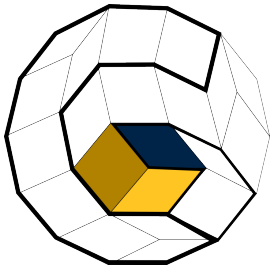
TU BRAUNSCHWEIG
Dr.-Ing. Martin Eisemann
Institut für Computergraphik
Matthias Überheide

January 19, 2015



Real-time Computer Graphics WS14/15
Assignment 7

Present your solution to this exercise on Monday, January 26th, 2015.

The exercises are going to take place in the CIP pool, room G40 in Mühlenpfordstraße 23. Please make sure your solutions compile and run on the CIP pool computers. Note that you need a y-number, which can be obtained at the Gauß-IT-Zentrum, to use the computers. If for some reason you are not able to attend the exercise, you may send your solution to ueberheide@cg.tu-bs.de instead.

In this assignment you will learn how to use multiple textures at the same time. Besides a simple multi-texturing approach, allowing to display the earth with a cloud layer over the earth's surface and an emissive city-light layer on the dark side of the planet, in the second task you will also implement normal mapping. This enables you to display detailed surface structures, even though the used geometry as much coarser than the final rendering suggests.

7.1 Multi-Texturing (40 Points)

Extend your program to handle not only one but multiple textures simultaneously. For this, the predefined structure `Texture` has to be initialized and filled with the needed data for every texture layer used in this assignment. (see the `#INFO#` comments in the code of `Ex08.cpp`)

We want to display the planet earth with a simple color layer describing the actual earth's surface. An emissive texture is used to display city lights on the dark side not facing the sunlight. The view down to the surface is blocked by a cloud layer, which is given by a cloud texture and a mask texture describing, how dense the clouds are a different locations in the sky.

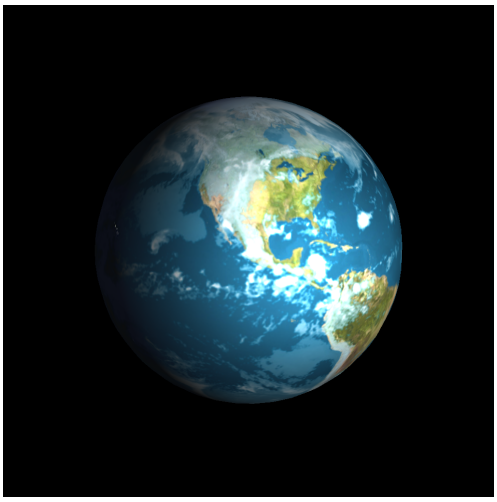
Alltogether, we need *four* textures - one for each information layer. The delivered texture resolution is 1k, but there is an additional texture package with 8k resolution on our website. Using this there will be more details visible on the surface. But make sure your GPU is able to load 8k textures.

Complete the missing parts for initializing and loading these textures and pass them to your shader using multiple texture units. Activate them using `glActiveTexture()` and pass the different texture layers to the corresponding uniforms in your fragment shader.

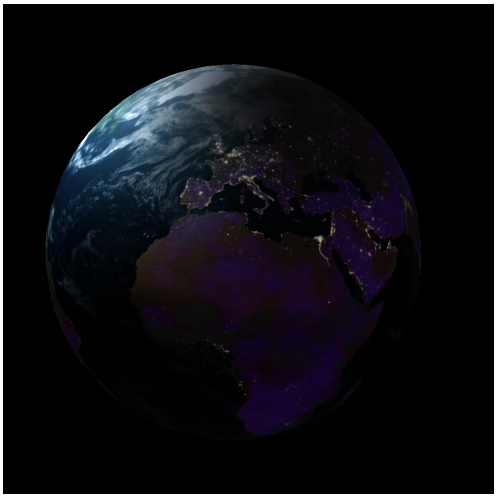
This shader should render the earth with the lighting used in the previous exercises. Remember that dense clouds block the view down to the surface (only clouds are visible) as well as the emitted light from the backside of the planet, while thin cloud layers block the light only partly. Where no clouds are visible, the light is not blocked at all and the earth's surface is clearly visible.

Use your shader to create this result and apply the emissive texture only on the sides of the globe, where no bright daylight is hitting the surface.

The light source for your *sun* is already set up at $(15, 15, 15)^T$ in *world space* emitting plain white light and the predefined material is also plain white, not altering the colors of the used textures. As the result, you should see renderings like the following:



(a) Dayside (1k resolution)



(b) Europe at night (8k resolution).

Figure 1: Different views of the Earth.

7.2 Normal Mapping (60 Points)

This task is divided into two parts. To be able to use normal mapping, you have to implement the computation of the required tangent space first. After this has been done, you can use the tangent and bitangent information in your shader code to compute the correct lighting.
You may start the program with an additional parameter 1 or 2, starting the program in normal map mode and loading the proper textures for the Moon resp. the planet Mars.
Implement your shader code in `normal_mapping.vert` and `normal_mapping.frag`.

1.) Tangent Space Computation:

As presented in the lecture, to use normal maps correctly, a orientation-independent *tangent space* has to be computed. Each viewing vector or light direction vector is transformed into this tangent space allowing to use normals defined in this space and stored as a RGB texture. The tangent space is defined that way, that the tangent vector T aligns directly with the u-axis of the uv-mapping space of the texture. The binormal B (or often referred as cotangent) is perpendicular to the tangent and also embedded to the uv-plane. Thus the binormal aligns with the v-axis of our texture space. Together with the surface normal N, being perpendicular to both vectors T and B, yields an orthonormal basis for the desired tangent space. Using a matrix

$$M = \begin{pmatrix} T_x & B_x & N_x & 0 \\ T_y & B_y & N_y & 0 \\ T_z & B_z & N_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

we are able to transform any vertex from tangent space to world space. Since we need to do this transformation from world space to tangent space, we need to use the inverse matrix M^{-1}. Knowing that the given matrix is based on an orthonormal basis, the transpose M' of the matrix M can be used as inverse.
But how to calculate the correct tangent space vectors T, B and N? The normal N is already given by every face definition and its vertices V_0, V_1, and V_2 using the cross product $N = (V_1 - V_0) \times (V_2 - V_0)$. This is what we've already done when reconstruction face normals.
To find the correct tangent an binormal vectors, see Figure 2 for an illustration. Having a triangle $\Delta = (V_0, V_1, V_2)$ in world space and its uv-mapping $\Delta' = (P_0, P_1, P_2)$ in uv-space 2D. Any point within the triangle can be described in both spaces. In uv-space $X' - P_0 = (X'_U - P_{0_U}) * T' + (X'_V - P_{0_V}) * B'$, T' and B' being the projections of tangent and binormal into tangent space. As we know, these vectors align with the u-axis resp. the v-axis in uv-space.

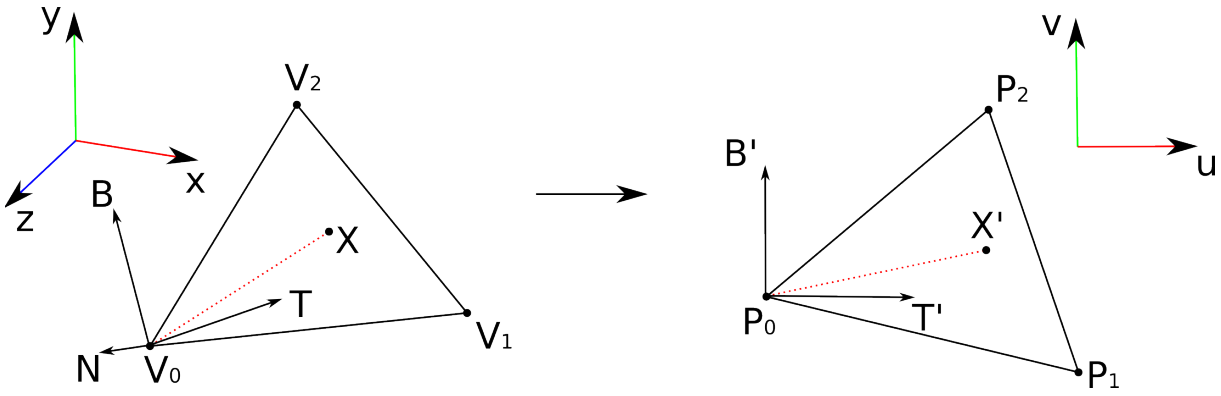


Figure 2: Tangent space per triangle vertex.

The same equation holds in 3D space given as $X - V_0 = (X'_U - P_{0_U}) * T + (X'_V - P_{0_V}) * B$. Note, that the scalar factors for T and B are still the same as in 2D. Using the other vertex points of the triangle as values for X yields a system of equations:

$$V_1 - V_0 = (P_{1_U} - P_{0_U}) * T + (P_{1_V} - P_{0_V}) * B$$
$$V_2 - V_0 = (P_{2_U} - P_{0_U}) * T + (P_{2_V} - P_{0_V}) * B$$

Which can be written as a matrix

$$\begin{pmatrix} V_1 - V_0 \\ V_2 - V_0 \end{pmatrix} = \begin{pmatrix} dU_1 & dV_1 \\ dU_2 & dV_2 \end{pmatrix} \cdot \begin{pmatrix} T \\ B \end{pmatrix}$$

with $dU_1 = P_{1_U} - P_{0_U}$ being the u-difference in uv-space. Analogous for dV_1, dU_2, and dV_2.
To compute our desired vectors T and B we need to invert the matrix of the uv-differences and multiply it by our edge vectors of the triangle.

$$\begin{pmatrix} T \\ B \end{pmatrix} = \begin{pmatrix} dU_1 & dV_1 \\ dU_2 & dV_2 \end{pmatrix}^{-1} \cdot \begin{pmatrix} V_1 - V_0 \\ V_2 - V_0 \end{pmatrix}$$

Having only a 2×2 matrix of uv-differences, we can invert it by simply using

$$\begin{pmatrix} dU_1 & dV_1 \\ dU_2 & dV_2 \end{pmatrix}^{-1} = \frac{1}{det(A)} \cdot \begin{pmatrix} dV_2 & -dV_1 \\ -dU_2 & dU_1 \end{pmatrix}$$

with $det(A) = dU_1 * dV_2 - dV_1 * dU_2$ the determinant of the original matrix.
One can now compute tangent and binormal from the uv-mapping of an object for each single vertex. By averaging the tangent and binormals of each vertex defined by every incident traingle (such as done with the reconstructed normals) we have a tangent and binormal to use for our mesh.
Your task is to implement the tangent and binormal reconstruction from an objects uv-mapping. Complete the method `computeTangentSpace` and iterate over every face doing the following:

- Compute the edge vectors $E_1 = V_1 - V_0$ and $E_2 = V_2 - V_0$ for the current triangle.
- Compute the uv-differences for the triangle's vertex points in uv-space. (dU_1, dV_1, dU_2, dV_2)
- Compute the determinant and calculate T and B using the equations above.
- Add T and B to every vertex of this triangle. (accumulate the values)

When your done, iterate over all vertices, averaging the tangent and binormal values by simply normalizing them for every vertex.
Unforntunately, due to the averaging of the vectors, the tangents and binormals of each vertex are not neccessarily perpendicular to each other any more. To resolve this, we use the *Gram-Schmidt* approach an retransform tangent and binormal. Compute

$$T' = T - (N \cdot T)N$$

for each vertex tangent and normalize T' again. Then reconstruct B' by using the cross-product and the vertex normal already given:

$$B' = T' \times N$$

Normalize B' and save both vectors T' and B' as vertex attributes `tangent` resp. `binormal`.

2.) Normal Mapping:

To implement normal mapping into the given framework you have to do the following:

- Pass the computed tangent and binormal values as VBO to your vertex shader. Extend your `render()` method of `MeshObj` and upload these values as vertex attributes. Setup proper attribute locations in your shader code for these two additional vertex attributes.
- In your vertex shader load tangent, binormal, and normal from the attribute locations you assigned them to and transform them from object space into world space using the modelview matrix. Now create your inverse matrix M' from the introduction of task 7.2.2 allowing to transform vectors from world space into tangent space. Note, that OpenGL/GLSL uses column-major matrix definitions. So `mat2(a, b, c, d)` creates a matrix of this form: $\begin{pmatrix} a & c \\ b & d \end{pmatrix}$. Compute your vertex-to-camera resp. vertex-to-light vectors as usual and multiply them with this matrix. They are now in tangent space and may be passed to your fragment shader.
- In your fragment program, use the normal map to read out the normal for each pixel. Be sure to reformat the loaded *color*-value to a proper vector within $[-1 \dots 1]^3$. Compute your lighting using the pixel-to-camera and pixel-to-light vectors in tangent space and the normal given by your normal map texture.

You should now be able to render the Moon or the Mars like in Figure 3:

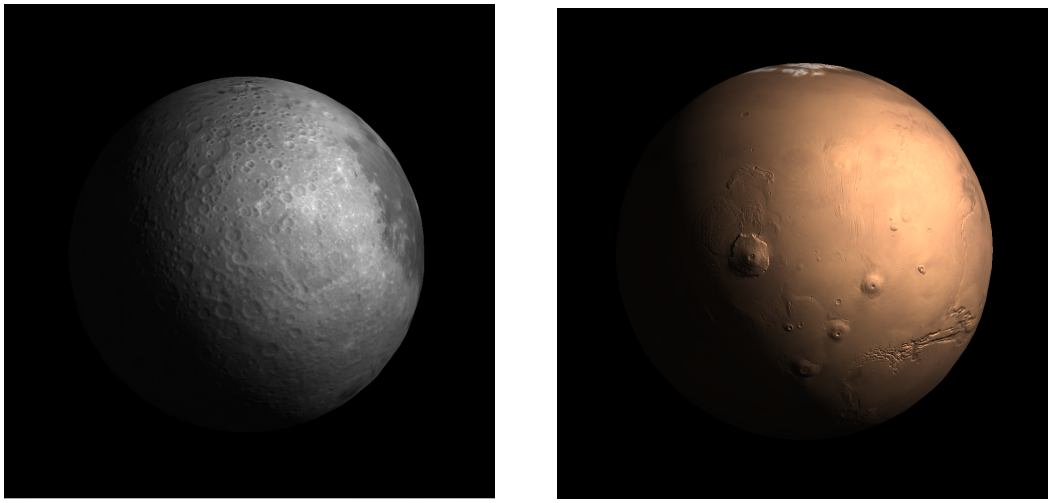


Figure 3: Views of the Moon and Mars.