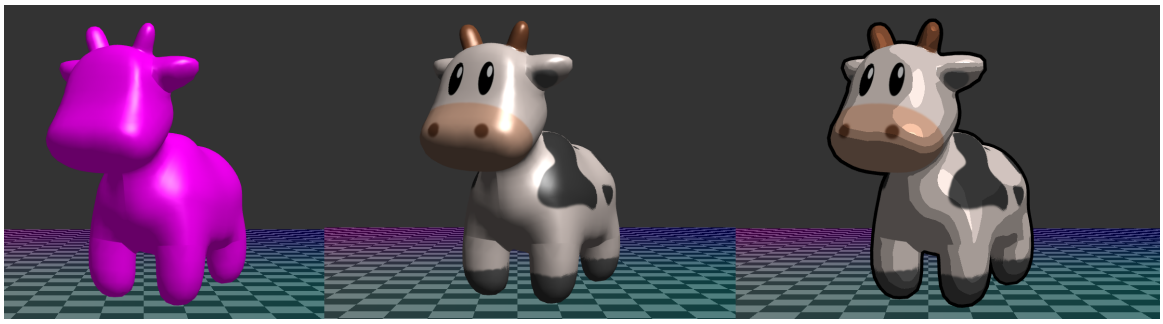
ETH Zürich
Computer Science Department

Visual Computing

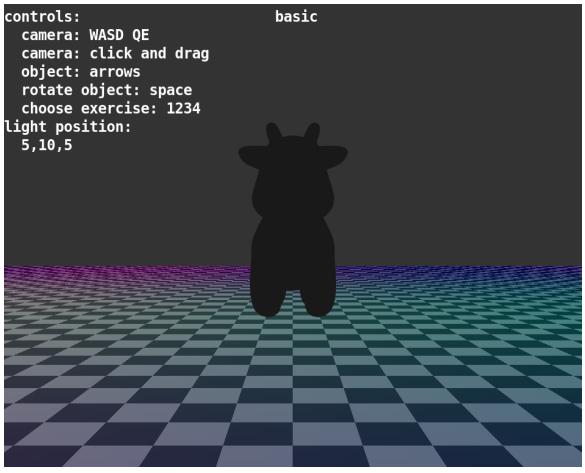
CG Exercise 9- The Graphics Pipeline



This exercise includes 5 different files:

- *tut9ex.html*: Used to start the scripts. Open this file in your browser.
- *gl-matrix.js*: Contains auxiliary functions.
- *tut9ex.js*: Script in charge of running and rendering.
- *spoon/*: Folder containing the 3d model assets (credits go to Keenan Crane).

You can have a peek at all the files (and change them if you want), but you can fully solve the exercise just by editing *tut9ex.js*.
Open the file *tut9ex.html* in a web browser. You should a see a cow, like this:



Follow the on-screen instructions onhow to control the camera, object and light.

As with the previous tutorial remember to start a simple *python** server in the exercise folder to avoid problems loading the textures:

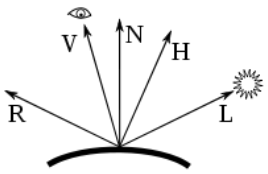
```
python3 -m http.server
```

and then go to http://localhost:8000/tut8ex.html

*You can get python from: https://www.python.org/downloads/release/python-390/

9.1 Phong Reflection Model

The Phong reflection model describes the way surface reflects light as a combination of diffuse and specular reflection. It also includes an ambient term to amount for scattered light in the entire scene.



Following the Phong reflection model, the illumination I of each surface point can be computed with:

$$I = I_a k_a + I_d k_d (\mathbf{N} \cdot \mathbf{L}) + I_s k_s (\mathbf{R} \cdot \mathbf{V})^{\alpha}$$

where

- I_a, I_d, I_s are the ambient, diffuse and specular intensities (usually rgb colors),
- k_a, k_d, k_s are the ambient, diffuse and specular reflection constant,
- α is the shininess constant,
- **N** is the surface normal of the illuminated point,
- **L** is the direction vector from the surface point toward the light source,
- **R** is the direction that a perfectly reflected ray of light would take, and it can be computed with $R = 2(\mathbf{L} \cdot \mathbf{N})\mathbf{N} - \mathbf{L}$.

Your task is to implement the Phong reflection model as a WebGL shader program.

a) For the Phong reflection model, the shader program needs to know the surface normals. In the code, the normals are computed per vertex and saved to `object.vertexNormals`. Pass the normals to the shader program (look at how this is done for vertex positions!).

b) Now that you have access to the normals in the shaders, implement the Phong reflection model. *Note: Think about where it makes most sense to compute the illumination. Per vertex or per fragment?*

c) In the last exercise you learned how to add texture to a 3d object. Combine the texture color with the Phong reflection model.

9.2 Toon Shading

While the Phong reflection model attempts to describe reflection behavior as seen in the real world, other reflection models try to give a particular artistic style to the virtual world. One such example is toon or cel shading, as seen in the above image on the right.

To create this cartoonish effect, two shading stages are needed:

1: An outline shader, that draws the object a little bit too large.

2: The toon shader, that shades the object in different tones.

In order that the second shading stage - the toon shader - always draws over the outline shader, the depth buffer has to be cleared in between with `gl.clear(gl.DEPTH_BUFFER_BIT)`.

Implement both, the outline and the toon shader. The result should look similar as the above image on the right. Note that the outline is of constant width, and the shading is limited to 4 tones of color.

Hint: For the toon shader, start from a Phong shader and modify it so that it draws only certain tones.