CG-PJ3—说明文档

开发运行环境

- **硬件**：带有支持OpenGL ES2.0显卡的计算机
- **软件**：支持HTML5, JavaScript和WebGL的浏览器，建议选择最新版本的Google Chrome浏览器，也可以选择Internet Explorer 11+或者最新版本的Firefox，Safari等。

用户手册

所有操作方法如下图所示

fog distance: 160

guideline:
W: move forward S: move backward
A: move left D: move right
J: rotate left K: rotate right
F: turn on the point light
arrow_up: increase fog distance
arrow_down: decrease fog distance

代码结构

- lib 相关配置文件
- src 源文件
  - camera.js 定义了一个Camera类，存储相机的元信息，用于生成viewProjectMatrix
  - fog.js 定义了一个Fog类，存储雾的颜色，雾的距离等，用于计算场景中每个物体的模糊度(mix方法)
  - keyController.js 定义了一个KeyController类，该类对象为物体（该pj中我们的物体就是相机）绑定键盘事件（移动和转动），为雾对象绑定调节深度事件（调节雾的距离范围）
  - MyVector3.js 定义了Vector的一些计算方法，在处理旋转平移涉及向量计算时会用到
  - objLoader.js 定义了加载对象的相关方法
  - objModelShader.js 用于obj对象物体渲染的着色器
  - textureModelShader.js 用于带纹理物体渲染的着色器
- 3DWalker.html 主页面

基本功能

- 3D场景纹理贴图: 用textureModelShader实现
- 模型载入: objLoader与objModelShader实现载入，用modelMatrix,viewMatrix进行模型变换
- 3D场景漫游: 用keyController绑定键盘事件改变fog与camera的参数
- 光照: 添加方向光，平行光及点光源(键盘事件F触发，随camera移动)
- 动画: 绘制小鸟时取一个时间戳，进行平移与旋转，距离与角度是一个跟时间戳%360有关的函数
- swichingShader: 每次新绘制物体时，切换program，绑定新的buffer来完成swichShader重新绘制

附加功能

- 雾气的添加

```
// 记录雾的颜色与距离
// 距离的第一个参数50表示，最小的距离，即调整的范围为（50，无穷大），第二个参数160为距离的初始值。
// 在160以内的物体会随着距离增加逐渐模糊（通过fogFactor实现），160以外的物体不可见。
class Fog {
    constructor() {
        this.fogColor = new Float32Array([0.137, 0.231, 0.423]);
        // Distance of fog [where fog starts, where fog completely covers object]
        this.fogDist = new Float32Array([50, 160]);
    }
}
```

遇到的问题

- 平移旋转会涉及到很复杂的数学运算，将'向前W''向后S'等平移转换到世界坐标系的平移的转换最开始想了很久，如果能够有当前方向向量的垂直法向量就可以辅助进行世界坐标系的转换了，怎么求法向量呢？用叉积。

```
translate(x,y,z) {
        let directionVector = VectorMinus(this.at, this.eye);

        let localZ = directionVector.normalize();
        let localY = this.up;
        let localX = VectorCross(localZ, localY);
        let increaseVector = VectorAdd(
            VectorAdd(VectorMultNum(localZ, z),VectorMultNum(localX, x)),
            VectorMultNum(localY, y));
        this.eye = VectorAdd(this.eye, increaseVector);
        this.at = VectorAdd(this.at, increaseVector);

}
rotate(angle) { //弧度制
        let directionVector = VectorMinus(this.at, this.eye);
        let baseVector = VectorMultNum(directionVector, Math.cos(angle));
        let verticalVector = VectorMultNum(VectorCross(directionVector, this.up),
                                           Math.sin(angle));
        let newDirectionVector = VectorAdd(baseVector, verticalVector);
        this.at = VectorAdd(this.eye, newDirectionVector);

}
```

- 动画实现过程中，需要传入时间差，根据时间差进行物体方向的改变和旋转相应的角度，我没有这样实现。这个我想到了一个非常简单的方法，直接根据时间戳进行旋转与平移，具体步骤如下
    - 根据时间戳求出当前秒数，秒数%360得到旋转角度thita
    - 将小鸟旋转thita，这个时候旋转就完成了
    - 根据小鸟的scene中的信息求出中心点到小鸟的方向向量（这个方向向量是初始值，在进行动画之前就求出，跟小鸟的初始位置和中心点有关）
    - 将上述方向向量旋转thita（与相机旋转方法类似），根据旋转后的方向向量求出小鸟的落脚点
    - 平移到小鸟的落脚点，这个时候平移就完成了
- 阴影的添加，阴影比较复杂，研究了网上的一些源代码，发现阴影的处理需要使用两个着色器来进行渲染，第一次负责产生阴影贴图，将每个片元的Z值写入纹理贴图中。第二次渲染时，计算相机坐标系中物体的距离与阴影贴图中的深度进行比较产生visibility,根据visibility进行绘制产生阴影效果。做阴影这一个功能我花了很长很长的时间，从入门到放弃，遇到了一系列非常奇怪的bug，我修改产生阴影的光源的高度在达到一定大小时，阴影就消失了，于是参考教程中的方法修改了我的片源着色器 source如下

```
// Fragment shader program for generating a shadow map
        const SHADOW_FSHADER_SOURCE =
            '#ifdef GL_ES\n' +
            'precision mediump float;\n' +
            '#endif\n' +
            'void main() {\n' +
            '  const vec4 bitShift = vec4(1.0, 256.0, 256.0 * 256.0,
256.0 * 256.0 * 256.0);\n' +
            '  const vec4 bitMask = vec4(1.0/256.0, 1.0/256.0,
1.0/256.0, 0.0);\n' +
            '  vec4 rgbaDepth = fract(gl_FragCoord.z * bitShift);\n' +
// Calculate the value stored into each byte
            '  rgbaDepth -= rgbaDepth.gbaa * bitMask;\n' + // Cut off
the value which do not fit in 8 bits
            '  gl_FragColor = rgbaDepth;\n' +
            '}\n';
```

    最后虽然高度的可控范围变大了，但仍然没有完美地解决这个问题。我在我的代码里为实现阴影已经编写了着色器，修改了draw方法（进行shader切换两次渲染）（虽然最后并没有产生阴影）。阴影的原理非常艰深难以理解（非常耗时间），这个问题先保留，如果后面有时间，希望能够再来完善这个功能。

建议

project3知识性比较强，虽然比较烦但还是挺有趣的，特别是当遇到一些很奇怪的bug比如上述的例子，深究下去会学到很多东西。针对课程建议，我觉得课程应该拓展一些诸如three.js的webgl的库。毕竟three.js写这个pj的话会非常简单，面向webgl变成是一件非常痛苦的事情= =。我觉得可以把pj3换成一个大的project，比如用three.js写一个简单的网页小游戏，等等。

附录

以下是我的shader

```
// Texture shader
        const SHADOW_VSHADER_SOURCE =
            'attribute vec4 a_Position;\n' +
            'uniform mat4 u_MvpMatrix;\n' +
            'void main() {\n' +
            '  gl_Position = u_MvpMatrix * a_Position;\n' +
            '}\n';
        const SHADOW_FSHADER_SOURCE =
            '#ifdef GL_ES\n' +
            'precision mediump float;\n' +
            '#endif\n' +
            'void main() {\n' +
            '  const vec4 bitShift = vec4(1.0, 256.0, 256.0 * 256.0, 256.0
* 256.0 * 256.0);\n' +
            '  const vec4 bitMask = vec4(1.0/256.0, 1.0/256.0, 1.0/256.0,
0.0);\n' +
            '  vec4 rgbaDepth = fract(gl_FragCoord.z * bitShift);\n' + //
Calculate the value stored into each byte
            '  rgbaDepth -= rgbaDepth.gbaa * bitMask;\n' + // Cut off the
value which do not fit in 8 bits
            '  gl_FragColor = rgbaDepth;\n' +
            '}\n';

        const VSHADER_SOURCE =
            'attribute vec4 a_Position;\n' +
            'attribute vec4 a_Normal;\n'+
            'attribute vec2 a_TexCoord;\n' +
            'attribute vec4 a_Color;\n' +


            'uniform mat4 u_MvpMatrix;\n' +
            'uniform mat4 u_ModelMatrix;\n' +   // Model matrix
            'uniform mat4 u_NormalMatrix;\n' +
            'uniform mat4 u_MvpMatrixFromLight;\n' +
            'uniform vec4 u_Eye;\n' +     // Position of eye point (world
coordinates)


            'uniform vec3 u_DiffuseLight;\n' +   // Diffuse light color
            'uniform vec3 u_LightDirection;\n' + // Diffuse light direction
(in the world coordinate, normalized)
            'uniform vec3 u_AmbientLight;\n' +   // Color of an ambient
light
            'uniform vec3 u_PointLightColor;\n' +
```

```
            'uniform vec3 u_PointLightPosition;\n' +

            'varying vec2 v_TexCoord;\n' +
            'varying vec4 v_Color;\n' +
            'varying vec4 v_PositionFromLight;\n' +
            'varying float v_Dist;\n' +


            'void main() {\n' +
            '  gl_Position = u_MvpMatrix * a_Position;\n' +
            '  v_TexCoord = a_TexCoord;\n' +

            '  vec3 normal = normalize(vec3(u_NormalMatrix * a_Normal));\n'
+
            '  vec4 vertexPosition = u_ModelMatrix * a_Position;\n'+

            '  vec3 pointLightDirection = normalize(u_PointLightPosition -
vec3(vertexPosition));\n' +
            '  float pointLightnDotL = max(dot(pointLightDirection,
normal), 0.2);\n' +
            '  vec3 pointLightDiffuse = u_PointLightColor *
pointLightnDotL;\n' +


            '  float nDotL = max(dot(u_LightDirection, normal), 0.1);\n' +
            '  vec3 diffuse = u_DiffuseLight * nDotL;\n' +
            '  vec3 ambient = u_AmbientLight * 0.1;\n' +

            '  v_Color = vec4(diffuse + ambient + pointLightDiffuse,
1.0);\n' +
            '  v_Dist = distance(u_ModelMatrix * a_Position, u_Eye);\n' +


            '}\n';

        // Fragment shader program
        const FSHADER_SOURCE =

            '#ifdef GL_ES\n' +
            'precision mediump float;\n' +
            '#endif\n' +
            'uniform sampler2D u_Sampler;\n' +
            'uniform sampler2D u_ShadowMap;\n' +

            'uniform vec3 u_FogColor;\n' + // Color of Fog
            'uniform vec2 u_FogDist;\n' +  // Distance of Fog (starting
point, end point)

            'varying vec2 v_TexCoord;\n' +
```

```
            'varying vec4 v_Color;\n' +
            'varying vec4 v_PositionFromLight;\n' +
            'varying float v_Dist;\n' +

            'float unpackDepth(const in vec4 rgbaDepth) {\n' +
            '  const vec4 bitShift = vec4(1.0, 1.0/256.0,
1.0/(256.0*256.0), 1.0/(256.0*256.0*256.0));\n' +
            '  float depth = dot(rgbaDepth, bitShift);\n' + // Use dot()
since the calculations is same
            '  return depth;\n' +
            '}\n' +

            'void main() {\n' +
            '  vec3 shadowCoord =
(v_PositionFromLight.xyz/v_PositionFromLight.w)/2.0 + 0.5;\n' +
            '  vec4 rgbaDepth = texture2D(u_ShadowMap, shadowCoord.xy);\n'
+
            '  float depth = unpackDepth(rgbaDepth);\n' + // Retrieve the
z-value from R
            '  float visibility = (shadowCoord.z > depth + 0.005) ? 0.5 :
1.0;\n' +

            '  float fogFactor = clamp((u_FogDist.y - v_Dist) /
(u_FogDist.y - u_FogDist.x), 0.0, 1.0);\n' +
            '  vec3 color = mix(u_FogColor, vec3(v_Color +
texture2D(u_Sampler, v_TexCoord)) * visibility, fogFactor);\n' +

            '  gl_FragColor = vec4(color, v_Color.a);\n' +
            //'  gl_FragColor = vec4(color, v_Color.a);\n' +

            '}\n';




// objShader
        const SHADOW_VSHADER_SOURCE =
            'attribute vec4 a_Position;\n' +
            'uniform mat4 u_MvpMatrix;\n' +
            'void main() {\n' +
            '  gl_Position = u_MvpMatrix * a_Position;\n' +
            '}\n';
        const SHADOW_FSHADER_SOURCE =
            '#ifdef GL_ES\n' +
            'precision mediump float;\n' +
            '#endif\n' +
            'void main() {\n' +
            '  const vec4 bitShift = vec4(1.0, 256.0, 256.0 * 256.0, 256.0
* 256.0 * 256.0);\n' +
```

```
            '  const vec4 bitMask = vec4(1.0/256.0, 1.0/256.0, 1.0/256.0,
0.0);\n' +
            '  vec4 rgbaDepth = fract(gl_FragCoord.z * bitShift);\n' + //
Calculate the value stored into each byte
            '  rgbaDepth -= rgbaDepth.gbaa * bitMask;\n' + // Cut off the
value which do not fit in 8 bits
            '  gl_FragColor = rgbaDepth;\n' +
            '}\n';
        const VSHADER_SOURCE =
            'attribute vec4 a_Position;\n' +
            'attribute vec4 a_Color;\n' +
            'attribute vec4 a_Normal;\n' +
            'uniform mat4 u_MvpMatrix;\n' +
            'uniform mat4 u_ModelMatrix;\n' +   // Model matrix
            'uniform mat4 u_NormalMatrix;\n' +
            'uniform vec3 u_DiffuseLight;\n' +   // Diffuse light color
            'uniform vec3 u_LightDirection;\n' + // Diffuse light direction
(in the world coordinate, normalized)
            'uniform vec3 u_AmbientLight;\n' +   // Color of an ambient
light
            'uniform vec3 u_PointLightColor;\n' +
            'uniform vec3 u_PointLightPosition;\n' +
            'uniform vec3 u_Color;\n' +
            'uniform vec4 u_Eye;\n' +     // Position of eye point (world
coordinates)
            'varying vec4 v_Color;\n' +
            'varying float v_Dist;\n' +
            'varying vec4 v_PositionFromLight;\n' +


            'void main() {\n' +
            '  gl_Position = u_MvpMatrix * a_Position;\n' +
            '  vec3 normal = normalize(vec3(u_NormalMatrix * a_Normal));\n'
+
            '  vec4 vertexPosition = u_ModelMatrix * a_Position;\n' +

            '  vec3 pointLightDirection = normalize(u_PointLightPosition -
vec3(vertexPosition));\n' +
            '  float pointLightnDotL = max(dot(pointLightDirection,
normal), 0.0);\n' +
            '  vec3 pointLightDiffuse = u_PointLightColor * u_Color *
pointLightnDotL;\n' +

            '  float nDotL = max(dot(u_LightDirection, normal), 0.0);\n' +
            '  vec3 diffuse = u_DiffuseLight * u_Color * nDotL;\n' +
            '  vec3 ambient = u_AmbientLight * u_Color;\n' +

            '  v_Color = vec4(diffuse + ambient + pointLightDiffuse,
a_Color.a);\n' +
```

```
            '  v_Dist = distance(u_ModelMatrix * a_Position, u_Eye);\n' +

            '}\n';
        const FSHADER_SOURCE =
            '#ifdef GL_ES\n' +
            'precision mediump float;\n' +
            '#endif\n' +
            'uniform sampler2D u_ShadowMap;\n' +

            'uniform vec3 u_FogColor;\n' + // Color of Fog
            'uniform vec2 u_FogDist;\n' +  // Distance of Fog (starting
point, end point)

            'varying vec4 v_Color;\n' +
            'varying float v_Dist;\n' +
            'varying vec4 v_PositionFromLight;\n' +

            'float unpackDepth(const in vec4 rgbaDepth) {\n' +
            '  const vec4 bitShift = vec4(1.0, 1.0/256.0,
1.0/(256.0*256.0), 1.0/(256.0*256.0*256.0));\n' +
            '  float depth = dot(rgbaDepth, bitShift);\n' + // Use dot()
since the calculations is same
            '  return depth;\n' +
            '}\n' +

            'void main() {\n' +
            '  vec3 shadowCoord =
(v_PositionFromLight.xyz/v_PositionFromLight.w)/2.0 + 0.5;\n' +
            '  vec4 rgbaDepth = texture2D(u_ShadowMap, shadowCoord.xy);\n'
+
            '  float depth = unpackDepth(rgbaDepth);\n' + // Retrieve the
z-value from R
            '  float visibility = (shadowCoord.z > depth + 0.005) ? 0.5 :
1.0;\n' +

            '  float fogFactor = clamp((u_FogDist.y - v_Dist) /
(u_FogDist.y - u_FogDist.x), 0.0, 1.0);\n' +
            '  vec3 color = mix(u_FogColor, vec3(v_Color) * visibility,
fogFactor);\n' +
            '  gl_FragColor = vec4(color, v_Color.a);\n' +
            '}\n';
```