Shading

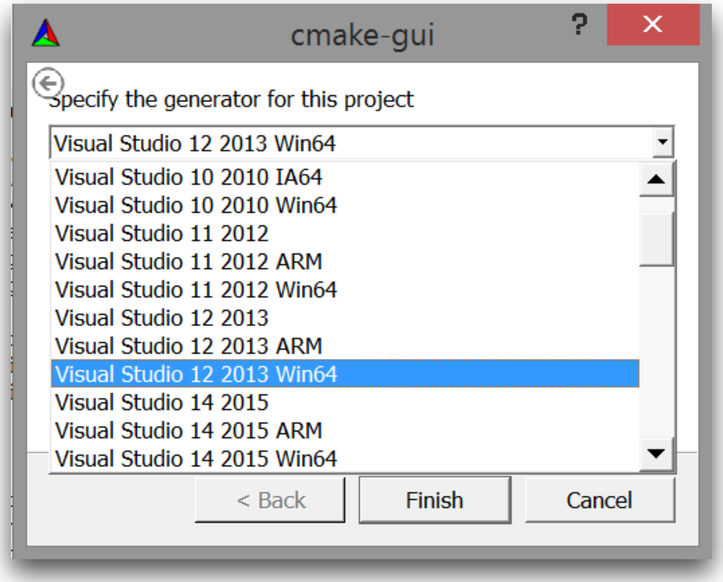
Minh Dang

Structure of Practicals

Week	Topic	Practical	Homework
2	2D OpenGL	Intro to GLSL Interfacing GLSL / C++ Working with Textures	Triangle Spirals Checkerboard 2D Planet System
3	3D OpenGL	3D Transformations Orthographic Projections Index Buffers	Perspective Projections Virtual Trackball Animated Triangle Grid
4	Screen Space Techniques	**invited talk:** R.Ziegler (45m) Anttweakbar: intro Gouraud shading	Shading: phong, flat, toon, artistic Spot light

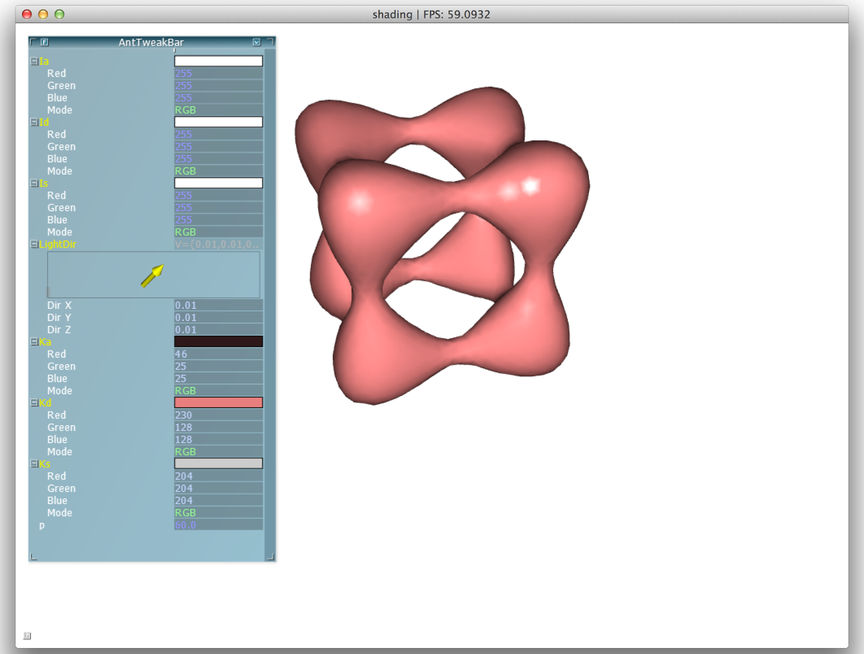
Preliminaries

- Code in the git repository
  - `git clone https://git.epfl.ch/repo/icg15.git`
  - `git pull` (to update already checkout code)
  - if `pull` doesn't work: `git reset –hard HEAD`
    ATTENTION: Backup before doing this
- For Windows users: Use x64 compiler
- These slides are in `PDF/Practical-Homework#4.pdf`



Anttweakbar

- Useful tool to tweak parameters
- Download and build source code from
  `http://anttweakbar.sourceforge.net`
- For Mac OsX: `brew install anttweakbar`
- Read `main.cpp` for an example



Loading a triangle mesh

- Triangles + connectivity (common edges or vertices)
- Draw using **`index buffer`**

get indices using mesh iterators

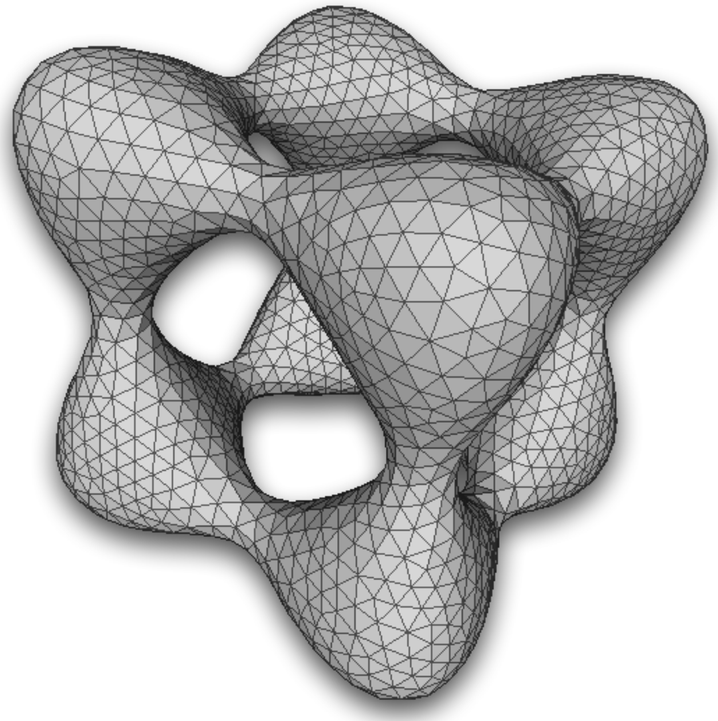
```
///--- Index Buffer
std::vector<unsigned int> indices;
for(Surface_mesh::Face_iterator fit = mesh.faces_begin(); fit != mesh.faces_end(); ++fit) {
    unsigned int n = mesh.valence(*fit);
    Surface_mesh::Vertex_around_face_circulator vit = mesh.vertices(*fit);
    for(unsigned int v = 0; v < n; ++v) {
        indices.push_back((*vit).idx());
        ++vit;
    }
}
```

create an index buffer

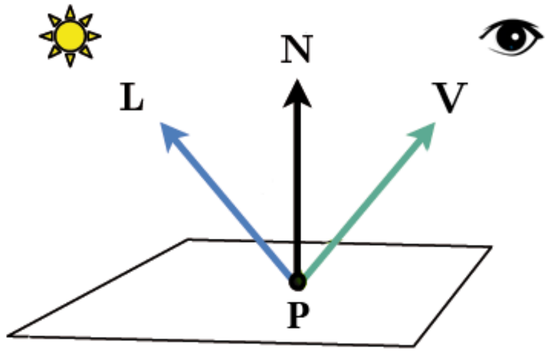
```
GLuint _vbo_indices;
glGenBuffers(ONE, &_vbo_indices);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, _vbo_indices);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(unsigned int), &indices[0], GL_STATIC_DRAW);
check_error_gl();
```

draw

```
glDrawElements(GL_TRIANGLES, /*#vertices*/ 3*mesh.n_faces(),
               GL_UNSIGNED_INT, ZERO_BUFFER_OFFSET);
```

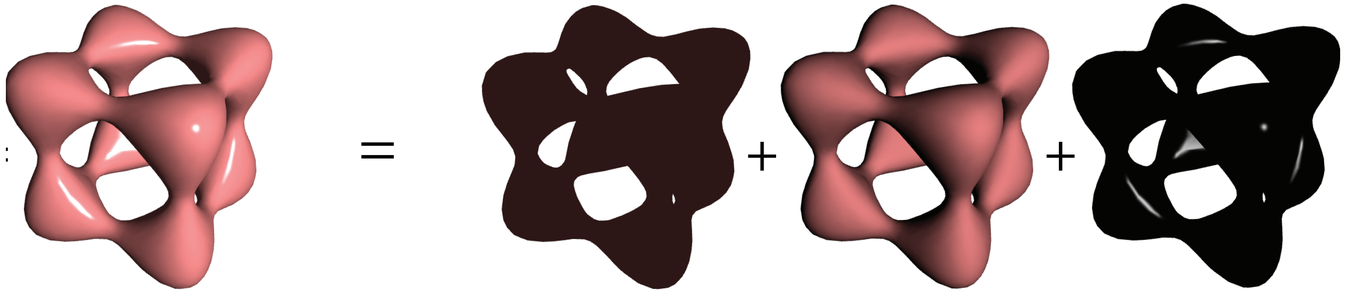


Phong lighting model

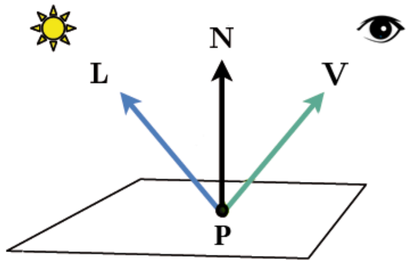


$$\mathbf{I} = \mathbf{I}_a\mathbf{k}_a + \mathbf{I}_d\mathbf{k}_d(\mathbf{N}\cdot\mathbf{L}) + \mathbf{I}_s\mathbf{k}_s(\mathbf{V}\cdot\mathbf{R})^p$$

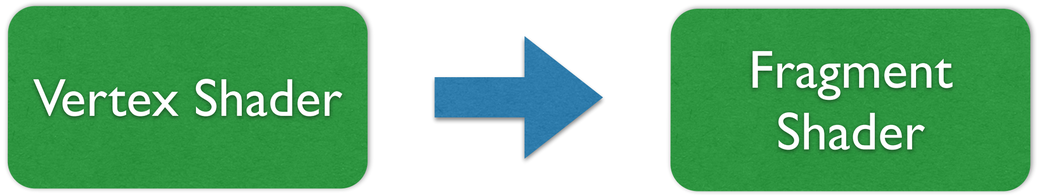
ambient diffuse specular



Practical 4: Gouraud shading



$$\mathbf{I} = \mathbf{I}_a\mathbf{k}_a + \mathbf{I}_d\mathbf{k}_d(\mathbf{N}\cdot\mathbf{L}) + \mathbf{I}_s\mathbf{k}_s(\mathbf{V}\cdot\mathbf{R})^p$$



Per vertex color

Per pixel color (interpolate)

TODO 0.1 Complete `gouraud_vshader.glsl`

a. Transform `vnormal` to camera space `normal_mv`

b. Compute light direction `light_dir`

c. Compute view direction `view_dir`

d. Compute vertex color `vcolor`

TODO 0.2 Complete `gouraud_fshader.glsl`

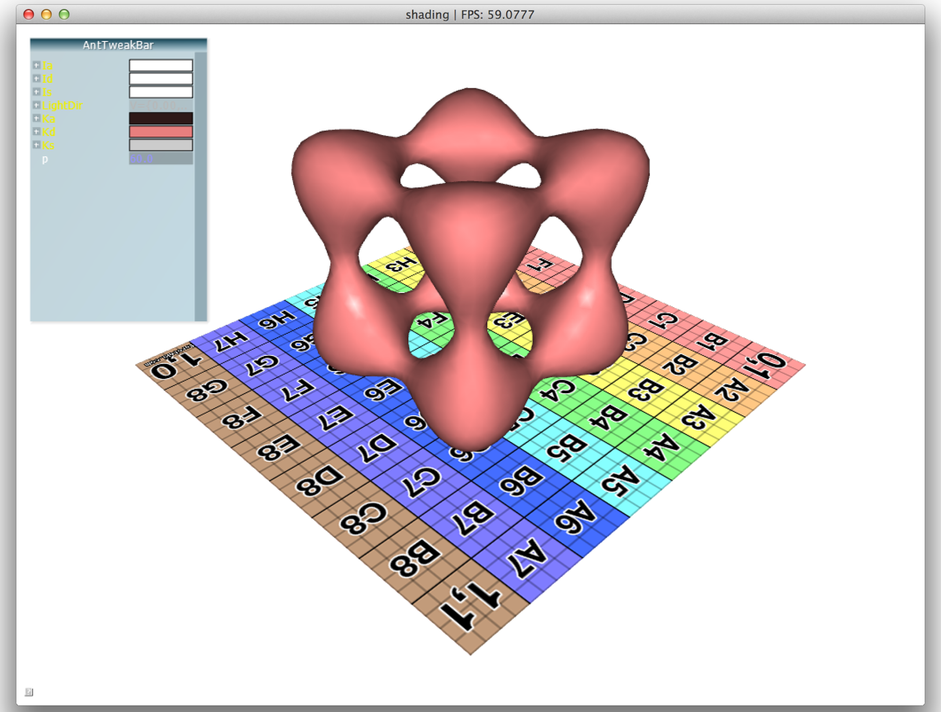
Clamp negative values of **N.L** and **V.R** to **0**!

Homework 4

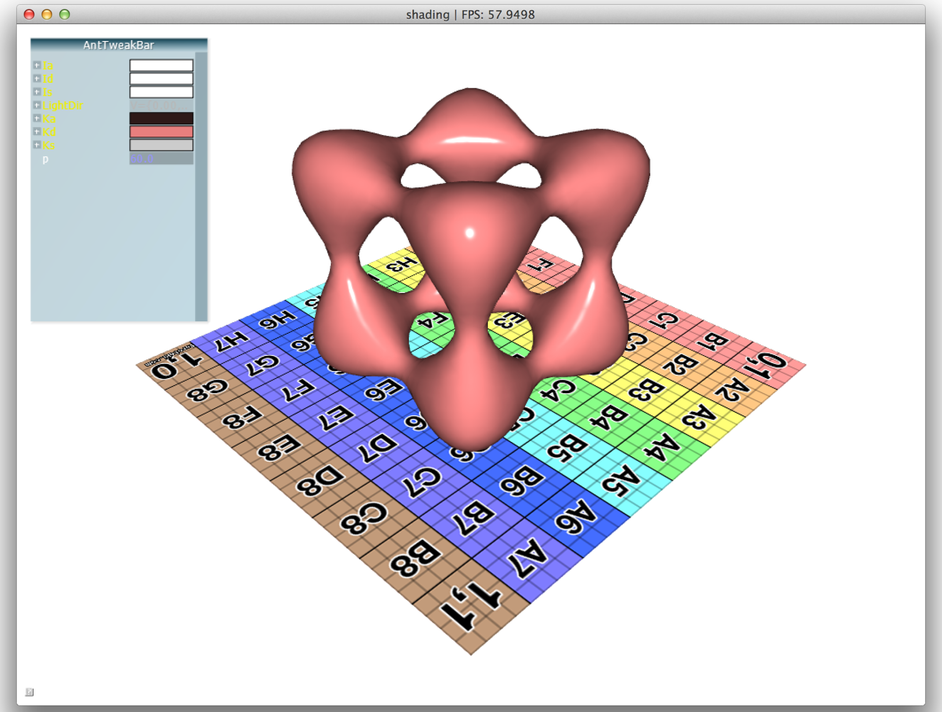
Homework 4

- 5 exercises
  - 1. Phong shading **1 point**
  - 2. Toon shading **1 point**
  - 3. Artistic shading **1 point**
  - 4. Flat shading **2 points**
  - 5. Spot light **1 point**
- Switch between shading methods by keys ‘0’ to ‘5’ (See `main.cpp`)
- Same submission criteria as before
- Moodle submission deadline **Thursday, 19 March 10 am**
- Homework code in `hw4_shading`

Phong vs Gouraud shading

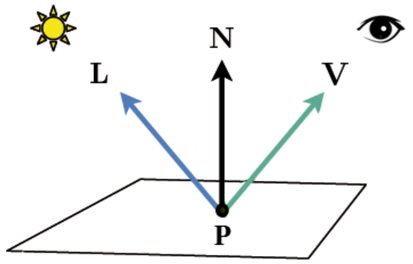


Gouraud

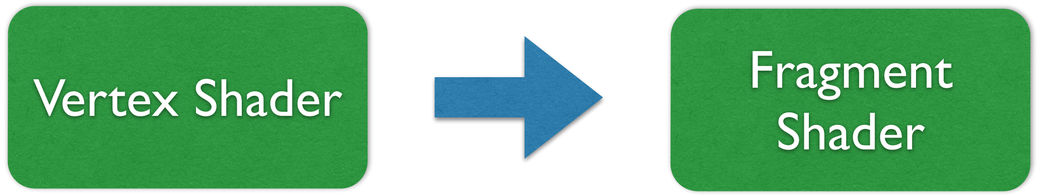


Phong

1. Phong shading



$$\mathbf{I} = \mathbf{I}_a\mathbf{k}_a + \mathbf{I}_d\mathbf{k}_d(\mathbf{N}\cdot\mathbf{L}) + \mathbf{I}_s\mathbf{k}_s(\mathbf{V}\cdot\mathbf{R})^p$$



Per vertex
normal

Per pixel
normal
(interpolate)

TODO 1.1 Complete `phong_vshader.glsl` (0.5 point)

a. Transform `vnormal` to camera space `normal_mv`

b. Compute light direction `light_dir`

c. Compute view direction `view_dir`

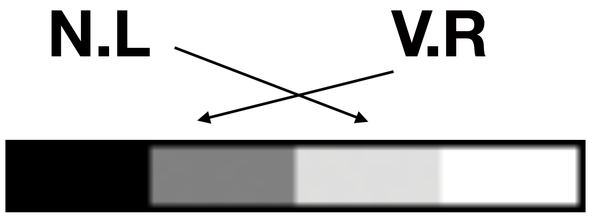
TODO 1.2 Complete `phong_fshader.glsl` (0.5 point)

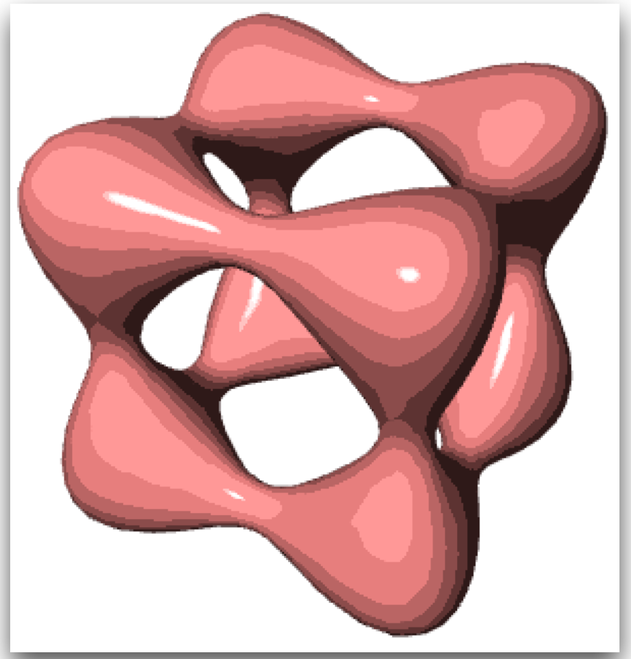
a. Get `normal_mv`, `light_dir` and `view_dir` from vertex shader

b. Compute per pixel color

2. Toon shading

- Mimic the rendering style of cartoon
- Also called “cel shading”
- Use a *discrete* range of tones to shade the surface
- In Phong lighting model:
  - Map **N.L** and **V.R** to a discrete range stored in an 1D texture (revise practical 2)
  - The texture is generated for you and stored at _tex1D



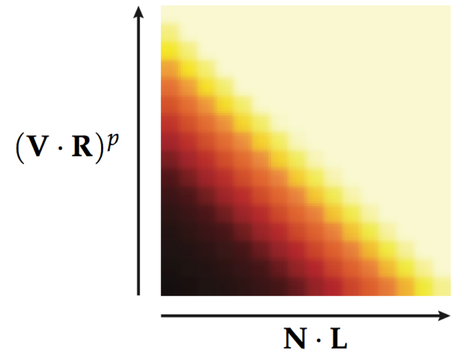


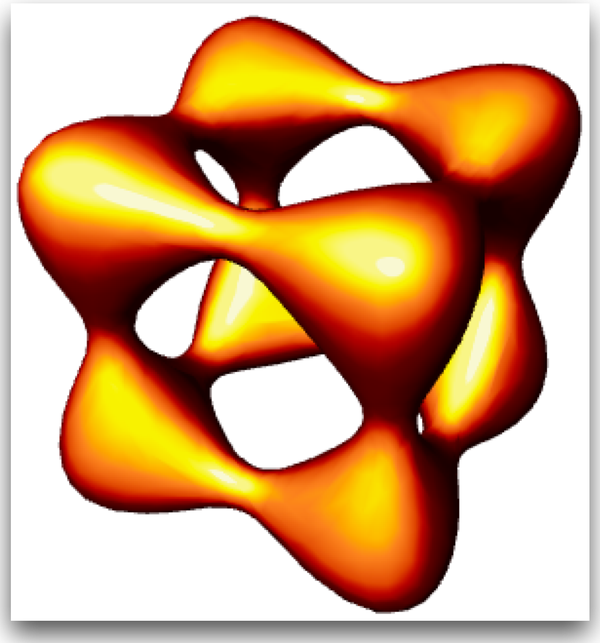
TODO 2.1 Bind `_tex1D` in `main.cpp/bindShader()` (0.25 point)
TODO 2.2 Complete `toon_fshader.glsl` (0.75 point)
You can use the same vertex shader as `phong_vshader.glsl`

3. Artistic shading

Use a 2D texture to shade the object in an artistic fashion

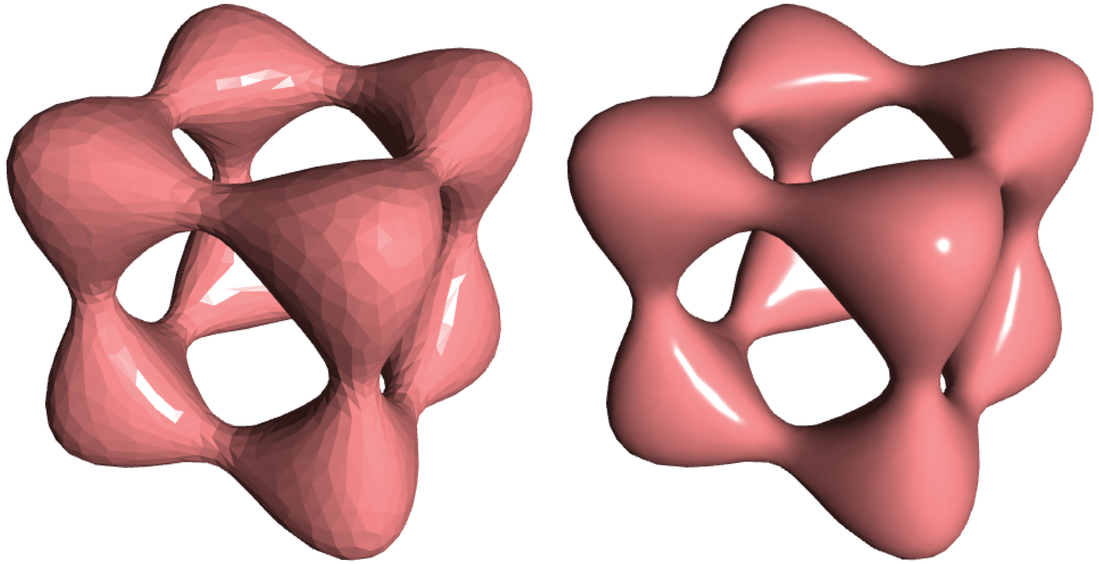
http://maverick.inria.fr/Publications/2006/BTM06a/x-toon.pdf





TODO 3.1 Bind `_tex2D` in `main.cpp/bindShader()` (0.25 point)
TODO 3.2 Complete `art_fshader.glsl` (0.75 point)
You can use the same vertex shader as `phong_vshader.glsl`

4. Flat shading



- Flat shading shades each triangle based on the angle between the triangle normal and the light direction.
- This is contrast to smooth shading (phong, gouraud)
- Using dFdx and dFdy in fragment shader to compute the triangle normal

Reading exercise!

4. Flat shading

TODO 4.1 Complete `flat_vshader.glsl` (0.5 point)

a. Compute light direction `light_dir`

b. Compute view direction `view_dir`

c. Pass `view_dir, light_dir, vpoint_mv` to the fragment shader

TODO 4.2 Complete `flat_fshader.glsl` (1.5 point)

a. Compute the triangle normal using dFdx, dFdy

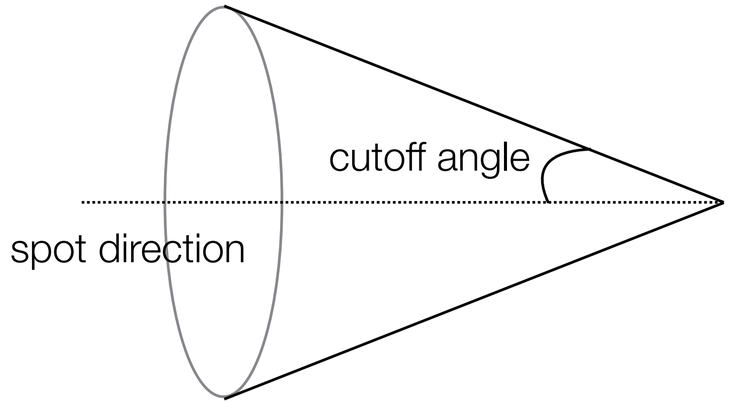
b. Add ambient, diffuse and specular terms to shade the mesh

5. Spot light (phong shading)

- Light rays are restricted by a cone of light
- Inside the cone

$$I_{spot} = I * SpotEffect$$

$$SpotEffect = (LightDir.SpotDir)^{SpotExp}$$



TODO 5 Complete `spot_fshader.glsl` (1 point)
You can use the same vertex shader as `phong_vshader.glsl`

Use Phong shading model!