Prof. Dr. Leif Kobbelt
Patrick Schmidt, Joe Jakobi

Basic Techniques in Computer Graphics

Assignment 10

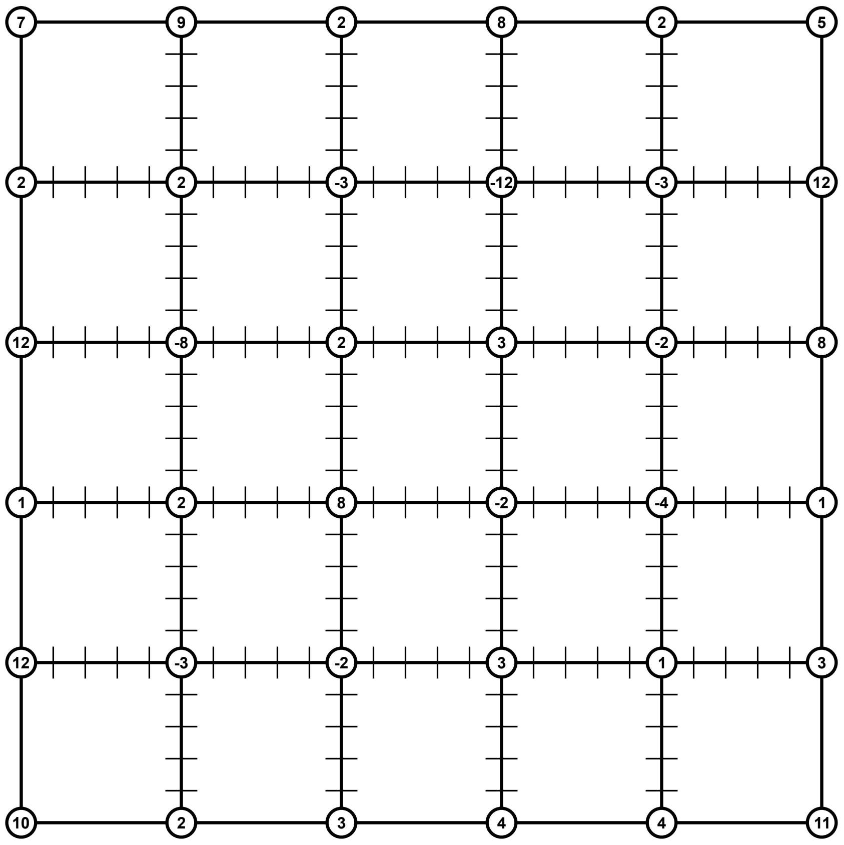
Date Published: January 10th 2023, Date Due: January 17th 2023

- All assignments (programming and theory) have to be completed in teams of 3–4 students. Teams with fewer than 3 or more than 4 students will receive no points.
- Hand in **one solution per team per assignment**.
- Every team must work independently. Teams with identical solutions will receive no points.
- Solutions are due on January 17th 2023 via Moodle. Late submissions will receive zero points. No exceptions!
- Instructions for **programming assignments**:
  - Make sure you are part of a Moodle group with 3-4 members. See "Group Management" in the Moodle course room.
  - Download the solution template (a zip archive) through the Moodle course room.
  - Unzip the archive and populate the `assignmentXX/MEMBERS.txt` file. The names and student ids listed in this file **must match** your moodle group **exactly**.
  - Complete the solution.
  - Prepare a new zip archive containing your solution. It must contain exactly the files that you changed. **Only change the files you are explicitly asked to change in the task description.** The directory layout must be the same as in the archive you downloaded. (At the very least it must contain the `assignmentXX/MEMBERS.txt`.)
  - One team member uploads the zip archive through Moodle before the deadline, using the group submission feature.
  - Your solution must compile and run correctly **on our lab computers** by only inserting your **assignment.cc** and **shader files** into the Project. If it does not compile on our machines, you will receive no points. If in doubt you can test compilation in the virtual machine provided on our website.
- Instructions for **text assignments**:
  - Prepare your solution as a single pdf file per group. Submissions on paper will not be accepted.
  - If you write your solution by hand, write neatly! Anything we cannot decipher will receive zero points. No exceptions!
  - Add the names and student ID numbers of all team members to every pdf.
  - Unless explicitly asked otherwise, always justify your answer.
  - Be concise!
  - Submit your solution via Moodle, together with your coding submission.

Prof. Dr. Leif Kobbelt
Patrick Schmidt, Joe Jakobi

Exercise 1 Indirect Rendering of Implicit and Volumetric Geometry **[12 Points]**

Instead of rendering implicit or volumetric functions directly using ray traversal techniques, one can also take an indirect route and first extract a polygonal mesh representation that is then rendered using the standard triangle rasterization pipeline. For this purpose the Marching Cubes (MC) algorithm has been presented. Given an implicit volumetric representation in form of discrete sample values at grid points, it constructs a polygonal representation of the zero-level surface (or any other iso-level).



(a) Surface Extraction **[8 Points]**

Execute the Marching Squares algorithm (the 2D equivalent of MC) on the discrete implicit data given above to extract a zero-level representation, i.e. draw the resulting set of line segments (instead of polygons in 3D) into the raster. Whenever there is an ambiguity, assume the square center has a positive value. Make sure to get the geometry right, not only the topology, i.e. position the samples at the correctly interpolated positions.

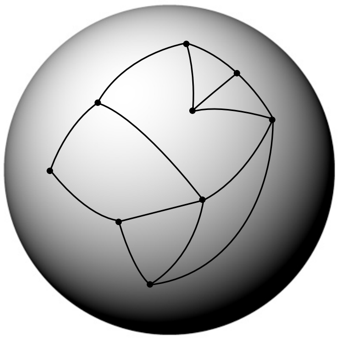
(b) Topological Equivalence **[4 Points]**

Is the zero level representation which is extracted from the Marching Squares algorithm always topologically equivalent to the original implicitly defined object? If it is, explain your answer. If not, give a counter example.

Exercise 2 Meshes [16 Points]

(a) [3 Points]

Consider the depicted closed mesh (drawn on a spherical surface to underline its closedness). Depict the dual mesh of this mesh (in a way that every vertex, edge, and face is visible, i.e. don't draw on the backside of the sphere). Remember that duality in its core is a purely topological concept, i.e. you can place the dual vertices arbitrarily in the primal faces, edges don't have to be straight, faces don't have to be planar.

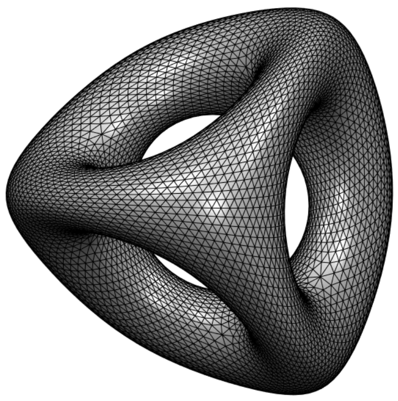


(b) [8 Points]

In the lecture you have seen how the average vertex valence of a triangle mesh as well as the relation between the number of faces and the number of vertices in a triangle mesh can be derived from the Euler formula. Remember that the Euler formula not only holds for triangle meshes, but also for arbitrary polygon meshes. Derive the average vertex valence of a closed hexagon mesh (of genus 1) as well as the relation between the number of faces and the number of vertices in such a mesh from the Euler formula.

(c) [5 Points]

Consider the depicted closed triangle mesh with 19968 edges. What is the genus of this object? Derive how many vertices and triangles this mesh has got.



Exercise 3 Environment Mapping **[10 Points]**

(a) **[4 Points]**

Given is an object surrounded by a cube map. The viewer is positioned at $v = \begin{pmatrix} 2 \\ 1 \\ 8 \end{pmatrix}$ and looking at point $p = \begin{pmatrix} 3 \\ 5 \\ 6 \end{pmatrix}$ with the normal $n_p = \begin{pmatrix} \sqrt{2}/2 \\ \sqrt{2}/2 \\ 0 \end{pmatrix}$ at that point. Calculate the reflected view vector r.

(b) **[3 Points]**

After obtaining the reflected vector, the color lookup is performed independently from the position the camera looks at. Briefly explain why this is the case.

(c) **[1 Points]**

Using the Environment Mapping algorithm for Cube Maps, which side of the Cube Map do we have to sample? Choose from $\{\text{posX}, \text{posY}, \text{posZ}, \text{negX}, \text{negY}, \text{negZ}\}$. Justify your answer!

(d) **[2 Points]**

Using the Environment Mapping algorithm for Cube Maps, compute the texel coordinates. Make sure to adjust your result to the correct interval! Assume a texture size of 1024×1024 pixels. What are the selected pixels coordinates?

Exercise 4 Programming **[8 Points]**

In computer graphics *Environment Mapping* is a common way to efficiently simulate reflections of the environment of an object on its reflective surface. In practice, *Cube-* and *Sphere-Mapping* are the methods of choice for many applications. In both techniques textures of the environment are precomputed and the reflected color is fetched according to the respective viewing ray reflected on the object's surface. Usually, a *Sphere Map* is a photograph of a reflective sphere that mirrors *almost* the entire surrounding environment. In contrast, a *Cube Map* consists of six images. Each image shows the surrounding scene as seen from the center of a cube in the direction of each of the cube's facets. In this week's practical exercise your task is to implement *Sphere-Mapping*, while a *Cube Map* is already implemented for comparison.

In OpenGL, sphere maps are treated as simple 2D textures. In order to get the correct texture coordinate for each fragment, you will have to create a mapping from the reflected viewing direction vector to the two-dimensional texture coordinates by yourself.

In the provided code you will find shaders that implement basic texturing and lighting using the Phong-Blinn model with Phong shading. The meshes are already equipped with lighting that is displayed in the initial scene. This we will refer to as the *original lighting* in the following. The scene also contains a *sky box* which is basically just a cube that encloses the visible scene onto which the respective cube map is projected. This is a very handy visual tool to verify whether the reflections look correct.

The reflection vectors for the texture look-up are in *Global Space*, which is already correct, so you do not have to do any transformations before computing the texture coordinates. Different from the previous assignments, this time the draw callback function has four parameters:

meshNumber (toggled with key 2**)** If this variable is `true` the bunny mesh should be displayed in the scene. If `false` the bunny should be replaced by the sphere.

cubeMapping (toggled with key 3**)** If this variable is `true` use the cube map to compute the environment map. If the value is `false` use the sphere map.

debugTexture (toggled with key 4**)** If this variable is `true` use the debug textures for the currently active environment map. These texture have the appendix "`Debug`" or "`x`" in its file name. If the value is `false` use the normal texture files.

environmentOnly (toggled with key 5**)** If this variable is `true` use texture blending to display the object's original lighting blended with 0.1 times the values from the currently active environment map. If `false` only show the environment map.

Furthermore, you can switch between rotation of the object itself and rotating the camera around the object via pressing key 1. The necessary transformations are already implemented in the given code. In this exercise you only have to edit the fragment shader *envmap.fsh*. Now, proceed as follows:

(a) **[6 Points]**

Implement the sphere mapping technique inside the fragment shader. Verify your results using the debug textures! The mapping of the reflected vectors to texture coordinates has to be done according to what was discussed in the lecture. Note that in the method from the lecture the computed values of the texture coordinates are in the range $[-1, 1]$. In order to comply with OpenGL's texture coordinate format, they will have to be scaled to the range $[0, 1]$.

(b) **[2 Point]**

Integrate texture blending between the respective mesh's original lighting and the environment map. Multiply a factor of 0.1 to the environment map.

If everything is implemented correctly, your scene should be similar to those shown in Fig. 1 for the different parameters.

Don't worry if the resulting images generated with the two different methods are not *exactly* identical!

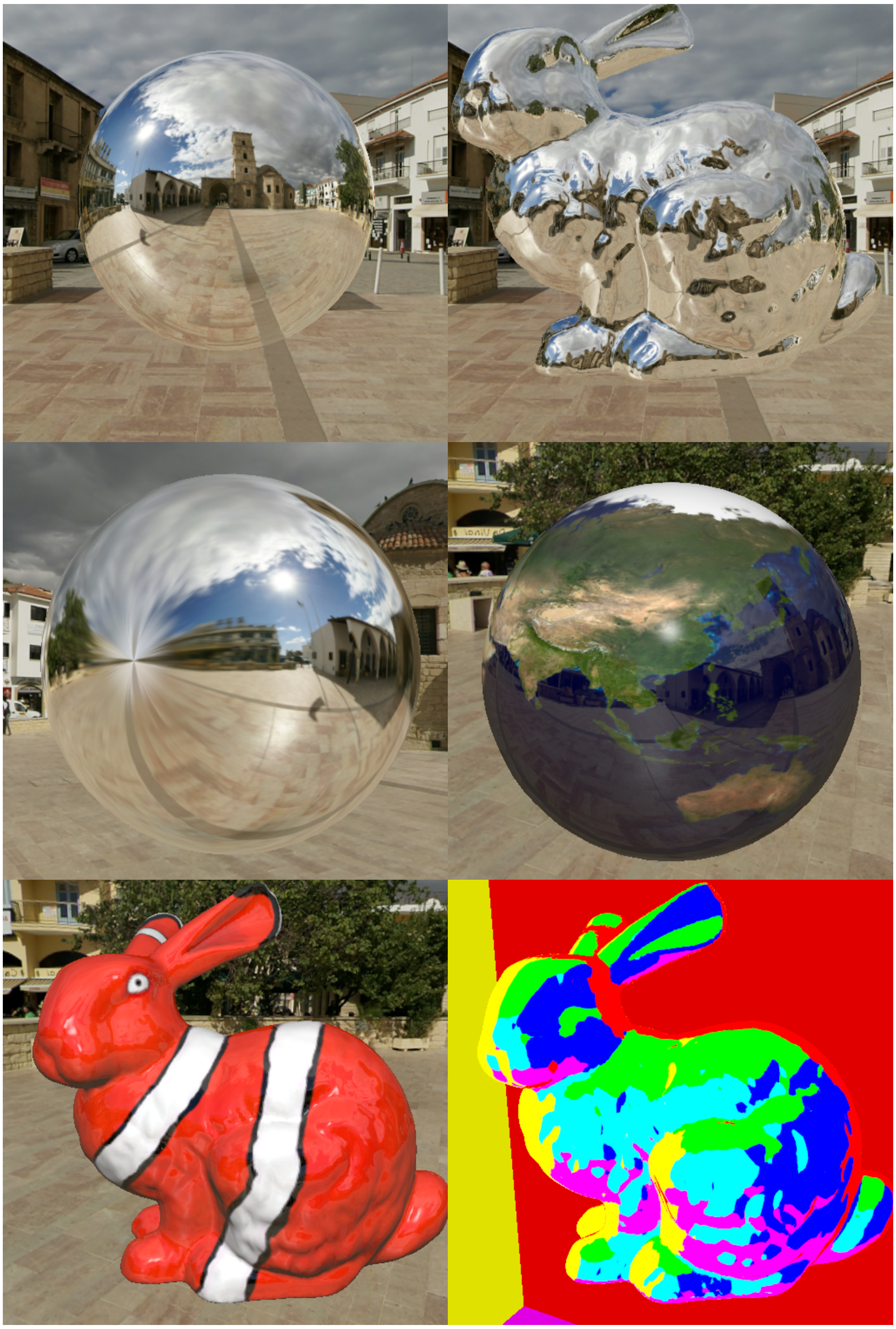


Figure 1: Screenshots showing the scene with different parameter values.