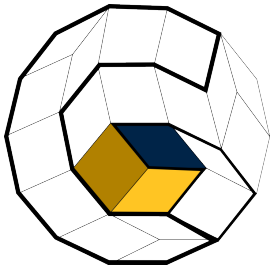
TU BRAUNSCHWEIG
Dipl. Inform. Martin Eisemann
Institut für Computergraphik
Lorenz Rogge (rogge@cg.tu-bs.de)

May 30, 2011



Echtzeit Computergraphik SS 2011
Assignment 7

In this exercise we implement the technique of *Normal Mapping* into our framework, allowing to display highly detailed surfaces using only rough geometry and the appropriate texture layers. Please present your solution to this exercise on Monday, June 6th, 2011.

7.1 Calculating the tangent space for mesh objects (20 Points)

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 bitangent B (or often falsely referred as binormal) is perpendicular to the tangent and also embedded to the uv-plane. Thus the bitangent 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 bitangent vectors, see Figure 1 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 bitangent 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 traingle as values for X yields a system of equations:

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

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.

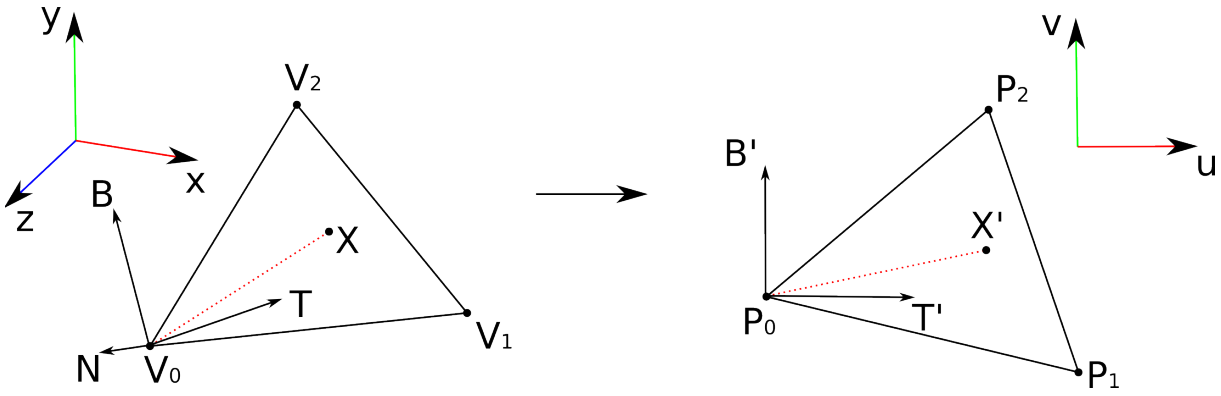


Figure 1: Tangent space per triangle vertex.

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 bitangent from the uv-mapping of an object for each single vertex. By averaging the tangent and bitangents of each vertex defined by every incident traingle (such as done with the reconstructed normals) we have a tangent and bitangent to use for our mesh.
Your task is to implement the tangent and bitangent 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 bitangent values by simply normalizing them for every vertex.
Unforntunately, due to the averaging of the vectors, the tangents and bitangents 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 bitangent. 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. `bitangent`.

7.2 Normal mapping (20 Points)

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

- Pass the computed tangent and bitangent values to your vertex shader. Extend your `render()` method of `MeshObj` and pass these values as vertex attributes. Since there are no predefined attribute variables provided by OpenGL, you may define your own or more simply just use `gl_MultiTexCoord1` and `gl_MultiTexCoord2` to do this job when rendering your vertex array.
- Load and pass the diffuse and normal texture to your shader program.
- In your vertex shader load tangent, bitangent, 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 7.1 allowing to transform vectors from world space into tangent space. Note, that OpenGL 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 eye-vector as usual and multiply it with this matrix. Now you can pass the eye-vector in tangent space to your fragment shader.
- In your fragment program, use the provided 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 (this time only use lambertian (eye-)lighting without any additional light sources) using the eye-vector in tangent space and the normal given by your normal map texture.

Your rendering should look like the following on the right, while the left view shows the exact same rendering without the use of normal mapping:

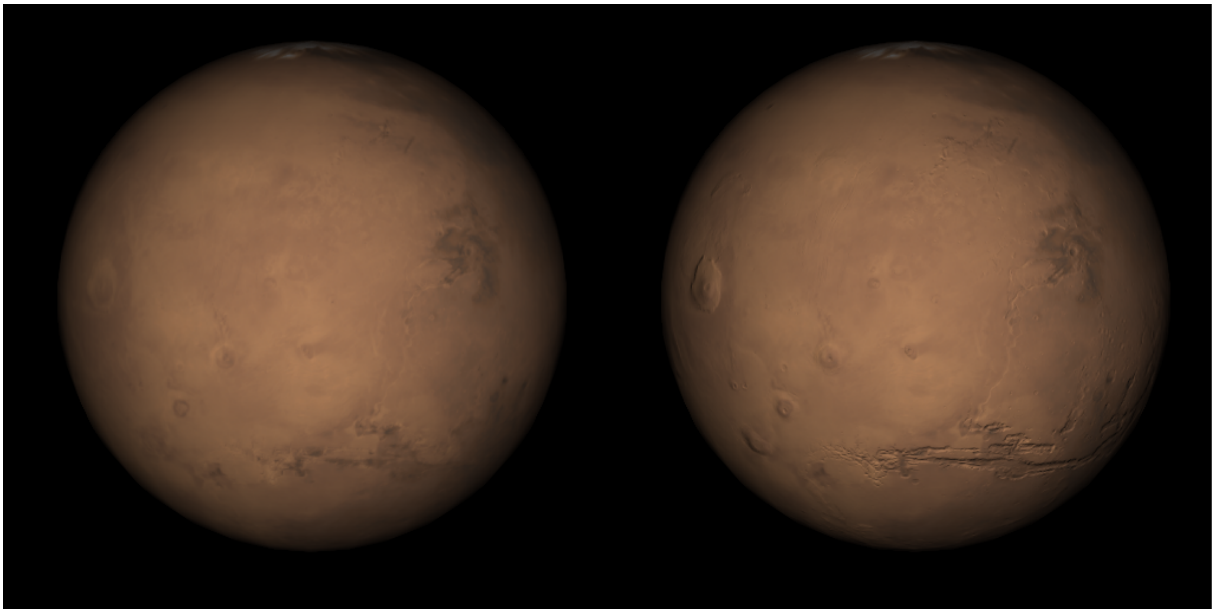


Figure 2: Left: without normal mapping. Right: with normal mapping