(5) 图像渲染 CUDA 实现部分: submodules/diff-gaussian-rasterization/rasterize\_points.cu中的RasterizeGaussiansCUDA类和RasterizeGaussiansBackwardCUDA类;

*// 创建并返回一个 lambda 表达式，该表达式用于调整 torch::Tensor 对象的大小，并返回一个指向它数据的原始指针*

std**::**function**<char\***(size\_t N)**>** resizeFunctional(torch**::**Tensor**&** t) {

**auto** lambda **=** [**&**t](size\_t N) {

t.resize\_({(**long** **long**)N});

**return** **reinterpret\_cast<char\*>**(t.contiguous().data\_ptr());

};

**return** lambda;

}

std**::**tuple**<int**, torch**::**Tensor, torch**::**Tensor, torch**::**Tensor, torch**::**Tensor, torch**::**Tensor**>**

RasterizeGaussiansCUDA(

**const** torch**::**Tensor**&** background,

**const** torch**::**Tensor**&** means3D,

**const** torch**::**Tensor**&** colors,

**const** torch**::**Tensor**&** opacity,

**const** torch**::**Tensor**&** scales,

**const** torch**::**Tensor**&** rotations,

**const** **float** scale\_modifier,

**const** torch**::**Tensor**&** cov3D\_precomp,

**const** torch**::**Tensor**&** viewmatrix,

**const** torch**::**Tensor**&** projmatrix,

**const** **float** tan\_fovx,

**const** **float** tan\_fovy,

**const** **int** image\_height,

**const** **int** image\_width,

**const** torch**::**Tensor**&** sh,

**const** **int** degree,

**const** torch**::**Tensor**&** campos,

**const** **bool** prefiltered,

**const** **bool** debug)

{

**if** (means3D.ndimension() **!=** 2 **||** means3D.size(1) **!=** 3) {

AT\_ERROR("means3D must have dimensions (num\_points, 3)");

}

**const** **int** P **=** means3D.size(0);

**const** **int** H **=** image\_height;

**const** **int** W **=** image\_width;

**auto** int\_opts **=** means3D.options().dtype(torch**::**kInt32);

**auto** float\_opts **=** means3D.options().dtype(torch**::**kFloat32);

torch**::**Tensor out\_color **=** torch**::**full({NUM\_CHANNELS, H, W}, 0.0, float\_opts); *// (3, H, W), 在指定的视角下, 对所有3D gaussian进行投影和渲染得到的图像*

torch**::**Tensor radii **=** torch**::**full({P}, 0, means3D.options().dtype(torch**::**kInt32)); *// (P,)*

torch**::**Device device(torch**::**kCUDA);

torch**::**TensorOptions options(torch**::**kByte);

torch**::**Tensor geomBuffer **=** torch**::**empty({0}, options.device(device)); *// (0,), 存储所有3D gaussian对应的参数(均值、尺度、旋转参数、不透明度)的tensor, 会动态分配存储空间*

torch**::**Tensor binningBuffer **=** torch**::**empty({0}, options.device(device)); *// (0,)*

torch**::**Tensor imgBuffer **=** torch**::**empty({0}, options.device(device)); *// (0,), 存储在指定视角下渲染得到的图像的tensor, 会动态分配存储空间*

std**::**function**<char\***(size\_t)**>** geomFunc **=** resizeFunctional(geomBuffer); *// 动态调整 geomBuffer 大小的函数, 并返回对应的数据指针*

std**::**function**<char\***(size\_t)**>** binningFunc **=** resizeFunctional(binningBuffer); *// 动态调整 binningBuffer 大小的函数, 并返回对应的数据指针*

std**::**function**<char\***(size\_t)**>** imgFunc **=** resizeFunctional(imgBuffer); *// 动态调整 imgBuffer 大小的函数, 并返回对应的数据指针*

**int** rendered **=** 0;

**if**(P **!=** 0)

{

**int** M **=** 0;

**if**(sh.size(0) **!=** 0)

{

M **=** sh.size(1);

}

rendered **=** CudaRasterizer**::**Rasterizer**::**forward(

geomFunc,

binningFunc,

imgFunc,

P, degree, M, *// 3D gaussian的个数, 球谐函数的次数, 球谐系数的个数 (球谐系数用于表示颜色)*

background.contiguous().data**<float>**(), *// 背景颜色, [0, 0, 0]*

W, H, *// 图像的宽和高*

means3D.contiguous().data**<float>**(), *// 每个3D gaussian的XYZ均值*

sh.contiguous().data\_ptr**<float>**(), *// 每个3D gaussian的球谐系数, 用于表示颜色*

colors.contiguous().data**<float>**(), *// 提前计算好的每个3D gaussian的颜色, []*

opacity.contiguous().data**<float>**(), *// 每个3D gaussian的不透明度*

scales.contiguous().data\_ptr**<float>**(), *// 每个3D gaussian的XYZ尺度*

scale\_modifier, *// 尺度缩放系数, 1.0*

rotations.contiguous().data\_ptr**<float>**(), *// 每个3D gaussian的旋转四元组*

cov3D\_precomp.contiguous().data**<float>**(), *// 提前计算好的每个3D gaussian的协方差矩阵, []*

viewmatrix.contiguous().data**<float>**(), *// 相机外参矩阵, world to camera*

projmatrix.contiguous().data**<float>**(), *// 投影矩阵, world to image*

campos.contiguous().data**<float>**(), *// 所有相机的中心点XYZ坐标*

tan\_fovx, *// 水平视场角一半的正切值*

tan\_fovy, *// 垂直视场角一半的正切值*

prefiltered, *// 是否预先过滤掉了中心点(均值XYZ)不在视锥（frustum）内的3D gaussian, False*

out\_color.contiguous().data**<float>**(), *// 在指定的视角下, 对所有3D gaussian进行投影和渲染得到的图像*

radii.contiguous().data**<int>**(), *// 存储每个2D gaussian在图像上的半径*

debug); *// False*

}

**return** std**::**make\_tuple(rendered, out\_color, radii, geomBuffer, binningBuffer, imgBuffer);

}