UNIVERSITÄT DES SAARLANDES
Prof. Dr.-Ing. Philipp Slusallek
Computer Graphics Group
Arsène Pérard-Gayot (perard@cg.uni-saarland.de)

06. January 2019

Introduction to Computer Graphics
Assignment 8

Submission deadline for the exercises: 13. January 2019

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 marked as release before the beginning of the lecture on the due date.
The code submitted for the programming part of the assignments is required to reproduce the provided reference images. The submission ought to tag the respective commit in your group's git repository as well as attach the mandatory generated images to the release. 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. A successful build by Drone CI is a good indicator. 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".
To pass the course you need for every assignment at least 50% of the points.

8.1 Perspective Projection (15 Points)

a) Compute the point where two arbitrary parallel lines seem to intersect after being projected by the perspective projection P. For which parallel lines does no such intersection point exist?

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

All parallel lines converge at infinity(thats the point we computed, w

b) Compute the center of projection of the following perspective projection Q.

$$Q = \begin{pmatrix} \frac{3}{4} & 1 & -\frac{1}{4} & -\frac{1}{4} \\ 0 & 0 & \frac{1}{4} & \frac{5}{4} \\ -1 & 0 & 0 & 1 \\ 0 & 0 & \frac{1}{4} & \frac{5}{4} \end{pmatrix}$$

Virtual camera, then center of projecti

c) Compute the projection plane of projection Q'

$$Q' = \begin{pmatrix} 1 & \frac{1}{4} & -\frac{1}{4} & 0 \\ 0 & \frac{3}{4} & \frac{1}{4} & 0 \\ 0 & -\frac{1}{4} & \frac{5}{4} & 0 \\ 0 & \frac{1}{4} & -\frac{1}{4} & 1 \end{pmatrix}$$

alpha = 1, as x cannot have fixed value(if it d

8.2 Bresenham for Parabola (20 Points)*

Develop a mid-point algorithm to draw the fixed parabola $y = \frac{1}{12}x^2 + 2$ in the range of 0 to 6. You do not have to implement the algorithm, but to show how to initialize x, y, and the decision variable d and how to update them in the two cases.

8.3 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 mappe will need to:

vscale applied to bump in textu

- Compute the bump map texture coordinates where the
- 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.

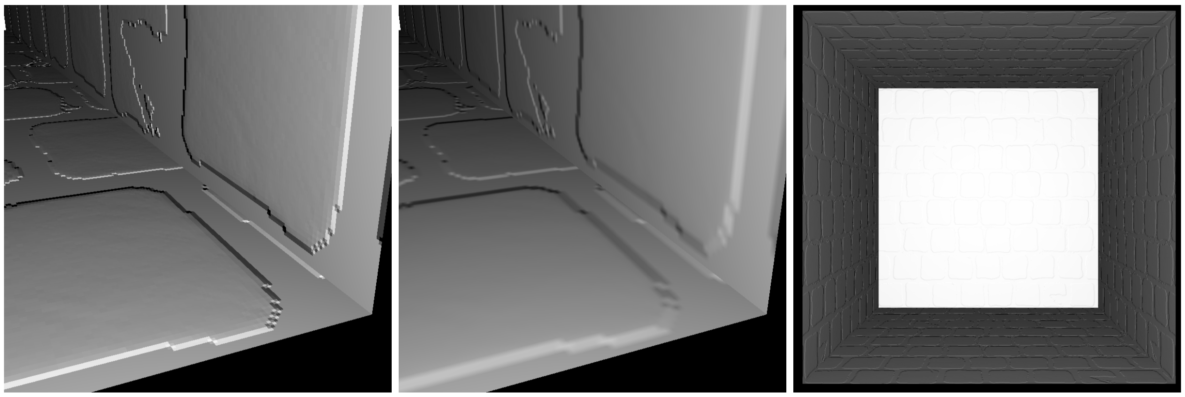

Figure 1: Bump mapping. Close-up, using nearest-neighbour and bilinear interpolation; and full scene.

8.4 Volume Rendering (10 + 15 + 15 Points)*

Extend the `World` to support a pervasive fog.
Implement a new integrator that simulates this fog.

- Assume that the fog is uniform, and lit by an ambient light source. You need to add the attenuation factor for your primary and shadow rays. The primary rays (and only those) should additionally account for the ambient light reflected by the fog.
- Assume that the fog is uniform, but is lit by actual light sources in the world. You will need to perform ray marching along the primary rays, but attenuation of shadow rays can still be done analytically.
- Assume the fog is heterogenous (e.g. defined by some 3D perlin noise texture). Perform the ray marching along the primary and shadow rays.

Pay attention to rays reaching to infinity. If you try to perform ray marching of constant step along such a ray, your program will hang.