Codeschnipsel zum Projekt helmade

Texturierung in WebGL

Für die realistische Darstellung der Helme im 3D-Konfigurator entwickelte Demodern eigene Shader und zeigt hier die Vorgehensweise einer auf Masken basierenden Texturierung in WebGL.

Code 1: Aufbau des Shell-Shaders

```
uniform sampler2D tDiffuse; //Tutorial: Farbtextur
uniform sampler2D tSurface; //Tutorial: Materialtextur

vec3 renderNormal()
{
	vec3 result = vec3();
	[siehe Code 5]
	return result;
}

vec3 renderMetallic()
{
	vec3 result = vec3();
	[...]
	return result;
}

vec3 renderFlake()
{
	vec3 result = vec3();
	[...]
	return result;
}

vec3 renderHoloFlake()
{
	vec3 result = vec3();
	[siehe Code 6]
	return result;
}

void main()
{
	[Preprocessing Code: Siehe Code 3]

	vec3 diffuseMap = texture2D(tDiffuse, vUv);
	vec3 surfaceMap = texture2D(tSurface, vUv);

	vec3 result = renderNormal( normal );

	float metallicMask = surfaceMap.r;
	if (metallicMask > 0.1)
	{
		vec3 metallicResult = renderMetallic();
		result = mix(result, metallicResult, metallicMask);
	}

	float flakeMask = surfaceMap.g;
	if (flakeMask > 0.1)
	{
		vec3 flakeResult = renderFlake();
		result = mix(result, flakeResult, flakeMask);
	}

	float holoFlakeMask = surfaceMap.b;
	if (holoFlakeMask > 0.1)
	{
		vec3 holoFlakeResult = renderHoloFlake();
		result = mix(result, holoFlakeResult, holoFlakeMask);
	}
```

```
    [Postprocessing Code Siehe Code 3]

    gl_FragColor = vec4(result, 1);
}
```

Code 2: Erstellung der Materialtextur

```
varying vec2 vUv;

uniform sampler2D tMask1;
uniform sampler2D tMask2;
uniform sampler2D tMask3;

uniform vec3 color0;
uniform vec3 color1;
uniform vec3 color2;
uniform vec3 color3;

void main()
{
    float t1 = texture2D(tMask1, vUv).r;
    float t2 = texture2D(tMask2, vUv).r;
    float t3 = texture2D(tMask3, vUv).r;

    vec3 rgb = color0;
    rgb = mix(rgb, color1, t1);
    rgb = mix(rgb, color2, t2);
    rgb = mix(rgb, color3, t3);

    gl_FragColor = vec4(rgb, 1);
}
```

Code 3: Pre- und Postprocessing im Shell Shader

```
void main() {

    [ prepare textures ]

    // compute environmap and plastic normals
    vec3 normal = normalize(vNormal);
    vec3 cameraToVertex = normalize(vWorldPosition - cameraPosition);

    //compute varnisch environment mapping
    vec3 normalPlasic = perturbNormal2Arb(
                tPlasticNormal,
                -vViewPosition,
                normal,
                vec2(1, 1),
                30.0);

    vec3 worldNormalPlastic = inverseTransformDirection(normalPlasic, viewMatrix);
    vec3 reflectVecPlastic = reflect(cameraToVertex, worldNormalPlastic);
    vec2 environmentUv = vec2(
        atan(reflectVecPlastic.z,reflectVecPlastic.x) * RECIPROCAL_PI2 + 0.5,
        reflectVecPlastic.y * 0.5 + 0.5
    );
    vec3 clearVarnish = texture2D(tEnv, environmentUv).xyz;

    [ compute resulting pixel per material, Siehe Code: 1 ]

    //hemispheric light
    float lambertHemi1 = dot(normalPlasic, vec3(0.4, 0.4, -0.4));
    float lambertHemi2 = dot(normalPlasic, vec3(-0.1, -0.8, -0.1));
    float hemi1 = clamp(lambertHemi1, 0.0, 0.3);
    float hemi2 = clamp(lambertHemi2, 0.0, 0.3);
    result += result * vec3(0.8,0.9,1.2) * hemi1 * 0.3;
    result += vec3(0.0,0.05,0.1) * hemi1 * 0.3;
    result -= result * vec3(0.8,1.0,1.2) * hemi2 * 0.3;
    result -= vec3(0.1,0.1,0.3) * hemi2 * 0.3;
```

```
        //ambient occlusion
        float occlusion = (0.3 + ambientMap.r * 0.7);

        gl_FragColor = vec4(result * occlusion, 1);

}
```

Code 4 Berechnung der Luminanz eines RGB-Wertes

```
float getLuminance(vec3 rgb)
{
  // Algorithm from Chapter 10 of Graphics Shaders.
  const vec3 W = vec3(0.2125, 0.7154, 0.0721);
  return dot(rgb, W);
}
```

Code 5 »Glossy Lack«

```
/*
 * Main fagment for normal varnish
 */
vec3 renderNormal(
        vec3 normal,
        vec3 diffuseMap,
        vec3 clearVarnish
) {
        float luminance =  getLuminance(diffuseMap);

        //soft noise
        vec3 normalSoftNoise = perturbNormal2Arb(
                tRandomNormal,
                -vViewPosition,
                normal,
                vec2(-0.05, -0.05),
                20.0);

        //diffuse and specular
        vec3 lightDir1 = normalize(vec3(0, 0.25, 1.0));
        float lambert1 = saturate(dot(normalSoftNoise, lightDir1));

        //fake specular light
        float cosineTerm1 = saturate(sin(lambert1 * 12.6));

        //finally mix all colors
        vec3 result = vec3(0,0,0);
        result += clearVarnish.xyz * lambert1 * 0.1;
        result += clearVarnish.xyz * 0.1;
        result += diffuseMap * 0.7;
        result += diffuseMap * lambert1 * 0.30;
        result += (vec3(0.4, 0.4, 0.4)) * cosineTerm1 * ( 0.62 - luminance * 0.52 ) * 0.3;

        return result;
}
```