二维SDF面部阴影方案

原文为：https://zhuanlan.zhihu.com/p/670837192

效果

https://www.bilibili.com/video/BV1Ac411D7gj/?t=2&spm_id_from=333.1007.seo_video.first&vd_source

一、前言

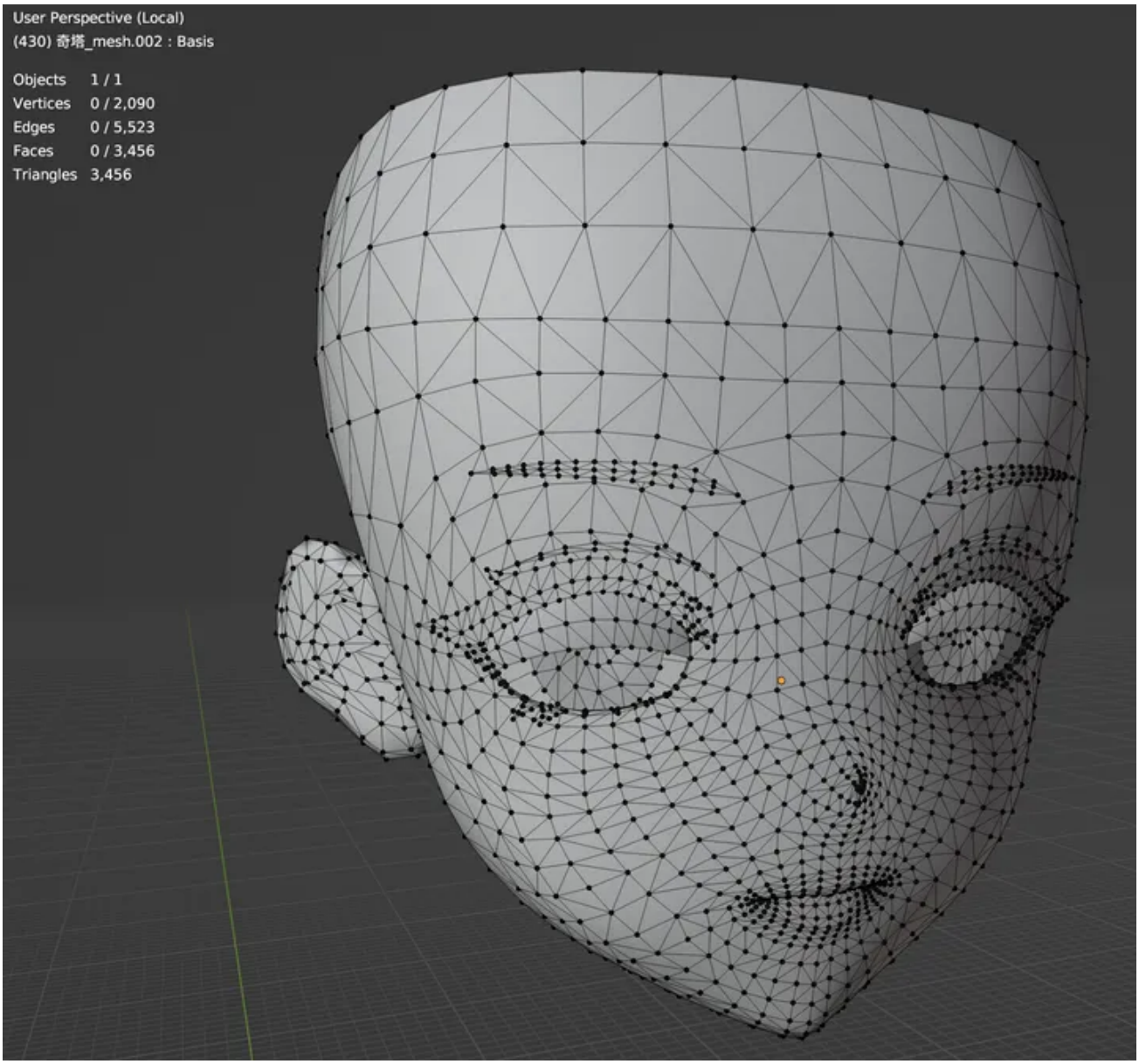
目前实时渲染领域卡通渲染脸部光照主要有两种实现方法：

1. 修改法线
2. 使用SDF贴图

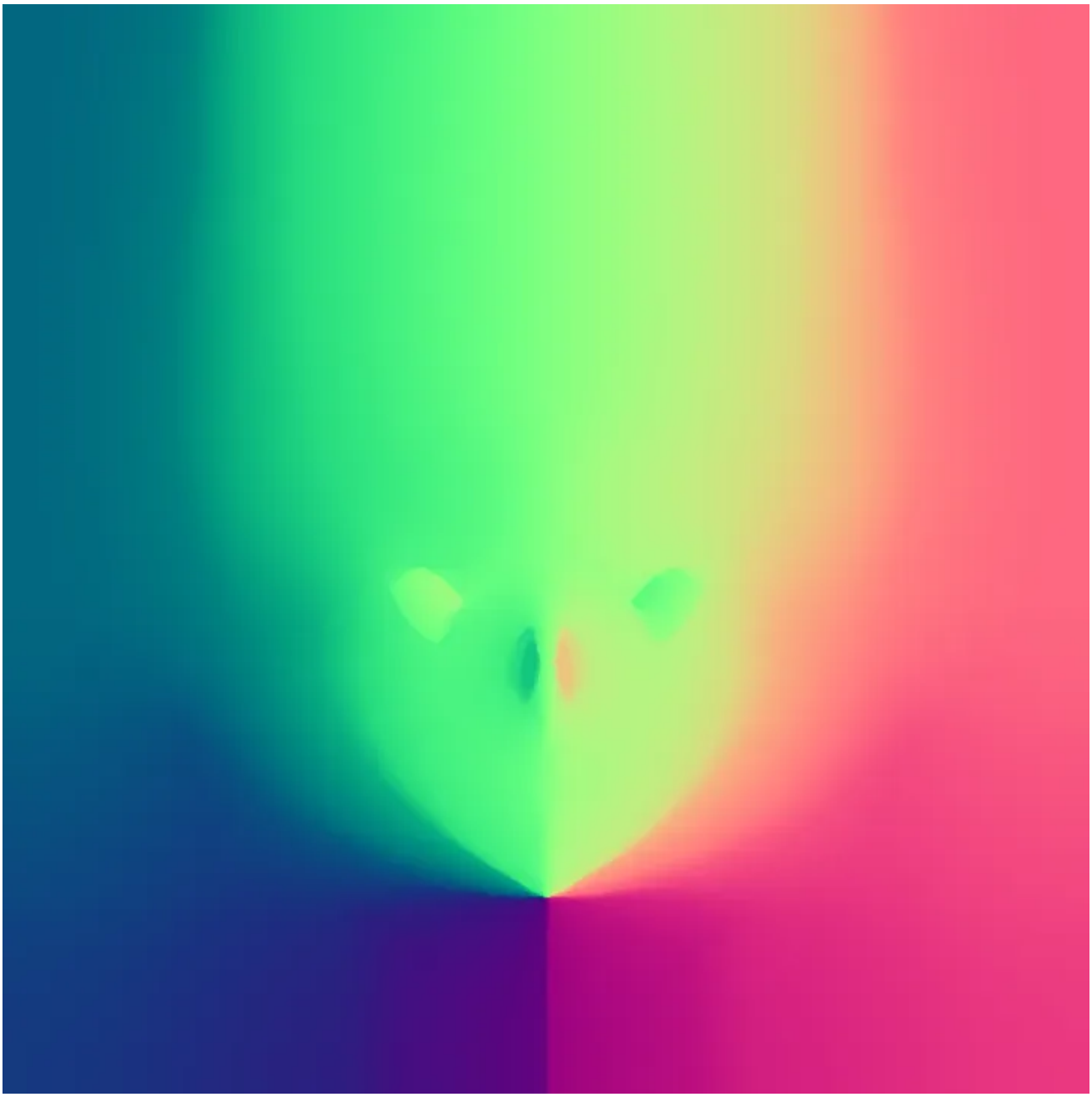
关于脸部卡通渲染方案的介绍可以看参考下面这篇文章：
https://zhuanlan.zhihu.com/p/411188212

下面说一下我对这两种方案的理解。

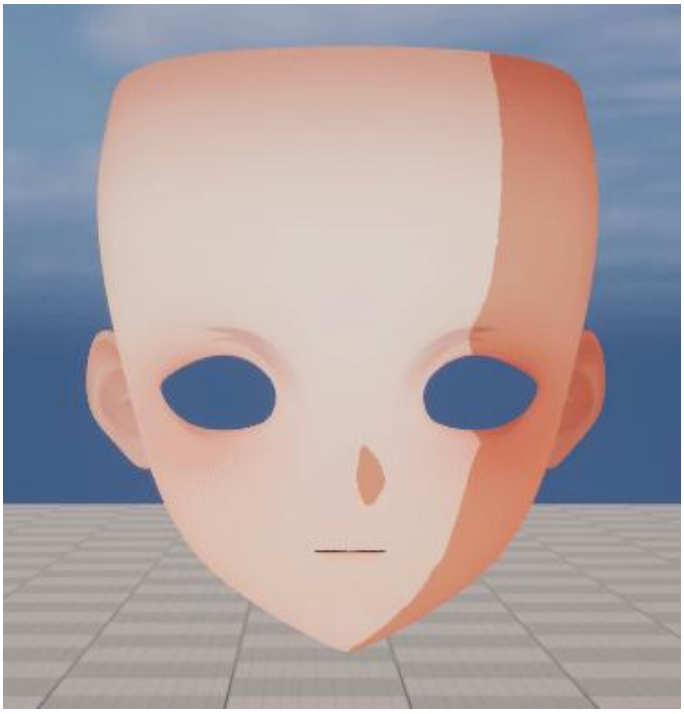
1.1修改法线的方案



首先，如果只是修改模型顶点的法线的话，模型顶点数就那么多，比如上面的模型已经做得很细致了，但顶点数也就5k，传递球形法线做比较柔和的光照效果还比较合适，想表现复杂的结构是比较困难的。
当然也有修改模型拓扑等方法，但我觉得这些方法都不是特别优雅。
我个人觉得使用法线贴图会更有潜力一些，自己私底下也有在研究把SDF烘培成法线贴图的方法，目前的结果我自己并不是很满意，如果后续有一些有意思的结果，我会再出一篇文章跟大家分享。



通过SDF生成的法线贴图

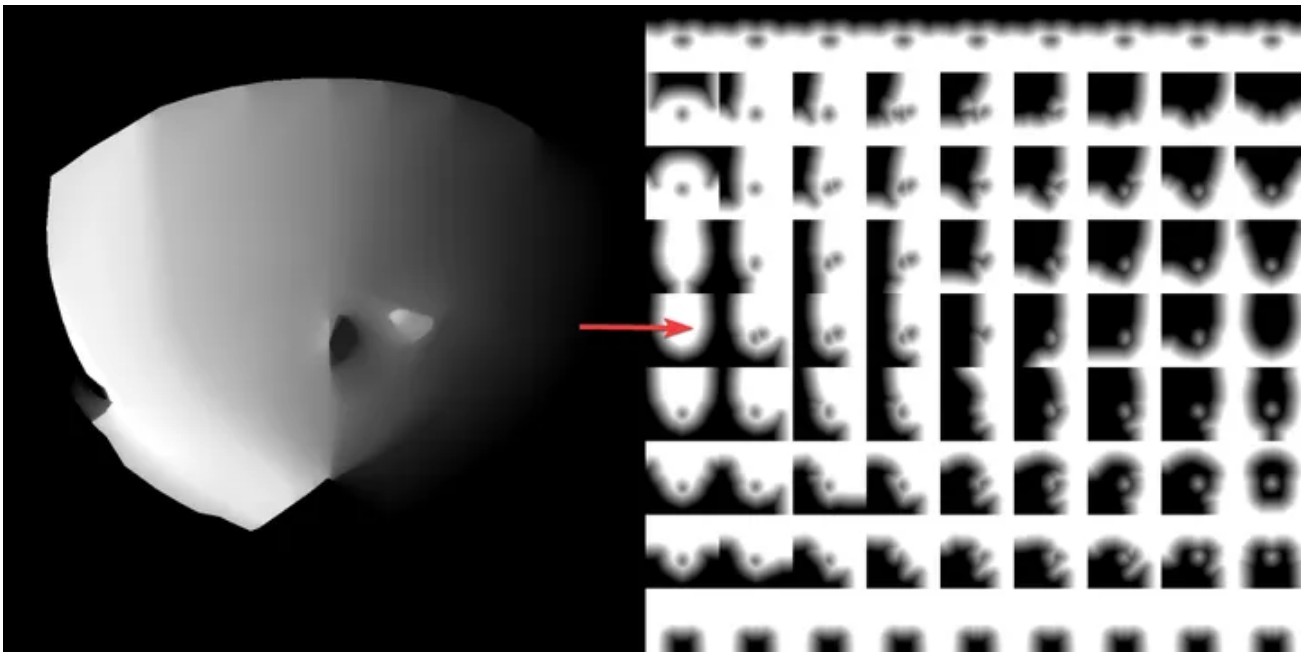


使用SDF法线贴图shading的效果

> 为了方便烘培成世界空间的法线，烘培到切线空间会更合适

1.2 使用SDF的方案

SDF的方案是目前市面上使用得最多的了，新出的二游基本都会用这种方式。
在实现上，SDF的方案并不困难，但是由于我自己使用unreal引擎比较多，自己想通过改管线的方法来实现的话很难改得很优雅，所以我以后还会继续研究法线贴图方案。
在表现上，目前SDF的方案最明显的问题是没有Z轴上的变化，这也是这篇文章主要想解决的问题。本文的实现思路其实很简单，目前的SDF都只画了一组水平轴的光照，那么我们把其他所有轴都画出来就行了。

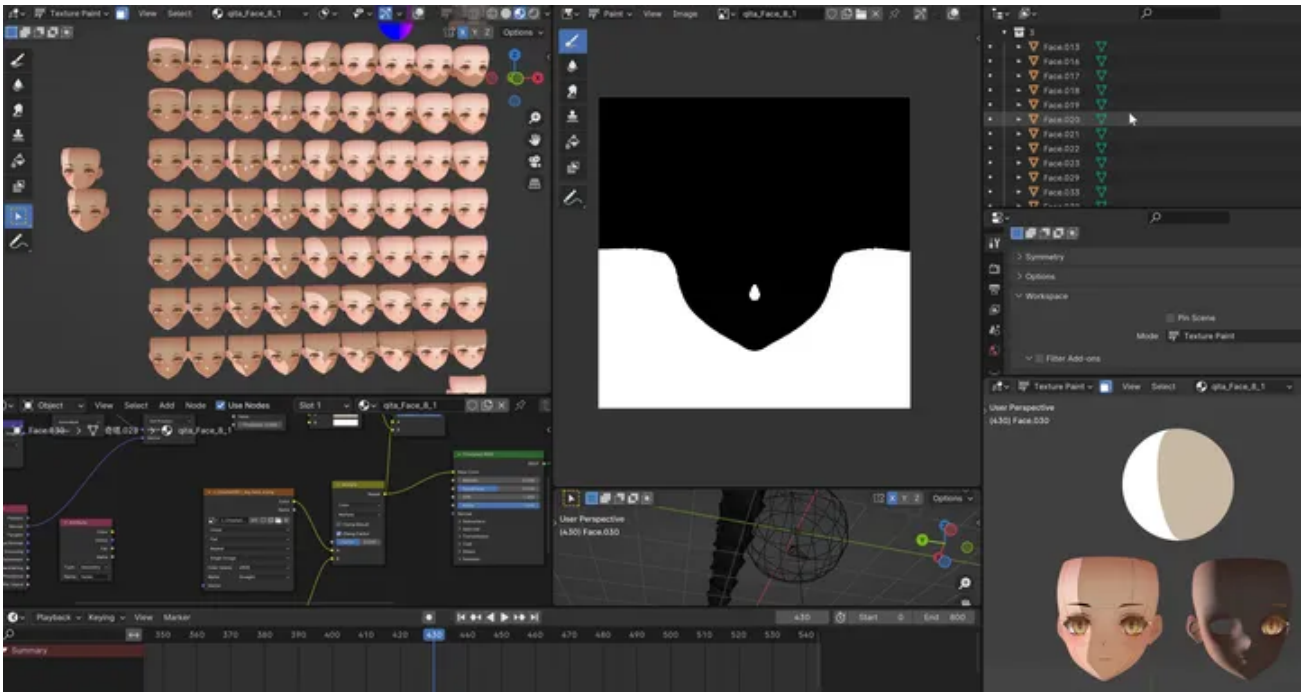


使用全角度的SDF图集代替水平轴的SDF
两三年前看到原神的SDF方案时就想到了这个方法，相信平时比较关注卡通渲染的各位都想到过这个方法，但是几年过去了，没有看到任何一篇文章实现了这个方案，看来大伙也都觉得把所有方向的光照都画出来工作量太大了不敢尝试，所以只能我自己尝试实现给大伙看了。

二、实现流程

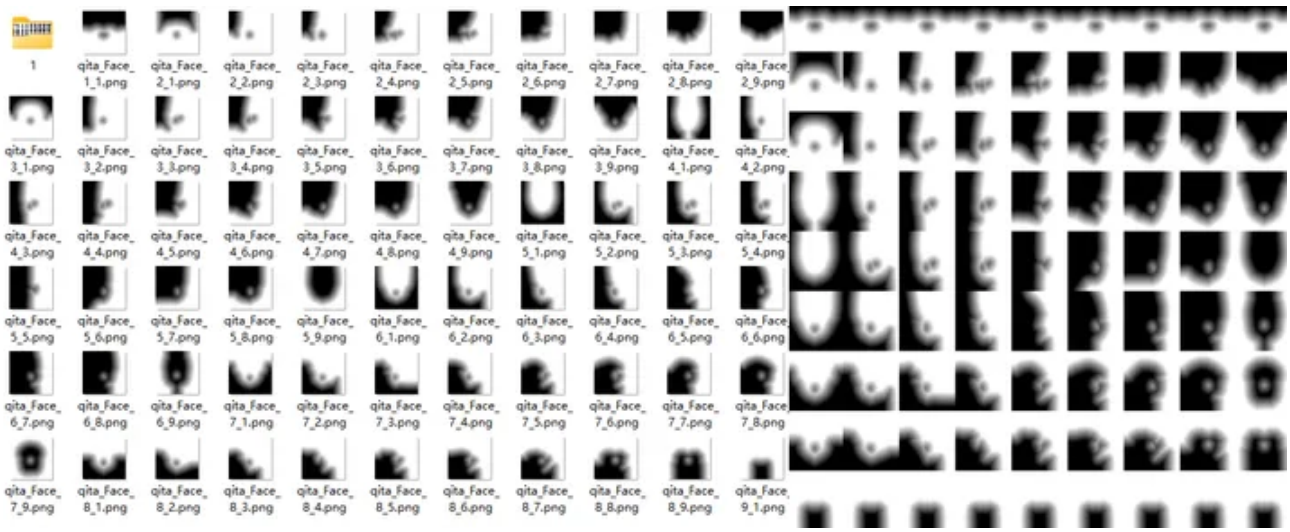
2.1 流程概况

三句话简单概括一下：
1.画好各个角度的光照，如下我一共画了65张，稍微偷懒点少画写应该也没问题。



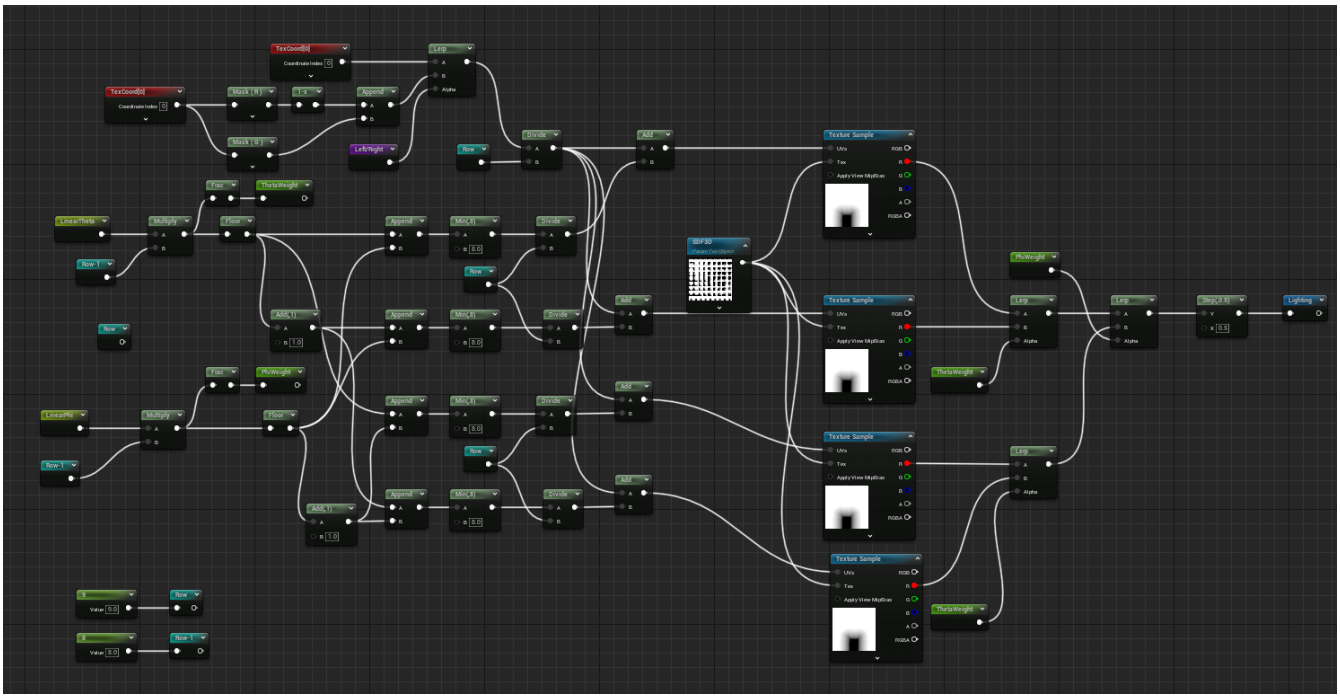
在blender中绘制光照图

2.把画完的光照图转成SDF，然后拼成一张图集



生成的SDF和SDF图集

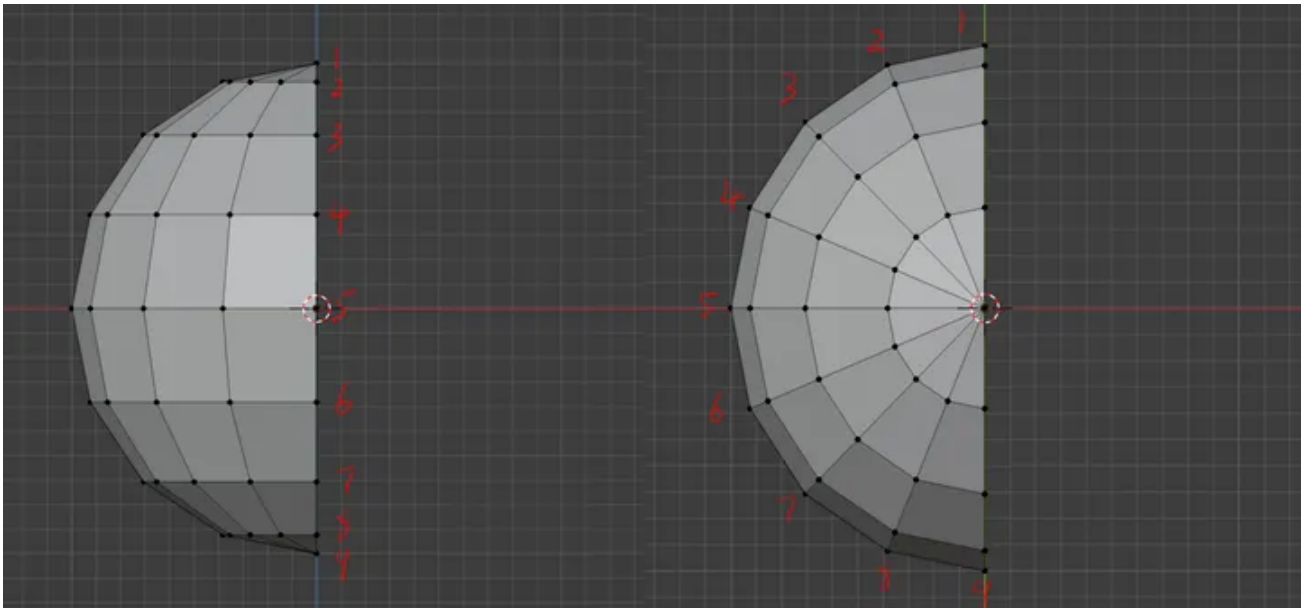
3.通过光源角度和计算合适的uv，采样4次进行插值。



采样四次SDF图集计算光照结果

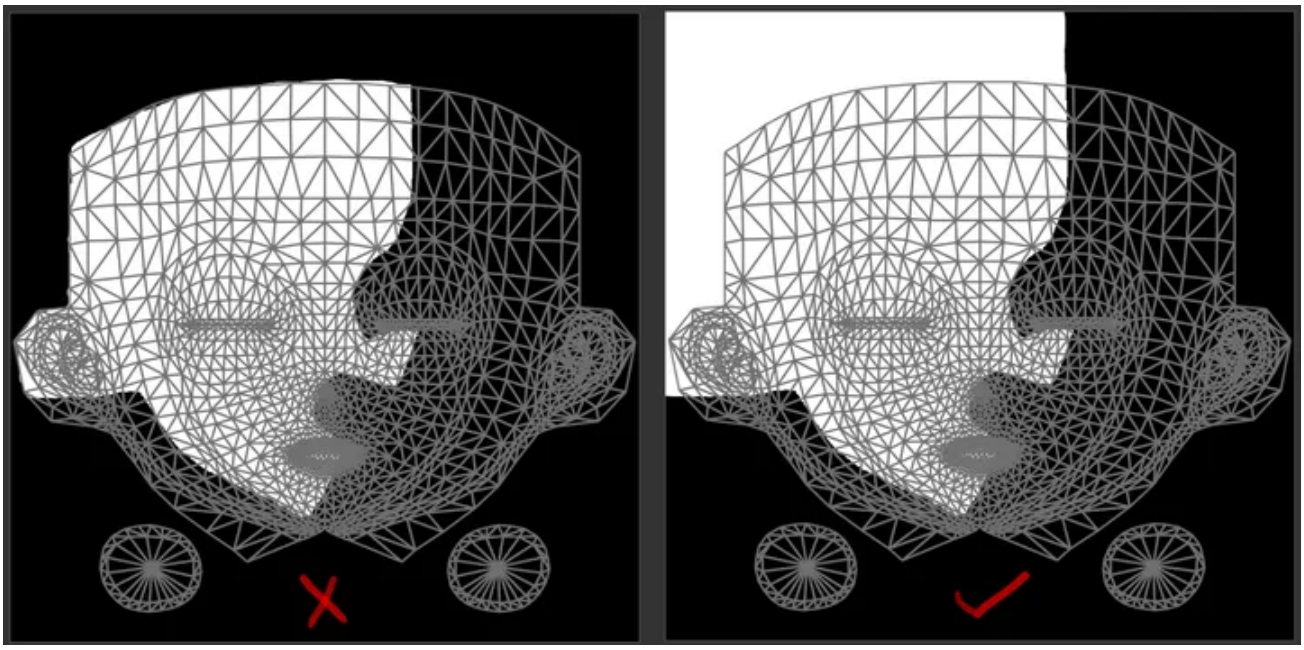
2.2 光照图绘制

绘制光照没有目前没想到什么很好的生成方式，纯靠手绘，我没啥美术能力，画这65张图，花了我一整个星期的空闲时间，如果有美术大佬愿意画的这东西的话可能效果会更好一些。（这一堆的模型，有点制作表情blend shape的感觉了）
光照图我画了9行9列（第一行和最后一行只有一张，所有一共9x7+2=65张）
光照图的每一行代表平行光从下面的半球从上到下的9种不同的经度投射到脸上，每一列代表9不同的纬度，第一行和最后一行平行光从正上和正下投射过来，所以只有一张。

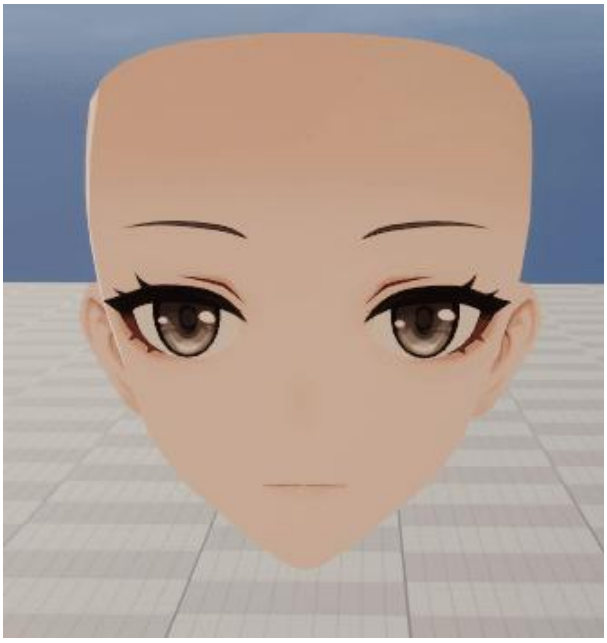


绘制的光照图相当于光照从左半球上各点照射过来的结果

有一点需要注意的，可以看到我绘制光照图的时候涂出了uv的边界，这样生成的SDF在uv边界效果不会出问题（不想让SDF超出uv边界的话等生成了SDF之后再把超出边界的部分去掉就行了）：

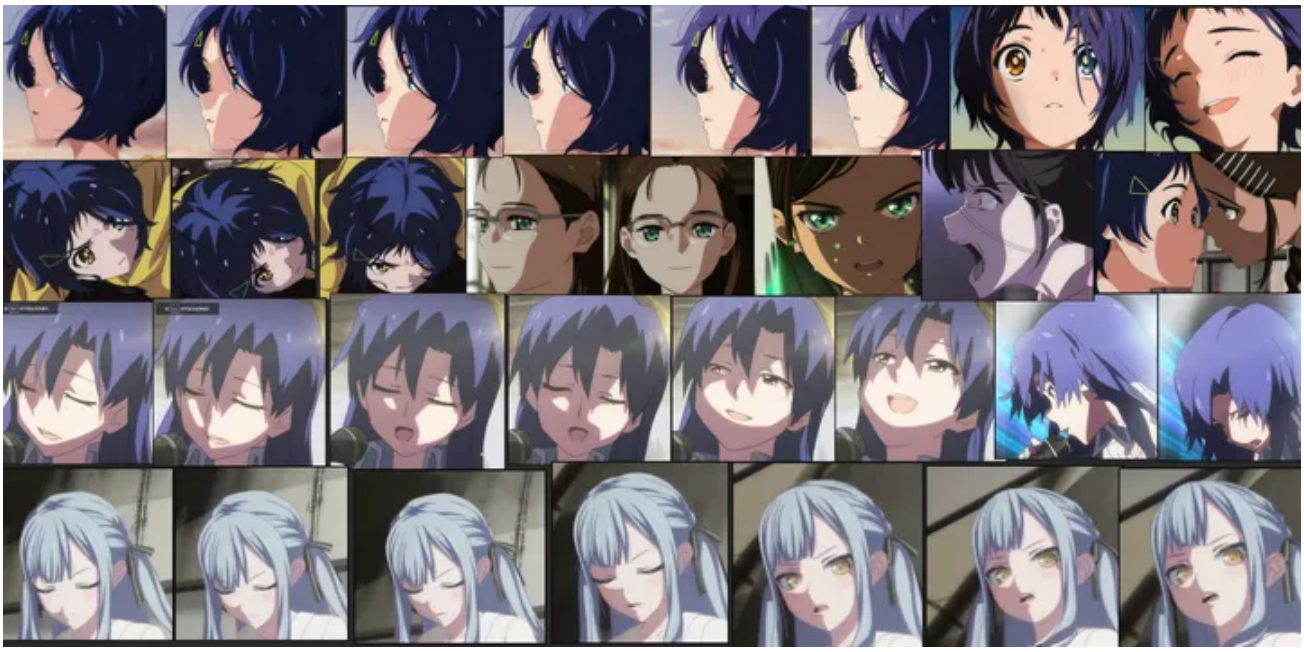


绘制光照图时建议涂出uv边界
下面这个是反面例子：



脸部上方uv边界处SDF插值效果不好
下面放点动画里的光影变化作为参考：

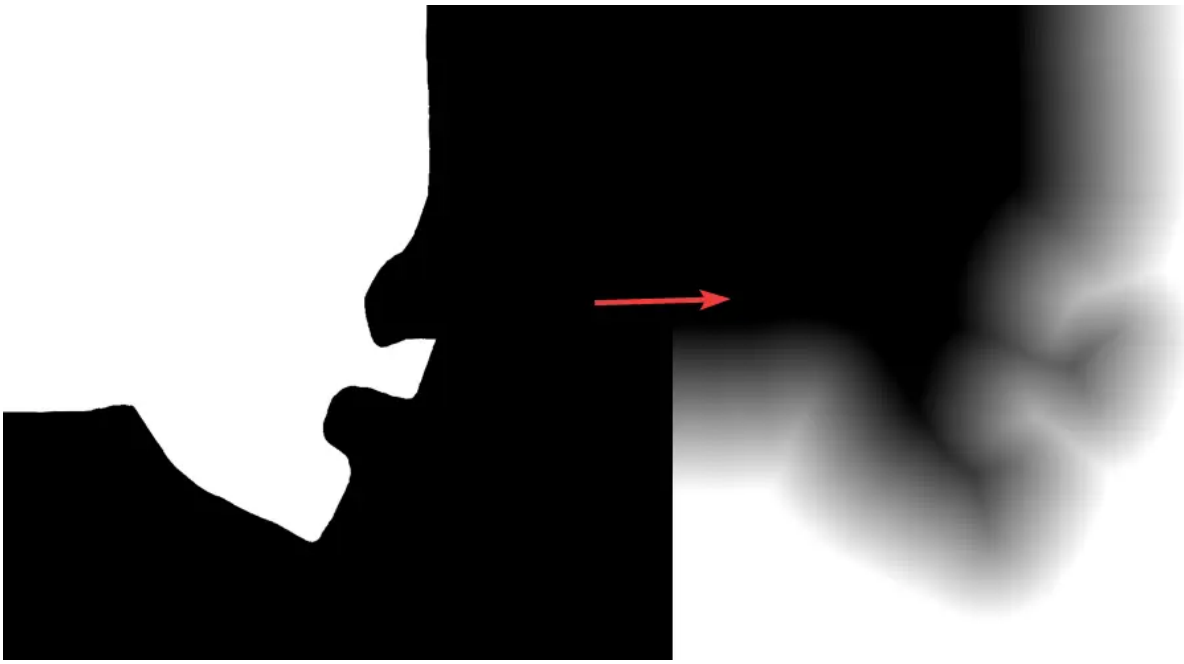
奇蛋物语



奇蛋物语、偶像大师、Mygo

轻音少女

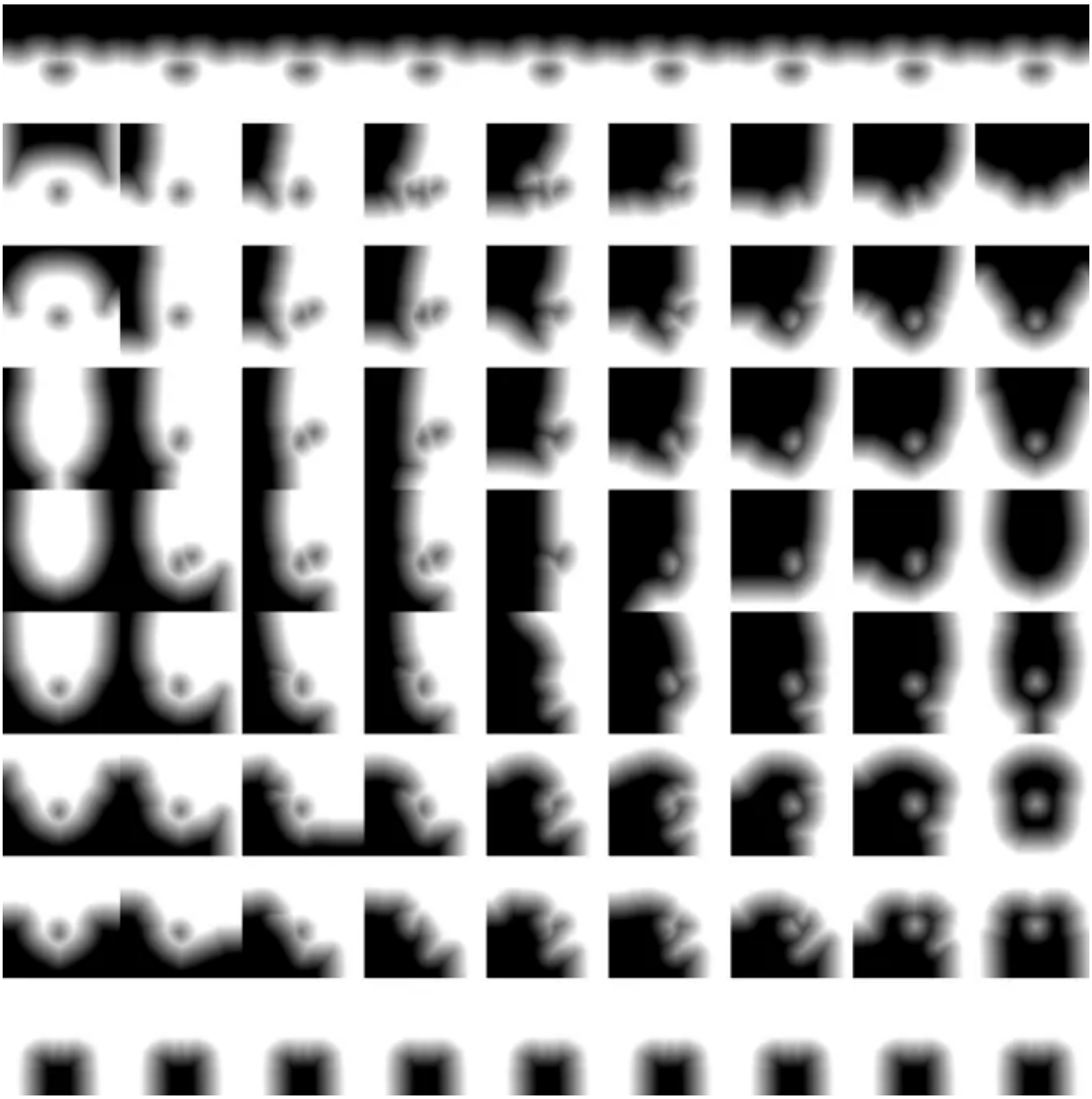
2.3 SDF图集制作



使用8ssedt算法生成SDF

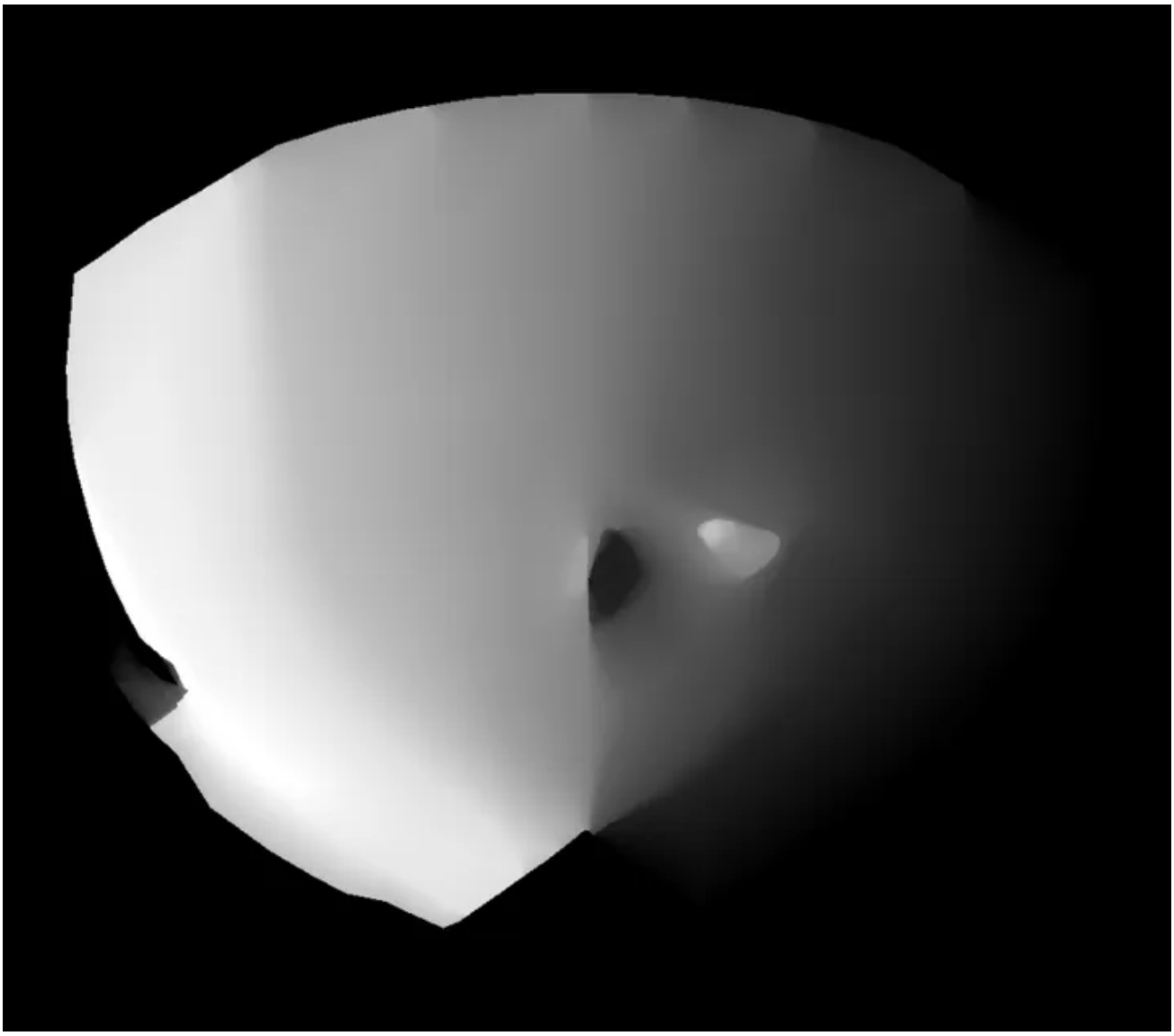
首先，我们需要把黑白的光照图转为SDF，我使用的是8ssedt算法生成的，关于8ssedt算法的详细可以看下面的这两篇文章：
https://zhuanlan.zhihu.com/p/337944099
https://blog.csdn.net/qq_41835314/article/details/128548073

接下来把所有的光照图拼成一张图集：

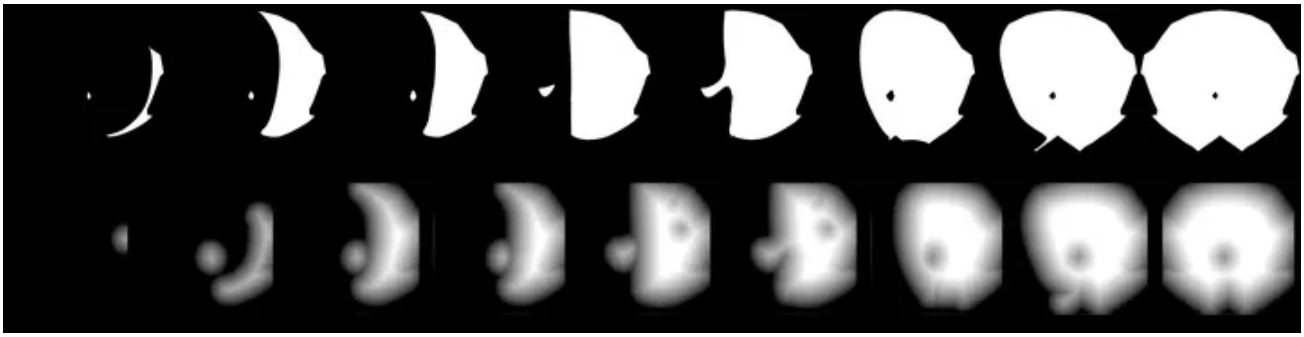


SDF图集

这里解释一下，我为什么不生成这种SDF（这种SDF在下文都使用“插值后的SDF”特指）。

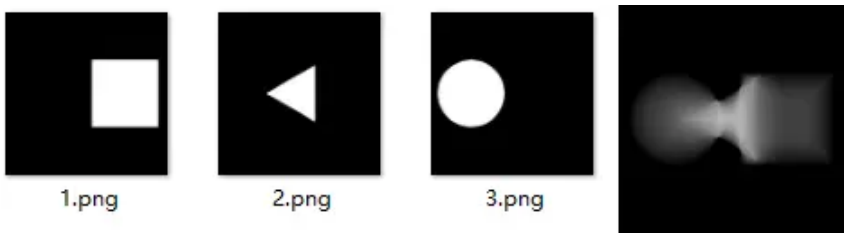


“插值后的SDF”
使用这种插值后的SDF其实是有一个条件的：后面的光照一定要覆盖前面的光照。
比如下面这套SDF图中，光照图的范围从左到右，一定是逐渐增大的

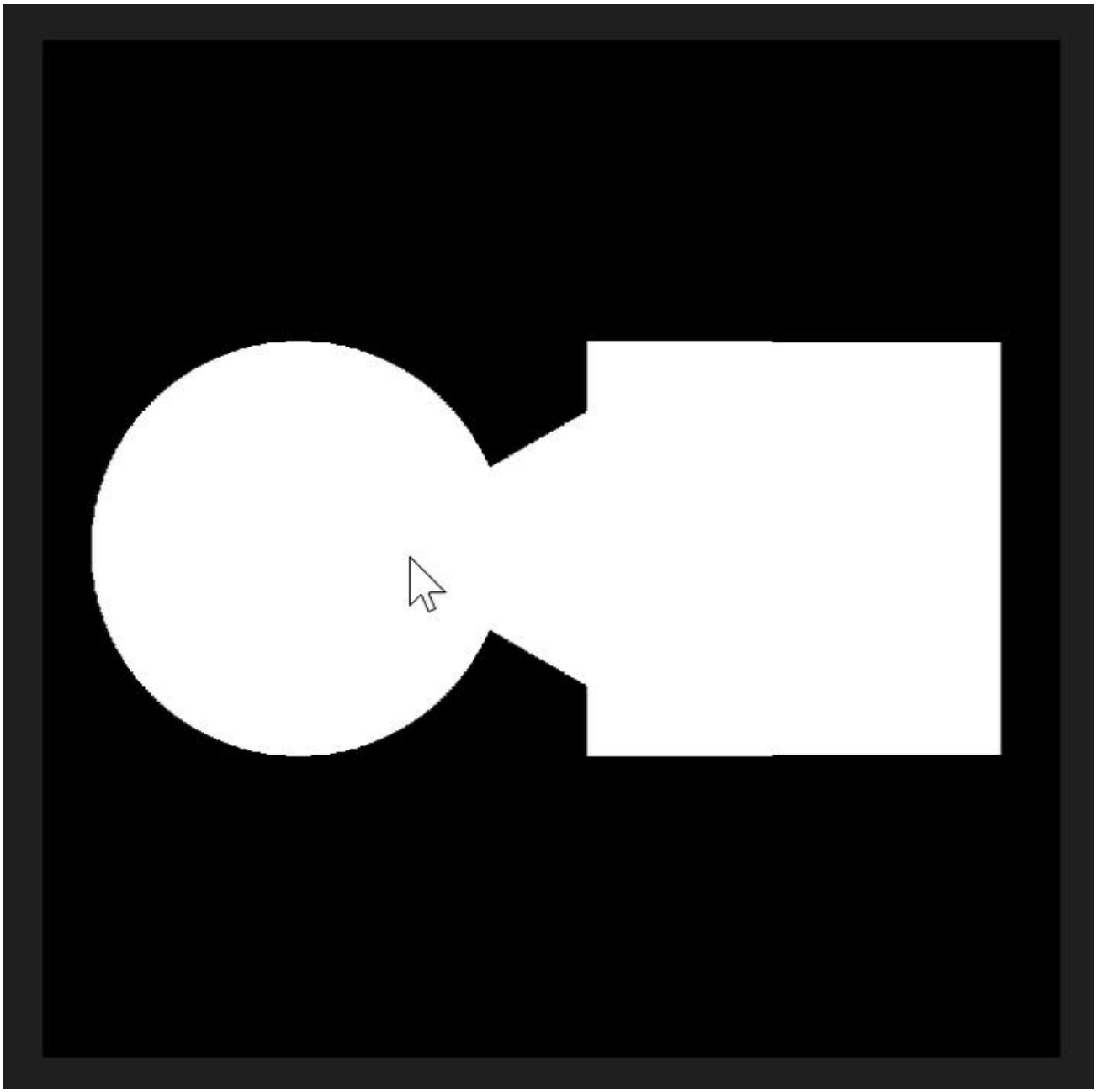


https://zhuanlan.zhihu.com/p/411188212
这样导致的结果就是当光从前方照过来的时候，如下图左边整个脸必须是完全亮的，不可能出现右边这种脸的后方没被照亮的效果。

“使用插值后的SDF的缺点”
再举一个例子，下面3张图之间没有包含关系，所有转成SDF再插值之后的效果是右边这样，是不是看起来就不对劲。



使用没有包含关系的图像生成“插值后的SDF”
看看效果，确实不对劲：



“插值后的SDF”错误的效果
导致这种错误的原因在于生成“插值后的SDF”的算法：

- 算法在255次循环中计算出了不同光源角度对应的光照结果，这一步是没问题的
- 但是为了能够将255种不同的结果保存在一张贴图里，算法里的做法是将所有结果累加起来，这一步破坏了插值信息

```C++
for (int y = 0; y < height; y++)
{
    for (int x = 0; x < width; x++) {
        for (int i = 0; i < 255; i++) {
            if (nextTexIndex >= int(grayImages.size())) {
                break;
            }
            float weight = lerpStep / levelStep;
            // 这里采样了相邻的两张SDF
            int curPixel = grayImages[curTexIndex].at<uchar>(y, x);
            int nextPixel = grayImages[nextTexIndex].at<uchar>(y, x);
```

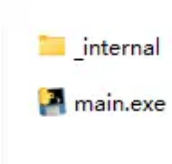
```
            int lerpPixel = curPixel * weight + nextPixel * (1 - weight);
            // 在这里计算出了不同光源角度对应光照结果
            result += lerpPixel > 127 ? 0 : 1;
            // 结果进行累加
            lerpStep++;
            if (lerpStep >= levelStep)
            {
                lerpStep = 0;
                curTexIndex++;
                nextTexIndex++;

            }
        }

        lerpedSDF.at<Vec3b>(y, x)[0] = int(result);
        lerpedSDF.at<Vec3b>(y, x)[1] = int(result);
        lerpedSDF.at<Vec3b>(y, x)[2] = int(result);

        result = 0;
        curTexIndex = 0;
        nextTexIndex = 1;
        lerpStep = 0;
    }
}
```

如果只做水平轴的SDF，其实“后面的光照范围必须大于前面的光照”的这个条件其实无伤大雅，而且用一张贴图代替了9张SDF，性价比很高
但是我们做全角度的SDF光照，本身就需要做图集，直接用SDF就好了，而且不会受上面条件的影响，效果会更好。

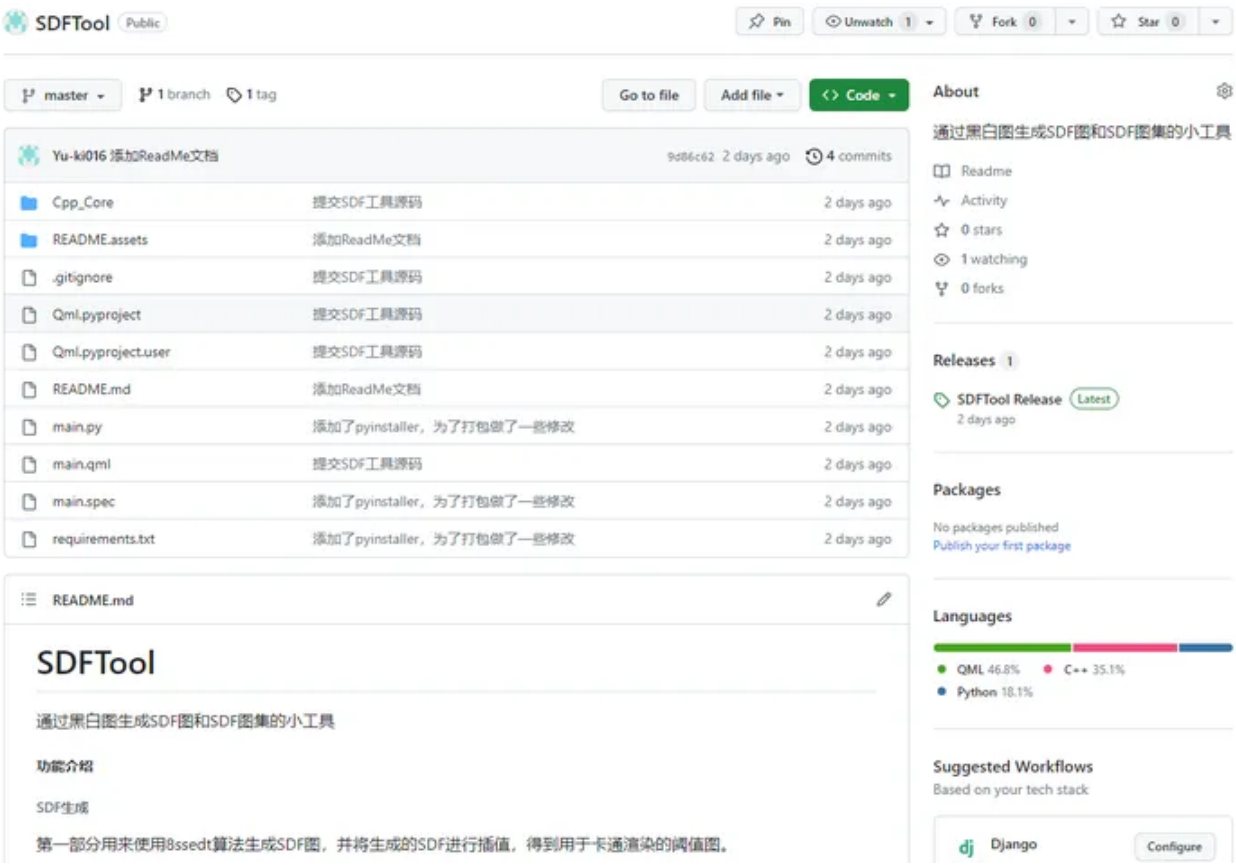
2.4 工具

以上转SDF和拼图集的操作我写了一个小工具，你们直接用我的工具来做就行了，不用再浪费时间去重复造轮子了。

生成SDF和图集的小工具
工具我和工程放在一起，解压SDFTool.zip然后双击main.exe就能打开

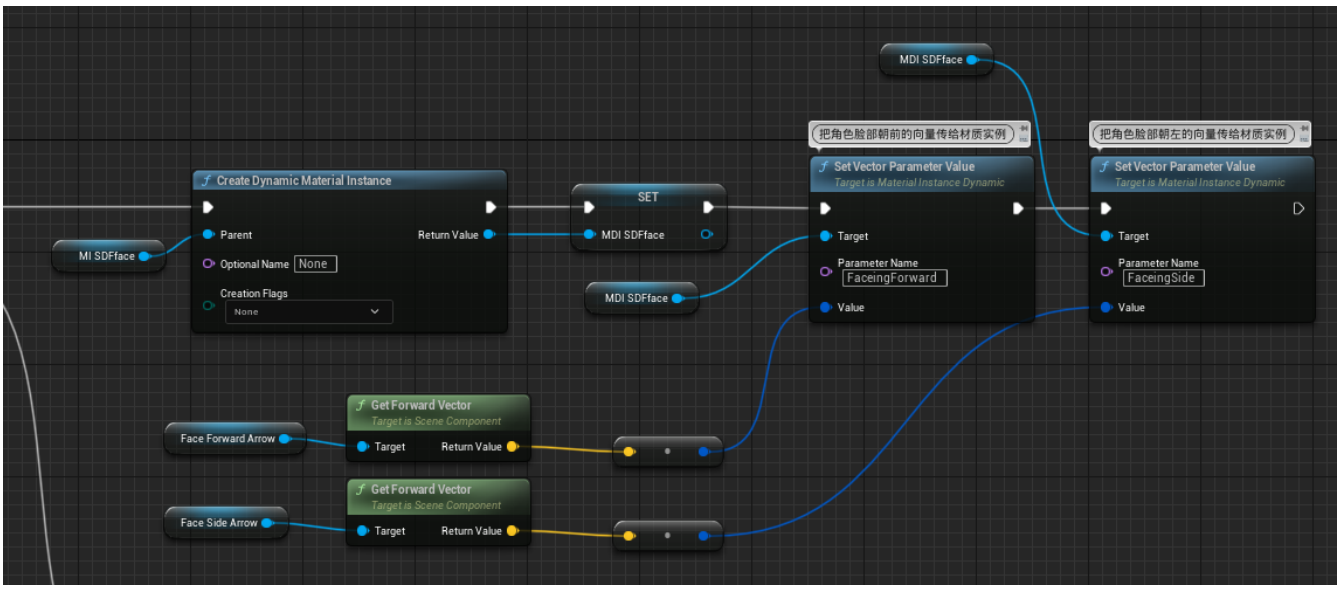


美术的同学直接使用打包好的工具就行了
工具具体的使用文档可以在我的github上查看，源码也在上面，工具写得不是很鲁棒，如果出了什么问题的话有能力的朋友直接改源码吧。

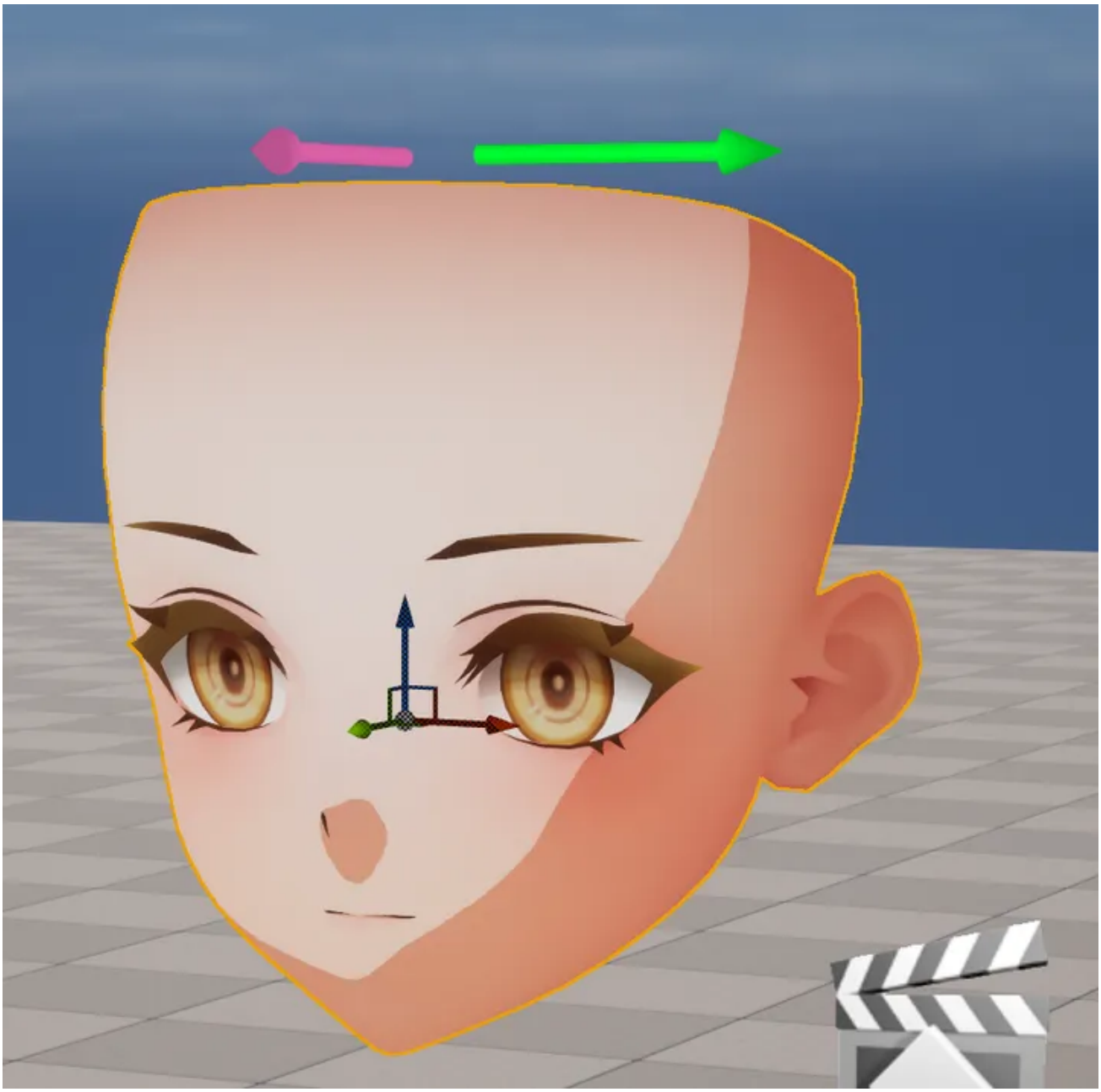


2.5 计算光源角度采样SDF图集

下面很多操作都跟正常的SDF脸部光照一样，我就不写得太详细了，具体直接打开我的工程看就行了。
1.在蓝图里把角色脸部向前的向量和先左的向量传到角色的材质里

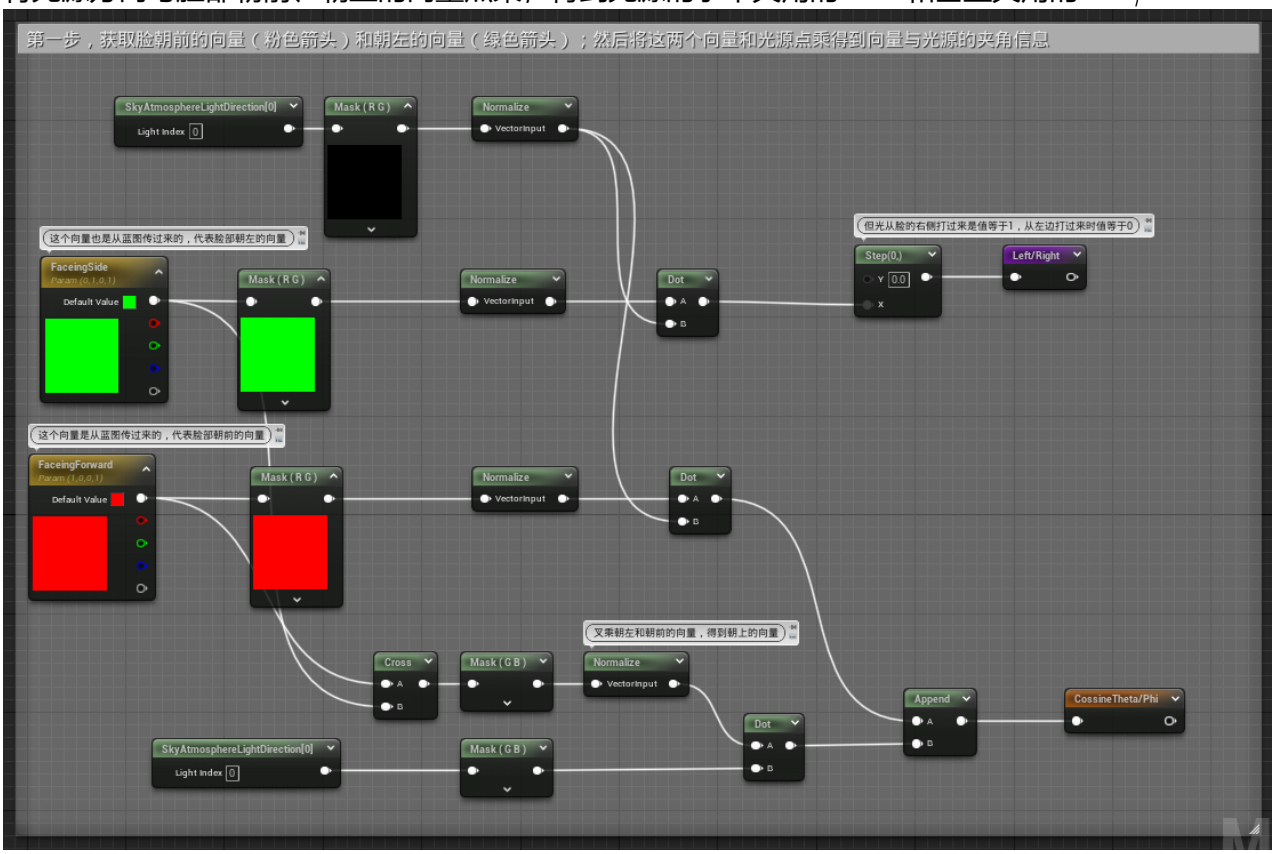


就是把下图下面这两个箭头的方向传到材质里：

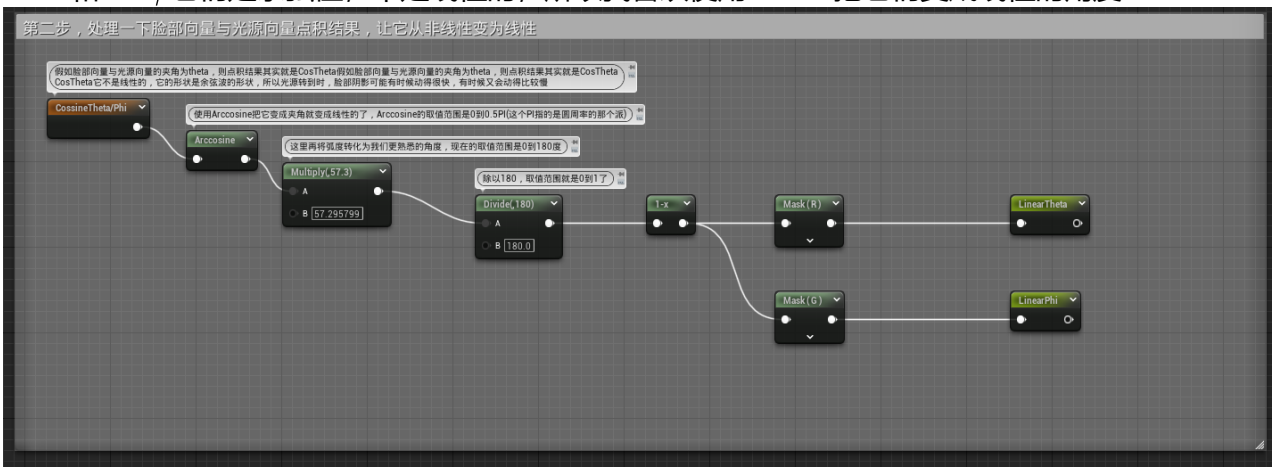


- 叉乘朝左和朝前的向量，得到朝上的向量
- 将光源方向与脸部朝左向量点乘，然后step一下，用于后面判断光源是从脸部左边还是右边照过来的

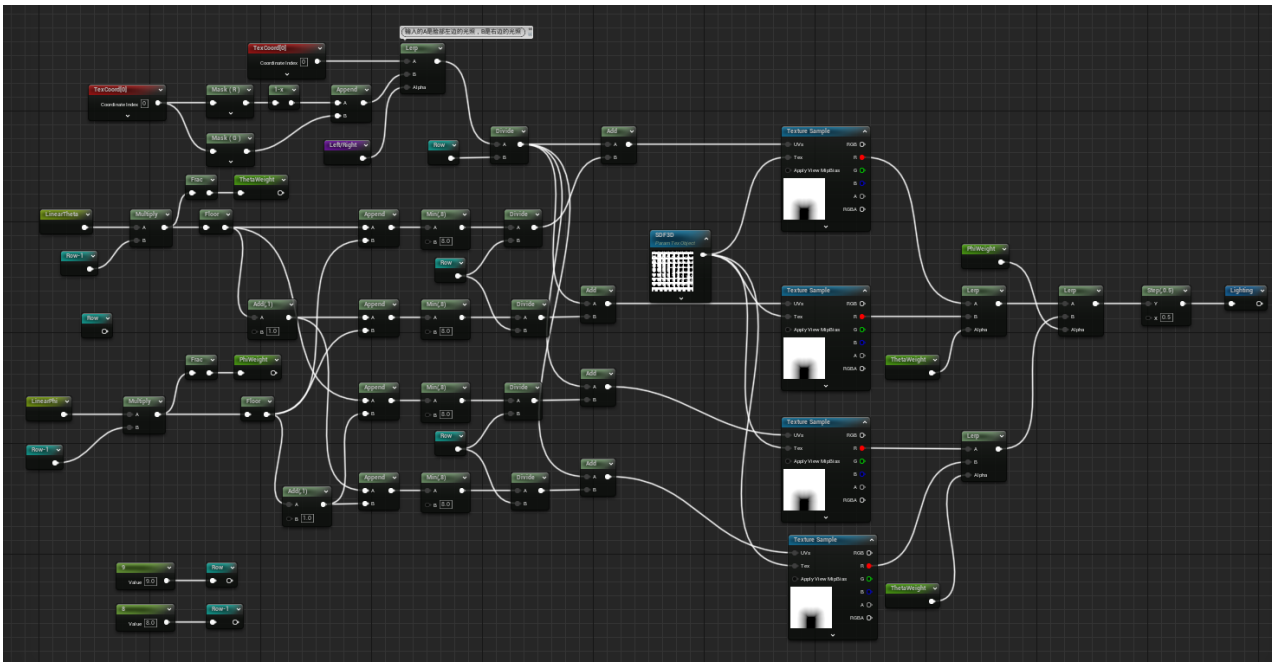
- 将光源方向与脸部朝前、朝上的向量点乘，得到光源的水平夹角的$cos\theta$和垂直夹角的$cos\phi$



- $cos\theta$和$cos\phi$它们是余弦值，不是线性的，所以我喜欢使用arccos把它们变成线性的角度：

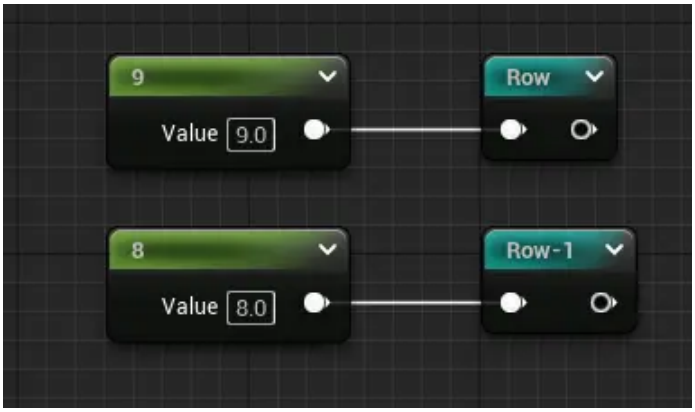


- 接下来采样的部分看起来比较杂乱

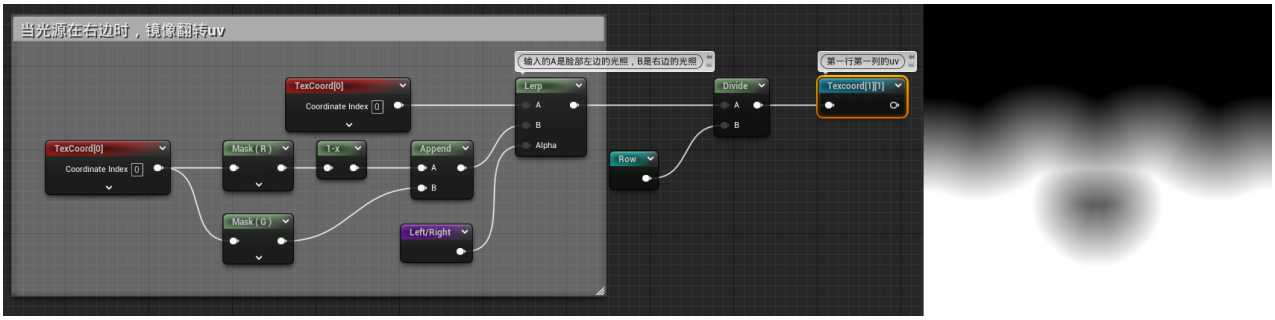


我们稍微拆开来看：

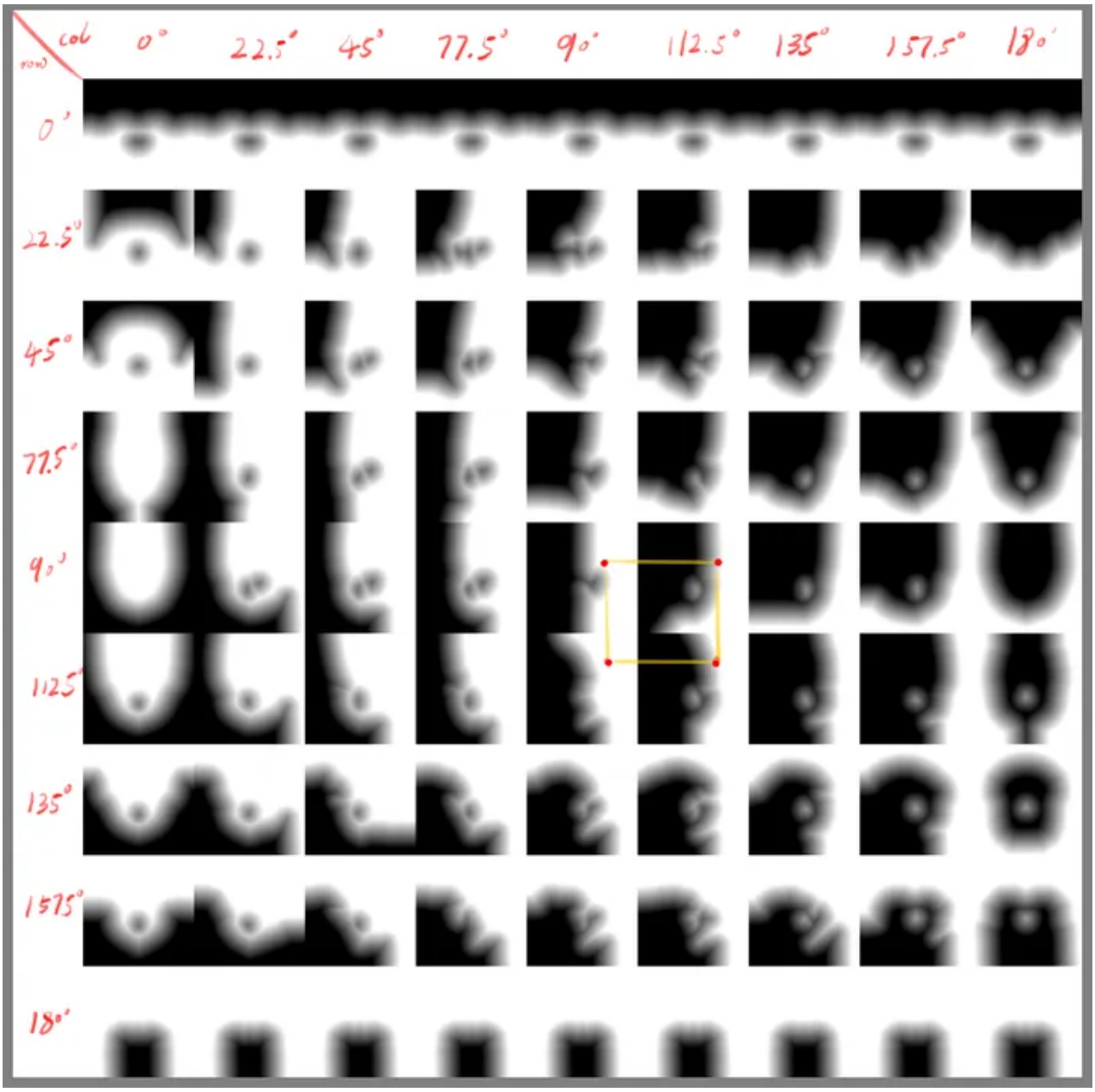
- 这里是定义了两个常数，应该图集是9x9的，所有定义了Row=1，Row-1的意思是Row-1=8



- 当光源在右边时，镜像翻转uv
- 然后uv除以行数9，就得到了贴图左上角的第一张SDF的uv

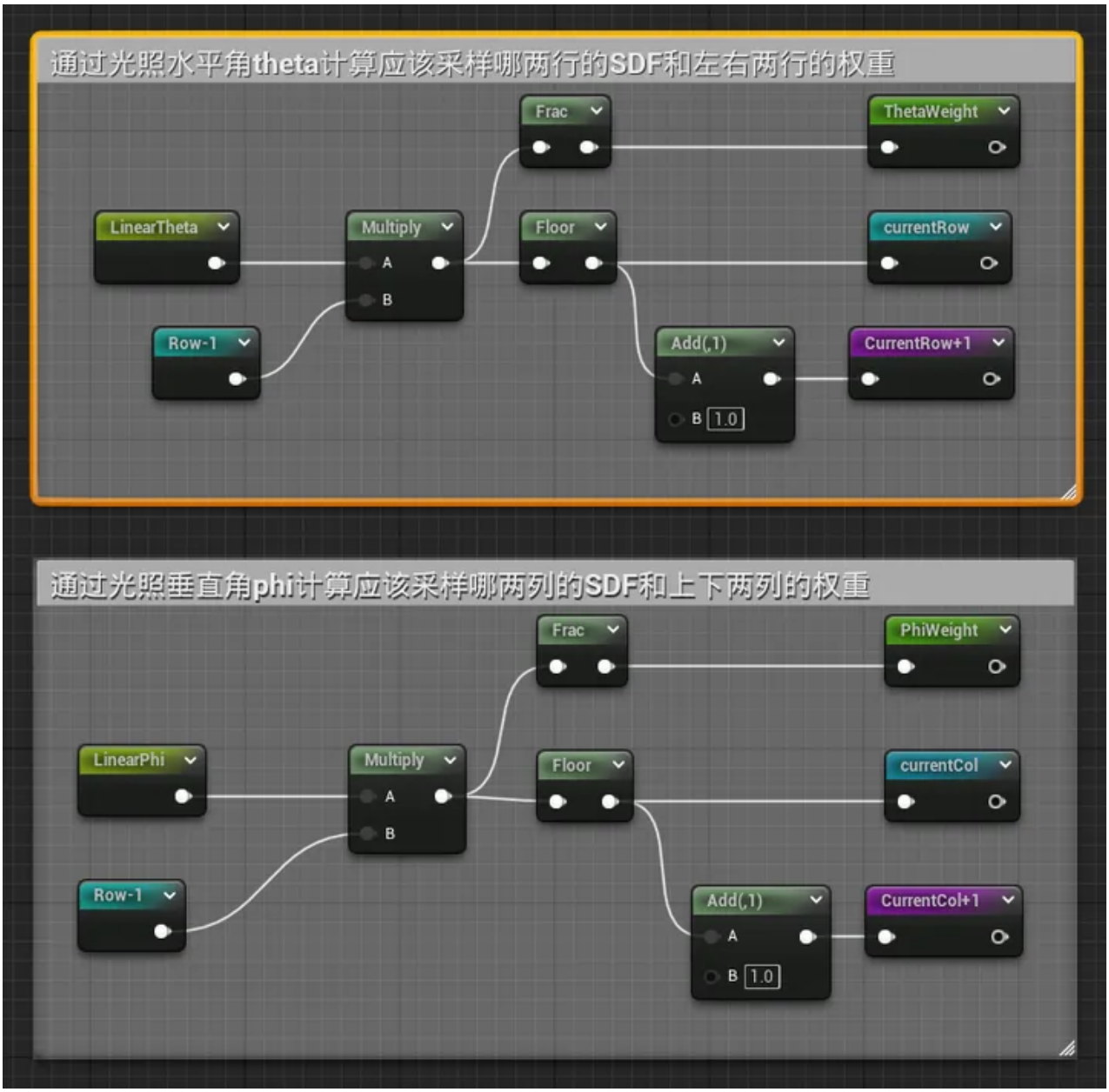


- 接下来要计算离当前光源角度最近的4张SDF的uv，比如当水平角度𥚃等于100度，垂直角度𥚃等于120度时会采样下面画的四个点的SDF，然后按照权重来插值，跟双线性插值很像。

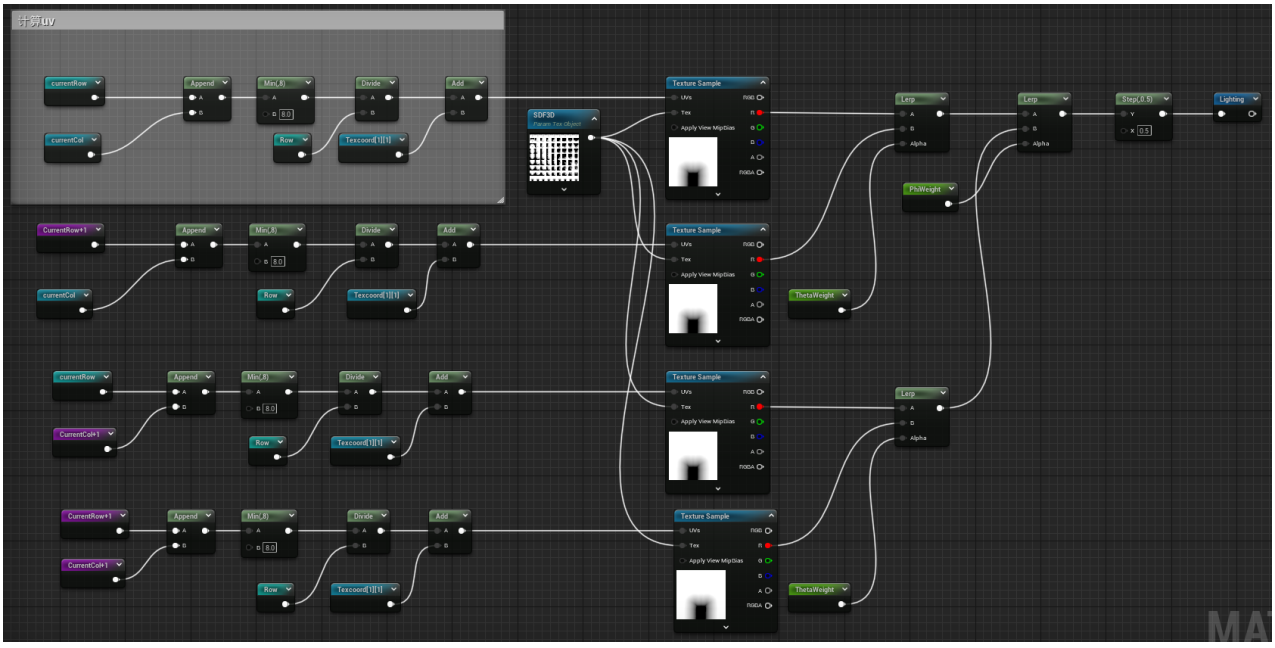


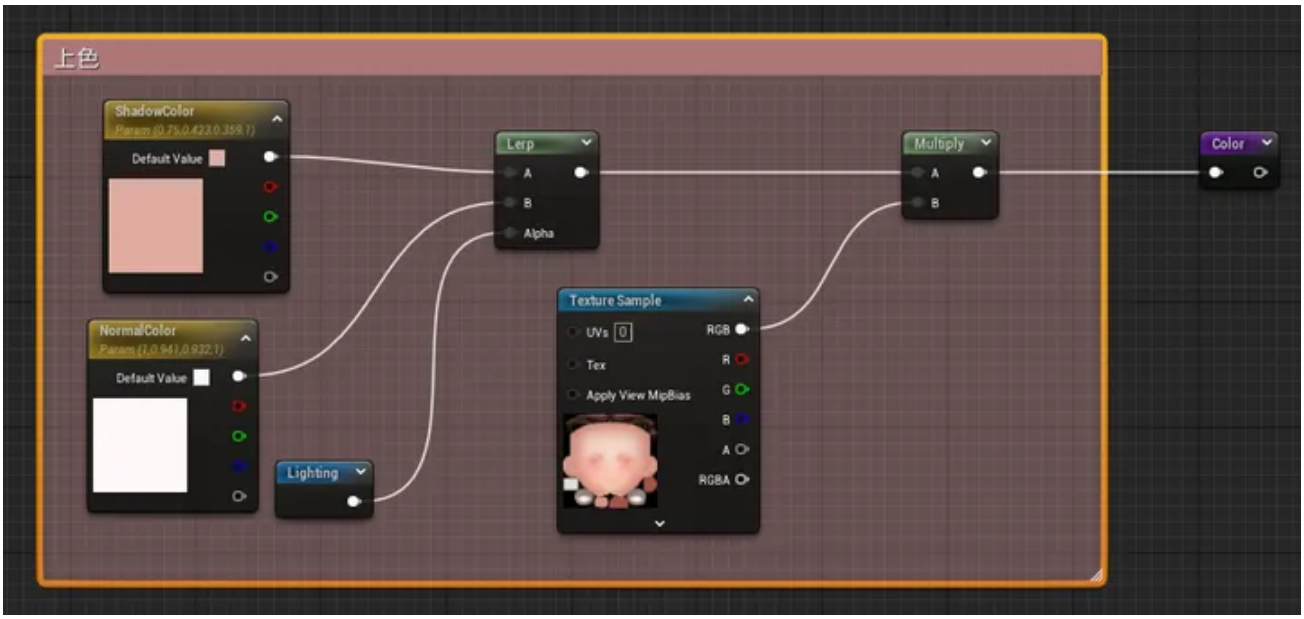
采样当前光源角度最近的四张SDF

- 这里计算离光源最近的行数和列数



- 接下来计算出uv，采样贴图并插值出光照结果最后上色




上色
ShadowColor
Default Value
Lerp
A
B
Alpha
Multiply
A
B
Color
NormalColor
Default Value
Lighting
Texture Sample
UVs 0
Tex
Apply View MipBias
RGB
R
G
B
A
RGBA