Prof. Dr. Michael Krone
Marco Schäfer
Marcel Bok
Big Data Visual Analytics in Life Sciences
Department of Computer Science & IBMI
Faculty of Science

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Scientific Visualization

SS/2021

Assignment 4

Due: June 2nd, 2021, 9:00 am

We supply you with a basic code template to use for this assignment but you are free to use the code you wrote for assignment one as a basis for this task. Our supplied code will contain code snippets assisting you in solving the tasks. So be sure to look at the supplied code if you choose to use your own code as a basis for this assignment.

1 Writing Custom WebGL2 Shaders (10 pts)

In this assignment you will implement two vertex and two fragment shaders to color the scene using GLSL-Language shaders. The scene should feature a box and a spheres both using the custom vertex and fragment shader.

To complete this assignment it is crucial to know several functions of the GLSL language to reduce the implementation expenditure:

- `pow(x,y)` $= x^y$
- `clamp(x, u, v)` $= \begin{cases} x, & u \leq x \leq v \\ u, & x < u \\ v, & x > v \end{cases}$
- `reflect(I, N)` = Reflect incident vector `I` at the normal vector `N` to get a reflection vector.
- `max(x,y)` = Return maximum of x and y
- `dot(v1, v2)` = Return dot product of v1 and v2
- `normalize(v1)` = Normalizes the values of a vector (v1) so that its length is 1

Furthermore you will need to know the following keyword:

- `uniform` = using `uniform` before a type definition makes this value available to all shader calls. This uniform are passed from the external program, i.e. your JS program.
- `in/out` = using `in/out` before a type definition makes it possible to transfer and interpolate this variable in the WebGL pipeline to the fragment shader. (e.g `"out pos"` vertex shader; `"in pos"` fragment shader)

Keep in mind that the GLSL language is strongly typed. This means you have to supply a data type for each variable you define. Furthermore, you also have to make sure that operations between variables are possible as the compiler will not convert the variables for you. This means that `1/2.5` will cause an error. Instead you must write `1.0/2.5`.

For this assignment you will have to:

- Write shaders coloring a box with varying amounts of R, G and B depending on the local coordinates of the vertex.
- Write a vertex and a fragment shader implementing per vertex phong shading.
- Link the lighting controls to the per pixel phong shader.
- Generate a control for the color of the sphere.

Utilize the `Three.js` documentation found at `https://threejs.org/` and the lecture slides to help you along. There are many examples for nearly every situation, as well as a comprehensive API-documentation for the functions you have to use. You also should comment your code and adhere to basic style guidelines, e.g. using reasonable variable names and indentation of lines.

1.1 Writing a Vertex Shader and a Fragment Shader Applying Position Dependent Coloring (5 pts)

All shaders are defined as special script-elements in different `index.html` .js files. You will have to write two separate shaders for this task: A vertex shader (`vertShader_cube.vert.js`) and a fragment shader (`fragShader_cube.frag.js`). The main role of the vertex shader, given a collection of vertex coordinates in local space, is to transform these local 3-dimensional coordinates into the 2-dimensional clip space. These are later used, interpolated between the vertices, as input by the fragment shader for coloring. Additional variables, to be interpolated for the fragment shader, can also be defined here using the `'out'` keyword.

You have to implement a function inside the vertex shader performing the transformation from local coordinates to clip space (The result is stored in `gl_Position` which is later automatically used from the fragment shader. It interpolates automatically between the coordinates given by the vertices.) You also have to generate a second `'out localposVertex'` which can be used by the fragment shader (`'in localposVertex'`) to color the fragments according to their position on the x-, y- and z-axis. Specifically: a cube with side length s the corner at $vec4(-s, -s, -s, 1.0)$ should be black (rgb(0.0,0.0,0.0)) and the corner at $vec4(s, s, s, 1.0)$ should be white ((rgb(1.0,1.0,1.0)). Figure 1 shows the result of this shading. In the fragment shader you write a function to assign colors to the generated fragments.

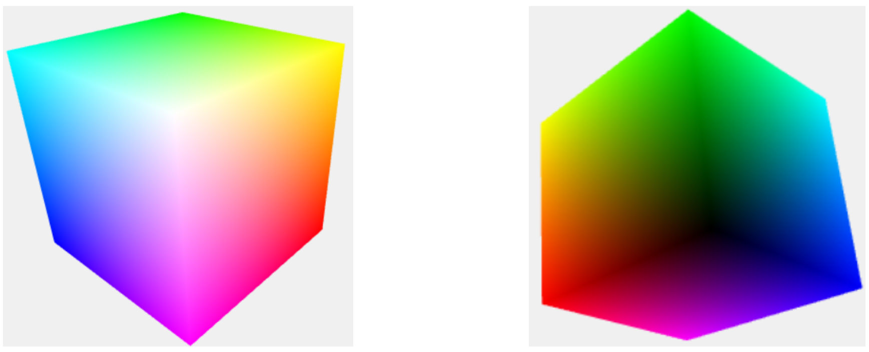


Figure 1: Final result of coordinate dependent shading

1.2 Writing a per-Vertex Phong Illumination Shader (5 pts)

In past times, in which hardware was not as potent as it is today, or even in certain situations it can be useful to calculate the shading on a vertex level instead of a pixel level to save resources. This is also known as Gouraud Shading. In the lecture the Phong reflection model was introduced. Write a vertex shader (`vertShader_sphere_perVertex`) calculating the Phong reflection model and the resulting per vertex shading/coloring. Subsequently, write a fragment shader (`fragShader_sphere_perVertex`) applying this per vertex colors, interpolated between the vertices, to the fragments. Your result should look like in Figure 2.

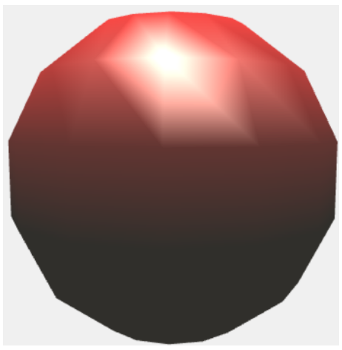


Figure 2: Per-vertex Phong Illumination

1.3 Bonus Exercise (1 pt)

Add controls to change the number of segments used to build the spheres (individual parameters for longitudinal and latitudinal direction, e.g., for widthSegments and heightSegments). See Figure 3 for example results.

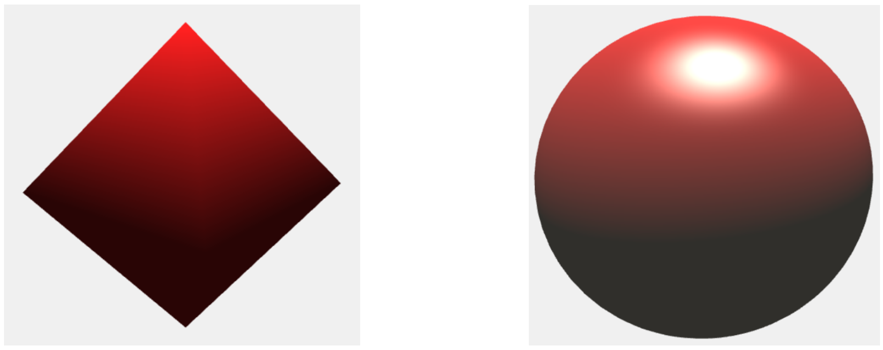
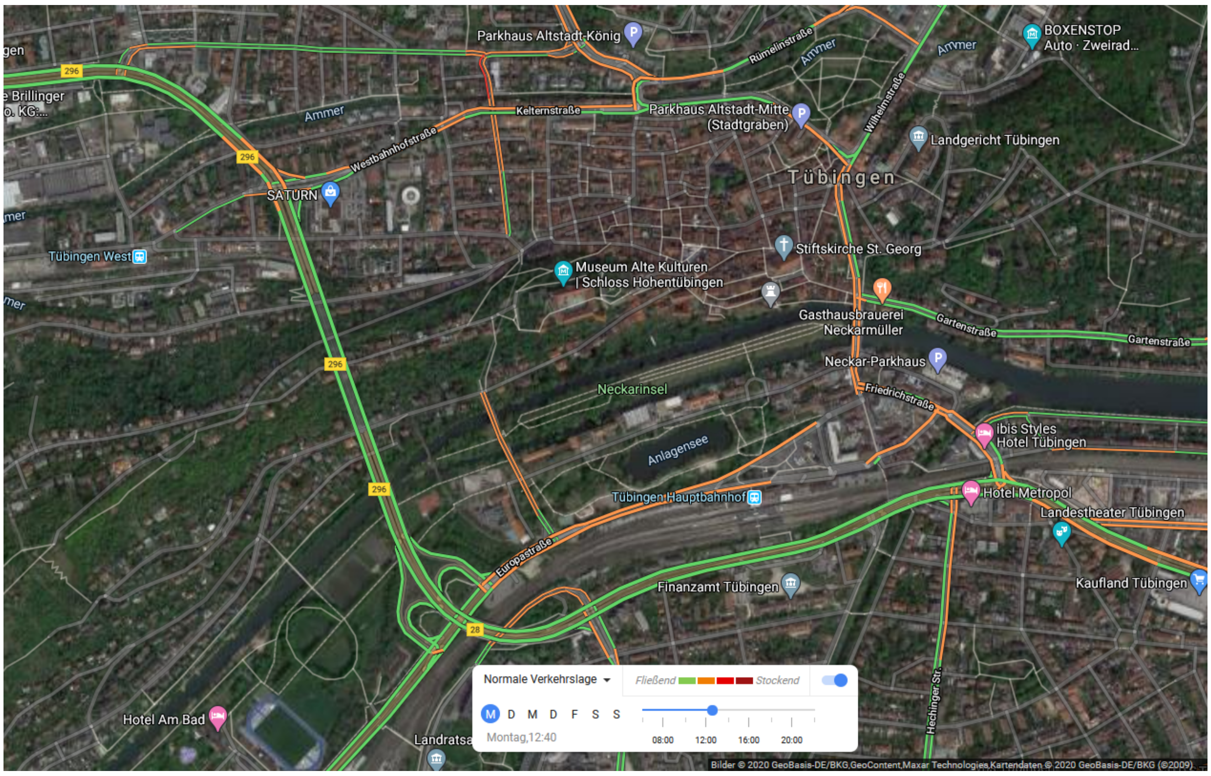


Figure 3: Left: "Sphere" with a low amount of width and height segments, Right: Sphere with a high amount of width and height segments.

2 Visualization Pipeline (5 pts)



Explain a possible solution of what happens in each step of the visualization pipeline for the traffic visualization from Google Maps shown above. Note that your solution does not have to be exactly what is actually implemented by Google, it should only be a *plausible* solution given what you see here in this visualization.

Handing in

For the Hand-in, write the names of both group members at the top of all code files and PDF documents. For the tasks involving transformation matrices, you can hand in a high quality and cropped picture of your calculations. Create a ZIP archive of your project folder and **PLEASE EXCLUDE** the folder **NODE_MODULES** from your archive. Name it with the last and first name of each group member, and the assignment number (e.g., *schaefer_marco___bok_marcel_assignment4.zip*) and upload it to `Hand-In` → `Assignment_4`.