017--视觉班第17次课程[OpenGL ES专题]

一. 视觉班课程安排:

- 课程日期: 2020 年 8 月 10 日 周一 第 17 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:
  - 快速回顾上节课程分屏滤镜内容以及实现九分屏的代码;
  - 使用OpenGL ES 实现灰度滤镜/颠倒滤镜
  - 使用OpenGL ES 实现方形马赛克滤镜/六边形马赛克滤镜/三角形马赛克滤镜;
- 课程作业:

- 实现使用OpenGL ES 实现方形马赛克/六边形马赛克滤镜/三角形马赛克滤镜
- 博客: 将马赛克滤镜算法总结于博客中;

课程笔记

灰度滤镜的片元着色器代码实现[加注释版]

```
precision highp float;
uniform sampler2D Texture;
varying vec2 TextureCoordsVarying;
//变换因子
const highp vec3 W = vec3(0.2125, 0.7154, 0.0721);

void main (void) {
    //获取对于纹理坐标下的颜色值
    vec4 mask = texture2D(Texture, TextureCoordsVarying);
    //将颜色mask 与 变换因子相乘得到灰度值.
    float luminance = dot(mask.rgb, W);
    //将灰度值转化成(luminance,luminance,luminance,mask.a)并填充到像素中
    gl_FragColor = vec4(vec3(luminance), mask.a);
}
```

正方形马赛克滤镜的片元着色器代码实现[加注释版]

```
precision mediump float;
//纹理坐标
varying vec2 TextureCoordsVarying;
//纹理采样器
uniform sampler2D Texture;
//纹理图片size
const vec2 TexSize = vec2(400.0, 400.0);
//马赛克Size
const vec2 mosaicSize = vec2(8.0, 8.0);
void main()
{
    //计算实际图像位置
```

```
    vec2 intXY = vec2(TextureCoordsVarying.x*TexSize.x, TextureCoordsVarying.y*TexSize.y);
    // floor (x) 内建函数,返回小于/等于X的最大整数值.
    // floor (intXY.x / mosaicSize.x) * mosaicSize.x 计算出一个小马赛克的坐标.
    vec2 XYMosaic = vec2(floor(intXY.x/mosaicSize.x)*mosaicSize.x, floor(intXY.y/mosaicSize.y)*mosaicSize.y);
    //换算回纹理坐标
    vec2 UVMosaic = vec2(XYMosaic.x/TexSize.x, XYMosaic.y/TexSize.y);
    //获取到马赛克后的纹理坐标的颜色值
    vec4 color = texture2D(Texture, UVMosaic);
    //将马赛克颜色值赋值给gl_FragColor.
    gl_FragColor = color;
}
```

六方形马赛克滤镜的片元着色器代码实现[加注释版]

```
precision highp float;
uniform sampler2D Texture;
varying vec2 TextureCoordsVarying;
//六边形的边长
const float mosaicSize = 0.03;

void main (void)
{
    float length = mosaicSize;
    //sqrt(3)/2;
    float TR = 0.866025;

    //纹理坐标:(0,0)(0,1)(1,0)(1,1)
    float x = TextureCoordsVarying.x;
    float y = TextureCoordsVarying.y;

    //wx,wy -> 表示纹理坐标在所对应的矩阵坐标为
    int wx = int(x /( 1.5 * length));
    int wy = int(y /(TR * length));

```

```
    vec2 v1, v2, vn;

    //判断wx,wy在矩形中的上半部还是下半部
    if (wx/2 * 2 == wx) {
        if (wy/2 * 2 == wy) {
            v1 = vec2(length * 1.5 * float(wx), length * TR * float(wy));
            v2 = vec2(length * 1.5 * float(wx + 1), length * TR * float(wy + 1));
        } else {
            v1 = vec2(length * 1.5 * float(wx), length * TR * float(wy + 1));
            v2 = vec2(length * 1.5 * float(wx + 1), length * TR * float(wy));
        }
    }else {
        if (wy/2 * 2 == wy) {
            v1 = vec2(length * 1.5 * float(wx), length * TR * float(wy + 1));
            v2 = vec2(length * 1.5 * float(wx + 1), length * TR * float(wy));
        } else {
            v1 = vec2(length * 1.5 * float(wx), length * TR * float(wy));
            v2 = vec2(length * 1.5 * float(wx + 1), length * TR * float(wy + 1));
        }
    }

    // 计算参考点与当前纹素的距离
    float s1 = sqrt(pow(v1.x - x, 2.0) + pow(v1.y - y, 2.0));
    float s2 = sqrt(pow(v2.x - x, 2.0) + pow(v2.y - y, 2.0));

    // 选择距离小的则为六边形中心点.则获取它的颜色
    if (s1 < s2) {
        vn = v1;
    } else {
        vn = v2;
    }

```

```
    //获取六边形中心点的颜色值.
    vec4 color = texture2D(Texture, vn);

    //将颜色值填充到内建变量gl_FragColor 中
    gl_FragColor = color;

}
```

注意①:

首先六边形可以分裂成一个矩形块. 而这个矩形块的宽为:√3 长为: 3;

那么长宽比为: 3: √3;

矩形的宽为: length * √3/2;

矩形的长为: length * 3/2

所以TR = sqrt(3)/2 = 0.866025

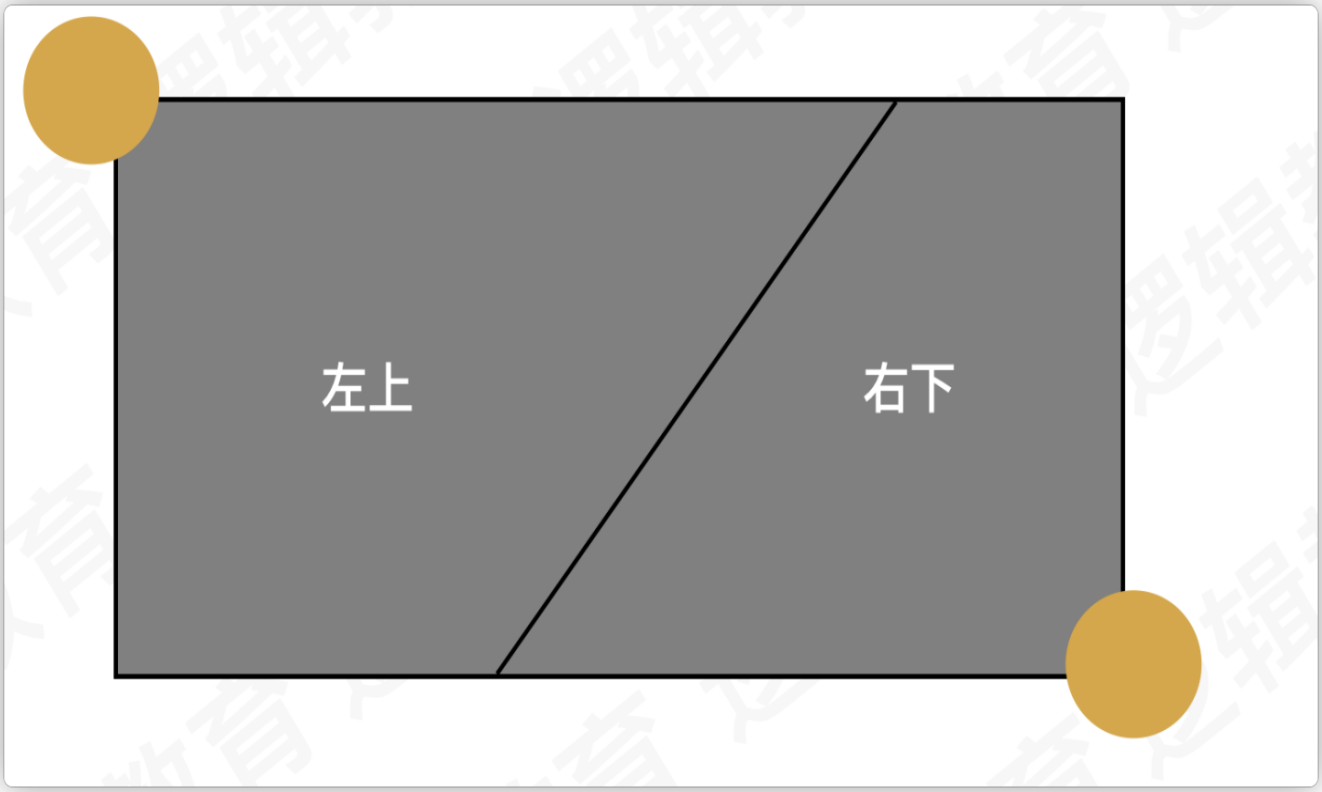
所以TB = 3/2 = 1.5;

注意②

第一种： 如上图表示当点在偶数行偶数列，或者奇数行奇数列的情形，这种情况下，只有左上右下有个点是六边形中心点。

左上点的纹理坐标为：v1 = vec2(length * 1.5 * float(wx), length * TR * float(wy));

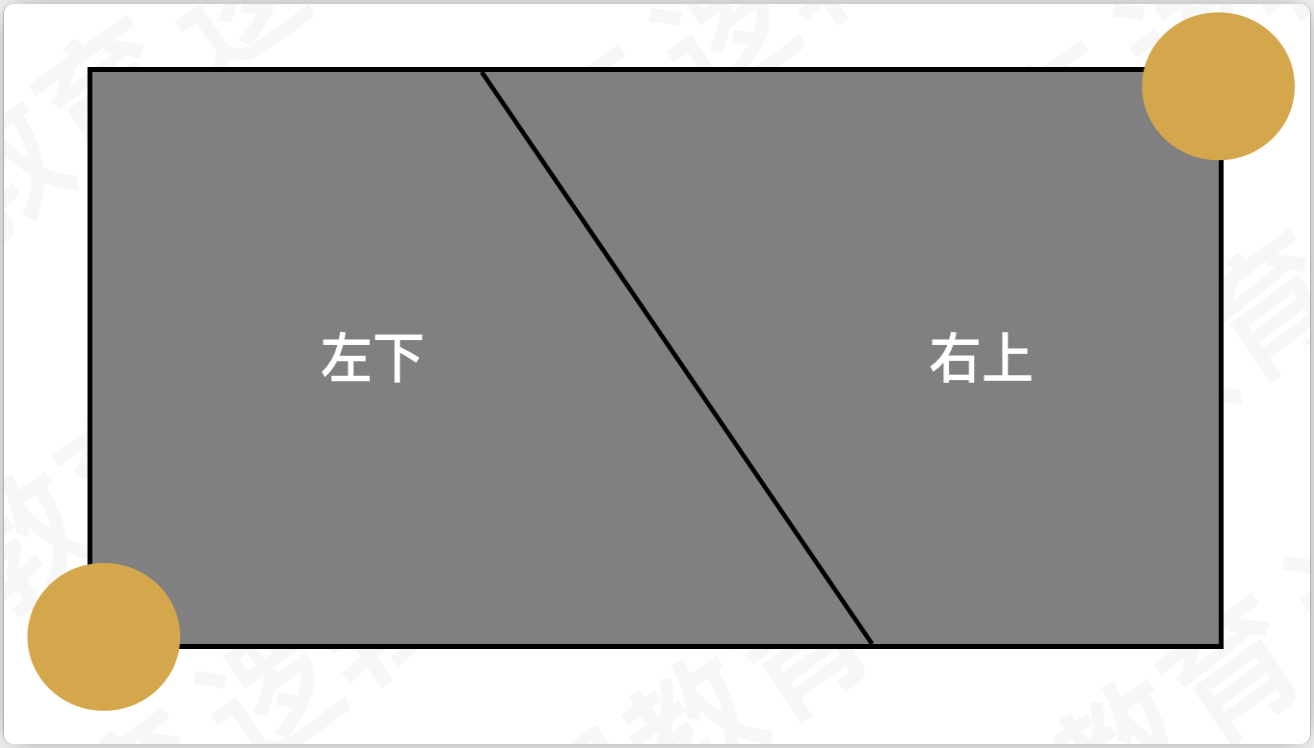
右下点的纹理坐标为:v2 = vec2(length * 1.5 * float(wx + 1), length * TR * float(wy + 1));



第二种：表示当点在奇数行偶数列，或者偶数行奇数列的情形，这种情况下，只有左下右上有个点是六边形中心点。

左下点的纹理坐标为：v1 = vec2(length * 1.5 * float(wx), length * TR * float(wy + 1));

右上点的纹理坐标为:v2 = v2 = vec2(length * 1.5 * float(wx + 1), length * TR * float(wy))。



三角形方形马赛克滤镜的片元着色器代码实现[加注释版]

```
precision highp float;
uniform sampler2D Texture;
varying vec2 TextureCoordsVarying;
//马赛克的边长
float mosaicSize = 0.03;

void main (void){
     //TR其实是√3/2
    const float TR = 0.866025;
    //π/6 = 30.0
    const float PI6 = 0.523599;

    //纹理坐标:(0,0)(0,1)(1,0)(1,1)
    float x = TextureCoordsVarying.x;
    float y = TextureCoordsVarying.y;

    //wx,wy -> 表示纹理坐标在所对应的矩阵坐标为
    int wx = int(x/(1.5 * mosaicSize));
```

```
    int wy = int(y/(TR * mosaicSize));

    vec2 v1, v2, vn;

    //判断wx,wy在矩形中的上半部还是下半部
    if (wx / 2 * 2 == wx) {
        if (wy/2 * 2 == wy) {
            v1 = vec2(mosaicSize * 1.5 * float(wx), mosaicSize * TR * float(wy));
            v2 = vec2(mosaicSize * 1.5 * float(wx + 1), mosaicSize * TR * float(wy + 1));
        } else {
            v1 = vec2(mosaicSize * 1.5 * float(wx), mosaicSize * TR * float(wy + 1));
            v2 = vec2(mosaicSize * 1.5 * float(wx + 1), mosaicSize * TR * float(wy));
        }
    } else {
        if (wy/2 * 2 == wy) {
            v1 = vec2(mosaicSize * 1.5 * float(wx), mosaicSize * TR * float(wy + 1));
            v2 = vec2(mosaicSize * 1.5 * float(wx+1), mosaicSize * TR * float(wy));
        } else {
            v1 = vec2(mosaicSize * 1.5 * float(wx), mosaicSize * TR * float(wy));
            v2 = vec2(mosaicSize * 1.5 * float(wx + 1), mosaicSize * TR * float(wy+1));
        }
    }

      // 计算参考点与当前纹素的距离
    float s1 = sqrt(pow(v1.x - x, 2.0) + pow(v1.y - y, 2.0));
    float s2 = sqrt(pow(v2.x - x, 2.0) + pow(v2.y - y, 2.0));

    // 选择距离小的则为六边形中心点. 此时可以了解点属于哪个六边形
    if (s1 < s2) {
        vn = v1;
    } else {
        vn = v2;
```

```
    }

    //纹理中该点的颜色值.
    vec4 mid = texture2D(Texture, vn);
    //获取a与纹理中心的角度.
    //atan算出的范围是-180至180度，对应的数值是-PI至PI
    float a = atan((x - vn.x)/(y - vn.y));

     //计算六个三角形的中心点
    vec2 area1 = vec2(vn.x, vn.y - mosaicSize * TR / 2.0);
    vec2 area2 = vec2(vn.x + mosaicSize / 2.0, vn.y - mosaicSize * TR / 2.0);
    vec2 area3 = vec2(vn.x + mosaicSize / 2.0, vn.y + mosaicSize * TR / 2.0);
    vec2 area4 = vec2(vn.x, vn.y + mosaicSize * TR / 2.0);
    vec2 area5 = vec2(vn.x - mosaicSize / 2.0, vn.y + mosaicSize * TR / 2.0);
    vec2 area6 = vec2(vn.x - mosaicSize / 2.0, vn.y - mosaicSize * TR / 2.0);

      //判断夹角a 属于哪个三角形.则获取哪个三角形的中心点坐标
   if (a >= PI6 && a < PI6 * 3.0) {
        //[30,90]
        vn = area1;
    } else if (a >= PI6 * 3.0 && a < PI6 * 5.0) {
        //[90,150]
        vn = area2;
    } else if ((a >= PI6 * 5.0 && a <= PI6 * 6.0)|| (a<-PI6*5 && a>-PI6*6)) {
       //[150,180],[-150,-180]
        vn = area3;
    } else if (a < -PI6 * 3.0 && a >= -PI6 * 5.0) {
        //[-90,-150]
        vn = area4;
    } else if(a <= -PI6 && a> -PI6 * 3.0) {
        //[-30,-90]
        vn = area5;
    } else if (a > -PI6 && a < PI6) {
        //[-30,30]
        vn = area6;
```

```
86      }
87
88      //获取对应三角形中心的颜色值
89      vec4 color = texture2D(Texture, vn);
90
91      //将颜色值填充到片元着色器内置变量gl_FragColor
92      gl_FragColor = color;
93  }
```