



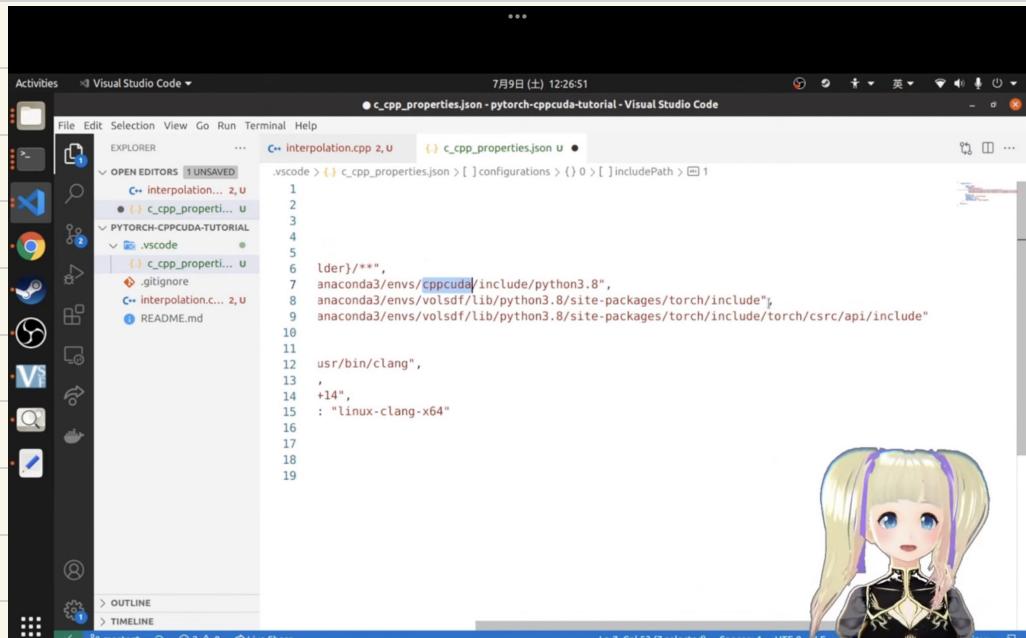
# Cuda 编程

## ① 目录结构

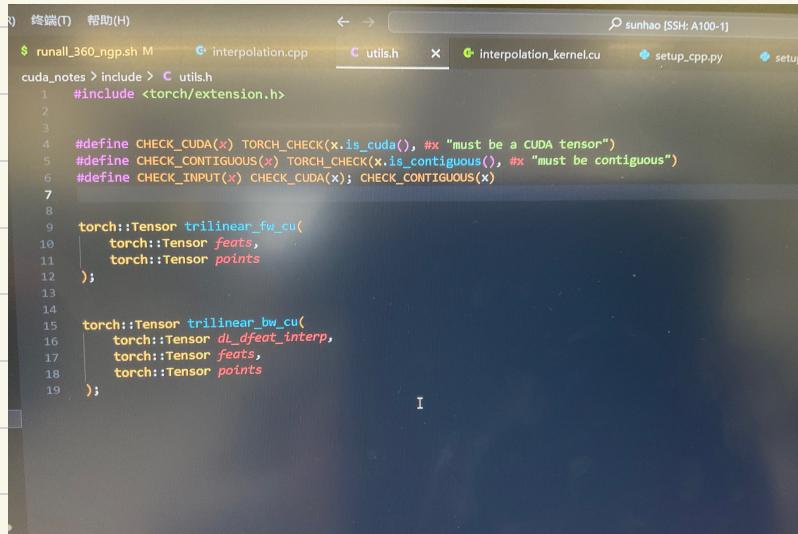
- include
- utils.h
- interpolation-kernel.cu
- interpolation.cpp
- setup.py
- test.py

## ② VScode 环境设置

- i)  $ctrl + shift + P$
- ii) 转入 edit configuration, 拉出 json 配置文件
- iii) 在 "includePath" 添加环境路径

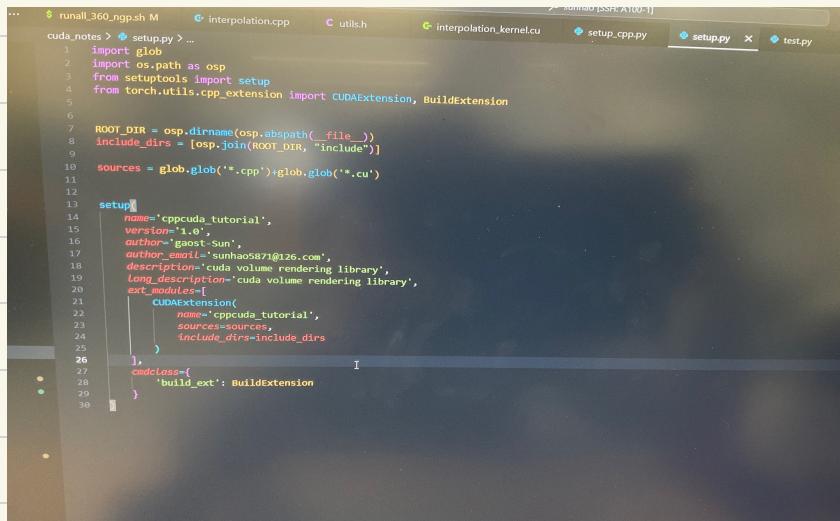


### ③ include/utils.h



```
$ runall_360_ngp.sh M interpolation.cpp utils.h interpolation_kernel.cu setup_cpp.py setup
cuda_notes > include > C utils.h
1 #include <torch/extension.h>
2
3
4 #define CHECK_CUDA(x) TORCH_CHECK(x.is_cuda(), #x "must be a CUDA tensor")
5 #define CHECK_CONTIGUOUS(x) TORCH_CHECK(x.is_contiguous(), #x "must be contiguous")
6 #define CHECK_INPUT(x) CHECK_CUDA(x); CHECK_CONTIGUOUS(x)
7
8
9 torch::Tensor trilinear_fw_cu(
10     torch::Tensor feats,
11     torch::Tensor points
12 );
13
14
15 torch::Tensor trilinear_bw_cu(
16     torch::Tensor dl_dfeat_interp,
17     torch::Tensor feats,
18     torch::Tensor points
19 );
```

### ④ setup.py



```
$ runall_360_ngp.sh M interpolation.cpp utils.h interpolation_kernel.cu setup_cpp.py setup.py test.py
cuda_notes > setup.py > ...
1 import glob
2 import os.path as osp
3 from setuptools import setup
4 from torch.utils.cpp_extension import CUDAExtension, BuildExtension
5
6 ROOT_DIR = osp.dirname(osp.abspath(__file__))
7 include_dirs = [osp.join(ROOT_DIR, "include")]
8 sources = glob.glob("*.cpp") + glob.glob("*.cu")
9
10
11
12
13 setup(
14     name='cudacuda_tutorial',
15     version='1.0',
16     author='gaost-Sun',
17     author_email='sunhao5871@126.com',
18     description='cuda volume rendering library',
19     long_description='cuda volume rendering library',
20     ext_modules=[
21         CUDAExtension(
22             name='cudacuda_tutorial',
23             sources=sources,
24             include_dirs=include_dirs
25         ),
26     ],
27     cmdclass={
28         'build_ext': BuildExtension
29     }
30 )
```

## ⑤ interpolation.py

```

$ runall_360_ngpah M interpolation.cpp c util.h interpolation_kernel.cu setup.cpp.py setup-py test.py
...
cuda_notes > interpolation > PYBIND11_MODULE(TORCH_EXTENSION_NAME, m)
1 #include <torch/extension.h>
2 #include <include/utils.h>
3
4 torch::Tensor trilinear_interpolation_fw(
5     torch::Tensor feats,
6     torch::Tensor points
7 ) {
8     CHECK_INPUT(feats);
9     CHECK_INPUT(points);
10    return trilinear_fv_cu(feats, points);
11 }
12
13 torch::Tensor trilinear_interpolation_bw(
14     torch::Tensor dl_dfeat_interp,
15     torch::Tensor feats,
16     torch::Tensor points
17 ) {
18     CHECK_INPUT(dl_dfeat_interp);
19     CHECK_INPUT(feats);
20     CHECK_INPUT(points);
21    return trilinear_bw_cu(dl_dfeat_interp, feats, points);
22 }
23
24 PYBIND11_MODULE(TORCH_EXTENSION_NAME, m) {
25     m.def("trilinear_interpolation_fw", &trilinear_interpolation_fw);
26     m.def("trilinear_interpolation_bw", &trilinear_interpolation_bw);
27 }
28

```

## ⑥ interpolation\_kernel.cu

```

$ runall_360_ngpah M interpolation.cpp c util.h interpolation_kernel.cu setup.cpp.py setup-py test.py
...
cuda_notes > interpolation > PYBIND11_MODULE(TORCH_EXTENSION_NAME, m)
1 #include <torch/extension.h>
2
3
4 template <typename scalar_t>
5 __global__ void trilinear_fv_kernel(
6     const torch::PackedTensorAccessor<scalar_t, 3, torch::RestrictPtrTraits, size_t> feats,
7     const torch::PackedTensorAccessor<scalar_t, 3, torch::RestrictPtrTraits, size_t> points,
8     torch::PackedTensorAccessor<scalar_t, 2, torch::RestrictPtrTraits, size_t> rec_weights
9 ) {
10    const int n = blockIdx.x * blockDim.x + threadIdx.x;
11    const int m = blockDim.y * blockDim.z + threadIdx.y;
12
13    if ((m > points.size(0)) || (n > feats.size(0))) return;
14
15    //point [-1,1]
16    const scalar_t u = (points[m][0] + 1) / 2;
17    const scalar_t v = (points[m][1] + 1) / 2;
18    const scalar_t _u = (points[m][0] - 1) / 2;
19    const scalar_t _v = (points[m][1] - 1) / 2;
20
21    const scalar_t t_a = (1 - v);
22    const scalar_t t_b = (1 - u);
23    const scalar_t t_c = v * u;
24    const scalar_t t_d = v * _u;
25
26    float feat_interp[n][t] = (1 - u) * (1 - v) * feats[n][0][t] +
27        (1 - u) * v * feats[n][1][t] +
28        u * (1 - v) * feats[n][2][t] +
29        u * v * feats[n][3][t];
30
31    feat_interp[n][t] += rec_weights[m][t];
32
33 }
34
35
36
37
38 torch::Tensor trilinear_fv_cu(
39     torch::Tensor feats,
40     torch::Tensor points
41 ) {
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59

```

```

$ runall_360_ngpah M interpolation.cpp c util.h interpolation_kernel.cu setup.cpp.py setup-py test.py
...
cuda_notes > interpolation > PYBIND11_MODULE(TORCH_EXTENSION_NAME, m)
1 #include <torch/extension.h>
2
3
4 template <typename scalar_t>
5 __global__ void trilinear_fw_kernel(
6     torch::Tensor feats,
7     torch::Tensor points
8 ) {
9
10    const int N = feats.size(0), F = feats.size(2);
11    torch::Tensor feat_interp = torch::zeros({N, F}, feats.options());
12
13    const dim3 threads(16, 16);
14    const dim3 blocks((N + threads.x - 1) / threads.x, (F + threads.y - 1) / threads.y);
15
16    AT_DISPATCH_FLOATING_TYPES(feats.type(), "trilinear_fv_cu",
17    [&] {
18        trilinear_fv_kernel<scalar_t>(<blocks, threads>>>
19            .feats_packed_accessor<scalar_t, 3, torch::RestrictPtrTraits, size_t>(),
20            .points_packed_accessor<scalar_t, 3, torch::RestrictPtrTraits, size_t>(),
21            .feat_interp_packed_accessor<scalar_t, 2, torch::RestrictPtrTraits, size_t>());
22    });
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59

```

trilinear-fw-kernel

trilinear-fw-cu

## • Loss 计算 (梯度反传):

二次插值函数:  $f = F(f_1, f_2, f_3, f_4) = uvf_4 + (1-u)vf_3 + u(1-v)f_2 + (1-u)(1-v)f_1$

$$\frac{\partial f}{\partial f_1} = (1-u)(1-v) \quad \frac{\partial f}{\partial f_2} = u(1-v) \quad \frac{\partial f}{\partial f_3} = (1-u)v \quad \frac{\partial f}{\partial f_4} = uv$$

trilinear-bw-kernel

trilinear-bw-cu

## ⑦ 測試結果：