### 传统摄影测量vs Nerf vs 3DGS

摄影测量利用大量重叠的图像序列，生成sparse 点云，生成dense 点云，生成mesh网格+外观纹理texture

Nerf利用相对较少的图像来隐式学习三维表达

3DGS仍然是显式的三维空间表达，但是不是点云，而是高斯splatting

### 球谐函数（Spherical Harmonics）

<https://zhuanlan.zhihu.com/p/351289217>

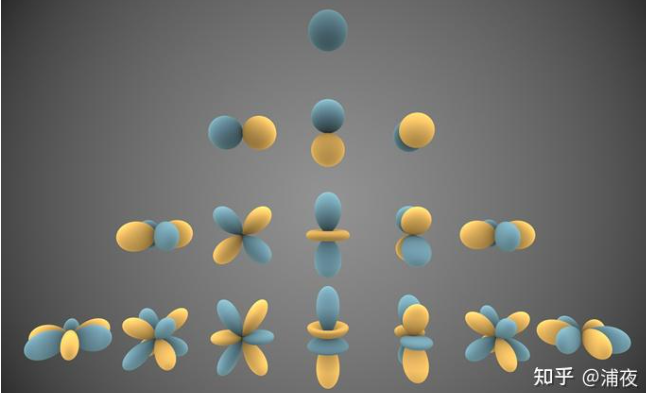
球谐函数本质就是一组基函数

泰勒展开：,这里的1,x,x^2都是基函数

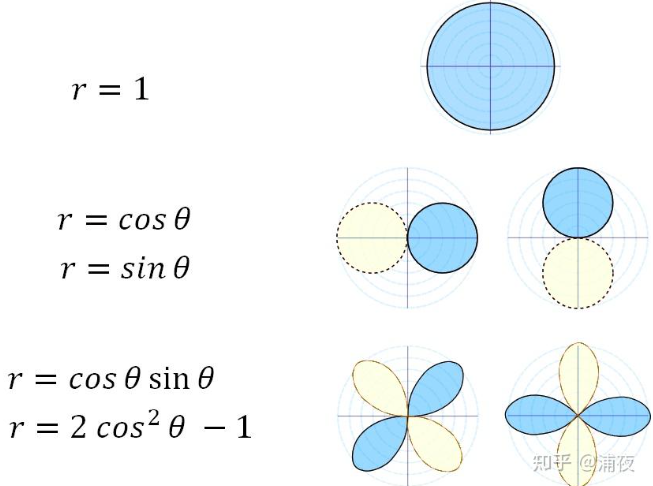
傅里叶变换也是一系列三角函数基函数的累和

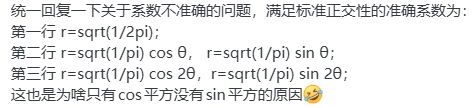
一般来说基函数越多，表达能力就越强（泰勒N展开越多，越精准）

在！！三维球面坐标系！！，著名的就是球谐函数，正交性，旋转不变性等。。

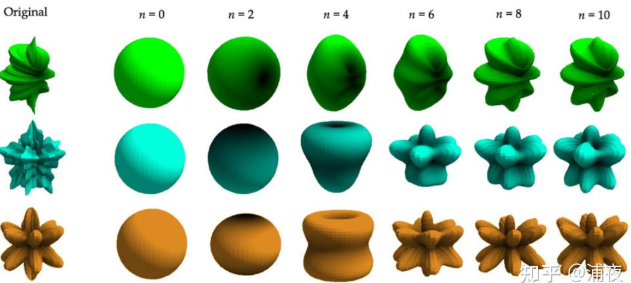


以二维为示例：





注：



球谐函数就是对应基函数的前面的系数，用不同的系数组合来表示不同三维曲面

在图形学，会用球谐函数（二阶或者三阶）来表示某一个位置在不同视角下的光照或rgb

### Splatting

A math equation with numbers and symbols

AI-generated content may be incorrect.

就是把3D高斯表示出来的椭球以某种方式mapping到相机平面，会获得一个椭圆，一大堆椭球splatting到相机平面上就会得到一个图像（整个过程是可微的，所以就可以和ground truth图像作loss获得梯度），**所以其实广义上看，一个3D可微的过程+神经网络+真实图像=新3d表示**



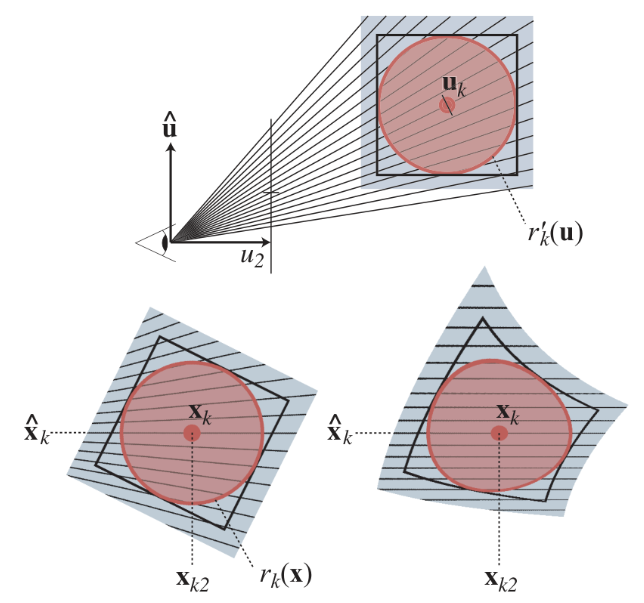
3DGS的高斯公式很明确，去掉了scale系数，去掉了坐标中心，后续方便添加世界坐标系的变换。



协方差矩阵有半正定的约束，对于优化问题不友好，改为优化R和轴向的scale变换。等于是先轴向放缩，再旋转。

Note：渲染管线：local， world， view（相机坐标系）， view坐标project之后为NDC坐标系， screen/pixel坐标。

为了满足现代图形学渲染管线的MVP变换，三角mesh是一组点的变换，不会出现问题，但是3D高斯分布本身是一个分布，project变换本身不满足仿射的性质。



根据[EWA Volume Splatting](https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=8a1a5500111ad35e8bae41e21fcfdcd41c2d8f12) ，

3D高斯分布的协方差矩阵可以近似上面式子，其中J是project变换的affine近似的雅可比矩阵。

### 代码

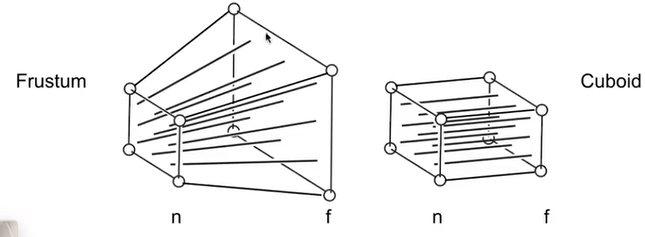
<https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/opengl-perspective-projection-matrix.html>

MVP= P V M

M：将模型空间变换到世界坐标

V：世界坐标变换到view观察空间

P：相机空间变换到clip space，再到NDC空间



为了使得frustum变换到cuboid，则有关系

假设右手系opengl，看向-z轴，n和f都是实际坐标也就是负数（有些推导会用正数）

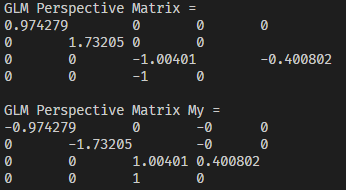
假如右手系opengl，看向-z轴，但是n和f为正数，实际near plane和far plane为z=-near, z=-far

下一步，把cubiod 通过translate和scale变换到-1,1立方体上去，原点为000，同时注意此时n要映射到-1，f映射到1

代入上面的第一个假设，**n和f都是坐标**

把两个矩阵合并，注意这里仍然是右手系bottom,top对应-1,1, right,left对应-1,1，**near,far对应-1,1**，

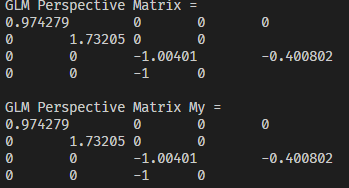
下面使用GLM来测试以上推导 (*代码参考myglm*)



而大多数opengl教材都是假定r l t b是带符号的坐标值，n和f为距离而非坐标值，那么我们有

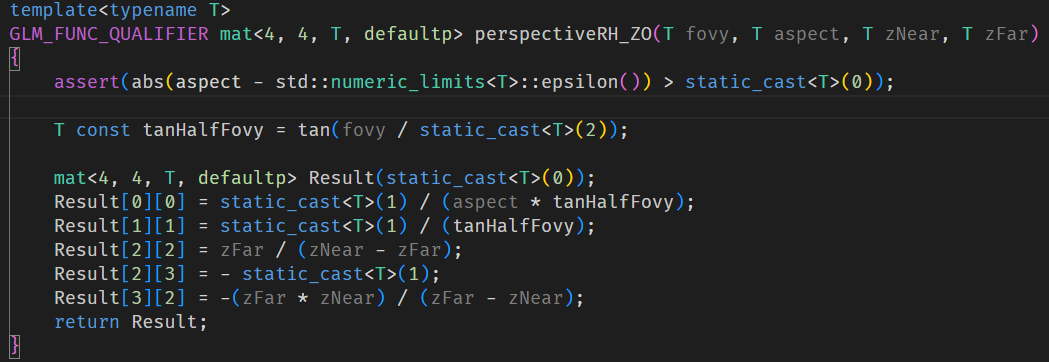
这个M\_persp-ortho实际把z=-near映射到了z=near，z=-far映射到了z=far

因为z=-near和z=-far已经映射到了沿着z轴方向



结果和GLM的匹配

<https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/opengl-perspective-projection-matrix.html> 提到有些平台会映射到[0,1]，我们把上面的式子稍加修改

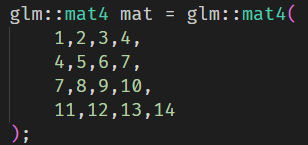


看3dgs的getProjectionMatrix的实现

## 代码

非常好的参考<https://www.guyuehome.com/userDetail?id=1824722788112207873>

**注意glm的实现都是col-major，如果按下初始化，实际矩阵会被初始化为**



而glm的矩阵乘法依照正常思路，比如上述矩阵的T乘以该矩阵实际上就是

整体流程分为

1. Preprocess. 预处理，也就是完成投影部分的计算。每个thread处理一个高斯，计算3D协方差，计算2D协方差，计算SH颜色，计算投影的2D高斯和tile的相交情况，为渲染做准备
2. 渲染前的处理. 参考下面图示
3. 实际的渲染。

## Preprocess

### 2D协方差的计算

假如世界坐标系下的点

则有相机坐标系下的点

假设相机坐标系的下的

投影矩阵为

投影坐标为

对应的雅可比矩阵为

根据论文，可以直接计算获得2D协方差（这里具体原理是3DGS引用了另一篇论文，有兴趣可以看看）

**如何计算2x2矩阵的Det行列式**

如何计算2阶方阵的逆

如何求2阶方阵的特征值

协方差矩阵的特征值意义

二维高斯的等密度线为，可以证明是椭圆方程，若协方差矩阵的特征值为，则椭圆的两个半轴长度为，代码中取了c=9，覆盖了99%的椭圆（高斯的性质）

根据[Mip-Splatting: Alias-free 3D Gaussian Splatting](https://openaccess.thecvf.com/content/CVPR2024/papers/Yu_Mip-Splatting_Alias-free_3D_Gaussian_Splatting_CVPR_2024_paper.pdf)提到，具体看这篇文章

1. 膨胀。防止2D高斯过小，将2D高斯协方差替换为
2. 抗锯齿。对2D高斯加入缩放因子，以实现抗锯齿

所以实际的2D高斯公式变为

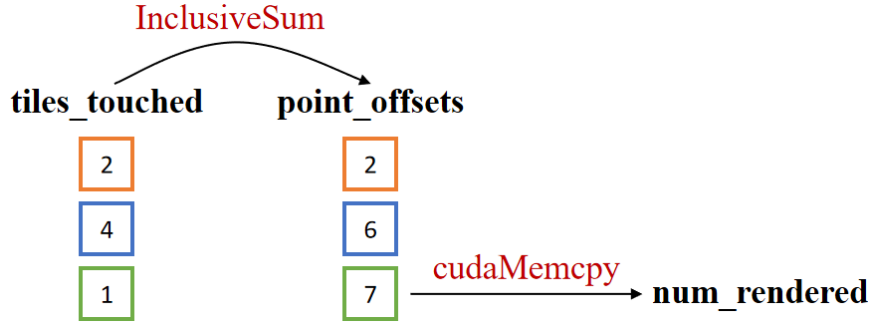
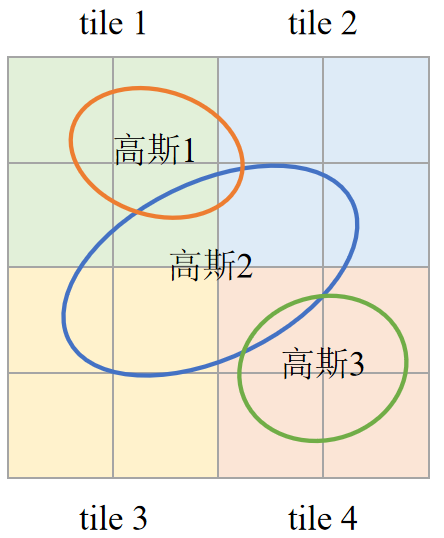
NDC空间如何变换到pixel （ndc2Pix函数）

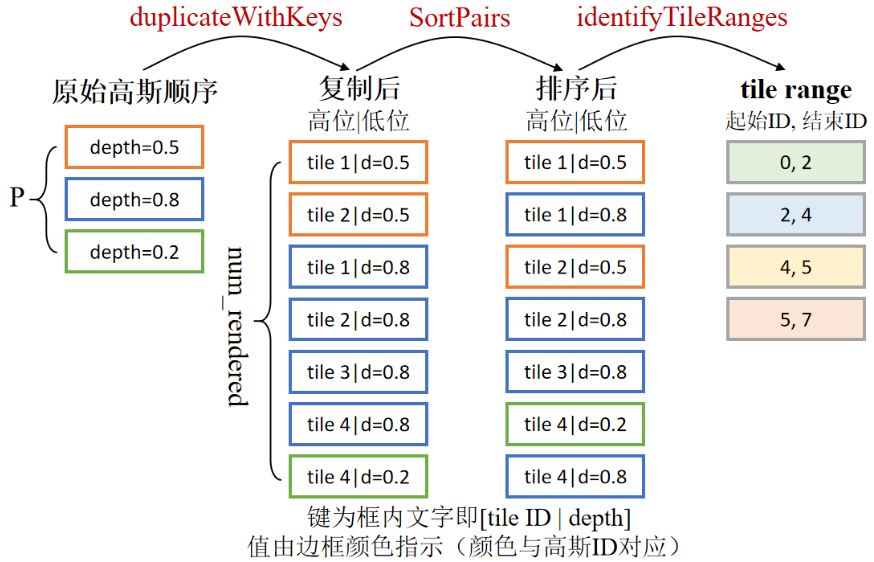
把x从[-1,1]变换到[-0.5, S-0.5]，S是image\_width或者image\_height，所以输出是像素坐标，为什么这么输出？将图像像素中心是为0,1,2,3,…,S-1，所以输出像素的最左/上端边缘为-0.5，最右/下端边缘为S-0.5

### 渲染前的准备

上面的流程主要完成了投影部分，下面开始完成渲染前的工作。

这里直接复制了参考的资料，图很好理解





1. （duplicateWithKeys）对于每个高斯，进行复制，复制次数为与tile相交的个数。并且生成[tile ID | depth]
2. （SortPairs）按tile ID排序，按深度为第二key排序，使得同一tile的高斯会相邻在一起。
3. （identifyTileRanges）获得每个排序后数组，每个tile的起始和结束index

### 渲染

每个线程负责一个像素

遍历每个当前tile相交的高斯（前面已经按深度排好序），每次取BLOCK\_SIZE个高斯：

把高斯的信息拉入shared\_mem，因为每个像素都要使用这BLOCK\_SIZE个高斯的信息，减少内存访存

内for循环：遍历拉取到shared\_mem的高斯

更新像素的透射率，颜色等

如果累积透射率被消耗完了，退出，不再迭代深度更深的高斯

实际计算公式为

其中为第i个高斯的颜色，为第i个高斯在像素的不透明度，为累积透射率，为第i个高斯x处的值

其中也就是2D高斯中心和像素的坐标的差值

在代码中采用

至此获得了每个像素的渲染结果