Prof. Dr. Leif Kobbelt
J. Born, M. Ibing,
J. Möbius, P. Schmidt

Visual Computing Institute | RWTH AACHEN UNIVERSITY

Basic Techniques in Computer Graphics

Assignment 7

Date Published: December 12th 2016, Date Due: December 19th 2016

- All assignments (programming and text) have to be done in teams of 3–4 students. Teams with less 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 18:00 on December 19th 2016. Late submissions will receive zero points. No exceptions!
- Instructions for **programming assignments**:
  - Download the solution template (a zip archive) through the L^{2}P course room.
  - Unzip the archive and populate the `assignmentXX/MEMBERS.txt` file. Any team member not listed in this file will not receive any points! (Also see the instructions in the file.)
  - Complete the solution.
  - Prepare a new zip archive containing your solution. It must contain exactly those files that you changed. **Only change those 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`.)
  - Upload your zip archive through the L^{2}P before the deadline.
  - Your solution must compile and run correctly **on our lab computers** using the exact same `Makefile` provided to you. If it does not, you will receive no points.
- Instructions for **text assignments**:
  - Prepare your solutions on paper.
  - If you write your solution by hand, write neatly! Anything we cannot decipher will receive zero points. No exceptions!
  - If you hand in more than one sheet, staple your sheets together. (No paper clips!)
  - Put the names and matriculation numbers of all team members onto every sheet.
  - Unless explicitly asked otherwise, always justify your answer.
  - Be concise!
  - Put your solution into the designated drop box at our chair before the deadline. (1st floor, E3 building.)

Exercise 1 Barycentric Coordinates [3 Points]

In the lecture we encountered barycentric coordinates multiple times already: In the rasterization they can be used to check whether a point lies within a triangle, in our shading algorithms they provide a way to interpolate colors or normals and they are an important part of texture mapping. The definition of barycentric coordinates directly gives a way to compute them for a given point by solving a linear equation system. However this is not the most efficient way when barycentric coordinates have to be computed for multiple points in a row. Here we will use an alternative approach.

(a) [1 Point]

Give a parametric representation $l(\lambda)$ of a line in 2D defined by points $P_1, P_2 \in \mathbb{R}^2$ such that $l(0) = P_1$ and $l(1) = P_2$. Compute the intersection parameter as well as the intersection point of such a line with a scanline(line parallel to the x-axis) implicitly defined by $y = y_0$ where y_0 is the offset from the x-axis.

(b) [0.5 Points]

The points $A = (1, 1)^T, B = (5, 2.5)^T, C = (4, 5)^T$ form a triangle on the screen. Now consider the scanline given by $y = 4.0$. Compute the intersection parameter λ_0 and intersection point E of line $\overline{BC}$ with the scanline. Do the same for line $\overline{CA}$ to get λ_1 and point D.

(c) [0.5 Points]

For point $Q = (3.5, 4.0)^T$ on the scanline compute the parameter λ_2 on the line $\overline{DE}$.

(d) [1 Point]

Finally give the formula to compute the barycentric coordinates using $\lambda_0, \lambda_1, \lambda_2$ and compute the barycentric coordinates of Q. Hint: the definition of barycentric coordinates gives you an easy way to check if your solution is correct.

Exercise 2 Shadow Maps [2 Points]

Another method to render shadows, besides using shadow volumes, is the Shadow Maps algorithm presented in the lecture. Its basic idea is to render the scene from the perspective of the light source and store the depth values observed by the light source into a special texture, the so-called shadow map. When rendering the scene from the view of the camera, the shadow map is used to determine which points seen from the camera are also seen from the light source and are therefore lit.

(a) [0.5 Points]

Let M_c be the view matrix of the camera and let M_l be the view matrix of the light source (with respect to which the shadow map has been rendered). Given an arbitrary point $p \in \mathbb{R}^3$ in the local coordinate system of the camera, specify the formula that transforms it into its representation p' in the coordinate system of the light source.

(b) [0.5 Points]

Let $f_{\text{SM}} : \mathbb{R}^3 \to \mathbb{R}$ be the function that takes a point q in the coordinate system of the light source and returns the depth value stored in the shadow map at the position onto which q projects on the image plane of the light source. Specify the condition (in terms of M_c, M_l, and f_{SM}) that answers whether a point p (represented in the local coordinate system of the camera) lies in the shadow according to the Shadow Maps technique.

(c) [1 Point]

What are the two types of aliasing that can occur when using shadow maps and which one can be fixed by using Perspective Shadow Maps? Explain your answer in **at most 5 sentences in your own words**.

Prof. Dr. Leif Kobbelt
J. Born, M. Ibing,
J. Möbius, P. Schmidt

Exercise 3 Anti-Aliasing [3 Points]

(a) [0.5 Points per column]

Aliasing in the context of rendering can have the following causes:

- Texture alias: Introduced by point sampling textures which leads to jagged edges in the case of magnification and moirée pattern in the case of minification.
- Geometry alias: Caused by the binary decision for each pixel whether a pixel is rasterized or not. Noticeable on geometry silhouettes.
- Shader alias: Produced by the fragment shader. An example would be a fragment shader that outputs a black and white checkerboard based on the world coordinates.

There exist several methods to reduce the effect of those artifacts such as Mipmapping/linear interpolation, Multisample Anti-Aliasing(MSAA), Post-processing Anti-Aliasing(for example FXAA) and Supersample Anti-Aliasing(SSAA). Indicate which method helps against which type of alias by filling the table below with either check marks(the method reduces the type of alias) or crosses(the method does not help with the type of alias).

	Mipmapping	MSAA	FXAA	SSAA
texture				
geometric				
shader				

(b) [1 Point]

Shortly explain, **in your own words**, what *Multisampling* and *Supersampling* techniques do. Discuss the difference between both techniques.

Exercise 4 Programming [2 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 'm') 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 'c') 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 'x') 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.

mixEnvMapWithTexture (toggled with key 'e') 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 '`r`'. 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) [1.5 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) [0.5 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!

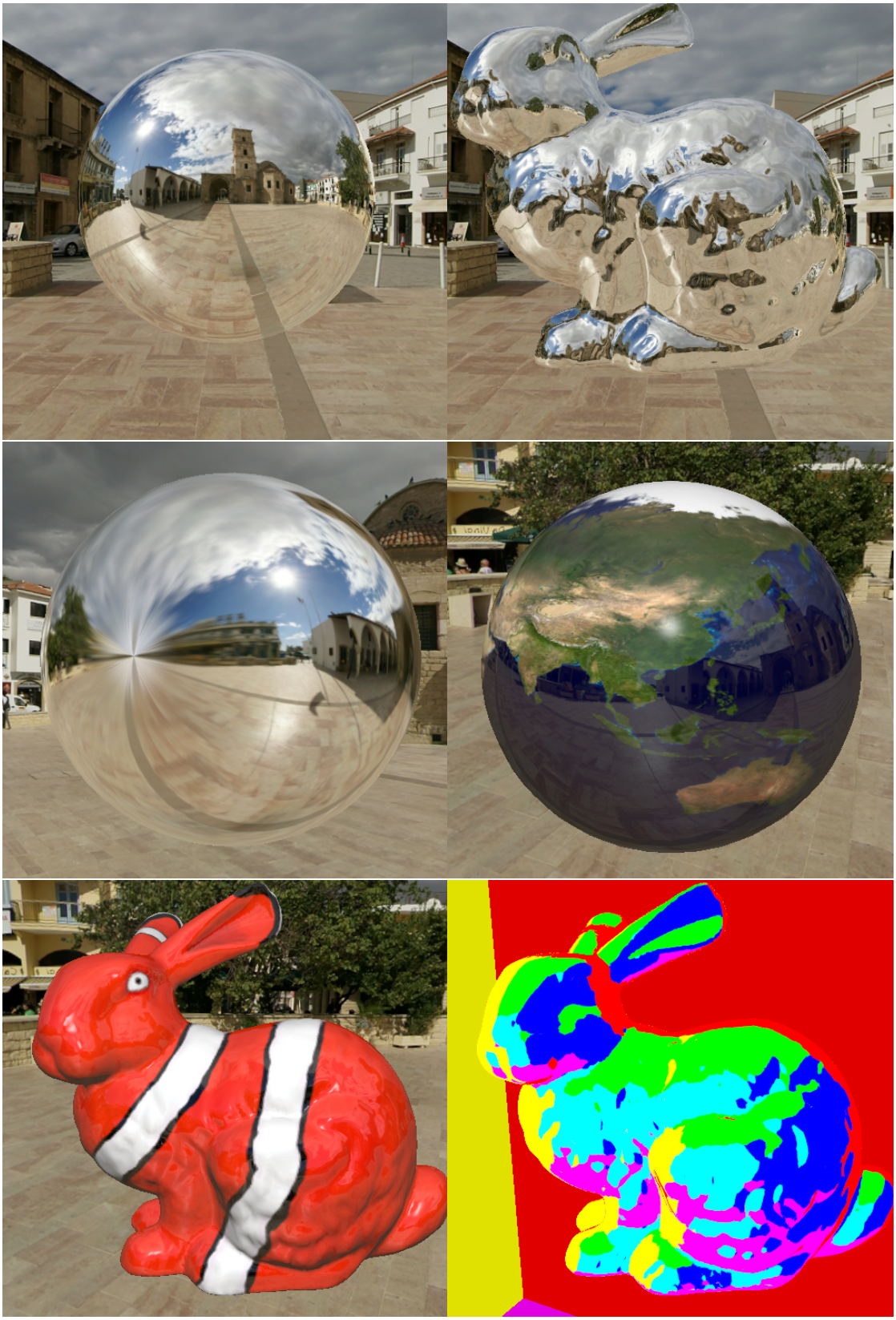

Figure 1: Screenshots showing the scene with different parameter values.