Normal Mapping

Niklas Ecker

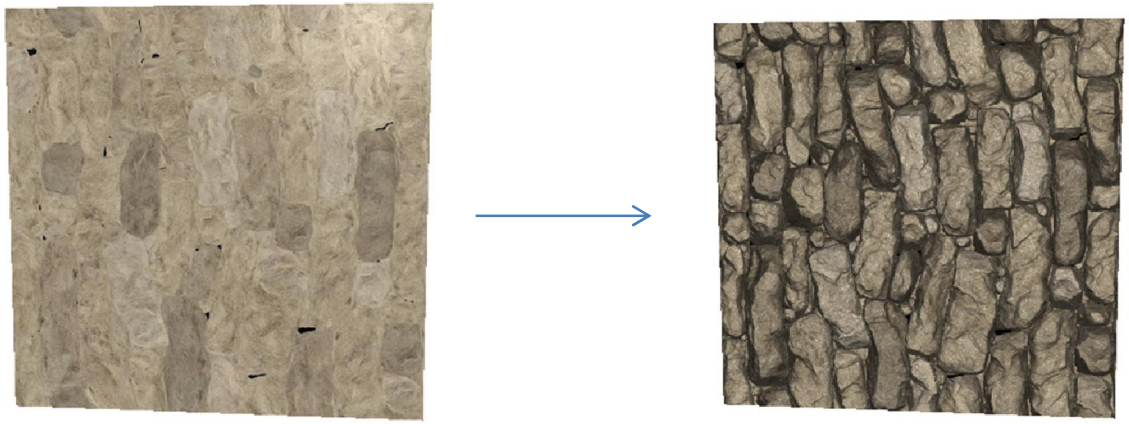
27.08.2014

Übersicht

- Was ist Normal Mapping ?
- Was wird benötigt ?
- Was passiert in den Shadern ?
- Beispiel-Demo
- Erweiterungen

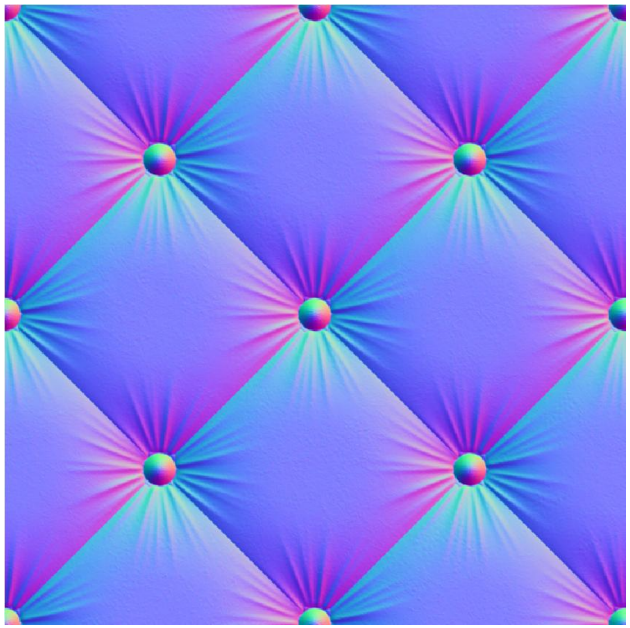
Was ist Normal Mapping

- Verfahren um Oberflächen schöner darzustellen
- Jedes Fragment hat eine eigene Normale
- Normalen sind in einer Normal Map gespeichert (rgb-Wert)



Was ist Normal Mapping

- Beispiel einer Normal Map
- rgb entsprechen dabei xyz der Normalen
- Normal Map kann aus einer Height-Map berechnet werden



Normal Map

Was wird benötigt (Vert)

Der Vertex Shader erhält :

- Position
- Normale
- UV-Koordinate
- Tangente
- Light-Position
- (MVP-Matrix, wie immer)

Was wird benötigt (Frag)

- Für den Fragment shader müssen die entsprechenden Texturen bereitstehen (Normal Map und Color Texture)
- Außerdem erhält der Fragment-shader noch die Material-Informationen und die Licht-Farbe

Was passiert in den Shadern (Vert)

- tbn Matrix berechnen :

```
mat4 normalMatrix = transpose(inverse(viewMatrix * modelMatrix));
vec3 normal_cs = normalize((normalMatrix * vec4(normal,0)).xyz);
vec3 tangent_cs = normalize((normalMatrix * vec4(tangent,0)).xyz);
vec3 bitangent_cs = cross(tangent_cs, normal_cs);

mat3 tbn = transpose(mat3(
    tangent_cs,
    bitangent_cs,
    normal_cs
));
```

- Wie immer :

```
gl_Position = projectionMatrix * viewMatrix * modelMatrix * position;
```

Was passiert in den Shadern (Vert)

- Umwandlungen von Spaces :

```
/*********** Calculate position in camera space ************************************/
vec3 position_cs = (viewMatrix * modelMatrix * position).xyz;

/*********** Calculate light vector (tangent space) ********************************/
// light position in world coordinates
vec3 lightPosition_cs = (light.pos).xyz;

// calculate vector that goes from the vertex to the light, in tangent space
passLightVector = tbn * normalize(lightPosition_cs - position_cs);

/*********** Calculate eye vector (tangent space) **********************************/
// calculate eye vector in camera space
vec3 eye_cs = normalize(-position_cs);

// calculate eye vector in tangent space
passEyeVector = tbn * eye_cs;

/*********** Pass uv of the vertex *************************************************/
// no special space for this one
passUV = uv;
```

Was passiert in den Shadern (Vert)

- Weitergereicht an den Fragment Shader werden :

Umgewandelt mit der tbn Matrix :

Lightvector (tangent space)

EyeVector (tangent space)

UV-Koordinate

Was passiert in den Shadern (Frag)

Beleuchtung mit Hilfe der Normalen aus der Normal-Map (Zugriff durch uv Koordinate) und dem Light- und Eye-Vector

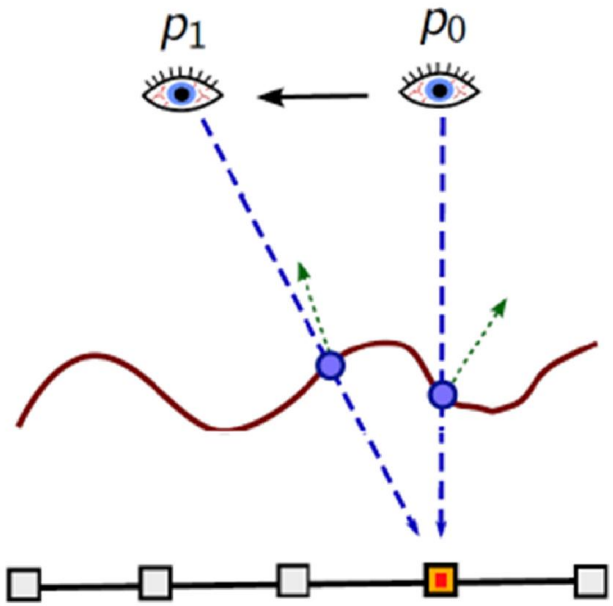
Was passiert in den Shadern (Frag)

```
/***************** Diffuse ************************************************************/
// local normal, in tangent space
vec3 normal = texture(normaltexture,passUV).rgb * 2.0 - 1.0;

// direction of the light (from the fragment to the light) in tangent space
vec3 lightVector = normalize(passLightVector);

//compute the diffuse lighting factor
float cos_phi = max(dot(normal, lightVector), 0);

/***************** Specular ***********************************************************/
// compute the normalized reflection vector using GLSL's built-in reflect() function
vec3 reflection = normalize(reflect(-lightVector, normal));

// eye vector in tangent space
vec3 eyeVector = normalize(passEyeVector);

// compute the specular lighting factor
float cos_psi_n = pow(max(dot(reflection, eyeVector), 0.0f), mat.shininess);

/***************** Material properties ************************************************/
vec3 diffuse_color;
if (useColorTexture != 0)
    diffuse_color = texture(colortexture, passUV).rgb;
else
    diffuse_color = mat.diffuse;

/***************** All together *******************************************************/
fragmentColor.rgb = diffuse_color * lightAmbient;
fragmentColor.rgb += diffuse_color * cos_phi * light.col;
fragmentColor.rgb += mat.specular * cos_psi_n * light.col;
fragmentColor.a = 1.0;
```

Beispiel

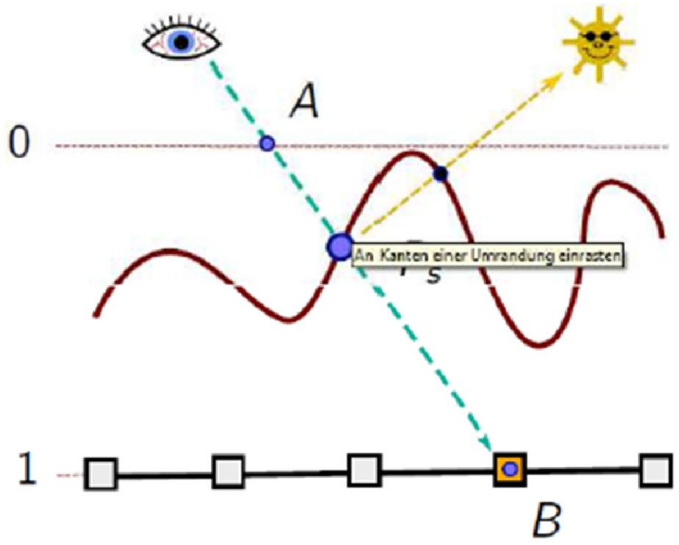
Erweiterungen

- Parallax Mapping
- Verschiebung der Textur-Koord in abhängigkeit der Blickrichtung



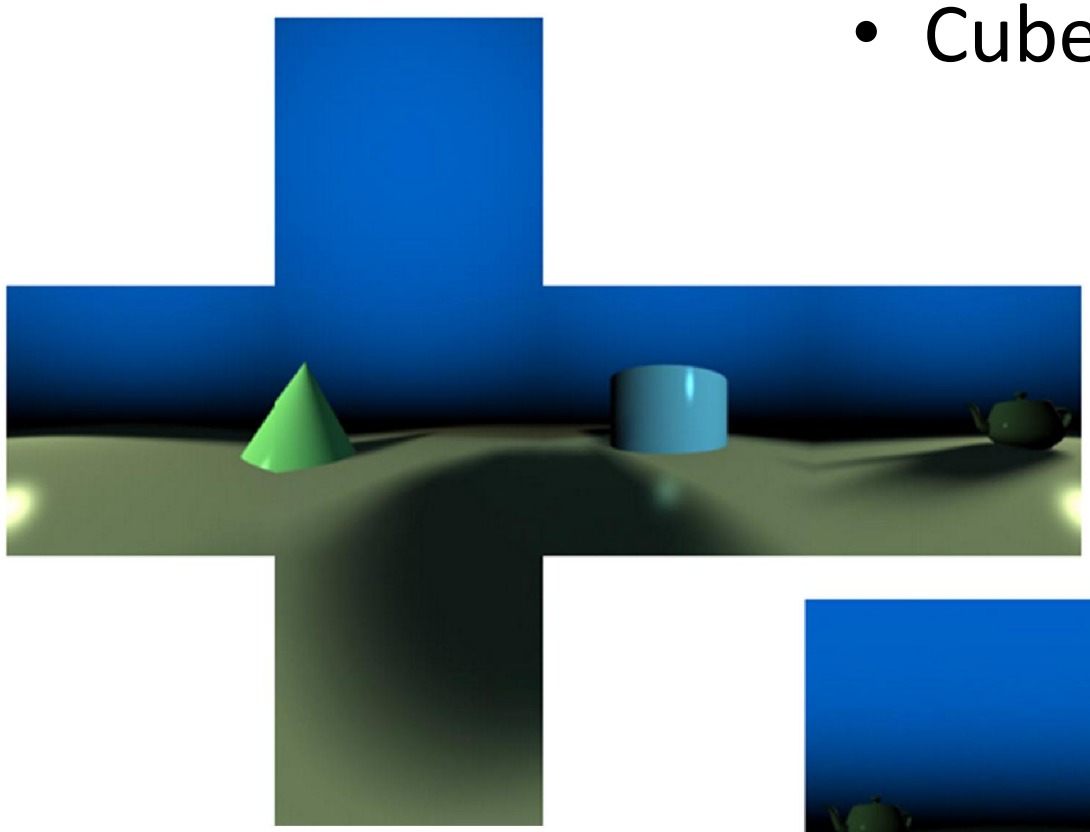
Erweiterungen

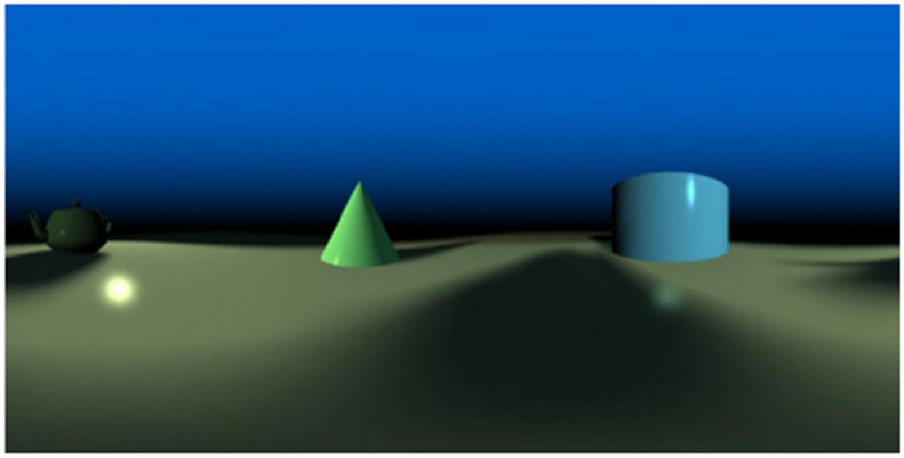
- Relief Mapping
- Schnittpunkt mit Höhenprofil finden
- Vom SP aus schauen ob der Licht-Vektor durch das Profil geht oder ob der SP die LQ „sehen“ kann
- Ähnelt dem Ray-Tracing



Environment Mapping

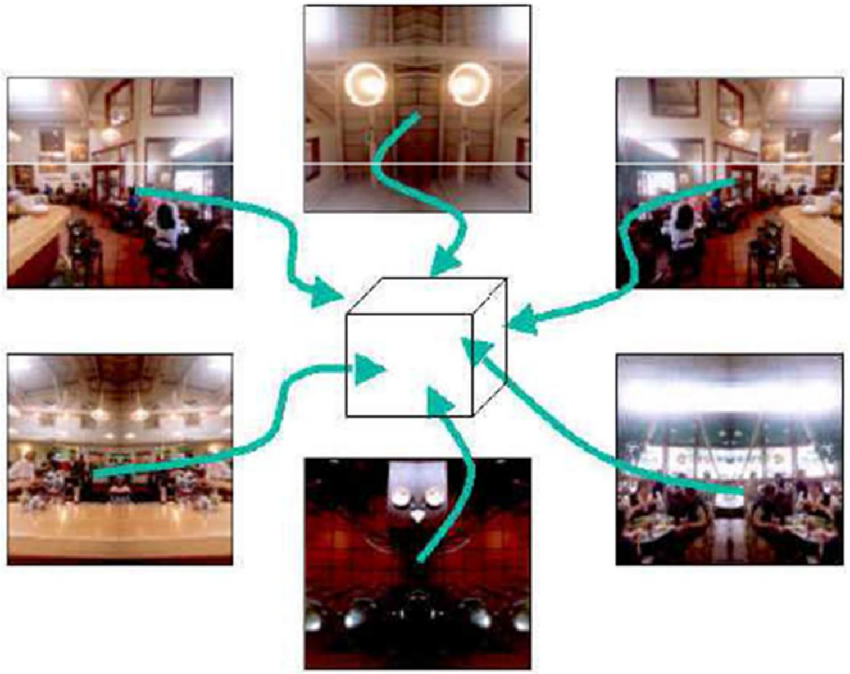
- Cube Mapping vs Sphere Mapping





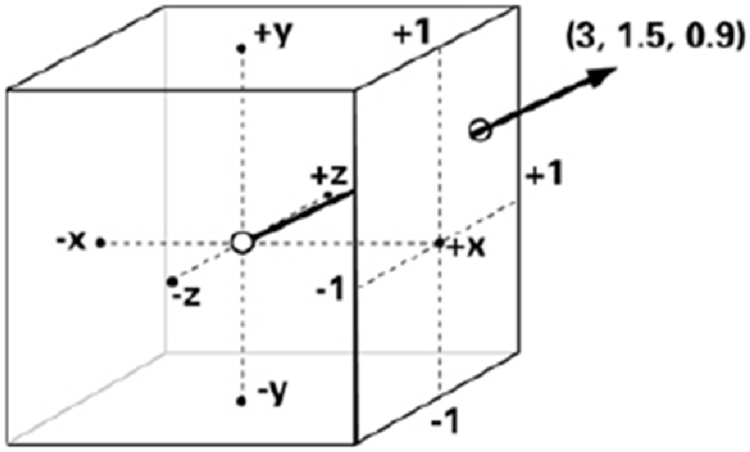
Environment Mapping

- Cube Mapping



Environment Mapping

- Cube Mapping
- Zugriff durch 3D-Koordinate
- Größter Wert bestimmt die Textur auf die zugegriffen wird



Environment Mapping

- Cube Mapping
- Von der Viewmatrix wird die Translation entfernt

```
layout (location = 0) in vec4 position;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;
out vec3 passTexCoord;

void main(){
	passTexCoord = position.xyz;
	mat4 vM = mat4(mat3(viewMatrix));
	gl_Position = projectionMatrix * vM * position;
}
```

Environment Mapping

- Cube Mapping
- Blickvektor an der Normalen gespiegelt ist der Zugriffsvektor auf die Cube-Map
- reflect(v,n) im Shader