Co-SLAM Mapping

Status	Not started

初始化

```
self.create_bounds()
self.keyframeDatabase = self.create_kf_database(config)
def create_kf_database(self, config):
        '''
        Create the keyframe database
        '''
        num_kf = int(self.dataset.num_frames // self.config['mapping']['keyframe_every'] + 1)   # 401
        print('#kf:', num_kf)
        print('#Pixels to save:', self.dataset.num_rays_to_save) # 40800 total * 0.05
        return KeyFrameDatabase(config,
                                self.dataset.H,
                                self.dataset.W,
                                num_kf,
                                self.dataset.num_rays_to_save,
                                self.device)


self.model = JointEncoding(config, self.bounding_box).to(self.device)
def __init__(self, config, bound_box):
        super(JointEncoding, self).__init__()
        self.config = config
        self.bounding_box = bound_box
        self.get_resolution()
        self.get_encoding(config)
        self.get_decoder(config)
```

first_frame_mapping

```
Step 1: 读取pose
Step 2: 图片上随机采取2048个pixel
Step 3: 得到对应的direction，rgb， depth
Step 4: 根据pose得到对应的rays_o. rays_d
Step 5: 根据depth值在表面随机采样11个z_vals，在free space采样32个z_vals
Step 6: 将采样点转换为实际的世界坐标系下的坐标
Step 7: 将采样点输入网络得到预测值 [rgb, sdf]
Step 8: rendering
```

模型训练

```
Step 1: 坐标归一化
Step 2: hash _ one_blob
def query_color_sdf(self, query_points):
        '''
        Query the color and sdf at query_points.

        Params:
            query_points: [N_rays, N_samples, 3]
        Returns:
```

```
            raw: [N_rays, N_samples, 4]
        '''
        inputs_flat = torch.reshape(query_points, [-1, query_points.shape[-1]])

        embed = self.embed_fn(inputs_flat) # hash encoding
        embe_pos = self.embedpos_fn(inputs_flat) # one-blob
        if not self.config['grid']['oneGrid']:
            embed_color = self.embed_fn_color(inputs_flat)
            return self.decoder(embed, embe_pos, embed_color)
        return self.decoder(embed, embe_pos)
```

rendering

```
rgb_map, disp_map, acc_map, weights, depth_map, depth_var = self.raw2outputs(raw, z_vals, self.config['training']['white_bkgd'])
```

```
def sdf2weights(self, sdf, z_vals, args=None):
      '''
      Convert signed distance function to weights.

      Params:
          sdf: [N_rays, N_samples]
          z_vals: [N_rays, N_samples]
      Returns:
          weights: [N_rays, N_samples]
      '''
      # 计算初始权重
      weights = torch.sigmoid(sdf / args['training']['trunc']) * torch.sigmoid(-sdf / args['training']['trunc'])

      # 计算哪些相邻的SDF值之间发生了符号的变化
      signs = sdf[:, 1:] * sdf[:, :-1]
      mask = torch.where(signs < 0.0, torch.ones_like(signs), torch.zeros_like(signs))

      # 找到符号变化的位置： argmax即每一条光线上首次符号变化的位置，即第一个表面的位置
      inds = torch.argmax(mask, axis=1)
      inds = inds[..., None]

      # 获取第一个表面的z值
      z_min = torch.gather(z_vals, 1, inds) # The first surface
      mask = torch.where(z_vals < z_min + args['data']['sc_factor'] * args['training']['trunc'], torch.ones_like(z_vals), torch.zeros_like(

      weights = weights * mask
      return weights / (torch.sum(weights, axis=-1, keepdims=True) + 1e-8)
```