BEVDet优化 : BEV Align 自研算子

一. 安装和编译

代码请移步 code

Configure

Please move this project to BEVDet, replace the old one

```
rm -rf ${THIS_PROJECT} ${BEVDet_Path}/mmdet3d/ops/bev_pool
mv ${THIS_PROJECT} ${BEVDet_Path}/mmdet3d/ops/bev_pool
```

Add import item in `${BEVDet_Path}/mmdet3d/ops/__init__.py` :

```
from .bev_pool.voxel_align import voxel_pool,voxel_align

__all__.extend(['voxel_align','voxel_pool'])
```

Add source file in `${BEVDet_Path}/setup.py` cuda extend item , replace the old one with :

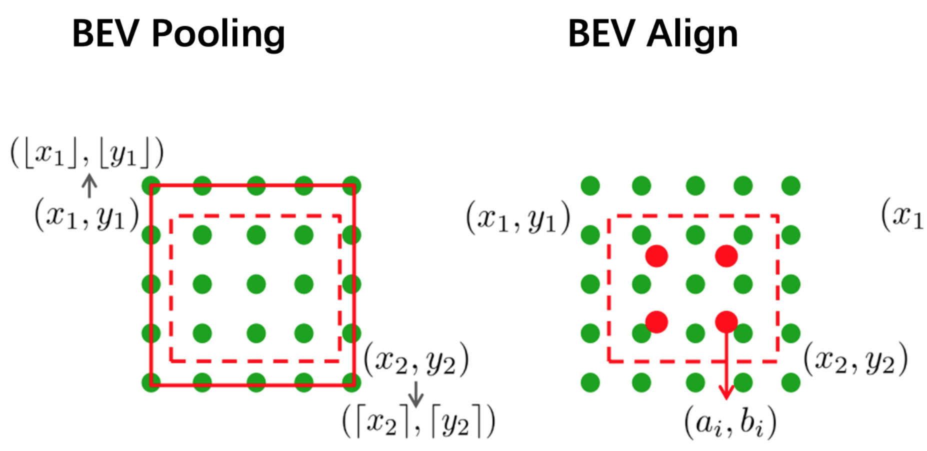
```
make_cuda_ext(
  name="bev_pool_ext",
  module="mmdet3d.ops.bev_pool",
  sources=[
    "src/bev_pool.cpp",
    "src/bev_pool_cuda.cu",
    "src/bev_align_cuda.cu"
  ],
),
```

Install

Then recompiling the bevdet source code

```
cd ${THIS_PROJECT} ${BEVDet_Path}
pip install -v -e .
```

二. 数学原理



在针对BEVDet模型透视图转俯视图的转换过程(view transformer)，针对每个像素对应的椎体voxel，需要将其放置在相应的BEV voxel中。

在已有的方案BEV Pooling 中，给定BEV 伪图像对应的浮点坐标，BEV Pooling采用***向零取整***的方法对计算得到的BEV 浮点坐标转化为定点坐标，这样每个坐标最多损失1个像素的精度，假设voxel size(0.8 x 0.8)则最多损失0.8米的精度，对于整个BEV feature，则是系统性精度损失。

解决方法是对椎体voxel的特征在BEV 伪图片上进行双线性插值，对每个浮点坐标，分别计算这个浮点坐标到最近的4个整型坐标的权重，并将这个feature依权重赋予临近4个整点坐标对应的voxel中。

计算公式参考如下代码：

```
            float cur_feat_channel_value = dev_volume[batch_idx*sn*d*fh*fw*c+ sensor_idx*d*fh*fw*c+dhwc_idx];
            int x0 = __float2int_rd(bev_idx_x);
            int y0 = __float2int_rd(bev_idx_y);
            int x1 = x0+1; int y1=y0+1;
            float wa = (x1-bev_idx_x) * (y1-bev_idx_y);
            float wb = (x1-bev_idx_x) * (bev_idx_y-y0);
            float wc = (bev_idx_x-x0) * (y1-bev_idx_y);
            float wd = (bev_idx_x-x0) * (bev_idx_y-y0);


            // top left
            if (x0 >= 0 && x0 < bev_w && y0 >=0 && y0 < bev_h)
            {
              atomicAdd( bev_feat + batch_idx * bev_h * bev_w * c +
                        channel_idx * bev_h * bev_w + y0 * bev_w + x0 ,
                        cur_feat_channel_value * wa);
            }
            __threadfence();

            // bottom left
            if (x0 >= 0 && x0 < bev_w && y1 >=0 && y1 < bev_h)
            {
              atomicAdd( bev_feat + batch_idx * bev_h * bev_w * c +
                        channel_idx * bev_h * bev_w + y1 * bev_w + x0 ,
                        cur_feat_channel_value * wb);
            }
            __threadfence();

            // top right
            if (x1 >= 0 && x1 < bev_w && y0 >=0 && y0 < bev_h)
            {
              atomicAdd( bev_feat + batch_idx * bev_h * bev_w * c +
                        channel_idx * bev_h * bev_w + y0 * bev_w + x1 ,
                        cur_feat_channel_value * wc);
            }
            __threadfence();

            // bottom right
            if (x1 >= 0 && x1 < bev_w && y1 >=0 && y1 < bev_h)
            {
              atomicAdd( bev_feat + batch_idx * bev_h * bev_w * c +
                        channel_idx * bev_h * bev_w + y1 * bev_w + x1 ,
                        cur_feat_channel_value * wd);
            }
            __threadfence();
```

三. 反向传播

在反向传播过程中，有了BEV feature的梯度，需要实现从BEV到椎体透视图feature map之间的梯度计算，由于前述使用了线性插值，在缓存中需要保存原bev浮点坐标，并在BP的过程中，重新计算浮点坐标到4个临近整型坐标的权重，从而从4个临近整型坐标对用的voxel中将梯度加权得到该浮点坐标对应的梯度。

计算公式参考如下代码：

```
            int x0 = __float2int_rd(bev_idx_x);
            int y0 = __float2int_rd(bev_idx_y);
            int x1 = x0+1; int y1=y0+1;
            float wa = (x1-bev_idx_x) * (y1-bev_idx_y);
            float wb = (x1-bev_idx_x) * (bev_idx_y-y0);
            float wc = (bev_idx_x-x0) * (y1-bev_idx_y);
            float wd = (bev_idx_x-x0) * (bev_idx_y-y0);

            float grad_value = 0;
            // top left
            if (x0 >= 0 && x0 < bev_w && y0 >=0 && y0 < bev_h)
            {
              grad_value += bev_feat_grad[batch_idx * bev_h * bev_w * c + channel_idx * bev_h * bev_w + y0 *
bev_w + x0] * wa;
            }

            // bottom left
            if (x0 >= 0 && x0 < bev_w && y1 >=0 && y1 < bev_h)
            {
              grad_value += bev_feat_grad[batch_idx * bev_h * bev_w * c + channel_idx * bev_h * bev_w + y1 *
bev_w + x0] * wb;
            }

            // top right
            if (x1 >= 0 && x1 < bev_w && y0 >=0 && y0 < bev_h)
            {
              grad_value += bev_feat_grad[batch_idx * bev_h * bev_w * c + channel_idx * bev_h * bev_w + y0 *
bev_w + x1] * wc;

            }

            // bottom right
            if (x1 >= 0 && x1 < bev_w && y1 >=0 && y1 < bev_h)
            {
              grad_value += bev_feat_grad[batch_idx * bev_h * bev_w * c + channel_idx * bev_h * bev_w + y1 *
bev_w + x1] * wd;
            }


            atomicAdd(dev_volume_grad+batch_idx*sn*d*fh*fw*c+ sensor_idx*d*fh*fw*c+dhwc_idx,
                      grad_value);
            __threadfence();
```

四. 实验记录

1. 精度对齐

由于我们的BEV align实现机制与原代码中实现机制不同(原代码实现过程太复杂，计算比较冗余)， 我们首先在BEV align同范式下实现另一个BEV Pool，为了证明我们的代码范式具有同等可用性，需要在BEV pool算子下实现精度对齐。

为此我们从预保存的Tensor中加载img feature和浮点坐标geom, 然后分别用原代码范式实现BEV Pool :

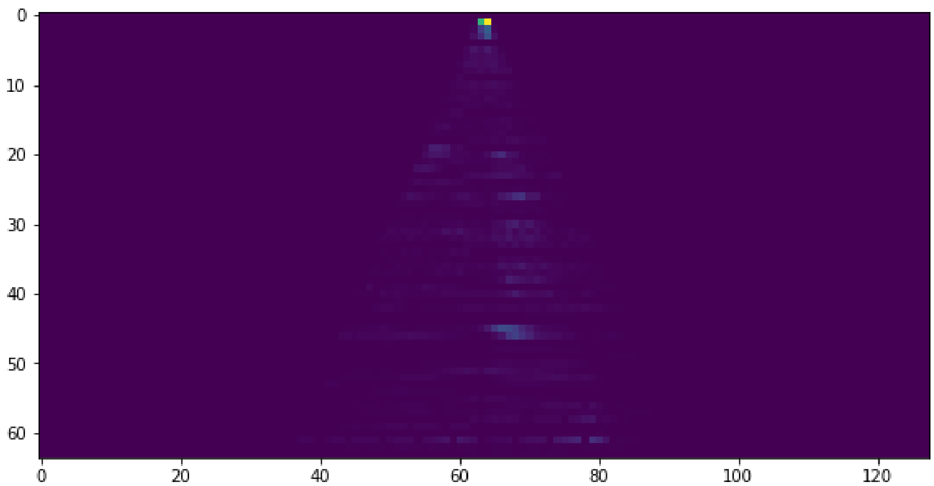
```
volume = torch.load("data/sg_features/volume_prev.pth")
# volume.requires_grad_()
geom_feats = torch.load("data/sg_features/geom_prev.pth")
bx = model.img_view_transformer.bx
nx = model.img_view_transformer.nx
dx = model.img_view_transformer.dx
# geom_feats : torch.Size([1, 6, 59, 16, 44, 3]), (B,N,D,H,W,3),pseudo-pointcloud
# B : batch_size , N : 6 sensors in stereo, D : 59 depth channel, H & W : img feature map size, C : img feature map c
B, N, D, H, W, C = volume.shape
Nprime = B * N * D * H * W
nx = nx.to(torch.long)
# flatten x
volume = volume.reshape(Nprime, C)
# flatten indices
geom_feats = ((geom_feats - (bx - dx / 2.)) / dx)
geom_feats = geom_feats.view(Nprime, 3)
batch_ix = torch.cat([torch.full([Nprime // B, 1], ix,
                                 device=volume.device, dtype=torch.long) for ix in range(B)])
geom_feats = torch.cat((geom_feats, batch_ix), 1)

# geom_feats = geom_feats.float()
geom_feats = geom_feats.int()

## filter out points that are outside box
kept = (geom_feats[:, 0] >= 0) & (geom_feats[:, 0] < nx[0]) \
       & (geom_feats[:, 1] >= 0) & (geom_feats[:, 1] < nx[1]) \
       & (geom_feats[:, 2] >= 0) & (geom_feats[:, 2] < nx[2])

volume = volume[kept]
geom_feats = geom_feats[kept]
volume.requires_grad_()
# geom_feats.requires_grad_()
tic = time.time()
final = bev_pool(volume.cuda(), geom_feats.cuda(), B, nx[2], nx[0], nx[1])
toc = time.time()
print(toc-tic)
final = final.transpose(dim0=-2, dim1=-1).detach().cpu()
final = torch.cat(final.unbind(dim=2), 1)
```

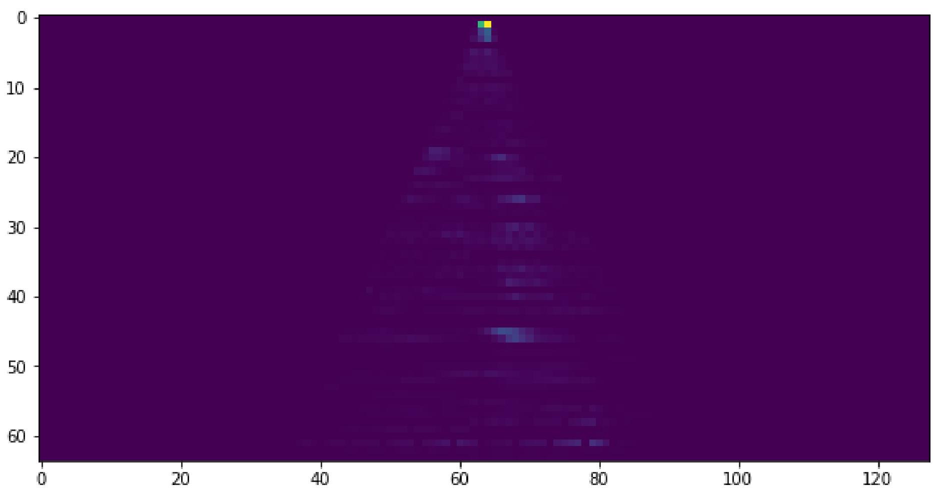
并用我们的代码范式实现BEV Pool

```
#voxel pool
from mmdet3d.ops import bev_pool,voxel_pool, voxel_align
tic = time.time()
output_pool = voxel_pool(volume.cuda(), geom_feats.cuda(), 64,128)
toc = time.time()
print(toc-tic)
```

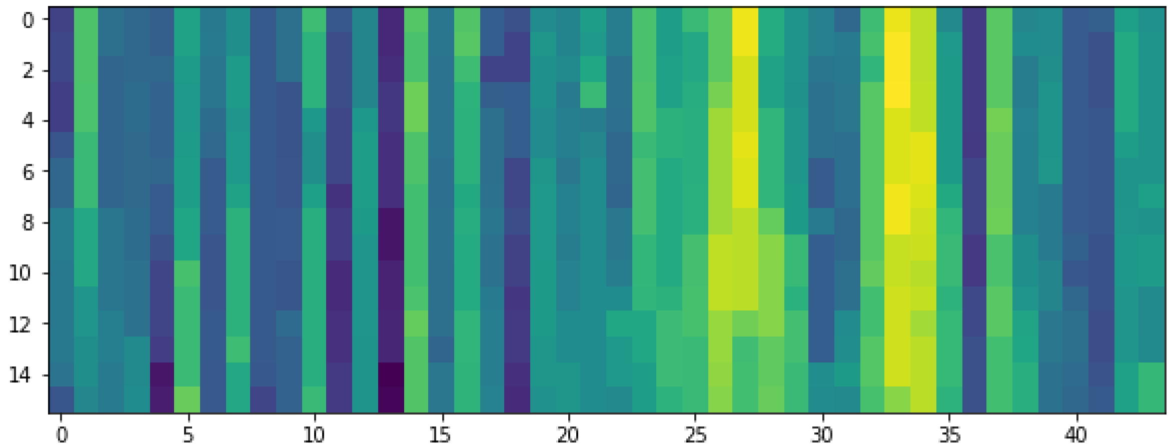
可视化效果如下

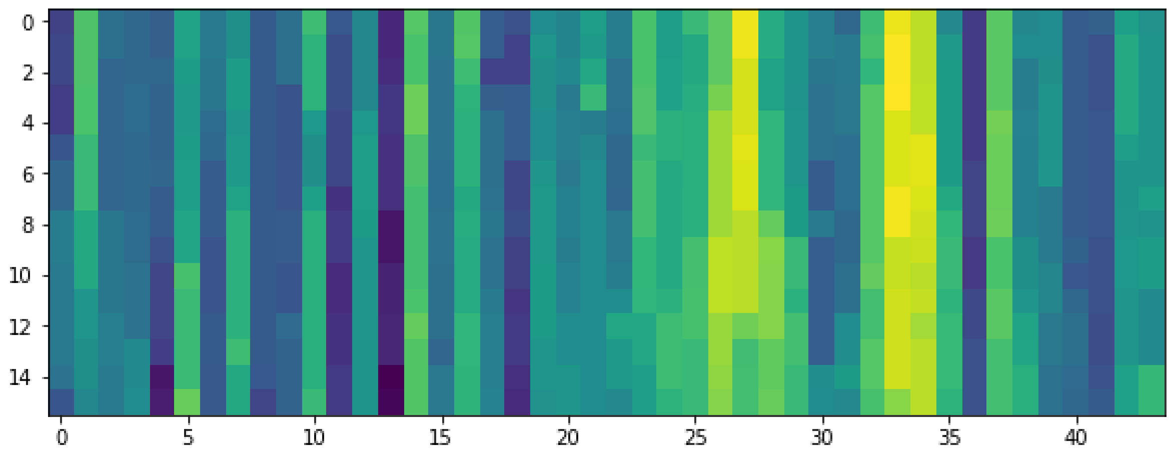
前向传播求BEV Feature map





反向传播求image feature volume





计算精度对应如下

	所有像素累计误差	每个像素平均误差
前向传播(img_feat->bev_feat)	0.001997639	3.810194e-09
反向传播(bev_feat->img_feat)	0.0	0.0

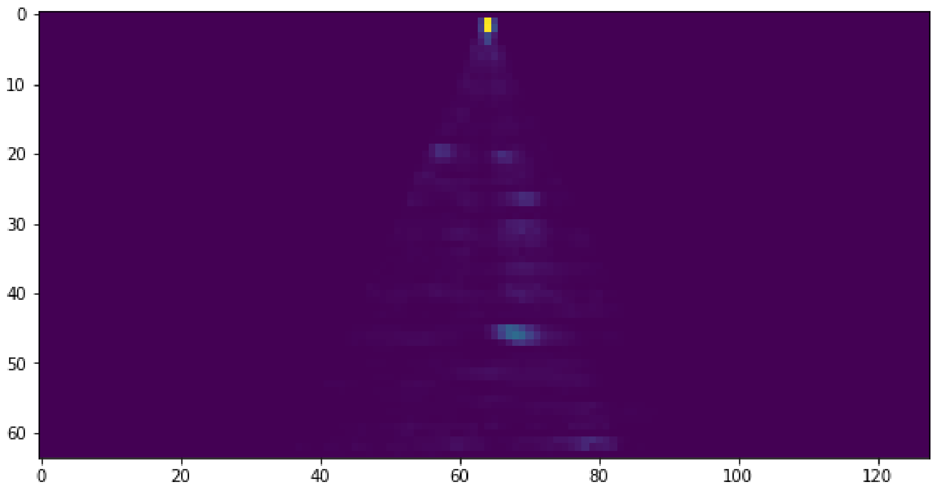
计算结果几乎完全一致，达到1e-9级别的误差

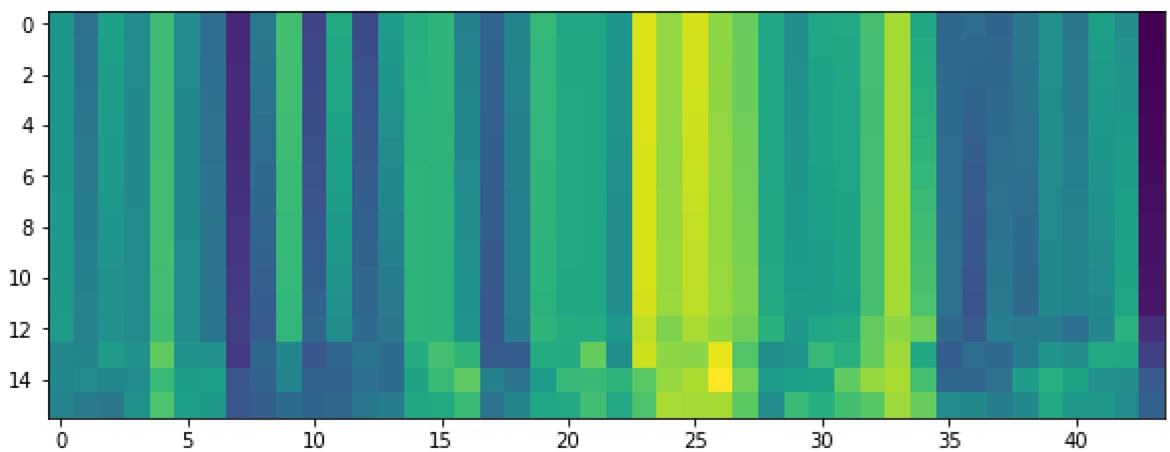
2. BEV Align与BEV Pooling

在同代码范式下，我们实现的BEV Align与BEV Pooling可视化如下：

```
#voxel align
from mmdet3d.ops import bev_pool,voxel_pool, voxel_align
tic = time.time()
output_align = voxel_align(volume.cuda(), geom_feats.cuda(), 64,128)
toc = time.time()
print(toc-tic)
```

前向传播和反向传播





BEV Align做插值之后，相当于对图像做虚化处理，feature整体特征形状一致，效果符合预期。

3. 计算速度(by millisecond)

BEV Pool(Org)	BEV Pool(ours)	BEV Align(ours)
1.03	1.05	1.08

计算速度几乎一致。