完整资源：

我的Github地址

前情提要：

从0开始编写minecraft光影包（0）GLSL，坐标系，光影包结构介绍

从零开始编写minecraft光影包（1）基础阴影绘制

从零开始编写minecraft光影包（2）阴影优化

@TOC

什么是泛光?

泛光是因为真实的相机拍摄时，尤其是从漆黑的房间拍摄窗外时，光从亮处溢出到暗处的一种现象：

~~如图，一个人（指自己）的宿舍随手抓拍~~

泛光是shader中的常见特效，~~其实现并不复杂~~，泛光的核心就是 ~~先富带动后富~~ “亮的像素” 照亮周围暗的像素。

首先要解决的是如何获取亮色，我们可以通过图像处理的一些公式来计算一个RGB值对应的亮度，选取亮度大于一定阈值的点。

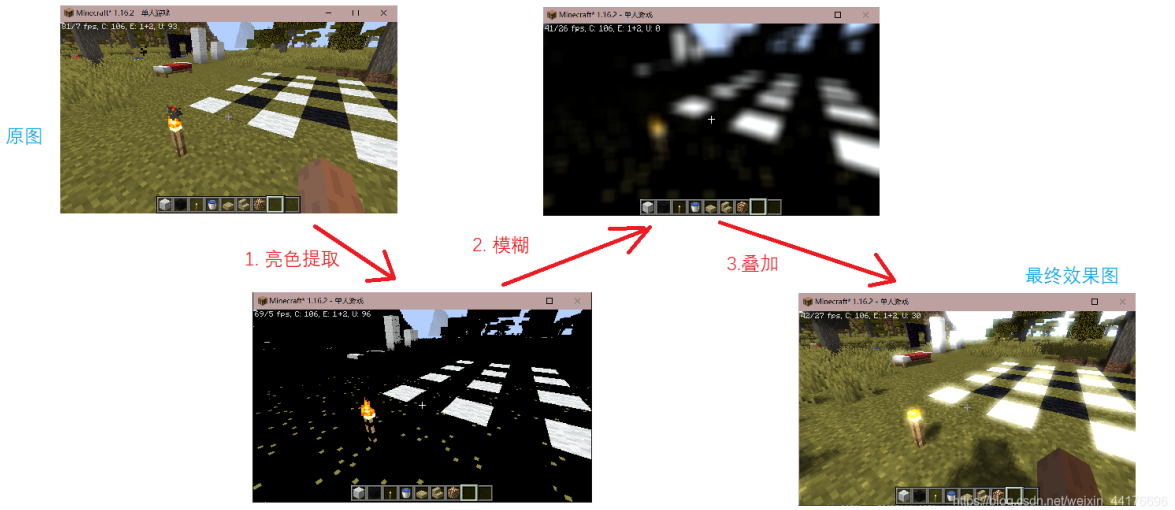
此外，如何实现 “亮的像素” 照亮周围暗的像素呢？我们通过**对高亮像素进行高斯模糊**来实现。

> 还记得上一篇博客提到的高斯模糊（均值滤波）吗？计算x像素邻近像素的RGB平均值作为x的RGB值。

泛光的实现分为三个步骤：

1. 亮色提取
2. 模糊
3. 叠加

下面的图演示了这三个步骤。首先从原图中提取亮色部分，然后对亮色部分进行高斯模糊（中值滤波），最后将处理的结果和原图叠加。



下面来看看如何实现三个步骤吧。。

亮色提取

判断一个像素亮度的标志，可以由公式得出：

```
B = (0.299*R) + (0.587*G) + (0.114*B)
```

即RGB通道的值简单相加即可。

我们选择亮度超过一定阈值的像素，比如 0.5，所以我们编写一个函数即可，在 composite.fsh中加入：

```
/*
 *  @function getBloomOriginColor : 亮色筛选
 *  @param color                  : 原始像素颜色
 *  @return                       : 筛选后的颜色
 */
vec4 getBloomOriginColor(vec4 color) {
    float brightness = 0.299*color.r + 0.587*color.g + 0.114*color.b;
    if(brightness < 0.5) {
        color.rgb = vec3(0);
    }
    return color;
}
```

高斯模糊与叠加

还记得上一篇博客提到的高斯模糊（均值滤波）吗?

这个函数会判断原像素是不是超过 0.5 是则输出，否则返回0。然后我们编写 getBloom() 函数，该函数主要做三件事:

1. 从texture纹理中取原始颜色数据
2. 根据阈值判断该像素是否为亮色并且获取提取结果
3. 对提取的结果进行高斯模糊

我们对亮色提取后的图像做一次半径为15个采样的高斯模糊，然后返回模糊后的图像。

```
/*
 *  @function getBloom : 亮色筛选
 *  @return            : 泛光颜色
 */
vec3 getBloom() {
    int radius = 15;    // 半径
    vec3 sum = vec3(0);

    for(int i=-radius; i<=radius; i++) {
        for(int j=-radius; j<=radius; j++) {
            vec2 offset = vec2(i/viewWidth, j/viewHeight);
            sum += getBloomOriginColor(texture2D(texture,
texcoord.st+offset)).rgb;
        }
    }

    sum /= pow(radius+1, 2);
    return sum*0.3;
}
```

> 注：这里并未拆分横向和纵向模糊，只是简单实现，之后会细🔒拆分版写法（提升性能）

此外，注意到两个新的变量，分别是 viewWidth 和 viewHeight。

这是两个 shadermod 提供的可用变量，他们表示了**屏幕的高度和宽度**。因为 texcoord（屏幕坐标）的范围是 [0, 1]，要将像素点采样的偏移量映射到 [0, 1] 区间才能对 texture 纹理正确的采样。

在 composite.fsh 的开头添加

```
uniform float viewWidth;
uniform float viewHeight;
```

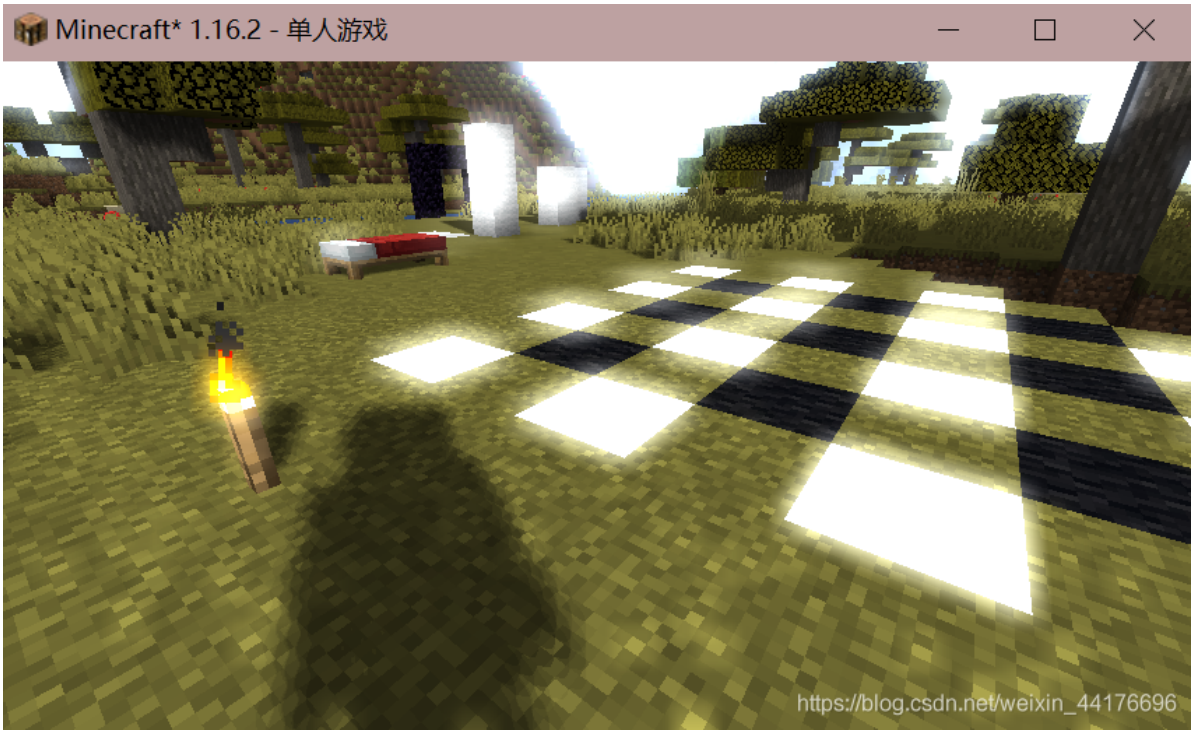
最后我们将模糊后的图像叠加到原图上即可。在 composite.fsh 计算阴影的代码之前加上：

```
color.rgb += getBloom();
```

~~大概这个位置：~~

```
/* DRAWBUFFERS: 0 */
void main() {
    //vec4 color = texture2D(shadow, texcoord.st);
    vec4 color = texture2D(texture, texcoord.st);

    float depth = texture2D(depthtex0, texcoord.st).x;

    // 利用深度缓冲建立带深度的ndc坐标
    vec4 positionInNdcCoord = vec4(texcoord.st*2-1, depth*2-1, 1);

    // 逆投影变换 -- ndc坐标转到裁剪坐标
    vec4 positionInClipCoord = gbufferProjectionInverse * positionInNdcCoord;

    // 透视除法 -- 裁剪坐标转到眼坐标
    vec4 positionInViewCoord = vec4(positionInClipCoord.xyz/positionInClipCoord.w, 1.0);

    // 逆 “视图模型” 变换 -- 眼坐标转 “我的世界坐标”
    vec4 positionInWorldCoord = gbufferModelViewInverse * positionInViewCoord;

    // 计算泛光
    color.rgb += getBloom();

    // 绘制阴影
    color = getShadow(color, positionInWorldCoord);

    gl_FragData[0] = color;
}
```

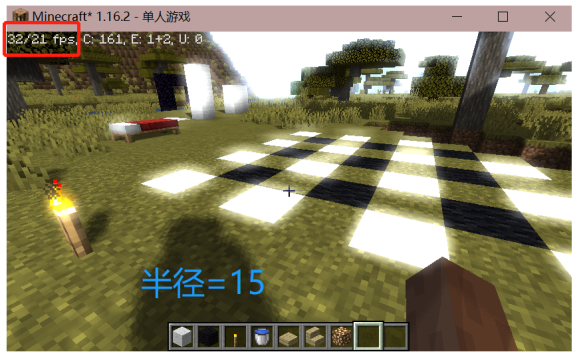

重新加载光影包，就能看到泛光的效果了：

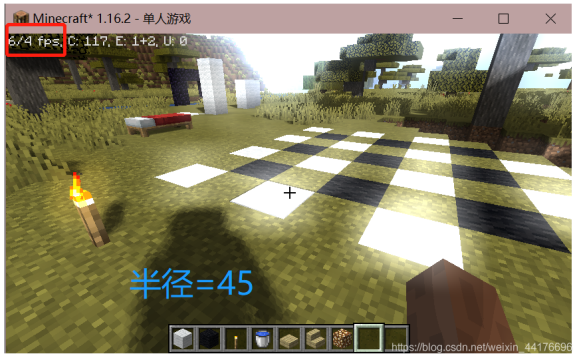


模糊半径与性能

如果想加大图像的模糊程度，就需要使用更大的半径，即使用更多的采样来计算均值。我们将模糊半径增大，可以取得很好的模糊效果：

更大的半径意味着更多的循环次数，更好的模糊效果，和更大的性能消耗





可以看到，45的半径，我电脑的帧数就降低到个位数了，因为横纵一起执行的高斯模糊是 `O(n^2)` 的复杂度。至于 `O(n^2)` 的代码效率有多么拉跨，~~相信懂的都懂~~

可以看到简单的加大模糊半径确实可以提升模糊的品质，可代价是昂贵的计算力。下一次博客会介绍几种优化的方案。 ~~咕子~~

代码

Github地址

和上一篇博客的 composite.fsh 相比，增加如下的新代码，其他并未做改变：

变量与函数声明部分：

```
uniform float viewWidth;
uniform float viewHeight;

/*
 *  @function getBloomOriginColor : 亮色筛选
 *  @param color                  : 原始像素颜色
 *  @return                       : 筛选后的颜色
 */
vec4 getBloomOriginColor(vec4 color) {
    float brightness = 0.299*color.r + 0.587*color.g + 0.114*color.b;
    if(brightness < 0.5) {
        color.rgb = vec3(0);
    }
    return color;
}

/*
 *  @function getBloom : 亮色筛选
 *  @return            : 泛光颜色
 */
vec3 getBloom() {
    int radius = 15;     // 半径
    vec3 sum = vec3(0);

    for(int i=-radius; i<=radius; i++) {
```

```
        for(int j=-radius; j<=radius; j++) {
            vec2 offset = vec2(i/viewWidth, j/viewHeight);
            sum += getBloomOriginColor(texture2D(texture,
texcoord.st+offset)).rgb;
        }
    }

    sum /= pow(radius+1, 2);
    return sum*0.3;
}
```

main函数部分：

```
// 计算泛光
color.rgb += getBloom();
```