作业 0: WebGL 框架的使用与 Blinn-Phong 着色模型
GAMES202, 2021 年春季

教授: 闫令琪

计算机图形学与混合现实研讨会

GAMES: Graphics And Mixed Environment Seminar

注意:

- 任何更新或更正都将发布在论坛上，因此请偶尔检查一下。
- 论坛链接 `http://games-cn.org/forums/forum/games202/`
- 你必须独立完成自己的作业。
- 你可以在论坛上发布帖子求助，但是发布问题之前，请仔细阅读本文档。

---

1 框架运行

提供的作业框架不会有跨平台问题，同时环境搭建也不会很棘手。学生在运行作业框架前仅需要简单的搭建一个本地服务器，除此之外不需要再手动配置任何环境。

1.1 Visual Studio Code 插件搭建本地服务器 (推荐)

1. 首先在 Visual Studio Code 中安装插件 **Live Server**(`https://github.com/ritwickdey/vscode-live-server`)
2. 在编辑器任意界面使用 **Ctrl+Shift+P** 调出命令行窗口，输入 **Live Server: Open with Live Server**
3. 随后浏览器自动打开指定地址的本地服务器，至此作业框架的运行就算完成了。

1.2 Node.js 搭建本地服务器

1. 首先，我们需要安装 Node.js 的包管理工具——npm ，有关安装教程请遵循: `https://www.npmjs.com/package/npm`
2. 接下来，我们就可以使用 npm 全局安装一个基于 Node.js 的轻量级 HTTP 服务器:

```
npm install http-server -g
```

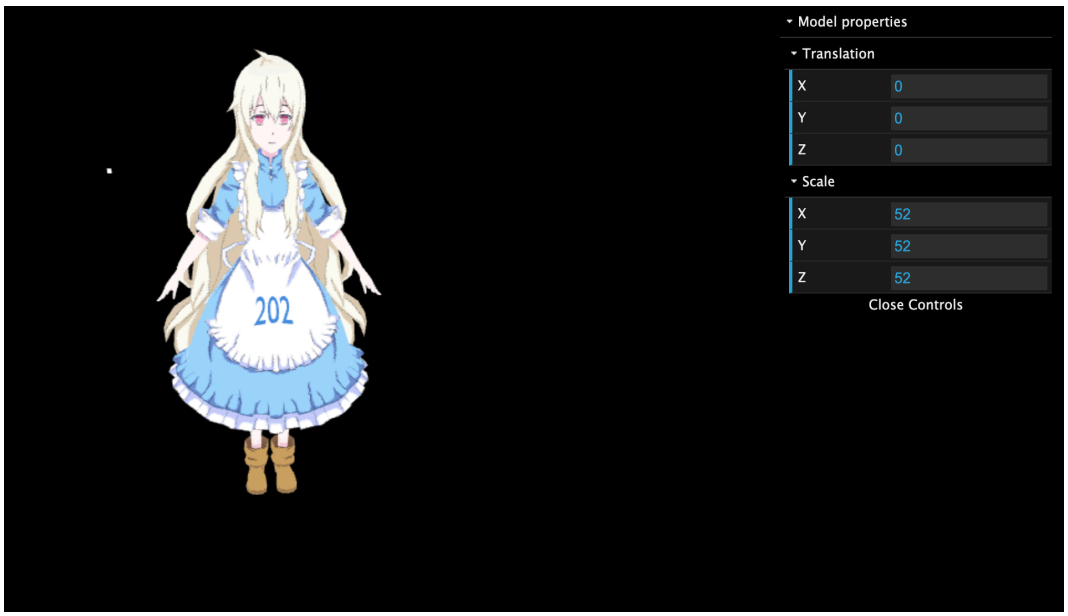
3. 安装完 http-server 之后，进入待打开网页的所在目录并开启本地服务器，即在终端中依次输入:

```
cd homework0
http-server . -p 8000
```

4. 随后复制给出的地址到浏览器打开，至此作业框架的运行就算完成了。

如果一切顺利的话，你将会看到一个加载出来的模型、一个移动的点光源以及一个可以对模型进行移动和缩放的面板。除此之外，你还可以通过鼠标与模型进行简单的交互：

- 按住鼠标右键旋转相机
- 通过鼠标滚轮缩放视角
- 按住鼠标左键移动相机
- 通过 GUI 调整模型的位置和大小



2 作业框架的说明

2.1 开发说明

课程推荐 VSCode 编辑代码，并在 Chrome 中运行调试。作业涉及到的知识包括简单的 js 语法、WebGL、GLSL。更多相关学习资源，请查阅`https://developer.mozilla.org/zh-CN/docs/Web/API/WebGL_API` 或 Stack Overflow 。

注意: 课程使用 WebGL

2.2 Three.js 库

作业框架中仅用到了少量 Three.js 库工具，例如本轮作业中包括 camera、mesh loader 以及 arcball，其余部分均为全新的封装实现。综上对作业任务的完成来说，并不会要求对该库有特别的掌握。

2.3 GLSL

作业的主要工作会集中在 shader 编写上，相关资料可以查阅 `https://www.khronos.org/opengles/sdk/docs/manglsl/docbook4/`

2.4 Blinn-Phong 着色模型

准备工作完成之后，下面将通过一个简单任务来熟悉基本作业流程。本轮任务将创建一种 PhongMaterial 材质，该材质使用 Blinn-Phong 着色模型，需要用户编写对应的 vertexShader 以及 fragmentShader。如果一切正常的话，最终渲染结果将会有不错的高光表现，同时可以和我们的移动光源呈现出有趣的交互效果。

为了简化作业流程，包括模型及其附属文件的读取解析，Shader 编译、Shader 和其参数的绑定、传值等都已封装完全并会自动完成。本次作业中我们重点需要了解的是 Material 系统，我们提供了一种创建材质的方法，对本例来讲我们可以通过从 Material 基类继承创建 PhongMaterial 子类，随后编写、传递给 PhongMaterial 对应的 PhongShader 主体与所需参数。

接下来我们就可以将关注点移动到 Shader 的编写、Material 的实现两个主要环节了。

主要步骤将包括:

- 使用 GLSL 语言实现 Blinn-Phong 模型的 vertexShader 和 fragmentShader。
- 使用 Javascript 语言实现 PhongMaterial 类。该类继承于 Material 类, 我们需要在其构造函数中传递对应字符串形式的 vertexShader 和 fragmentShader, 以及 Shader 所需要的变量。
- 为了使用编写好的 PhongMaterial 类，在 index.html 中将其导入。
- 将框架中原来使用的 Material 替换为我们编写好的 PhongMaterial 类。

首先,创建 PhongMaterial 要使用的 PhongVertexShader、PhongFragmentShader。打开 shaders 文件夹中的 InternalShader.js 文件,添加下列两个字符串作为 PhongMaterial 将用到的 Shader（注意：本次作业采用从字符串加载 Shader，从文本加载 Shader 可根据兴趣练习）:

```
const PhongVertexShader = `
attribute vec3 aVertexPosition;
attribute vec3 aNormalPosition;
attribute vec2 aTextureCoord;

uniform mat4 uModelViewMatrix;
uniform mat4 uProjectionMatrix;

varying highp vec2 vTextureCoord;
varying highp vec3 vFragPos;
varying highp vec3 vNormal;

void main(void) {

  vFragPos = aVertexPosition;
  vNormal = aNormalPosition;

  gl_Position = uProjectionMatrix * uModelViewMatrix * vec4(aVertexPosition, 1.0);

  vTextureCoord = aTextureCoord;
}
```

```
`;

const PhongFragmentShader = `
#ifdef GL_ES
precision mediump float;
#endif
uniform sampler2D uSampler;
//binn
uniform vec3 uKd;
uniform vec3 uKs;
uniform vec3 uLightPos;
uniform vec3 uCameraPos;
uniform float uLightIntensity;
uniform int uTextureSample;

varying highp vec2 vTextureCoord;
varying highp vec3 vFragPos;
varying highp vec3 vNormal;

void main(void) {
  vec3 color;
  if (uTextureSample == 1) {
    color = pow(texture2D(uSampler, vTextureCoord).rgb, vec3(2.2));
  } else {
    color = uKd;
  }

  vec3 ambient = 0.05 * color;

  vec3 lightDir = normalize(uLightPos - vFragPos);
  vec3 normal = normalize(vNormal);
  float diff = max(dot(lightDir, normal), 0.0);
  float light_atten_coff = uLightIntensity / length(uLightPos - vFragPos);
  vec3 diffuse =  diff * light_atten_coff * color;

  vec3 viewDir = normalize(uCameraPos - vFragPos);
  float spec = 0.0;
  vec3 reflectDir = reflect(-lightDir, normal);
  spec = pow (max(dot(viewDir, reflectDir), 0.0), 35.0);
  vec3 specular = uKs * light_atten_coff * spec;

  gl_FragColor = vec4(pow((ambient + diffuse + specular), vec3(1.0/2.2)), 1.0);
}
`;
```

接着，我们将在 Materials 文件夹中创建 PhongMaterial.js，并在该文件中实现继承自 Material 基类的 PhongMaterial 类。在 PhongMaterial 的构造函数中，会为 PhongShader 所需要的 uniform 和 attribute 变量传递相应的 map 和 array，以及指定要用到的 vertexShader 和 fragmentShader，上述会一起到基类 (Material) 的构造函数中。（需要注意，Material 基类中的 uniforms 默认包含 uModelViewMatrix，uProjectionMatrix，uCameraPos，uLightPos 四个 Shader 常用的 uniform 变量，这可以简化继承类材质的构造过程）。

在实例化 PhongMaterial 并以此进一步构造 MeshRender 类时，MeshRender 的构造函数会调用 material.compile(gl)，该方法自动完成材质用到的 Shader 编译，以及参数和 Shader 的绑定。由此用户在向材质构造函数中传递完 Shader 用到的 uniform、attribute 变量后，在 Shader 中就可以认为这些对应的变量可用了。

你可以将以下代码复制粘贴到刚刚创建的 PhongMaterial.js 中。

```
class PhongMaterial extends Material {

    /**
     * Creates an instance of PhongMaterial.
     * @param {vec3f} color The material color
     * @param {Texture} colorMap The texture object of the material
     * @param {vec3f} specular The material specular coefficient
     * @param {float} intensity The light intensity
     * @memberof PhongMaterial
     */
    constructor(color, colorMap, specular, intensity) {
        let textureSample = 0;

        if (colorMap != null) {
            textureSample = 1;
            super({
                'uTextureSample': { type: '1i', value: textureSample },
                'uSampler': { type: 'texture', value: colorMap },
                'uKd': { type: '3fv', value: color },
                'uKs': { type: '3fv', value: specular },
                'uLightIntensity': { type: '1f', value: intensity }
            }, [], PhongVertexShader, PhongFragmentShader);
        } else {
            //console.log(color);
            super({
                'uTextureSample': { type: '1i', value: textureSample },
```

```
                'uKd': { type: '3fv', value: color },
                'uKs': { type: '3fv', value: specular },
                'uLightIntensity': { type: '1f', value: intensity }
            }, [], PhongVertexShader, PhongFragmentShader);
        }

    }
}
```

随后，为了让我们的新建材质（类）能够被应用在模型上，我们需要在 index.html 文件中添加下列代码（第 32 行下），将刚刚创建的材质文件导入项目中。这里需要注意导入文件的顺序，因为 PhongMaterial 继承自 Material 类，所以要在导入 Material.js 之后导入 PhongMaterial.js。

```
<script src="src/materials/PhongMaterial.js" defer></script>
```

最后，将 load 文件夹中 loadOBJ.js 的下列代码删除（第 40-56 行，该部分代码只负责创建物体默认基础材质，即表面基础颜色属性，其使用的默认 Shader 不负责高光和环境光着色。除此之外，为了支持无纹理贴图的材质，我们加上了一点逻辑判断以特殊处理）:

```
  // MARK: You can change the myMaterial object to your own Material instance

  let textureSample = 0;
  let myMaterial;
  if (colorMap != null) {
    textureSample = 1;
    myMaterial = new Material({
      'uSampler': { type: 'texture', value: colorMap },
      'uTextureSample': { type: '1i', value: textureSample },
      'uKd': { type: '3fv', value: mat.color.toArray() }
    },[],VertexShader, FragmentShader);
  }else{
    myMaterial = new Material({
      'uTextureSample': { type: '1i', value: textureSample },
      'uKd': { type: '3fv', value: mat.color.toArray() }
    },[],VertexShader, FragmentShader);
  }
```

将上述删除的代码替换成下列代码，这会创建并使用之前新建的 PhongMaterial 实例：

```
let myMaterial = new PhongMaterial(mat.color.toArray(), colorMap, mat.specular.toArray(),
  renderer.lights[0].entity.mat. intensity);
```

至此，Blinn-Phong 着色模型的应用就算完成了。如果严格按照上述步骤，此时刷新网页你应该可以看到我们的模型能够与光源进行有趣的交互，并有着不错的光影效果。

2.5 JavaScript 与 WebGL 调试注意事项

- 对于 JavaScript 脚本语言的调试，推荐使用 Chrome 浏览器的开发者工具进行调试，结合 **console.log** 进行打印输出
- 因为 JavaScript 语言带有某些异步操作, 建议将所有异步操作串行化, 可以参考`https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/await`
- 对于 WebGL 的调试，建议使用 Chrome 插件 **Spector.js** 进行可视化输出，该插件能够获取每一帧绘制时所调用的所有指令、相关输出以及汇编后的 Shader 代码

3 作业描述与提交

3.1 提交

作业提交使用的平台为 **Smartchair** 平台，地址为`http://www.smartchair.org/GAMES202/`，平台的具体操作说明请在`http://games-cn.org/submit_homework/`下载。

3.2 评分

由于本次作业主要目的是让同学们熟悉课程所使用的 WebGL 框架、使用 js 与 GLSL 编写简单的程序，所以本次作业不进行评分。同学们只需将编写好的程序打包好后提交即可，顺便可以熟悉 **Smartchair** 平台提交作业的流程。