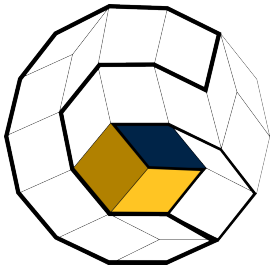
TU BRAUNSCHWEIG
Dr. Georgia Albuquerque
Institut für Computergraphik
Matthias Überheide

June 14, 2018



Real-time Computer Graphics, Summer 2018
Assignment 7

Present your solution to this exercise on Thursday, June 21, 2018.

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 ecg@cg.cs.tu-bs.de instead.

This exercise and the corresponding version of the framework can be found on the lectures website (`http://www.cg.cs.tu-bs.de/teaching/lectures/ss18/ecg/`).

Theoretical Tasks

7.1 Short Presentation (Extra credit 10 Points)

Prepare a 5min talk about an interesting topic related to the current lecture. Send your topic proposal via email to ecg@cg.cs.tu-bs.de at least 24 hours before the presentation.

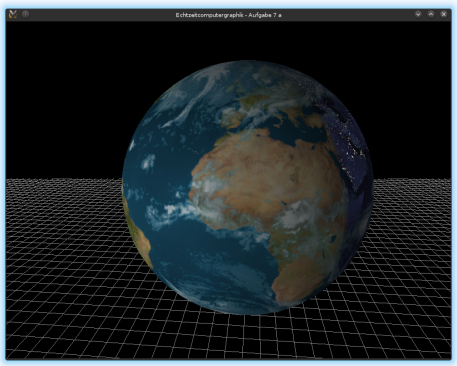
Practical Tasks

7.2 Multi-Texturing (20 Points)

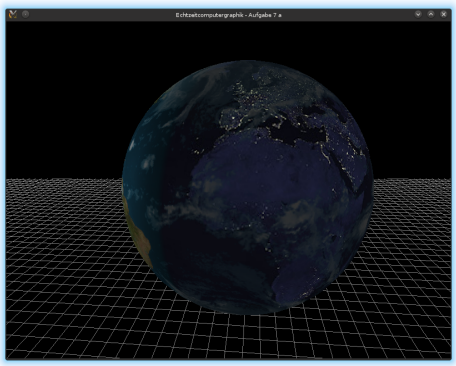
In this exercise we want to display the planet earth with a simple color layer describing the actual earth's surface. An emissive texture is used to display city lights on the dark side not facing the sunlight. The view down to the surface is blocked by a cloud layer, which is given by a cloud texture 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.

`OGLTexturedMeshObject` is already capable of loading multiple textures. To finish this exercise complete `shader/earth.frag` which will do the light computation and texture blending. You can assume that the sun is always lightsource 0. Compute if the current pixel should be on the night or day side of the earth and use the correct texture. Add the cloud layer with the given density. Note that the night texture contains emissive information but should still respect the clouds. Also think about smoothing the transition from night to day areas.
Although not completely accurate in our scene the sun is automatically rotating around the earth (still no reason for a flat shading of earth!) so you do get an automated day cycle. As the result you should see renderings similar to the following:



(a) Europe at day.



(b) Europe at night.

7.3 Normal Mapping (60 Points)

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.

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.
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:

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

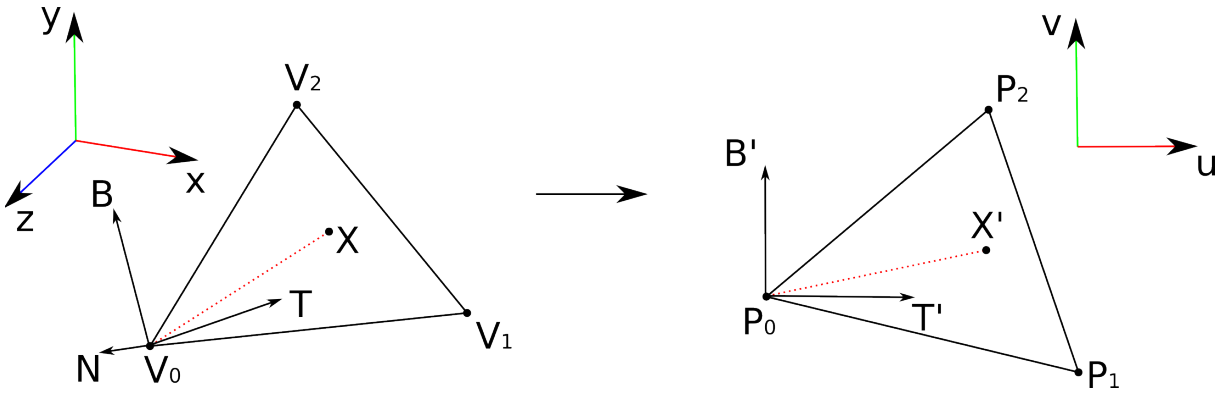


Figure 2: Tangent space per triangle vertex.

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 `Utils::OBJWavefrontFile::computeTangentSpace` in `utility/OBJWavefrontFile.cpp` 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.

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 and 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:

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

The normalized tangent and binormal are stored in vectors for later use by mesh objects

2.) Normal Mapping:

To implement normal mapping the tangents and binormals have to be available in the shader. This is done through `gl/oglobjects/OGLTexturedMeshObject2` which additionally uploads the corresponding vertex attributes to layout locations 3 and 4.
We will base our shader code again on the multi light shader. Extend the vertex shader code `normalmapping.vert` to transform the tangents and binormals into world space. Next, derive a transform from camera space to tangent space and apply this transform to eye direction and light directions. Review the corresponding fragment shader where now the normal information from the texture is used. Finally, you should now be able to render the Moon or the Mars like in Figure 3:

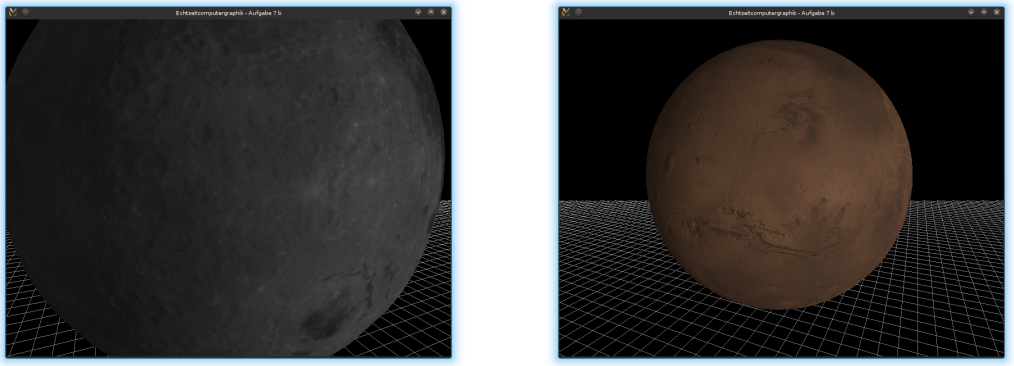


Figure 3: Views of the Moon and Mars.

7.4 Coding (Extra credit 10 Points)

Present one of the following:

- a bug in the framework
- a helpful test case to solve an exercise
- the implementation of an additional feature
- a nice demo using the current framework

Note that a short documentation is required and code has to be handed in. Please report bugs immediately (via email) so they can be fixed as soon as possible.