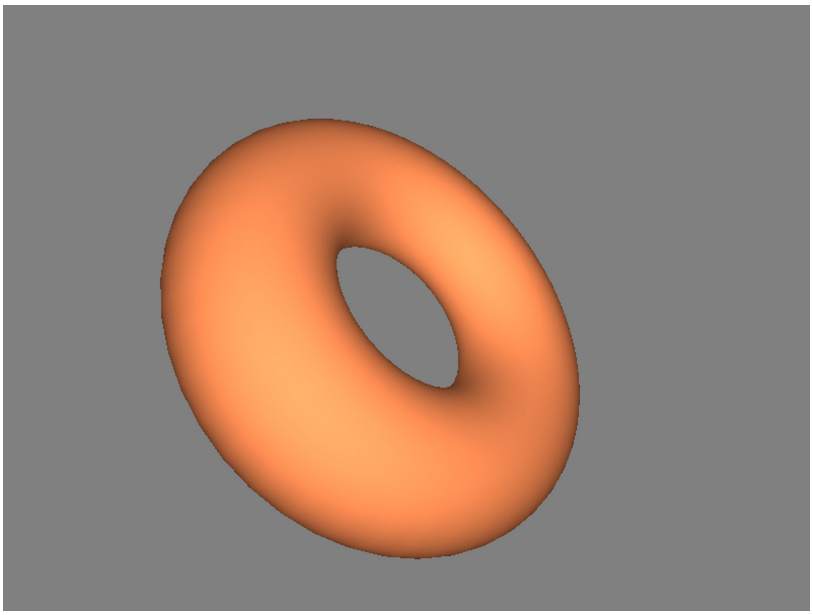


Diffuse, omnidirectional scattering only

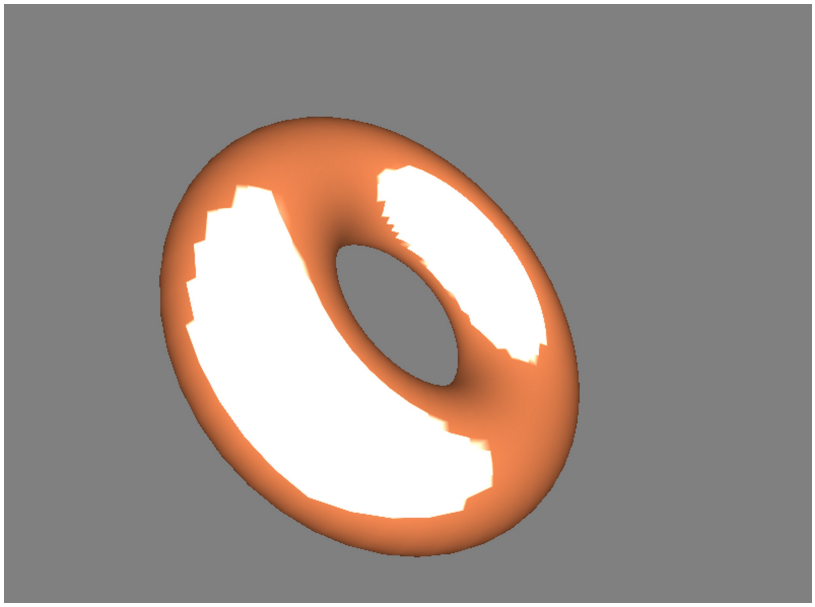


Ambient + diffuse

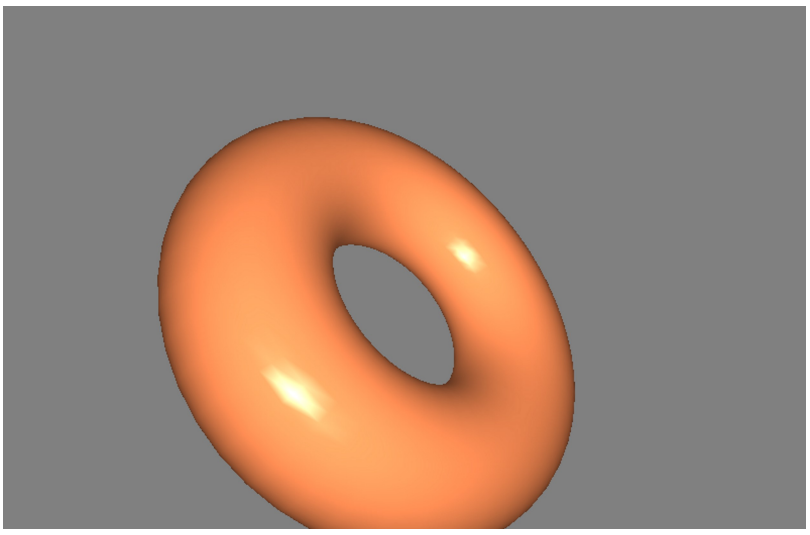
Note how the ambient light brightens up everything, including the darker regions

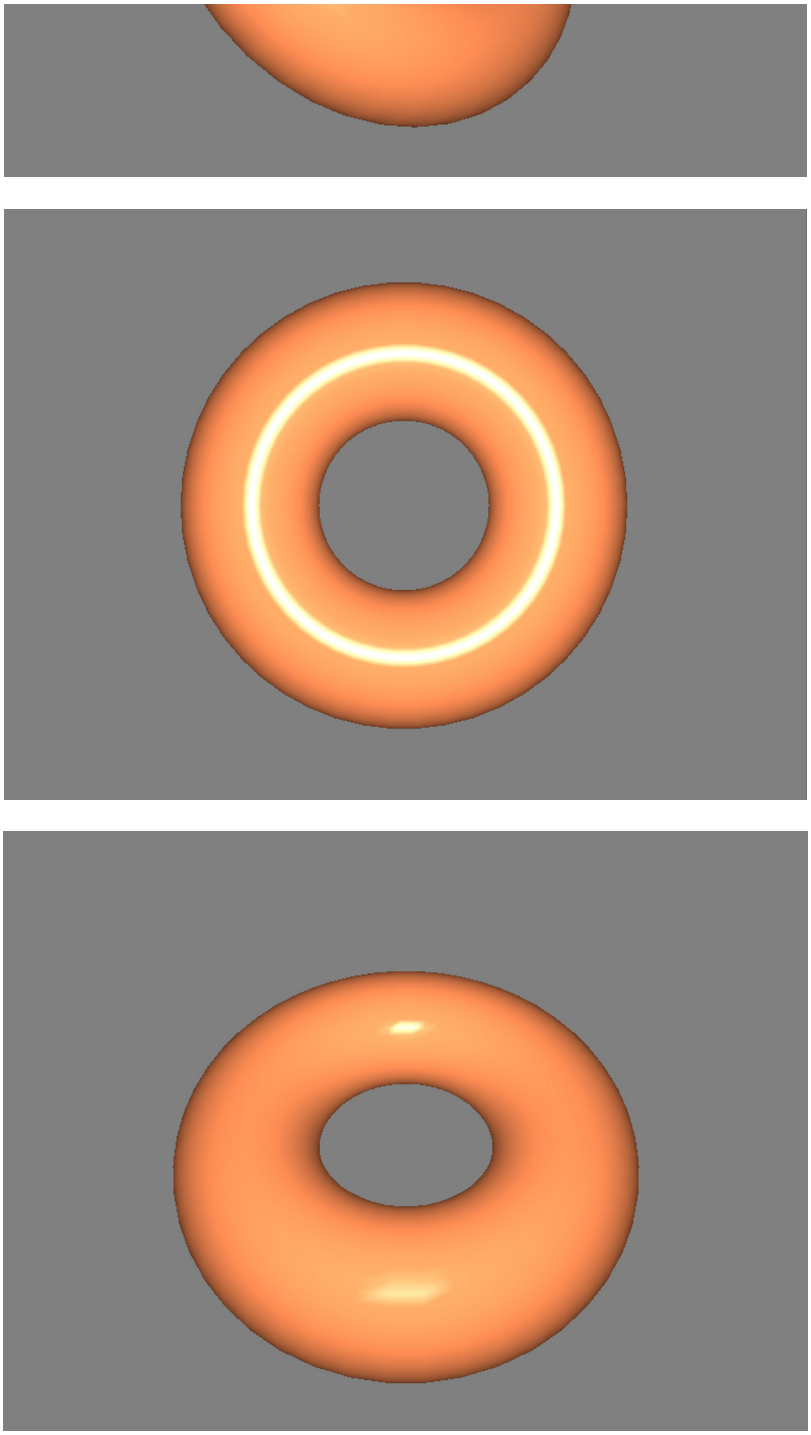


Specular + diffuse + ambient, first attempt

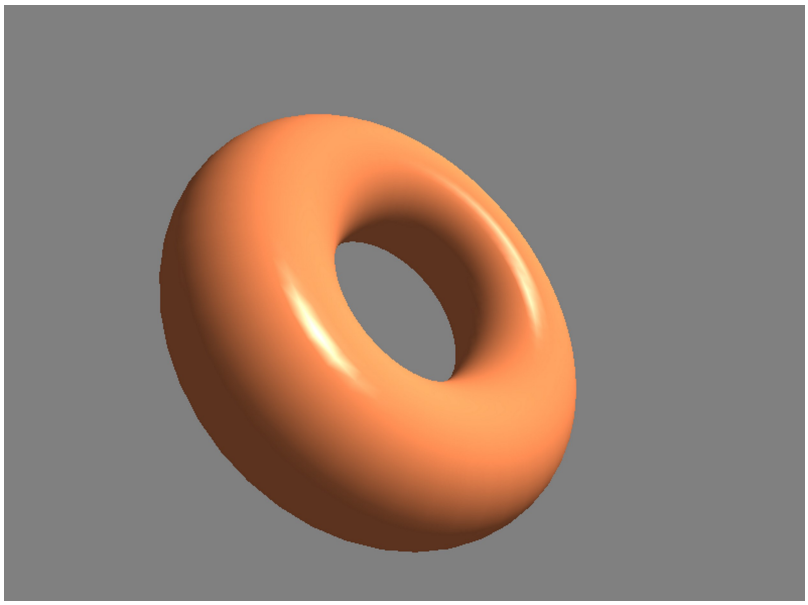


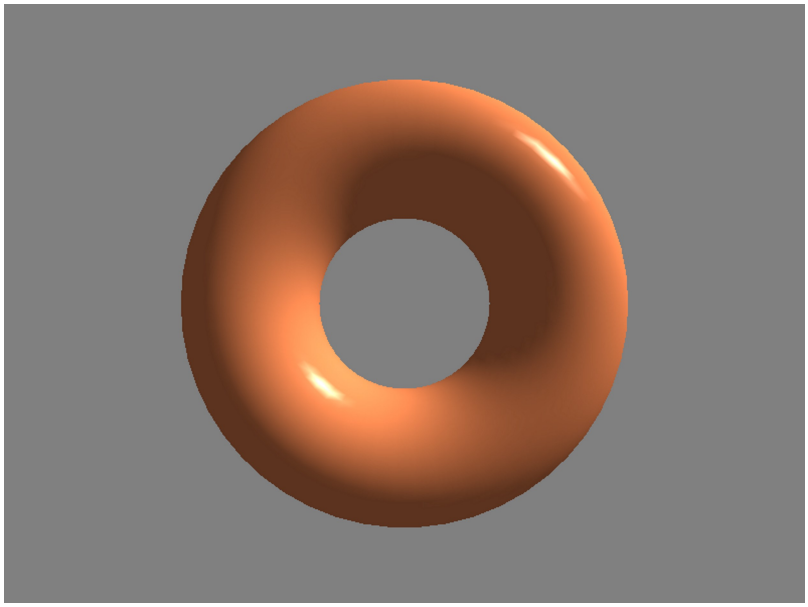
Specular + diffuse + ambient, light source is at the view origin

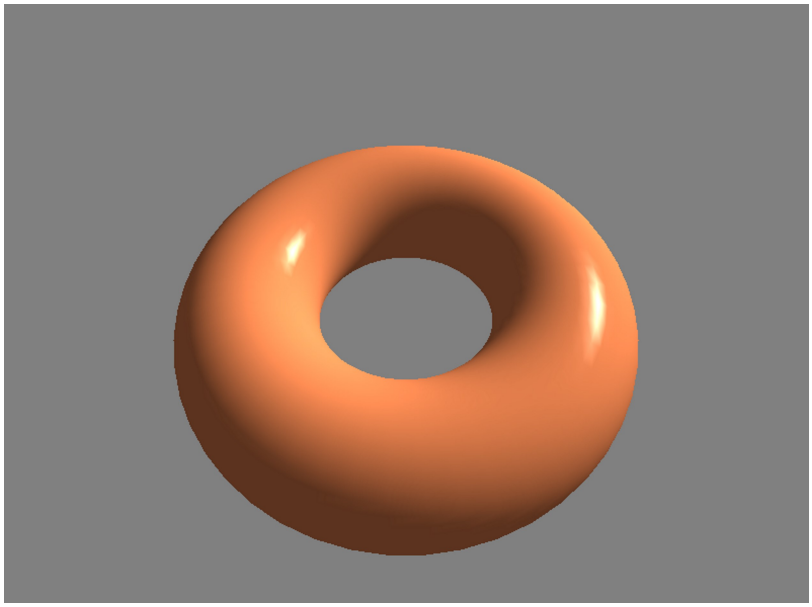




Specular + diffuse + ambient, light source is on the right hand side





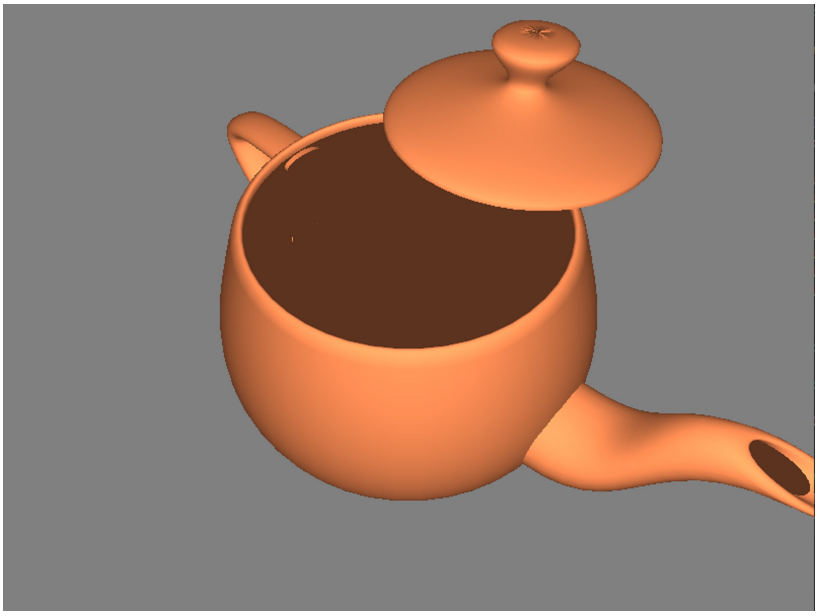


A shading equation may be ***evaluated*** in the vertex shader or in the fragment shader. When in the vertex shader, it gets evaluated only at the vertices (of course), whereas when in the fragment shader, at every fragment. Per-vertex evaluation has limitations; for example, there's no specularity on the surface of this cube, despite there being a light in front of it; this is because the shading equation was evaluated, for this face, only at the 4 vertices; at those 4 vertices, given the reflection vector and normal at those points, perhaps the specular component turned out to be 0; an outgoing radiance that had only ambient and diffuse components (and a specular component of 0), was interpolated across the face. Per-vertex evaluation and interpolation across the face is known as ***Gouraud shading***.

Shading with the 3 ADS components

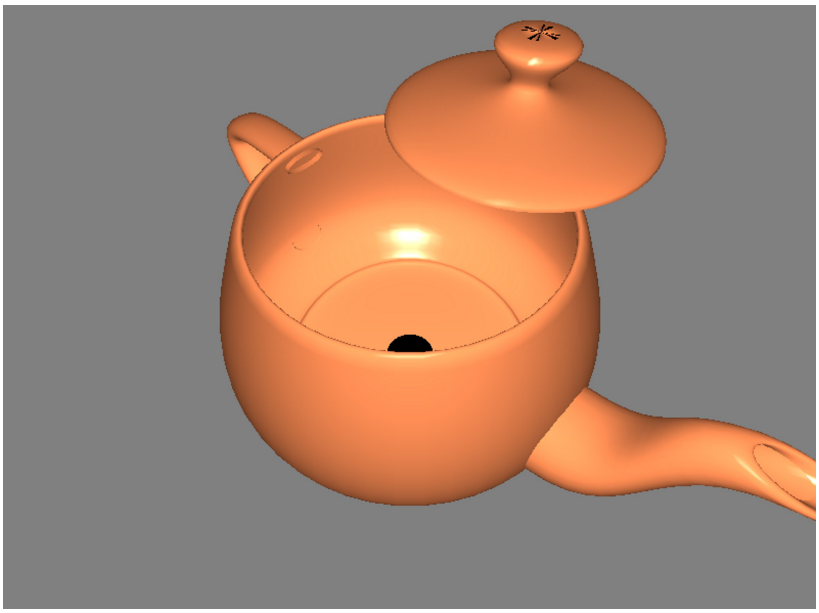
Removing the specular term of the equation altogether yields the same result

To shade ***2-sided geometry***, you need to invert the normal as needed. The normal is crucial in the computation of the diffuse and specular components. When a ***back face*** of the geometry is ***visible***, its front face (and the normal) will most likely point away from the camera; if you don't invert the normal, the view vector v and the normal will point in opposite directions, their dot product will be negative, and the ***specular component*** will be 0. When a back face is ***exposed to a light***, its front face won't be exposed to it; if you don't invert the normal, the light direction vector ℓ and the normal will point in opposite directions, their dot product will be negative, and the ***diffuse component*** will be 0.



Inverting the normal when v and the normal point in opposite directions,
to compute the specular component of a visible back face.

Inverting the normal when ℓ and the normal point in opposite directions,
to compute the diffuse component of a back face exposed to ℓ's light source.

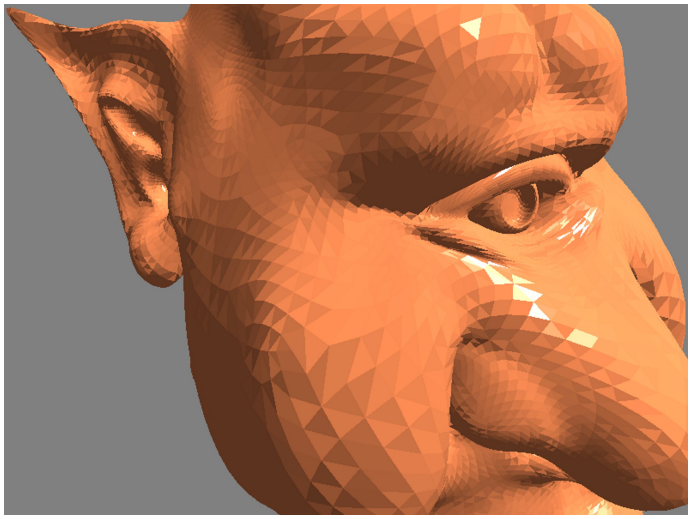


Gouraud vertex interpolation shading vs Flat shading

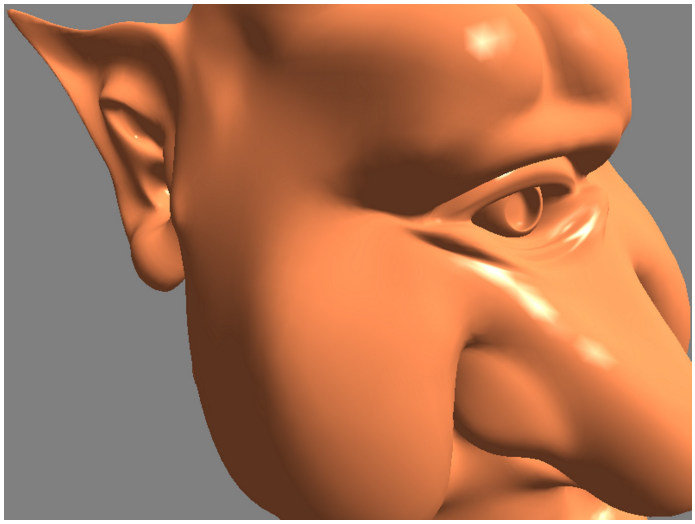
While Phong is a reflection model, Gouraud is a shading model. In ***Gouraud shading***, a color is computed at every vertex and the primitive's surface color is the result of interpolating these colors.

An alternative to vertex interpolation is ***flat shading***: a primitive has a single color.

Flat shading

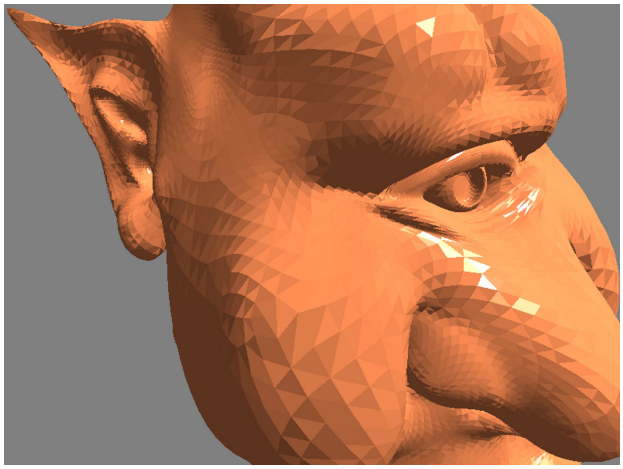


Gouraud shading



Switching between flat and Gouraud in GLSL is very easy: the default is Gouraud; if an ***out variable*** of the vertex shader is qualified with ***flat*** (and its corresponding input variable in the fragment shader), it won't be interpolated. Now, since each of the 3 vertices of a triangle might have a different color, which one gives the fragment its color? The ***provoking vertex***. The provoking vertex is either the first one (`glProvokingVertex(GL_FIRST_VERTEX_CONVENTION)`) or the last one (`glProvokingVertex(GL_LAST_VERTEX_CONVENTION)`).

Last *vertex is provoking vertex*



First *vertex is provoking vertex*

Compare the specular reflections on the forehead and nose with those of the previous render
(The specular component at each of the 3 vertices of any given triangle is likely to be different; only one of them gives the triangle its color.)