ECOLE POLYTECHNIQUE DE BRUXELLES

INFO-H502 - Virtual Reality

OpenGL project

Luka BONHEURE

000494375

January 8, 2024

1 Presentation

The project is fairly simple, containing just 4 objects:

- A simple, plain dark green floor. The mesh is stored in Object format (.obj)".
- A low poly tree. It's also a solid dark green colour, but it has lighting. The mesh is stored in Collada (.dae) format.
- A guard, swinging his lantern to see better in the dark. This is an animated object, with skinning and skeleton animation. The mesh is stored in md5 format.
- A cubeMap with a starry sky that frames the whole thing.

You can find the project with all the source code on github `https://github.com/lbonheure/INFO-H502_OpenGL_project_2023-2024`.

1.1 Controls

To navigate in the application, some commands are available:

- "Z" to move forward (zoom) ("Q" in QWERTY),
- "S" to move backward (de-zoom),
- "Q" to move left ("A" in QWERTY),
- "D" to move right,
- directional keys ("up", "down", "left", "right") to control the rotations of the camera,
- "Escape" to exit,
- "Alt" (press and hold) to activate mouse control (and deactivate mouse for rotation)

While Alt key is not pressed, mouse movements control camera rotations.

2 Shaders

There are 4 vertex shaders and 4 fragement shaders (1 of each per object).

2.1 Ground

The shaders used for the ground are very simple.

The vertex shader takes as input the position and normals, as well as the uniforms M, V and P, respectively the model, view and perspective transformation matrices. The output is the vertex position and a vertex colour $v\_col = vec4(normal * 0.5 + 0.5, 1.0)$ passed to the fragement shader.

The fragement shader is even simpler. It takes the input v_col from the vertex shader and outputs the fragement colour, which is simply $v_col * 0.2$.

2.2 Tree

The tree vertex shader is very similar to the ground vertex shader, except that it can receive and transmit texture coordinates and also transmits the vertex position (mutltiplied by M), and the normals without modification instead of *v_col*.

The fragment shader, on the other hand, is much more complex. As well as receiving the local vertex position, normals and texture coordinates, it also receives the camera position via uniforms, as well as all the parameters needed to apply a material and calculate the complete equations of light.

```
struct BaseLight
{
    vec3 Color;
    float AmbientIntensity;
    float DiffuseIntensity;
};

struct DirectionalLight
{
    BaseLight Base;
    vec3 Direction;
};

struct Attenuation
{
    float Constant;
    float Linear;
    float Exp;
};

struct PointLight
{
    BaseLight Base;
    vec3 LocalPos;
    Attenuation Atten;
};

struct SpotLight
{
    PointLight Base;
    vec3 Direction;
    float Cutoff;
};

struct Material
{
    vec3 AmbientColor;
    vec3 DiffuseColor;
    vec3 SpecularColor;
};
```

In this way, the total light can be calculated. First with the 'internal' light: the base light, the light linked to the material and the light linked to the texture. Then we add the light coming from points light and spots light.

```
vec4 CalcLightInternal(BaseLight Light, vec3 LightDirection, vec3 Normal)
```

```
{
    vec4 AmbientColor = vec4(Light.Color, 1.0f) *
                        Light.AmbientIntensity *
                        vec4(gMaterial.AmbientColor, 1.0f);

    float DiffuseFactor = dot(Normal, -LightDirection);

    vec4 DiffuseColor = vec4(0, 0, 0, 0);
    vec4 SpecularColor = vec4(0, 0, 0, 0);

    if (DiffuseFactor > 0) {
        DiffuseColor = vec4(Light.Color, 1.0f) *
                       Light.DiffuseIntensity *
                       vec4(gMaterial.DiffuseColor, 1.0f) *
                       DiffuseFactor;

        vec3 PixelToCamera = normalize(gCameraLocalPos - LocalPos0);
        vec3 LightReflect = normalize(reflect(LightDirection, Normal));
        float SpecularFactor = dot(PixelToCamera, LightReflect);
        if (SpecularFactor > 0) {
            float SpecularExponent = texture(gSamplerSpecularExponent, TexCoord0).r * 255.0;
            SpecularFactor = pow(SpecularFactor, SpecularExponent);
            SpecularColor = vec4(Light.Color, 1.0f) *
                            Light.DiffuseIntensity *
                            vec4(gMaterial.SpecularColor, 1.0f) *
                            SpecularFactor;
        }
    }
    return (AmbientColor + DiffuseColor + SpecularColor);
}

vec4 CalcPointLight(PointLight l, vec3 Normal)
{
    vec3 LightDirection = LocalPos0 - l.LocalPos;
    float Distance = length(LightDirection);
    LightDirection = normalize(LightDirection);

    vec4 Color = CalcLightInternal(l.Base, LightDirection, Normal);
    float Attenuation =  l.Atten.Constant +
                         l.Atten.Linear * Distance +
                         l.Atten.Exp * Distance * Distance;

    return Color / Attenuation;
}

vec4 CalcSpotLight(SpotLight l, vec3 Normal)
{
    vec3 LightToPixel = normalize(LocalPos0 - l.Base.LocalPos);
    float SpotFactor = dot(LightToPixel, l.Direction);

    if (SpotFactor > l.Cutoff) {
        vec4 Color = CalcPointLight(l.Base, Normal);
        float SpotLightIntensity = (1.0 - (1.0 - SpotFactor)/(1.0 - l.Cutoff));
        return Color * SpotLightIntensity;
    }
    else {
        return vec4(0,0,0,0);
    }
}
```

As the tree has no texture, the output colour is simply a green colour multiplied by the total light.

2.3 Guard

The vertex shader for animation offers additional functionality. It receives input via VBO of up to 10 bone ids and 10 weights. And via uniforms, it receives up to 100 (maximum number of bones) transformation matrices. The total transformation linked to the movement of the bones can therefore be calculated efficiently by adding up the influence of each bone:

```
mat4 boneTransform = mat4(0.0);
boneTransform += gBones[int(BoneIDs0_3.x)] * Weights0_3.x;
boneTransform += gBones[int(BoneIDs0_3.y)] * Weights0_3.y;
boneTransform += gBones[int(BoneIDs0_3.z)] * Weights0_3.z;
boneTransform += gBones[int(BoneIDs0_3.w)] * Weights0_3.w;
boneTransform += gBones[int(BoneIDs4_7.x)] * Weights4_7.x;
boneTransform += gBones[int(BoneIDs4_7.y)] * Weights4_7.y;
boneTransform += gBones[int(BoneIDs4_7.z)] * Weights4_7.z;
boneTransform += gBones[int(BoneIDs4_7.w)] * Weights4_7.w;
boneTransform += gBones[int(BoneIDs8_9.x)] * Weights8_9.x;
boneTransform += gBones[int(BoneIDs8_9.y)] * Weights8_9.y;
```

Bones and weights must each be sent in 3 pieces because the largest data structure that can be transmitted via a single VBO is a vector of size 4.

The vertex position is then calculated by multiplying this transformation matrix with the position received via VBO.

```
vec4 PosL = boneTransform * vec4(position, 1.0);
gl_Position = P*V*M * PosL;
```

The fragment shader is identical to the tree fragment shader, except that as the guard has a texture, this is used instead of a solid colour.

2.4 CubeMap

The vertex shader used for the cubeMap is very simple: it removes the translations from the V matrix and places the image at the maximum z distance (z=1). The fragment shader simply links the coordinate textures to the fragment.

3 Comments

This project has very little content. This is due to the fact that I wanted to embark on an over-ambitious project, and that, limited by time, I had to fall back on the simplest possible with what I had already managed to do.

4 Acknowledgement and references

I have to thank Etay Meiri for his excellent tutorials, which I may have abused a little, and without which I wouldn't have much to present. A large part of my code is directly inspired by his youtube 28 tutorial (`https://github.com/emeiri/ogldev/tree/master`), and some parts are taken directly from it. The guard comes from this tutorial too.

However, I needed another source to get through the skeleton animation: `https://github.com/hasinaxp/skeletal_animation-_assimp_opengl`.

These 2 references are my main sources.

The cubeMap comes from the `https://polyhaven.com/` platform (as an HDRI) and the tree was made by myself on Blender following Alex's tutorial "Comment modéliser un arbre LowPoly ? Blender" (`https://www.youtube.com/watch?v=XAsd5Oph6jc`).