UNIVERSITÄT DES SAARLANDES
Prof.Dr.Philipp Slusallek and Dr. Tim Dahmen
Computer Graphics Group
Arsène Pérard-Gayot (perard@cg.uni-saarland.de)
Han Du (du@cg.uni-saarland.de)
Erik Herrmann (herrmann@cg.uni-saarland.de)

5. January 2017

Introduction to Computer Graphics
Assignment 9

Submission deadline for the exercises: 12. January 2017, before the lecture (14. January 2017 for the programming part)

The paper copies for the theoretical parts of the assignments will be collected at the beginning of the lecture on the due date. The programming parts must instead be sent by email to your respective tutors.
The code submitted for the programming part of the assignments is required to reproduce the provided reference images, and the submission ought to include the mandatory generated images. The submission should also contain a creative image show-casing all extra-credit features that have been implemented.
The projects are expected to compile and work out of the box. If it fails to do so on our Windows or Linux computers, we will try to do so on the CIP-pool students' lab as a "fallback".

9.1 Transforms in color space (10 Points)

Many of today's monitors allow to change the color temperature of the image, by moving the white point in the color diagram. .
A monitor corresponding to the standard "ITU-R BT.709? (which is identical to sRGB) uses the following color coordinates $(x_r, y_r) = (0.640, 0.330), (x_g, y_g) = (0.300, 0.600), (x_b, y_b) = (0.150, 0.060)$, and a D65 white point $(x_w, y_w) = (0, 3127, 0.3290)$.

a) While monitors can have their white point changed, its primary colors of the monitor cannot move in the color diagram. With this restriction, can you think of an easy way to perform a movement of the white point on a monitor?

b) Calculate the necessary transformation matrix to convert a color from the color space given above into a new one with the same primaries but with the white point at $(w_x, w_y) = (0.400, 0.330)$. Explain your solution. Since this transformation is not used to change the brightness, assume a normalized white point. Note that while sRGB colors are gamma corrected with gamma 2.2, in this exercise you are only asked to derive the transromation matrix and ignore the gamma correction.

9.2 Smooth Edges (10 Points)

Most models are described by a collection of flat surfaces. This becomes very apparent when doing lighting as each surface has a single, constant normal.
A common technique for keeping the polygon count low but making it less apparent, is to perturbe the normal vector depending on the location where the ray hit the flat surface. For example, a triangle can be given three fake normals on its vertices; when a ray hits the primitive, the normals are interpolated for the given hit point.
Your task is to implement a `SmoothTriangle` solid, which should perform this normal interpolation. The triangle intersection routine should remain unchanged with the exception for the returned normal within the `Intersection` structure.

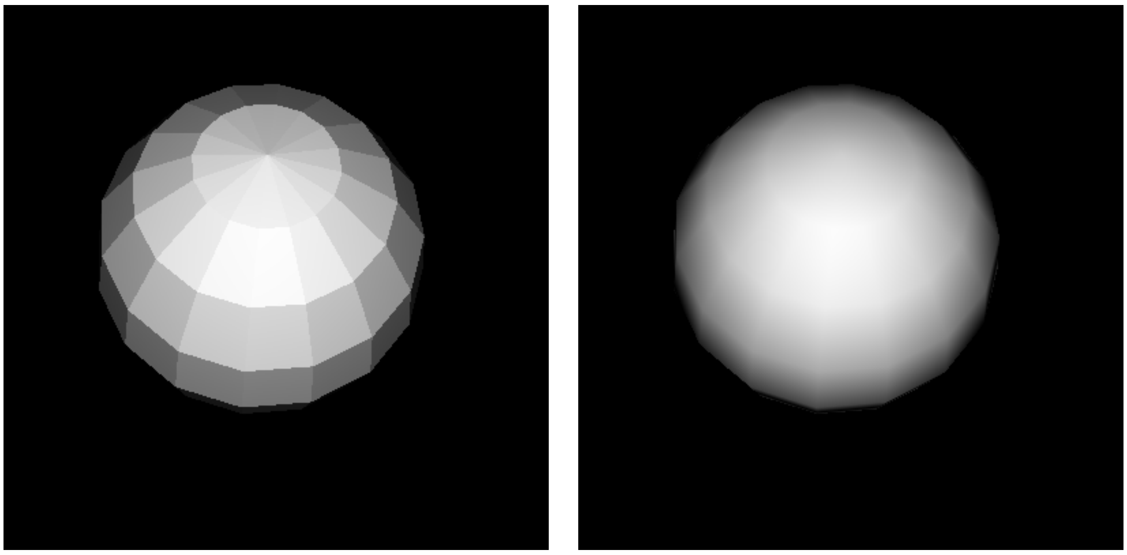

Figure 1: A tesselated sphere, without and with interpolated normals.

9.3 Bump Mapping (20 Points)

A common technique of increasing model complexity without actually adding to the geometry is to perturb the normal vectors returned when the geometry is intersected. In this exercise your task is to implement *bump mapping* where the normal vector perturbation is defined by a texture which represents an imaginary heightfield. This way, when the surface is lit, it seems to have bumps although it remains a single, flat geometry.
Your task is to implement `BumpMapper`, which holds a triangle solid as its base, and adds a bump texture onto it. The location of the texture is defined by texture coordinates at the triangle vertices. The parameter `vscale` controls the magnitude of the bumps.
When the underlying triangle is intersected, the bump mapper should perturb the normal. You will need to:

- Compute the bump map texture coordinates where the hit occured.
- Compute the gradient at the given texture coordinates. Assume that the image is gray. (`Texture::getColorDX`, `Texture::getColorDY`)
- Compute how the texture base vectors e_x, e_y map to world space w_x, w_y.
- Multiply w_x, w_y and the gradient to perturb the normal in world space.

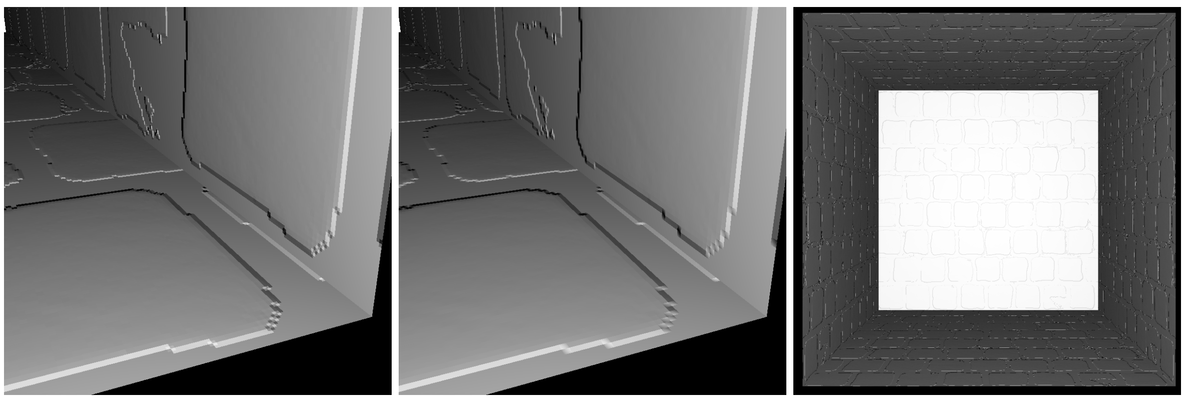

Figure 2: Bump mapping. Close-up, using nearest-neighbour and bilinear interpolation; and full scene.