Fondamenti di Computer Graphics LM

GLSL

Questa esercitazione puo' essere eseguita sia in ambiente Windows che Linux.

✓ 1. **Exercise 1: Wave motion**

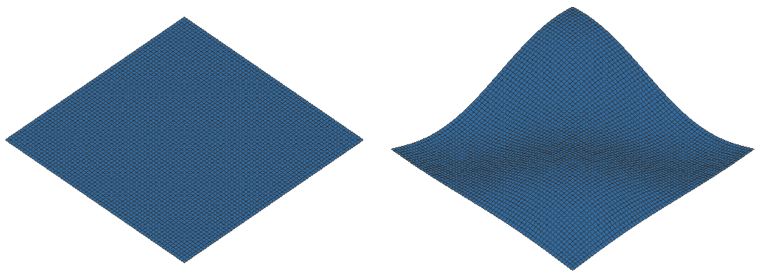


Figure 1: A screenshot from the `WAVE` application. (a) the initialized field. (b) after modifying the vertex positions via vertex shader.

The `WAVE` application demonstrates the possibility of geometry modification of an height field mesh by modifying the vertex positions in the vertex shader (see Fig. 1). The `WAVE` application is implemented in the source files:

- WAVE/wave.cpp
- WAVE/v.glsl
- WAVE/f.glsl

Extend the `WAVE` application implementing the following features:

(a) using elapsed time t, given wave amplitude a and frequency ω, perform the height modification (y-coordinate) of vertex v in vertex shader utilizing formula

$$v_y = a \sin(\omega t + 5v_x) \sin(\omega t + 5v_z) \, .$$

(b) when the user clicks the left mouse button, set a different amplitude, cycling through the values: $[0.05, 0.1, 0.2]$.

(c) when the user clicks the right mouse button, set a different oscillation frequency, cycling through the values: $[0.0005, 0.001, 0.002]$.

✓ 2. **Exercise 2: Particle System**

The `PARTICLE` application implements a simple particle system simulation (see Fig. 2). The particle position is computed in the vertex shader based on velocity and acceleration (gravity) at the current simulation time step. Press the space key to restart the system and randomize particles velocities and color. The `PARTICLE` application is implemented in the source files:

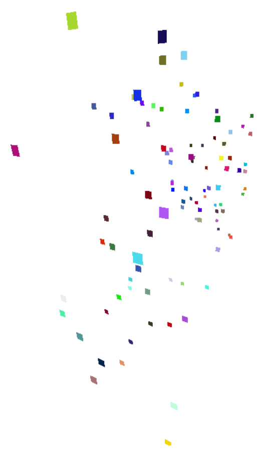


Figure 2: A screenshot from the `PARTICLE` application after implementing particles movement in the z direction.

- PARTICLE/particle.cpp
- PARTICLE/v.glsl
- PARTICLE/f.glsl

Extend the `PARTICLE` application implementing the following features:

(a) height dependent point size. The higher, the bigger (*TIP:* use the built-in variable `gl_PointSize`)

(b) make the particles move also in the z direction

✓ 3. **Exercise 3: Phong Lighting**

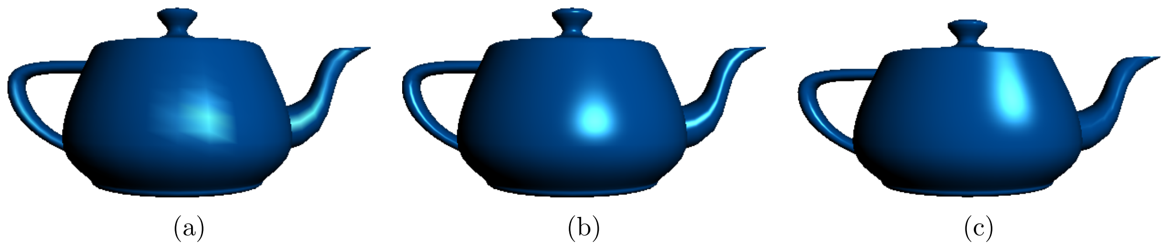


Figure 3: Phong lighting model implemented in the `PHONG` application with fragment color interpolation (a) and with normals interpolation in the fragment shader (b). After implementing the exact reflection ray (c).

The `PHONG` application implements the Phong lighting model in two different GLSL programs with fragment color interpolation and with a more precise normal interpolation performed in the fragment shader (see Fig. 3). The `PHONG` application is implemented in the source files:

- PHONG/phong.cpp
- PHONG/v1.glsl

- PHONG/f1.glsl
- PHONG/v2.glsl
- PHONG/f2.glsl

Extend the `PHONG` application implementing the following features:

(a) add a third shader (starting from `v2.glsl` and `f2.glsl` that, in the computation of the specular coefficient K_s, implements the exact light reflection ray instead of the approximate half-way vector. See Fig. 3c for the expected result and note that it's not very different since the half-way vector represents a good approximation of the reflection ray.

(b) show three teapots in the same scene with the three different shaders

✓ 4. **Exercise 4: Toon Shading**

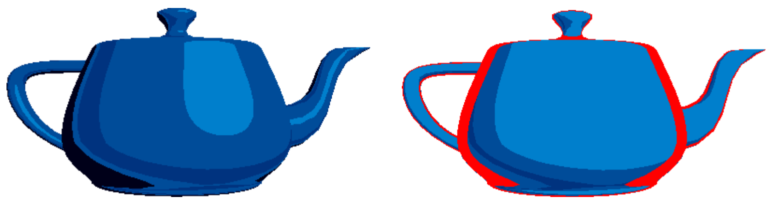


Figure 4: A screenshot from the `NONPHOTO` application (a). After adding a red outline (b).

The `NONPHOTO` implements a non-photorealistic rendering commonly known as "Toon shading" via vertex and fragment shaders (see Fig. 4a). The `NONPHOTO` application is implemented in the source files:

- NONPHOTO/nonphoto.cpp
- NONPHOTO/v.glsl
- NONPHOTO/f.glsl

Extend the `NONPHOTO` application implementing the following features:

(a) add an outline/silhouette effect (see Fig. 4b). *TIP:* Think about the angle formed by the eye vector $\overrightarrow{E}$ with the normal vector $\overrightarrow{N}$.

✓ 5. **Exercise 5: Morphing**

The `MORPH` application implements a morphing between silhouettes of two triangles (see Fig. 5) via vertex shader. The implementation source files are located in:

- MORPH/morph.cpp
- MORPH/v.glsl
- MORPH/f.glsl

Extend the `MORPH` application implementing **one** of the following features:

(a) draw the filled triangle with colours assigned to its vertices and implement the colour morphing/variation over time using the vertex shader.

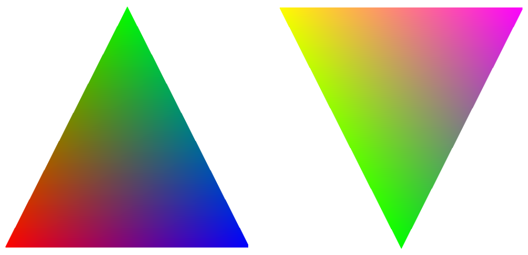


Figure 5: Screenshots of the two different states from the `MORPH` application after adding a per-vertex color change.

(b) via the vertex shader, implement a morphing between triangle and another shape outline (e.g. square, hexagon).

✓ 6. **Example 6: Bump Mapping**

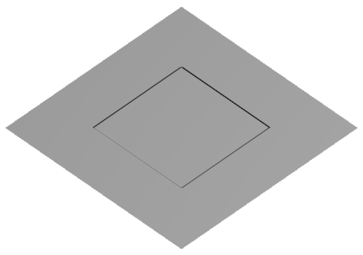


Figure 6: A screenshot from the `BUMPMAP` application.

The `BUMPMAP` implements a bump mapping texture for a given height field via vertex shader (see Fig. 6). The implementation source files are located in:

- BUMPMAP/bumptex.cpp
- BUMPMAP/v.glsl
- BUMPMAP/f.glsl

Extend the `BUMPMAP` application implementing the following features:

(a) change the input data height field shape into something else (e.g. stripes of different heights).

✓ 7. **Exercise 7: Cube environment mapping**

The `CUBEMAP` implements an cube environment texture mapping on a reflective object – teapot, while the environment cube sides are coloured by different colours (see Fig. 7). The implementation source files are located in:

- CUBEMAP/cubetexshader.cpp
- CUBEMAP/v.glsl
- CUBEMAP/f.glsl

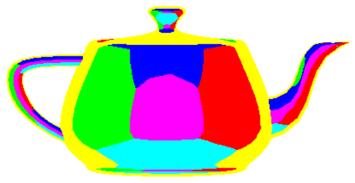
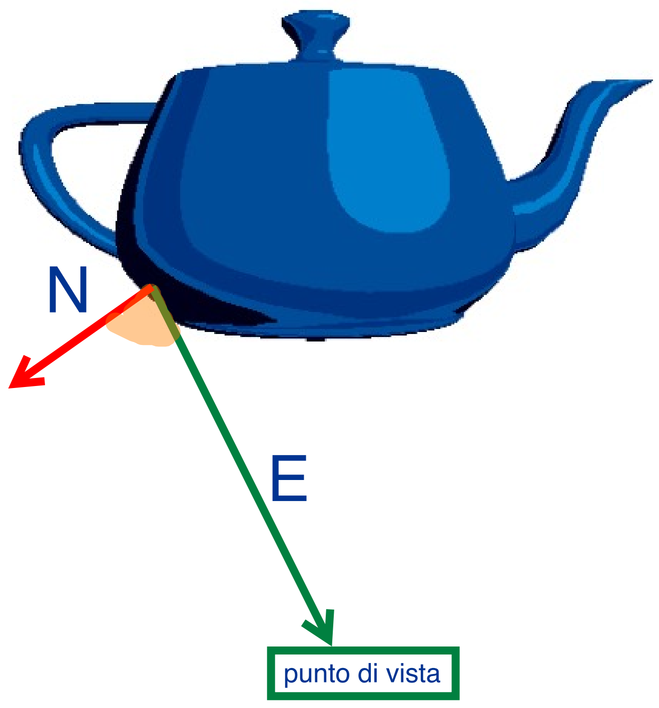


Figure 7: A screenshot from the `CUBEMAP` application.

Extend the `CUBEMAP` application implementing the following features:

(a) map onto the cube sides, instead of single colours, some texture images (e.g. one set of those provided in `Lab-05`).


N
E
punto di vista