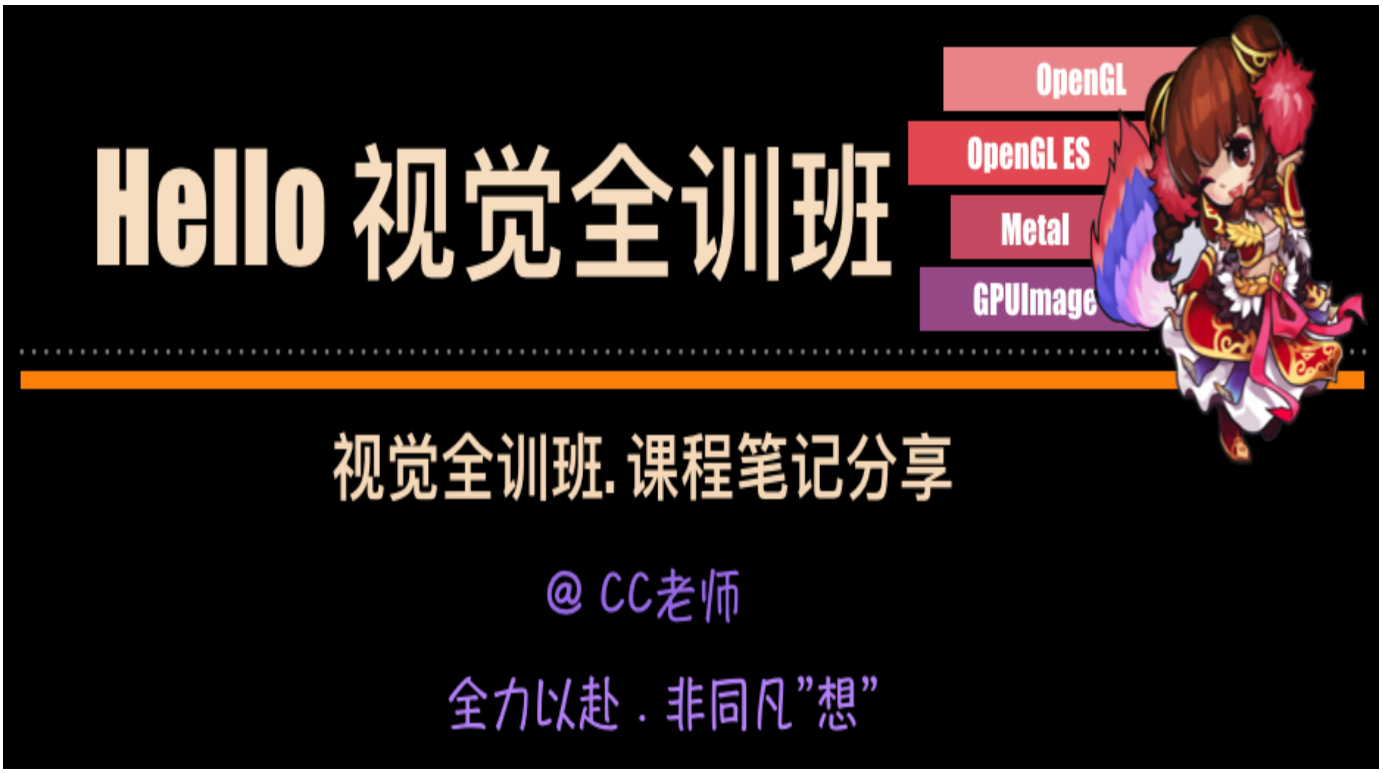
018--视觉班第18次课程[OpenGL ES专题]



一. 视觉班课程安排:

- 课程日期: 2020 年 8 月 12 日 周三 第 18 次课程 (共 21 次课程)
- 授课老师: CC 老师 (QQ: 1323177506)
- 研发老师: CC 老师
- 班主任老师:
  - 大大老师 (QQ: 188706023)
  - 朵朵老师 (QQ: 1550934962)
  - 婷婷老师 (QQ: 3470520842)
- 课程时长: 2小时
- 课程时间安排:
  - 上课: 20:00 – 21:00
  - 休息: 21:00 – 21:10
  - 上课: 21:10 – 22:00
- 课程内容:
  - 缩放滤镜
  - 灵魂出窍滤镜
  - 抖动滤镜;
  - 闪白滤镜:

    - 毛刺滤镜
    - 幻觉滤镜
- 课程作业:
    - 练习课堂上的案例
    - 博客: 将滤镜算法总结于博客中;

缩放滤镜.vsh

```
//顶点坐标
attribute vec4 Position;
//纹理坐标
attribute vec2 TextureCoords;
//纹理坐标
varying vec2 TextureCoordsVarying;
//时间撮(及时更新)
uniform float Time;
//PI
const float PI = 3.1415926;

void main (void) {
    //一次缩放效果时长 = 0.6ms
    float duration = 0.6;
    //最大缩放幅度
    float maxAmplitude = 0.3;

    //表示传入的时间周期.即time的范围被控制在[0.0~0.6];
    //mod(a,b),求模运算. a%b
    float time = mod(Time, duration);

    //amplitude 表示振幅，引入 PI 的目的是为了使用 sin 函数，将 amplitude 的范围控制在 1.0 ~ 1.3 之间，并随着时间变化
    float amplitude = 1.0 + maxAmplitude * abs(sin(time * (PI / duration)));

    //放大关键: 将顶点坐标的 x 和 y 分别乘上一个放大系数，在纹理坐标不变的情况下，就达到了拉伸的效果。
    //x,y 放大; z和w保存不变
    gl_Position = vec4(Position.x * amplitude, Position.y * amplitude, Position.zw);

    ////纹理坐标传递给TextureCoordsVarying
    TextureCoordsVarying = TextureCoords;
```

```
}
```

灵魂出窍滤镜.fsh

```
//「灵魂出窍」看上去是两个层的叠加，并且上面的那层随着时间的推移，会逐渐放大且不透明度逐渐降低。这里也用到了放大的效果，我们这次用片段着色器来实现

precision highp float;
//纹理采样器
uniform sampler2D Texture;
//纹理坐标
varying vec2 TextureCoordsVarying;
//时间撮
uniform float Time;

void main (void) {
    //一次灵魂出窍的时长
    float duration = 0.7;
    //透明度上限
    float maxAlpha = 0.4;
    //放大图片上限
    float maxScale = 1.8;

    //进度(0~1)
    float progress = mod(Time, duration) / duration; // 0~1
    //透明度(0-0.4)
    float alpha = maxAlpha * (1.0 - progress);
    //缩放比例(1.0 - 1.8)
    float scale = 1.0 + (maxScale - 1.0) * progress;

    //放大纹理坐标
    //将顶点坐标对应的纹理坐标的 x 值到纹理中点的距离，缩小一定的比例。这次我们是改变了纹理坐标，而保持顶点坐标不变，同样达到了拉伸的效果
    float weakX = 0.5 + (TextureCoordsVarying.x - 0.5) / scale;
    float weakY = 0.5 + (TextureCoordsVarying.y - 0.5) / scale;

    //获取放大的纹理坐标
    vec2 weakTextureCoords = vec2(weakX, weakY);
```

```
    //通过上面的计算，我们得到了两个纹理颜色值 weakMask 和 mask， weakMask 是在 mask 的基础上做了放大处理
    //读取到放大后的纹理坐标的纹素的颜色值
    vec4 weakMask = texture2D(Texture, weakTextureCoords);

    //读取原始的纹理坐标对应的纹素的颜色值
    vec4 mask = texture2D(Texture, TextureCoordsVarying);

    //在GLSL 实现颜色混合方程式.默认颜色混合方程式 = mask * (1.0 – alpha) + weakMask * alpha
    //参考OpenGL 第二节课中的颜色混合
    //计算最终的颜色赋值给片元着色器的内置变量: gl_FragColor
    gl_FragColor = mask * (1.0 – alpha) + weakMask * alpha;
}
```

抖动滤镜.fsh

```
precision highp float;
//纹理采样器
uniform sampler2D Texture;
//纹理采样器
varying vec2 TextureCoordsVarying;
//时间撮
uniform float Time;

void main (void) {

    //一次抖动滤镜的时长
    float duration = 0.7;
    //放大图片上限
    float maxScale = 1.1;
    //颜色偏移步长
    float offset = 0.02;

    //进度 0~1
    float progress = mod(Time, duration) / duration; // 0~1
    //颜色偏移值(0-0.02)
    vec2 offsetCoords = vec2(offset, offset) * progress;
    //缩放比例(1.0 – 1.1)
    float scale = 1.0 + (maxScale – 1.0) * progress;
```

```
    //放大后的纹理坐标
    vec2 ScaleTextureCoords = vec2(0.5, 0.5) + (TextureCoordsVarying - vec2(0.5, 0.5)) /
scale;

    //获取3组颜色
    //原始颜色 + offsetCoords
    vec4 maskR = texture2D(Texture, ScaleTextureCoords + offsetCoords);
    //原始颜色 - offsetCoords
    vec4 maskB = texture2D(Texture, ScaleTextureCoords - offsetCoords);
    //原始颜色
    vec4 mask = texture2D(Texture, ScaleTextureCoords);

    //从3组颜色中分别取出: 红色R,绿色G,蓝色B,透明度A填充到片元着色器内置变量gl_FragColor内.
    gl_FragColor = vec4(maskR.r, mask.g, maskB.b, mask.a);
}
```

闪白滤镜.fsh

```
precision highp float;
//纹理采样器
uniform sampler2D Texture;
//纹理坐标
varying vec2 TextureCoordsVarying;
//时间撮
uniform float Time;
//PI 常量
const float PI = 3.1415926;

void main (void) {

    //一次闪白滤镜的时长
    float duration = 0.6;
    //表示将传入的时间转换到一个周期内，即 time 的范围是 0 ~ 0.6
    float time = mod(Time, duration);

    //白色颜色遮罩
    vec4 whiteMask = vec4(1.0, 1.0, 1.0, 1.0);
    //amplitude 表示振幅，引入 PI 的目的是为了使用 sin 函数，将 amplitude 的范围控制在 0.0 ~ 1.0
之间，并随着时间变化
```

```
    float amplitude = abs(sin(time * (PI / duration)));
    //获取纹理坐标对应的纹素颜色值
    vec4 mask = texture2D(Texture, TextureCoordsVarying);

    //利用混合方程式: 将纹理颜色与白色遮罩融合.
    //注意: 白色遮罩的透明度会随着时间变化做调整
    //我们已经知道了如何实现两个层的叠加.当前的透明度来计算最终的颜色值即可。
    gl_FragColor = mask * (1.0 - amplitude) + whiteMask * amplitude;
}
```

毛刺滤镜.fsh

```
precision highp float;
//纹理采样器
uniform sampler2D Texture;
//纹理坐标
varying vec2 TextureCoordsVarying;
//时间撮
uniform float Time;
//PI 常量
const float PI = 3.1415926;

//随机数
float rand(float n) {
    //fract(x),返回x的小数部分
    //返回: sin(n) * 43758.5453123
    return fract(sin(n) * 43758.5453123);
}

void main (void) {

    //最大抖动
    float maxJitter = 0.06;
    //一次毛刺滤镜的时长
    float duration = 0.3;
    //红色颜色偏移
    float colorROffset = 0.01;
    //绿色颜色偏移
    float colorBOffset = -0.025;
```

```
    //表示将传入的时间转换到一个周期内，即 time 的范围是 0 ~ 0.6
    float time = mod(Time, duration * 2.0);
    //amplitude 表示振幅，引入 PI 的目的是为了使用 sin 函数，将 amplitude 的范围控制在 1.0 ~ 1.3 之间，并随着时间变化
    float amplitude = max(sin(time * (PI / duration)), 0.0);
    // -1~1 像素随机偏移范围(-1,1)
    float jitter = rand(TextureCoordsVarying.y) * 2.0 - 1.0; // -1~1

    //判断是否需要偏移,如果jtter范围< 最大抖动*振幅
    bool needOffset = abs(jitter) < maxJitter * amplitude;

    //获取纹理x 坐标,根据needOffset,来计算它的X撕裂,如果是needOffset = yes 则撕裂大;如果needOffset = no 则撕裂小;
    float textureX = TextureCoordsVarying.x + (needOffset ? jitter : (jitter * amplitude * 0.006));

    //获取纹理撕裂后的x,y坐标
    vec2 textureCoords = vec2(textureX, TextureCoordsVarying.y);

    //颜色偏移
    //获取3组颜色: 根据撕裂计算后的纹理坐标,获取纹素的颜色
    vec4 mask = texture2D(Texture, textureCoords);
    //获取3组颜色: 根据撕裂计算后的纹理坐标,获取纹素的颜色
    vec4 maskR = texture2D(Texture, textureCoords + vec2(colorROffset * amplitude, 0.0));
    //获取3组颜色: 根据撕裂技术后的纹理坐标,获取纹素颜色
    vec4 maskB = texture2D(Texture, textureCoords + vec2(colorBOffset * amplitude, 0.0));

    //颜色主要撕裂: 红色和蓝色部分.所以只调整红色
    gl_FragColor = vec4(maskR.r, mask.g, maskB.b, mask.a);
}
```

幻觉滤镜.fsh

```
precision highp float;
//纹理采样器
uniform sampler2D Texture;
//纹理坐标
varying vec2 TextureCoordsVarying;
//时间撮
uniform float Time;
//PI 常量
```

```
const float PI = 3.1415926;
//一次幻觉滤镜的时长
const float duration = 2.0;

//这个函数可以计算出，在某个时刻图片的具体位置。通过它我们可以每经过一段时间，去生成一个新的层
vec4 getMask(float time, vec2 textureCoords, float padding) {

    //圆周坐标
    vec2 translation = vec2(sin(time * (PI * 2.0 / duration)),
                            cos(time * (PI * 2.0 / duration)));

    //纹理坐标 = 纹理坐标+偏离量 * 圆周坐标
    vec2 translationTextureCoords = textureCoords + padding * translation;

    //根据这个坐标获取新图层的坐标
    vec4 mask = texture2D(Texture, translationTextureCoords);

    return mask;
}

//这个函数可以计算出，某个时刻创建的层，在当前时刻的透明度。
float maskAlphaProgress(float currentTime, float hideTime, float startTime) {

    //duration+currentTime-startTime % duration
    float time = mod(duration + currentTime - startTime, duration);
    return min(time, hideTime);
}

void main (void) {

    //表示将传入的时间转换到一个周期内，即 time 的范围是 0 ~ 2.0
    float time = mod(Time, duration);

    //放大倍数
    float scale = 1.2;
    //偏移量
    float padding = 0.5 * (1.0 - 1.0 / scale);
    //放大后的纹理坐标
    vec2 textureCoords = vec2(0.5, 0.5) + (TextureCoordsVarying - vec2(0.5, 0.5)) / scale;
```

```
    //隐藏时间
    float hideTime = 0.9;
    //时间间隔
    float timeGap = 0.2;

    //注意: 只保留了红色的透明的通道值,因为幻觉效果残留红色.
    //新图层的-R色透明度 0.5
    float maxAlphaR = 0.5; // max R
    //新图层的-G色透明度 0.05
    float maxAlphaG = 0.05; // max G
    //新图层的-B色透明度 0.05
    float maxAlphaB = 0.05; // max B

   //获得新的图层坐标!
    vec4 mask = getMask(time, textureCoords, padding);
    float alphaR = 1.0; // R
    float alphaG = 1.0; // G
    float alphaB = 1.0; // B

    //最终图层颜色
    vec4 resultMask = vec4(0, 0, 0, 0);

    //循环
    for (float f = 0.0; f < duration; f += timeGap) {

        float tmpTime = f;
        //获取到0-2.0秒内所获取的运动后的纹理坐标
        vec4 tmpMask = getMask(tmpTime, textureCoords, padding);

        //某个时刻创建的层，在当前时刻的红绿蓝的透明度
        float tmpAlphaR = maxAlphaR - maxAlphaR * maskAlphaProgress(time, hideTime,
tmpTime) / hideTime;
        float tmpAlphaG = maxAlphaG - maxAlphaG * maskAlphaProgress(time, hideTime,
tmpTime) / hideTime;
        float tmpAlphaB = maxAlphaB - maxAlphaB * maskAlphaProgress(time, hideTime,
tmpTime) / hideTime;

        //累积每一层每个通道乘以透明度颜色通道
        resultMask += vec4(tmpMask.r * tmpAlphaR,
```

```
                    tmpMask.g * tmpAlphaG,
                    tmpMask.b * tmpAlphaB,
                    1.0);
        //透明度递减
        alphaR -= tmpAlphaR;
        alphaG -= tmpAlphaG;
        alphaB -= tmpAlphaB;
    }

    //最终颜色 += 红绿蓝 * 透明度
    resultMask += vec4(mask.r * alphaR, mask.g * alphaG, mask.b * alphaB, 1.0);

    //将最终颜色填充到像素点里.
    gl_FragColor = resultMask;
}
```