DAE Maths Level Test 2017-18

STUDY MATERIALS

- **Game Development, Independent Game Production majors**: course book *Animation Maths (Ivo De Pauw, Bieke Masselis), LannooCampus (2016), ISBN 978-94014-32047*, which will be used in the course units Applied Maths and Physics 1 plus 2,

  at edu pricing via `http://www.lannooshop.be/en/howest` or around the globe via `http://www.amazon.com`.

- **Game Graphics Production, 3D Production & Visual Effects**: course book *Multimedia Maths (Bieke Masselis, Ivo De Pauw), LannooCampus (2016)*

  at edu pricing via `http://www.lannooshop.be/en/howest` or around the globe via `http://www.amazon.com`.

- In case of study material issues, please contact: `ivo.de.pauw@howest.be`.

CHAPTERS TO COVER

From your appropriate course book, study these chapters thoroughly and practice their paragraph of exercises. You can download the solutions to all exercises from your book's companion website,

either `http://www.animationmaths.be` or `http://www.multimediamaths.be`

- TRIGONOMETRY
- FUNCTIONS
- VECTORS
- MATRICES
- LINEAR TRANSFORMATIONS
- LIGHTING (outside of the course book but hereafter appended)

Chapter 7

Lighting

In this chapter we discuss the physical and mathematical principles that are the basics for lighting calculations in a 3D game. Most game engines use a **rasterizer** for 3D rendering and this means that an approximate model of real life lighting is used, or that realism is thrown out of the window altogether for the creation of a 'cartoony' style of rendering.

One of the weaknesses of the rasterizer is that shadows and reflections are somewhat harder to accomplish. The main reason is that a rasterizer works on individual objects on the scene, and the environment of the object can only be communicated via special means (e.g. environment textures, shadow maps, ...).

LIGHTING, by courtesy of Koen.Samyn@howest.be

We conclude the chapter with a discussion about the rendering pipeline in a typical 3D or game application. A modern GPU is very flexible in terms of programming but some basic building blocks are always present.

7.1 Color

Lighting mostly takes place in RGB space or SRGB space. This color space is not necessarily the best color space for lighting calculations. There are other color spaces where a brighter color can be calculate more correctly. However, in a RGB color space we just multiply all the channels with the same parameter and hope for the best:

$$c_1 = (124\ 12\ 64)$$
$$c_2 = 2 \cdot c1$$
$$c_2 = (248\ 24\ 128)$$

Color c_1 represents the following color: , and after multiplying c_1 with 2 we get color c_2 : .

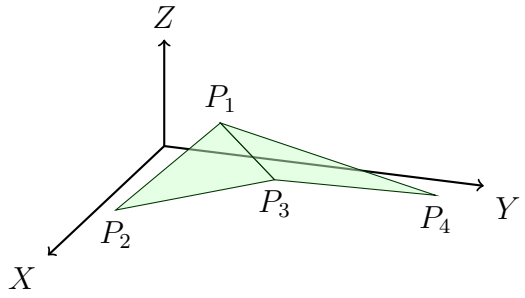
This principle is used in a lot of renderers to illuminate (or brighten) the diffuse color of an object. However, in a lot of cases this multiplication also changes the **tint** of the color. A possible solution would be to use an alternate color space that supports brightness, but this would require major changes in the art production toolchain, so it is probably not very realistic to hope for this to happen.

7.2 Normal calculation

The concept of a **normal** is very important when calculating the lighting on a 3D mesh. The **normal** is a vector that is perpendicular to a triangle that

is part of the mesh. For lighting calculations the normal must have length **one**.

We will start with a simple example. The following simple mesh is a single quad, which contains two triangles. Triangle 1 is defined by the points P_1, P_2 and P_3. Triangle 2 is defined by the points P_1, P_3 and P_4.



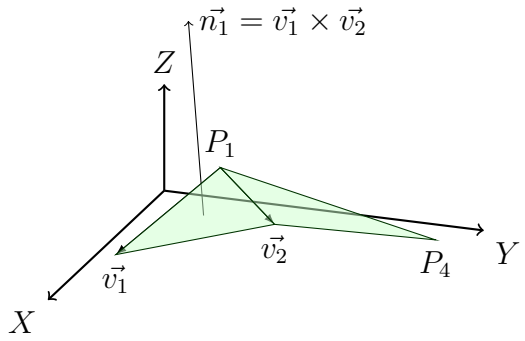
To calculate the normal for triangle 1 (P_1, P_2 and P_3) we need to calculate two vectors in this triangle:

$$\vec{v_1} = P_2 - P_1$$
$$\vec{v_2} = P_3 - P_1$$

The normal is now simply the **normalized** cross product of these two vectors:

$$\vec{c_1} = \vec{v_1} \times \vec{v_2}$$
$$\vec{n_1} = \frac{\vec{c_1}}{||\vec{c_1}||}$$

The result of the cross product is a vector that is perpendicular to the first triangle:



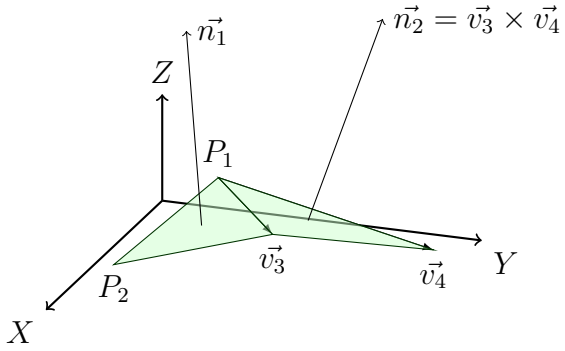
We can now perform the same operation for the second triangle. However, in a 3d mesh we typically want the normal vectors to point in the same direction. In other words, for a closed object, all the normal vectors should point outwards for example. For the second triangle the calculate becomes:

$$\vec{v_3} = P_3 - P_1$$
$$\vec{v_4} = P_4 - P_1$$

The normal is now simply the cross product of these two vectors:

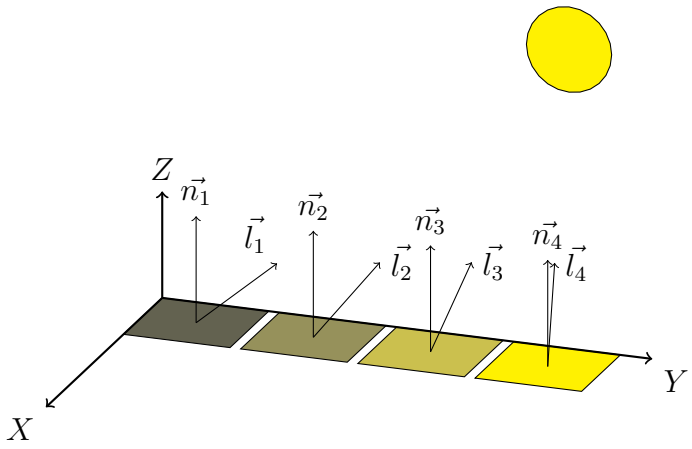
$$\vec{c_2} = \vec{v_3} \times \vec{v_4}$$
$$\vec{n_2} = \frac{\vec{c_2}}{||\vec{c_2}||}$$

And we can draw $\vec{n_2}$ on the second triangle:



7.3 Diffuse lighting

Diffuse lighting depends only on the position or direction of the light source and the surface normal. To illustrate this concept, consider the following situations:



The normal vectors ($\vec{n_1}$, $\vec{n_2}$, $\vec{n_3}$ and $\vec{n_4}$) are in this case the same vector: (0 0 1). The light vectors ($\vec{l_1}$, $\vec{l_2}$, $\vec{l_3}$ and $\vec{l_4}$) are the vectors that are directed from the plane towards the light source.

The angle between these two vectors determine how much **diffuse** light will hit the surface. It is easy to see that if the angle between the two vectors is

small or **zero**, the intensity on the surface will be **maximal** (situation on the right). If the angle between the two vectors is **large** (for example 90°), then the surface intensity will be minimal.

The formula that we can use to calculate the diffuse intensity on the surface is the dot product:

$$I_d = \vec{l} \cdot \vec{n}$$

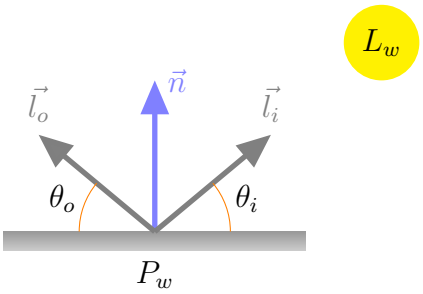
The light vector $\vec{l}$ depends on the type of light. For a directional light the light vector will be independent of the position of the pixel that we want to calculate. For a pointlight or spotlight the light vector will be the difference between the light position in the world and the pixel position in the world:

$$\vec{l} = \frac{L_w - P_w}{||L_w - P_w||}$$

When calculating vectors for lighting calculations it is always important to use **normalized** vectors. It is therefore necessary to divide the vector by its length.

7.4 Specular lighting

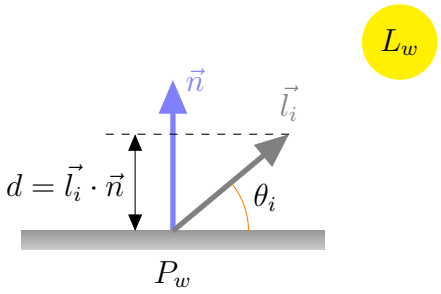
Specular lighting can be seen as the reflection of the light source by the material. A typical reflection will **mirror** the light ray based on the normal of the surface. In other words, the incoming angle of the light θ_i will be equal to the outgoing angle θ_o.



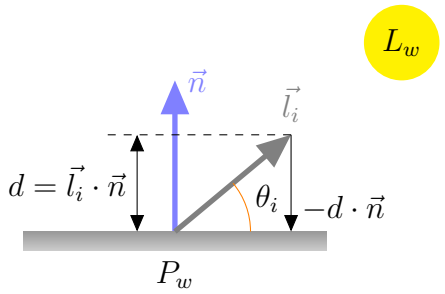
Just like the light vector calculation for diffuse we will calculate the light vector as follows:

$$\vec{l} = \frac{L_w - P_w}{||L_w - P_w||}$$

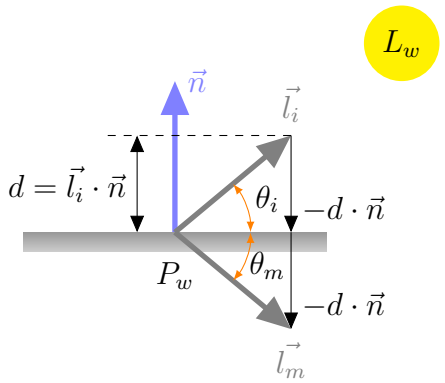
To calculate the outgoing vector $\vec{l_o}$ starting from the incoming angle $\vec{l_o}$ and the normal $\vec{n}$ we will use the dot product to calculate the projection of $\vec{l_i}$ on $\vec{n}$:



If $||\vec{n}||$ equals **one** and $||\vec{l_i}||$ equals **one**, then the length d is the dot product of the vectors $\vec{n}$ and $\vec{l_i}$. We can now use this length and the normal vector $\vec{n}$ to create a vector that is mirrored by the horizontal plane:



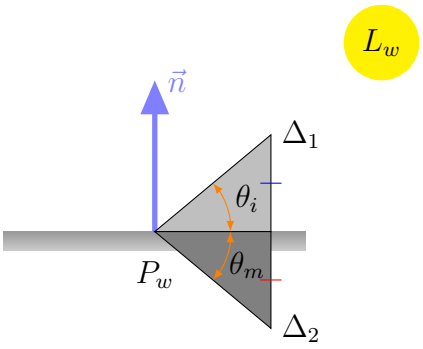
if we calculate the result of $\vec{l_i} - d\vec{n}$ we reach the horizontal plane. To mirror the vector $\vec{l_i}$ we need to apply the same vector again :



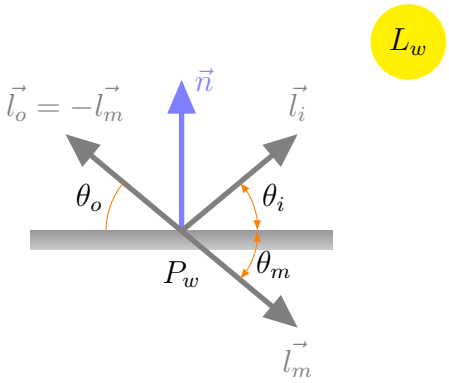
The mirrored vector $\vec{l_m}$ is calculated as:

$$\vec{l_m} = \vec{l_i} - 2d \cdot \vec{n}$$

It is easy to show that θ_i is equal to θ_m because these two angles are in **congruent** triangles. As you can see in the following figure the height of the triangles Δ_1 and Δ_2 is the same and they share an edge:



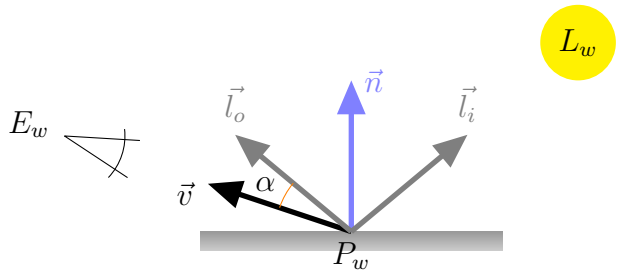
Finally we can calculate the **reflected** vector by negating the vector $\vec{l_m}$. Negating the coordinates of a vector effectively rotates the vector over 180°.



This reflected vector can now be used to calculate the specular intensity. However, there is also a technique which is called the **half vector** technique that can also be used to calculate the specular intensity.

7.4.1 Specular intensity

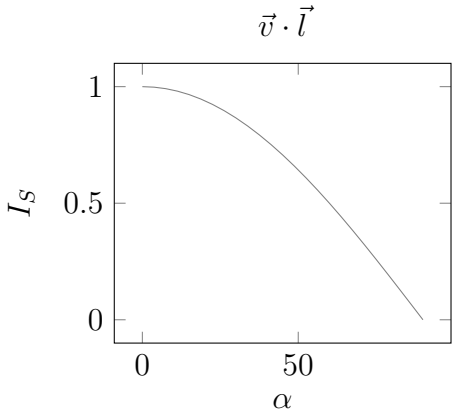
The specular intensity is dependent on the position of the viewer. This means that we need to define a view vector $\vec{v}$ that is normalized and again pointing from the pixel we want to calculate (p_w) towards the position of the viewer or eye (E_w).



A first approach for the specular component is again the dot product but now with the vectors $\vec{v}$ and $\vec{l_o}$. The specular intensity I_S is then calculated as :

$$I_S = \vec{v} \cdot \vec{l_o}$$

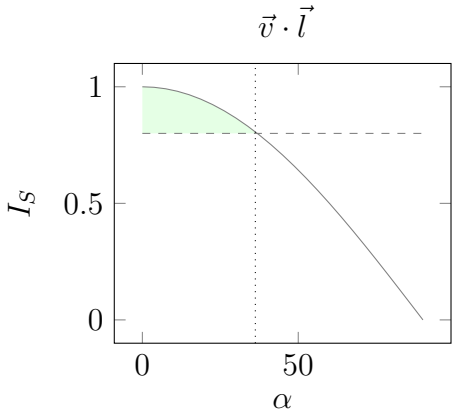
The dot product is related to the cosine of the angle between the vectors, thus the intensity in terms of the angle α can be presented as:



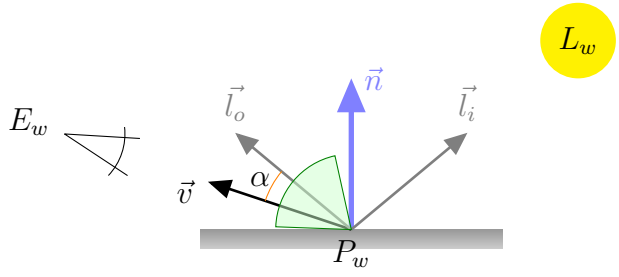
The problem with the dot product is that the area where the intensity is higher then 80% is quite large. We can visualize this by calculating the angle α for which the cosine equals 0.8 :

$$\begin{aligned}\cos(\alpha) &= 0.8\\ \alpha &= \arccos(\alpha)\\ \alpha &\approx 37^\circ\end{aligned}$$

This area is represented in the figure by the light green area:



This means that around the vector $\vec{l_o}$ there is a large area of high intensity (where the intensity is larger then 0.8), again shown as a light green area in the following figure:



This large sector of high intensity leads to a result that is **overexposed**. This situation can be corrected by raising the specular intensity I_S to a power:

$$I_S = (\vec{l_o} \cdot \vec{v})^{shininess}$$

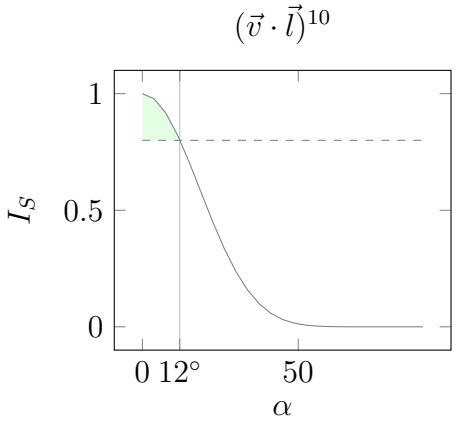
If the shininess parameter is assigned the value 10, we get the following formula for the specular power:

$$I_S = (\vec{l_o} \cdot \vec{v})^{10}$$

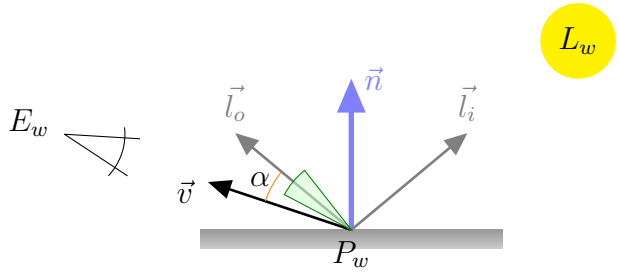
We can again calculate the value for α if we want to know the area where the intensity is higher then 0.8 or 80

$$\begin{aligned}(\cos(\alpha))^{10} &= 0.8\\ \cos(\alpha) &= \sqrt[10]{0.8}\\ \alpha &= \arccos(\sqrt[10]{0.8})\\ \alpha &\approx 12^\circ\end{aligned}$$

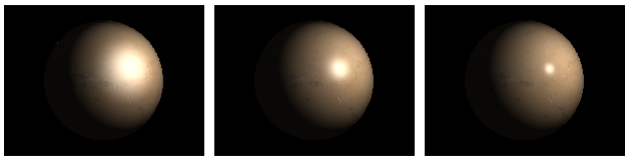
If we plot the specular power I_S again with the area where the specular power is larger then 0.8, we see that the are has been significantly reduced:



If we show this area around the vector $\vec{l_o}$ with the specular power value of ten, it is also clear to see the area with a large specular intensity is reduced:



The following figure shows the influence on the specular power. On the left, the specular power is two, the object in the middle has a specular power of 10, and the object on the right has a specular power of 50. As can be seen, the area with high specularity diminishes when the power increases:



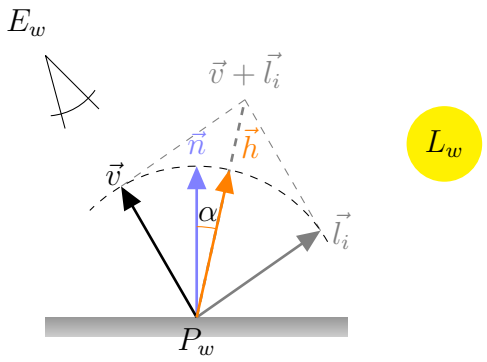
7.4.2 Specular intensity with halfvector

Another technique exists to calculate the specular intensity. With this technique the vector halfway between the view vector $\vec{v}$ and the incoming light vector $\vec{l_i}$ is calculated. The specular intensity I_S is then calculated as the dot product of this new half vector $\vec{h}$ and the normal $\vec{n}$. The half vector $\vec{h}$ can be calculated by adding $\vec{v}$ and $\vec{l_i}$ and normalizing the result:

$$\vec{h} = \frac{\vec{l_i} + \vec{v}}{||\vec{l_i} + \vec{v}||}$$

The following figure demonstrates that the sum of $\vec{v}$ and $\vec{l_i}$ is not always a

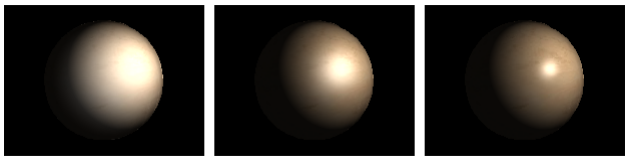
unit vector. It is necessary to normalize this sum vector to ensure that the vector $\vec{h}$ has length 1.



Now that the vector $\vec{h}$ has been calculated, the specular intensity can be calculated with the formula:

$$I_S = (\vec{h} \cdot \vec{n})^{shininess}$$

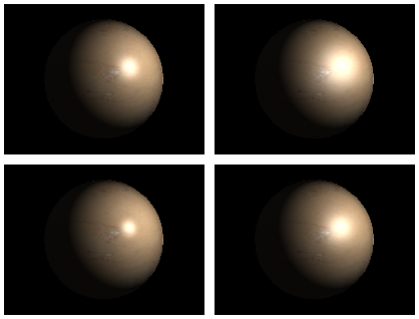
The shininess parameter provides control over the specular area. A small value for shininess leads to a large specular area, a large value generates a small specular area.



7.4.3 Comparison of reflection versus halfvector

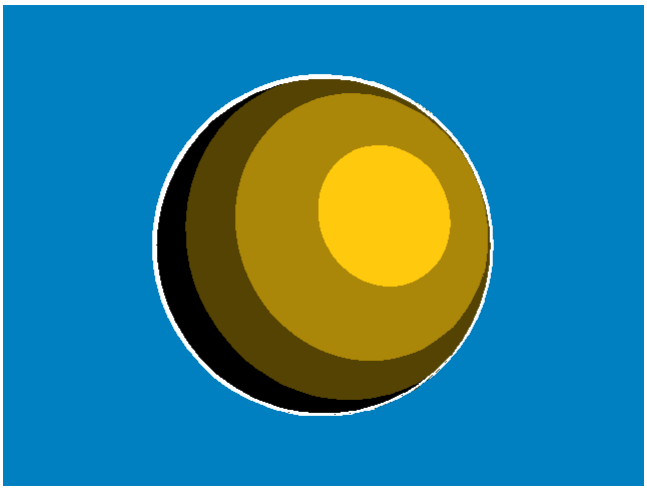
On the left hand side the results are shown for the phong shading, on the right hand side the results for blinn-phong shading with the half vector technique is shown. The top row has a shininess parameter of 10 and the bottom row has a shininess of 20.

We can immediately see tha phong shading results in a smaller highlight. Because the reflection vector is calculated in the phong shading technique, the specular highlight is **circular**. For the blinn-phong shading technique the highlight is larger in size, and also has an **ellipsoid** form.



7.5 Cell shading

Cell shading is a technique that is used to create lines (soft or hard) and in some cases also a separation of colors into a number (typically 2) of **tones** (different shades of the same colors). An example with 3 tones and a white line (for emphasis) is given in the following picture:

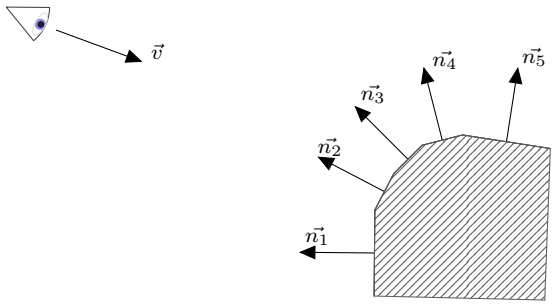


One part of the cell shading implementation determines if a pixel is a part of the line. If the pixel is on the line the line color will be used.

If the pixel is **not** on the line portion of the rendering the tone of the pixel will be determined.

7.5.1 Line determination

If a pixel is on the line part of the rendering, the dot product of the view vector with the normal of that pixel will be close to zero. The following figure shows the general concept. The view vector $\vec{v}$ is oriented towards a pixel that is to be rendered. Each pixel belongs to a surface that defines a normal vector $\vec{n_i}$.



For **smooth** surfaces the normal vector will be **interpolated**, which means that the actual normal is a **blend** of the normals of the vertices of a surface. See section 7.2 for detailed information about the calculation of the normal.

We can now

7.6 The render pipeline - Rasterization

To write shaders (programs that calculate lighting effects in games and 3d applications) it is necessary to understand the render pipeline. The rendering