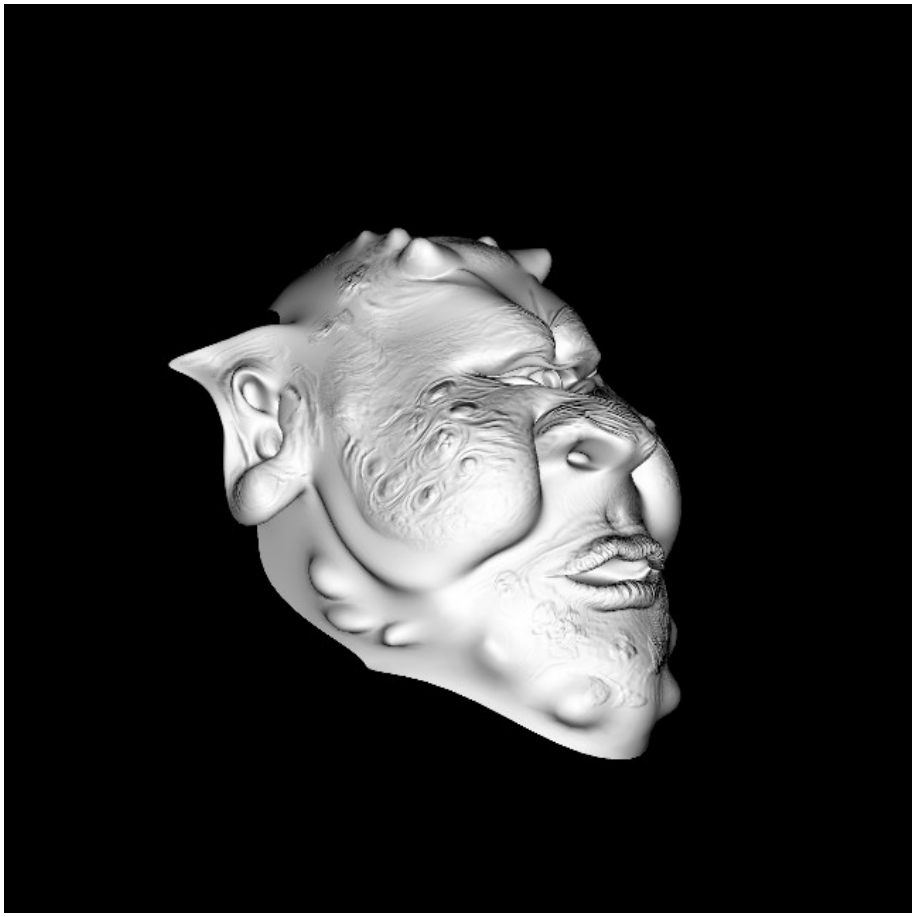
Normalmapping:

- Der Oger hat Tangent Space Vektoren & eine Normalmap
- Die Normalmap wird bereits gebunden und an den Shader übergeben
- Tangente und BiTangente bekommen Sie im Vertex Shader auf Location 2 bzw. 3
- Texturkoordinaten auf Location 4

Tangente und Bitangente müssen, genau wie die Normale, in den World Space transformiert werden. (Shading Berechungen werden diesmal im World Space statt wie sonst im Eye Space durchgeführt – daher gib es im Fragment Shader auch die Kameraposition als Extra Attribut.

Im FragmentShader bringen Sie die 3 TangentSpace Vektoren zunächst auf Einheitslänge. Mit mat3(T, B, N) stellen Sie die Rotationsmatrix vom Tangent Space in den World Space auf. (Hätten Sie die 3 Vektoren nicht im Vertex Shader in den World Space geholt, wäre es stattdessen die Matrix vom Tangent Space in den Object Space). Die Normale aus der Textur muss zuerst in den Wertebreich [-1, 1] gebracht werden – Bilder können ja keine negativen Werte speichern – un dann mit der Matrix transformiert werden.

Der Fragment Shader gibt zur Zeit dot(N, V) aus – benutzen Sie statt der geometrischen Normale die Normale aus der Textur.



Cube Mapping

- Die Cubemap wird im Shader als samplerCube deklariert.
- Im C++ Code wird bereits die Cubemap geladen, vorm Rendern müssen Sie sie allerdings noch binden. Das Texture Target is hier GL_TEXTURE_CUBE_MAP (anstelle vom sonst üblichen GL_TEXTURE_2D)
- Der Lookup funktioniert immer mit *texture* und einem vec3 Richtungsvektor.

Textur binden nicht vergessen!
Benutzen Sie den an der Normale reflektierten View Vektor für den Lookup.
Beim Rendern des Cubes (shadingType == RENDER_ENVIRONMENT) interpretieren Sie die FragmentPosition als Vektor und benutzen diese für den Lookup.

Environment Maps ersetzen üblicherweise das Umgebungslicht. Daher macht es Sinn, den Wert aus der Textur mit dem AO Wert des Modells – sofern vorhanden – zu multiplizieren.