GLSL shader for glossy specular reflections on an cubemapped surface

I wrote a shader for environmental cubemapping

***Vertex shader ***

```
varying vec3 Normal;
varying vec3 EyeDir;
uniform samplerCube cubeMap;

void main()
{
        gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
        Normal = gl_NormalMatrix * gl_Normal;
        EyeDir = vec3(gl_ModelViewMatrix * gl_Vertex);
}
```

***Fragment shader ***

```
varying vec3 Normal;
varying vec3 EyeDir;

uniform samplerCube cubeMap;

 void main(void)
 {
    vec3 reflectedDirection = normalize(reflect(EyeDir, normalize(Normal)));
    reflectedDirection.y = -reflectedDirection.y;
    vec4 fragColor = textureCube(cubeMap, reflectedDirection);
    gl_FragColor = fragColor;
}
```

This is the classical result:  Now I want to add some specular white highlight in order to obtain a more glossy effect, like motherpearl. How is it possible to add this kind of highlight? Like the one in this image Should I sum a specular component to `gl_FragColor` ? A first attempt is to compute specular reflection in vertex shader

```
vec3 s = normalize(vec3(gl_LightSource[0].position - EyeDir));
vec3 v = normalize(EyeDir);
vec3 r = reflect( s, Normal );
vec3 ambient = vec3(gl_LightSource[0].ambient*gl_FrontMaterial.ambient);
```

```
float sDotN = max( dot(s,Normal), 0.0 );
vec3 diffuse = vec3(gl_LightSource[0].diffuse * gl_FrontMaterial.diffuse * sDotN);
vec3 spec = vec3(0.0);
if( sDotN > 0.0 )
    spec = gl_LightSource[0].specular * gl_FrontMaterial.specular * pow( max( dot(r,v),
2.0 ), gl_FrontMaterial.shininess );

LightIntensity = 0*ambient + 0*diffuse +  spec;
```

and to multiply it to `gl_FragColor` but the effect I obtain is not convincing.

Someone has idea how to do it?

c++ opengl graphics glsl

asked Aug 3 '12 at 10:51

linello
2,726 6 33 68

1 I would expect you to add it rather than multiply it, as it's additional light, not modulating the amount of the reflection - unless I misunderstood :) – jcoder Aug 3 '12 at 11:08

Why adding? and what if fragColor RGB values are too big? Is there a kind of weighted means I should adopt? – linello Aug 3 '12 at 14:02

Well my understanding is that you want to calculate the reflection, and then in addition add some specular highlights, so you'd want to add the light for both (and presumably clamp the values). I'm not 100% sure though and can't help on the detail (which is why this is a comment, not an answer... sorry) – jcoder Aug 3 '12 at 14:18

It seems that a reasonable model for the fragment color is `gl_FragColor = texture*(ambient+diffuse) + specular` – linello Aug 3 '12 at 14:45

Can you show us the effect you obtained that you find unconvincing? – LarsH Aug 3 '12 at 16:00

1 Answer

Here's an example of how you **could** do it:

Mother-of-Pearl-Effect **OFF**:

Mother-of-Pearl-Effect **ON**:

Vertex shader:

```
uniform vec3 fvEyePosition;

varying vec3 ViewDirection;
varying vec3 Normal;

void main( void )
{
   gl_Position = ftransform();
   vec4 fvObjectPosition = gl_ModelViewMatrix * gl_Vertex;

   ViewDirection  = fvEyePosition - fvObjectPosition.xyz;
   Normal         = gl_NormalMatrix * gl_Normal;
}
```

Fragment shader:

```
uniform samplerCube cubeMap;

varying vec3 ViewDirection;
varying vec3 Normal;

const float mother_pearl_brightness = 1.5;

#define MOTHER_PEARL

void main( void )
{
   vec3  fvNormal         = normalize(Normal);
   vec3  fvViewDirection  = normalize(ViewDirection);
   vec3  fvReflection     = normalize(reflect(fvViewDirection, fvNormal));

#ifdef MOTHER_PEARL
   float view_dot_normal = max(dot(fvNormal, fvViewDirection), 0.0);
   float view_dot_normal_inverse = 1.0 - view_dot_normal;

   gl_FragColor = textureCube(cubeMap, fvReflection) * view_dot_normal;
   gl_FragColor.r += mother_pearl_brightness * textureCube(cubeMap, fvReflection +
vec3(0.1, 0.0, 0.0) * view_dot_normal_inverse) * (1.0 - view_dot_normal);
   gl_FragColor.g += mother_pearl_brightness * textureCube(cubeMap, fvReflection +
vec3(0.0, 0.1, 0.0) * view_dot_normal_inverse) * (1.0 - view_dot_normal);
   gl_FragColor.b += mother_pearl_brightness * textureCube(cubeMap, fvReflection +
vec3(0.0, 0.0, 0.1) * view_dot_normal_inverse) * (1.0 - view_dot_normal);
#else
   gl_FragColor = textureCube(cubeMap, fvReflection);
#endif
}
```

Of course the way the R, G and B components are being calculated is not very correct, but I'm posting this code to show you the way, not the solution.

EDIT:

Here's the promised "proper" version of the iridescent shader:

Vertex shader:

```
varying vec3 v_view_direction;
varying vec3 v_normal;
varying vec2 v_texture_coordinate;

void main(void)
{
   gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
   v_texture_coordinate = gl_MultiTexCoord0.xy;
   v_view_direction = -gl_ModelViewMatrix[3].xyz;
```

```
    v_normal = gl_NormalMatrix * gl_Normal;
}
```

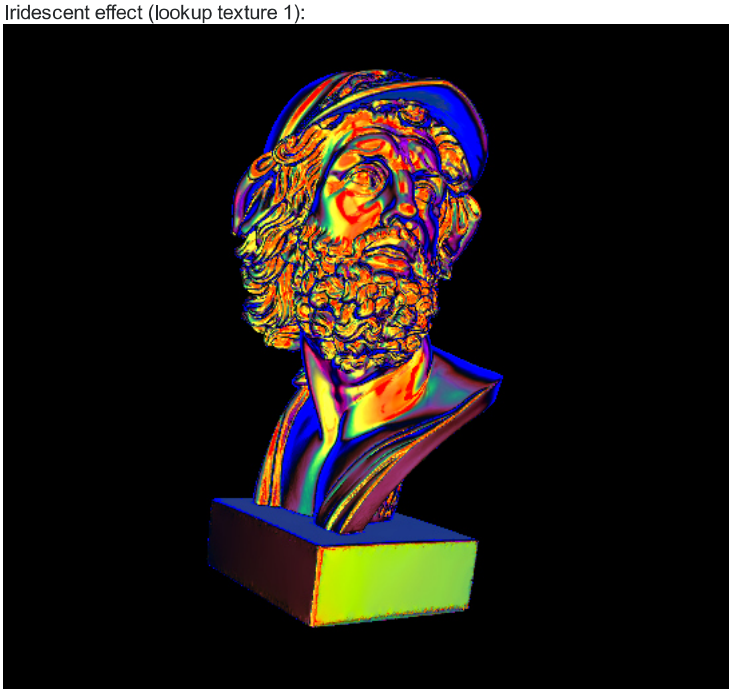
Fragment shader:

```
uniform samplerCube texture_reflection;
uniform sampler2D texture_iridescence;
uniform sampler2D texture_noise;

varying vec3 v_view_direction;
varying vec3 v_normal;
varying vec2 v_texture_coordinate;

const float noise_strength = 0.5;

void main(void)
{
   vec3 n_normal = normalize(v_normal);
   vec3 n_wiew_direction = normalize(v_view_direction);
   vec3 n_reflection = normalize(reflect(n_wiew_direction, n_normal));

   vec3 noise_vector = (texture2D(texture_noise, v_texture_coordinate).xyz - vec3(0.5)) *
noise_strength;

   float inverse_dot_view = 1.0 - max(dot(normalize(n_normal + noise_vector),
n_wiew_direction), 0.0);
   vec3 lookup_table_color = texture2D(texture_iridescence, vec2(inverse_dot_view,
0.0)).rgb;

   gl_FragColor.rgb = textureCube(texture_reflection, n_reflection).rgb *
lookup_table_color * 2.5;
   gl_FragColor.a = 1.0;
}
```

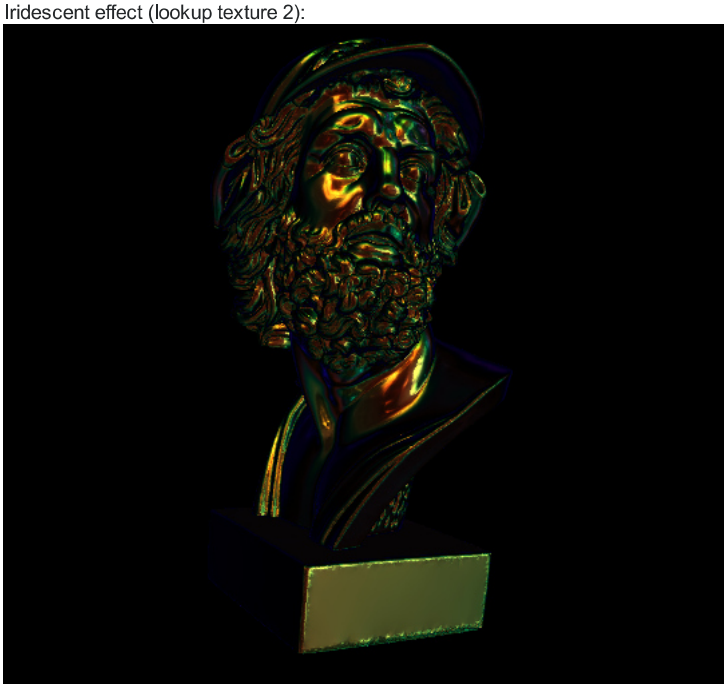
Results

No iridescent effect:

Iridescent effect (lookup texture 1):

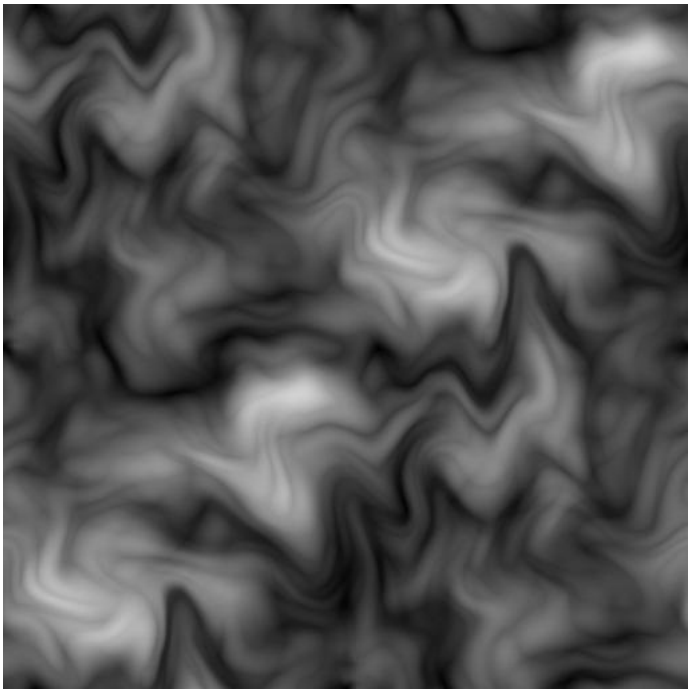


Iridescent effect (lookup texture 2):



Iridescence lookup texture 2:

Noise texture:



Remarks:
The iridescence lookup texture could also be a 1D texture, which would be much more memory efficient.
Also, the way the noise vector is calculated is actually nonsense. The right solution would be to use bump mapping. But hey, it works! :D

edited Nov 20 '13 at 9:51

answered Jan 3 '13 at 16:23

Dudeson
764 9 19

that's very cool indeed, thanks! – linello Jan 4 '13 at 10:11

No problem, my friend! – Dudeson Jan 4 '13 at 10:48

1 Okay my friend. The new version is up. – Dudeson Oct 11 '13 at 9:42

This is really neat stuff. I'm wondering how you would go about getting rid of aliasing artifacts around the edges of the visualization? – dev_nut Feb 26 at 17:00

@dev_nut: Are you talking about the aliasing on the bust model? That is caused by the model itself (it's a 3D scan). There shouldn't be any aliasing as mip-mapping should take care of it along the edges. – Dudeson Feb 26 at 22:49