SV Lab2 Shader Explanation

Rafael Ávila
10/01/2021
2020-2021/Q1
MIRI-SV

raycast.vert

```
#version 330

layout (location = 0) in vec3 vert;

uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;

smooth out vec3 tex_coords;
smooth out vec3 position;
smooth out vec3 camera; // we need the camera position for the fragment shader

void main(void)  {
	tex_coords = vert + vec3(0.5);
	position = vert;
	// obtain the camera position from the inverse of the view matrix (the 4th column)
	camera = inverse(view)[3].xyz;

	gl_Position = projection * view * model * vec4(vert, 1);
}
```

raycast.frag

```
#version 330

smooth in vec3 tex_coords;
smooth in vec3 position;
smooth in vec3 camera;

uniform sampler3D volume;
```

```
// Uniform values that will determine the color ranges on the transfer function for each color component. You can see how they are obtained from the UI and transferred to the shaders as uniforms on the "glwidget" and "mainwindow" files (the new parts of the code are documented as well)
uniform float red_min, green_min, blue_min;
uniform float red_max, green_max, blue_max;

uniform vec3 light_pos; // The light position, as the previous uniforms, you can modify it on the UI

out vec4 frag_color;

// A fixed transfer function. Returns a color depending on the user defined intervals for each primary color
vec4 transfer_function(float acc_color) {
    vec4 rgba = vec4(0, 0, 0, 0);

        if (red_color >= red_min && red_color <= red_max) rgba += vec4(red_color, 0, 0, red_color/3); // Each component will have ⅓ of the total opacity
        if (red_color >= green_min && red_color <= green_max) rgba += vec4(0, red_color, 0, red_color/3);
        if (red_color >= blue_min && red_color <= blue_max) rgba += vec4(0, 0, red_color, red_color/3);

    return rgba;
}

// Computes the normal vector of a given point using its neighbour points
vec3 compute_normal(vec3 curr_pos) {
    float dx = 1.0 / 256; // Small delta to compute neighbour pixels

    // Take the 6 samples
    float negx  = texture(volume, vec3(curr_pos.x - dx, curr_pos.y, curr_pos.z)).x;
    float posx  = texture(volume, vec3(curr_pos.x + dx, curr_pos.y, curr_pos.z)).x;
    float negy  = texture(volume, vec3(curr_pos.x, curr_pos.y - dx, curr_pos.z)).x;
    float posy  = texture(volume, vec3(curr_pos.x, curr_pos.y + dx, curr_pos.z)).x;
    float negz  = texture(volume, vec3(curr_pos.x, curr_pos.y, curr_pos.z - dx)).x;
    float posz  = texture(volume, vec3(curr_pos.x, curr_pos.y, curr_pos.z + dx)).x;

    // The difference between each sample (on the same axis) will represent the normal vector (then it's normalized)
    return normalize(vec3(negx - posx, negy - posy, negz - posz));
}

void main (void) {
    vec3 pos = tex_coords.xyz;
```

```
    vec3 dir = normalize(position - camera); // Vector from the camera to the point
    vec3 light_color = vec3(1, 1, 1); // Light color set to white

    float steps = 512; // Maximum number of steps that the ray will make when traversing the
cube
    float alpha_threshold = 0.95f; // Empirical threshold

    // Limits of the volume bounding box (the textured cube)
    vec3 volExtentMin = vec3(0.0);
    vec3 volExtentMax = vec3(1.0);

    vec4 current_color;
    vec4 acc_color = vec4(0, 0, 0, 0); // Initialise accumulated color and opacity

    float ka = 0.2, kd = 0.7, ks = 1.2; // Fixed k values for the blinn-phong shading computation
    float specularN = 10.0; // This one will be fixed too

    // Ray traversal loop
    for (int i = 0; i < steps; i++) {
        // Get the current color (red component) and apply transfer function to it to get the rgba
color which we will work on
        current_color = transfer_function(texture(volume, pos).x);

        vec3 norm = compute_normal(pos); // Compute the normal vector
        vec3 light_dir = normalize(light_pos - (pos - vec3(0.5))); // Compute the light direction
(the -0.5 offset is necessary due to the cube's position)

        // Compute Blinn-Phong Shading
        float diffuse = max(dot(norm, light_dir), 0.0); // Diffuse component

        vec3 light_reflect = reflect(-light_dir, norm);  // Calculate the reflection of the light with
respect to the already computed normal vector
        float specular = pow( max(dot(dir, light_reflect), 0.0) , specularN ); // Specular component

        vec3 diffuse_color = diffuse * light_color; // Multiply it by the color of the light
        vec3 specular_color = specular * light_color;

        vec3 phong = ka * light_color + kd * diffuse_color + ks * specular_color; // Add all the
terms together...
        current_color.rgb = phong * current_color.rgb; // … and apply it to the current color

        // Add rgb-color and alpha to the accumulated result (front-to-back compositing)
        acc_color.rgb += (1.0 - acc_color.a) * current_color.rgb * current_color.a;
        acc_color.a += (1.0 - acc_color.a) * current_color.a;
```

```
        // Advance ray position (taking into account the number of steps decided earlier)
        pos += dir * vec3(1.0 / steps);

        // Test if we are outside the volume (the method from the slides)
        vec3 temp1 = sign(pos - volExtentMin);
        vec3 temp2 = sign(volExtentMax - pos);
        float inside = dot(temp1, temp2);
        if (inside < 3.0) break; // Exit the loop

       // Test if the alpha value is high enough (over the empirical threshold previously defined)
        if (acc_color.a >= alpha_threshold) break;
    }

    frag_color = acc_color; // The final accumulated color
}
```