



一、Colmap稀疏重建

安装Colmap

1. 下载 Colmap，如果不需要使用稠密重建无需下载带 CUDA 版本的（稍大），这里给一个国内下载链接：

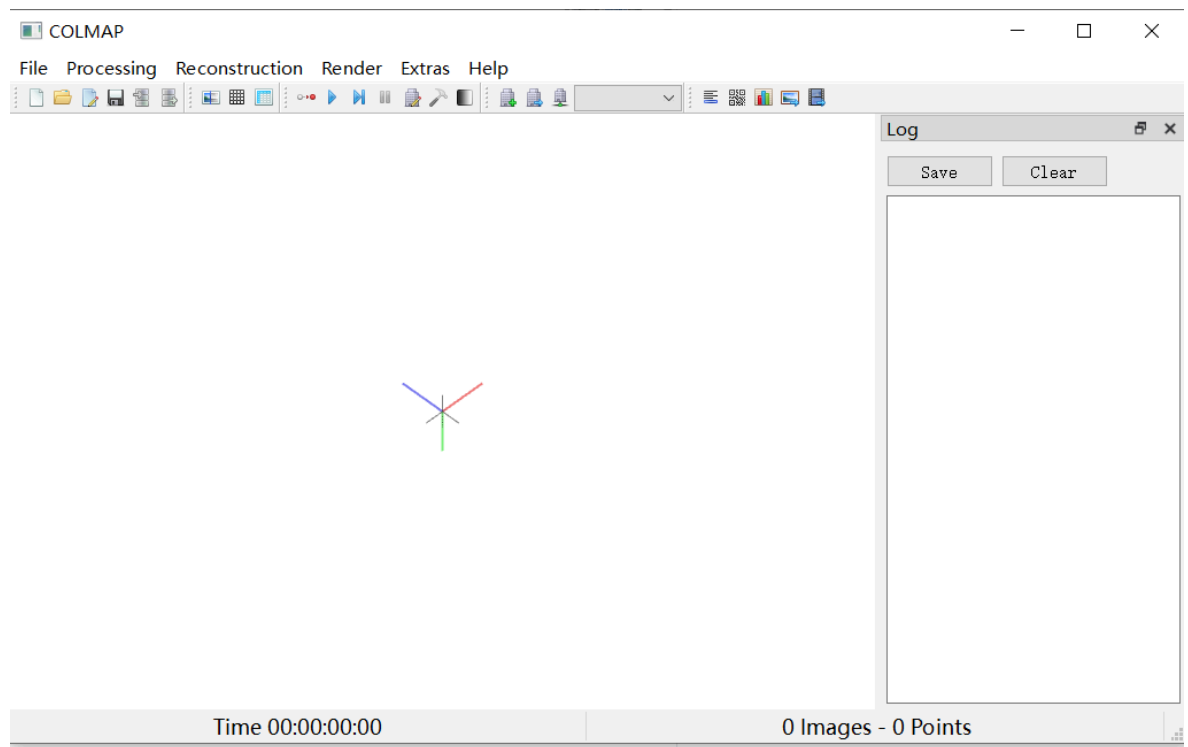
链接：<https://pan.baidu.com/s/1KUbkhDCvjo0ws2DkFON3pQ?pwd=ddnp>

提取码：ddnp

2. 解压后文件夹应是这样的：

名称	修改日期	类型	大小
bin	2022/8/4 16:59	文件夹	
image folder	2022/8/4 17:13	文件夹	
lib	2022/8/4 16:59	文件夹	
workplace	2022/8/4 17:21	文件夹	
COLMAP.bat	2022/1/25 21:11	Windows 批处理...	3 KB
PR_a19256	2022/8/4 18:33	文件	7,168 KB
PR_b19256	2022/8/4 18:33	文件	11,264 KB
RUN_TESTS.bat	2022/1/25 21:13	Windows 批处理...	2 KB
test_from_file_transform.txt	2022/8/4 17:01	文本文档	1 KB

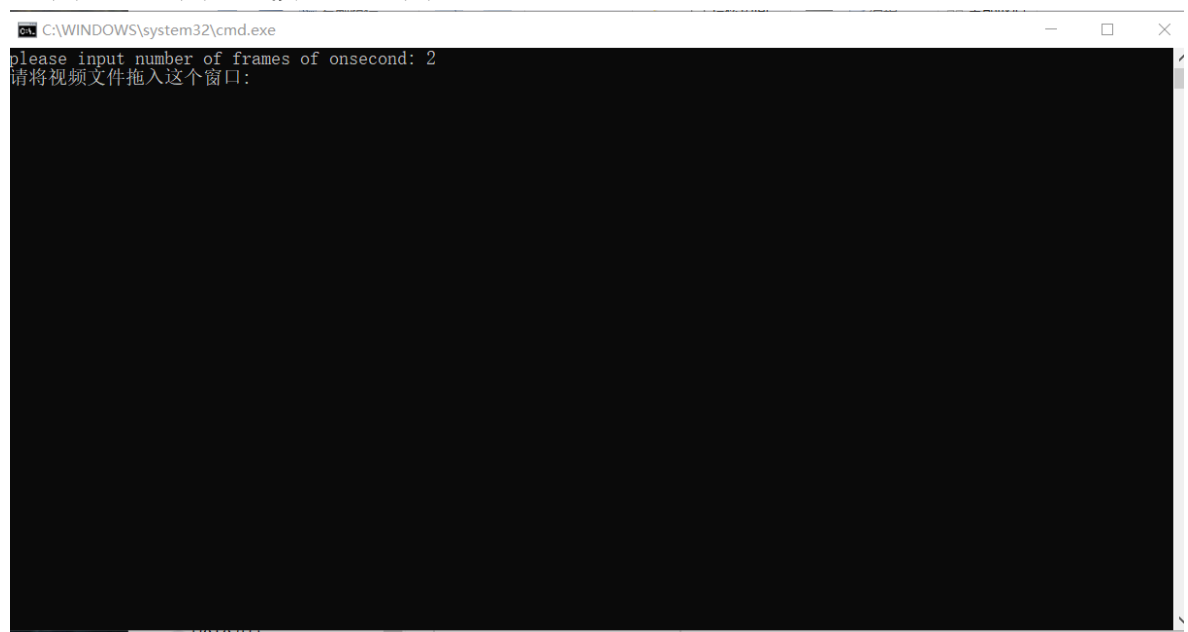
双击 COLMAP.bat 打开命令行，稍等片刻即可见到 colmap 图形界面：



准备你的数据

1. 通过视频抽帧获取图片(可选)：

- 打开视频抽帧脚本，输入每秒要抽帧数量，这里输入2（一般视频每秒有 24/30/60 帧），抽帧脚本可以咸鱼下单或者在qq上找作者要。
- 将视频拖入视频抽帧脚本（视频目录不要包含中文和空格）

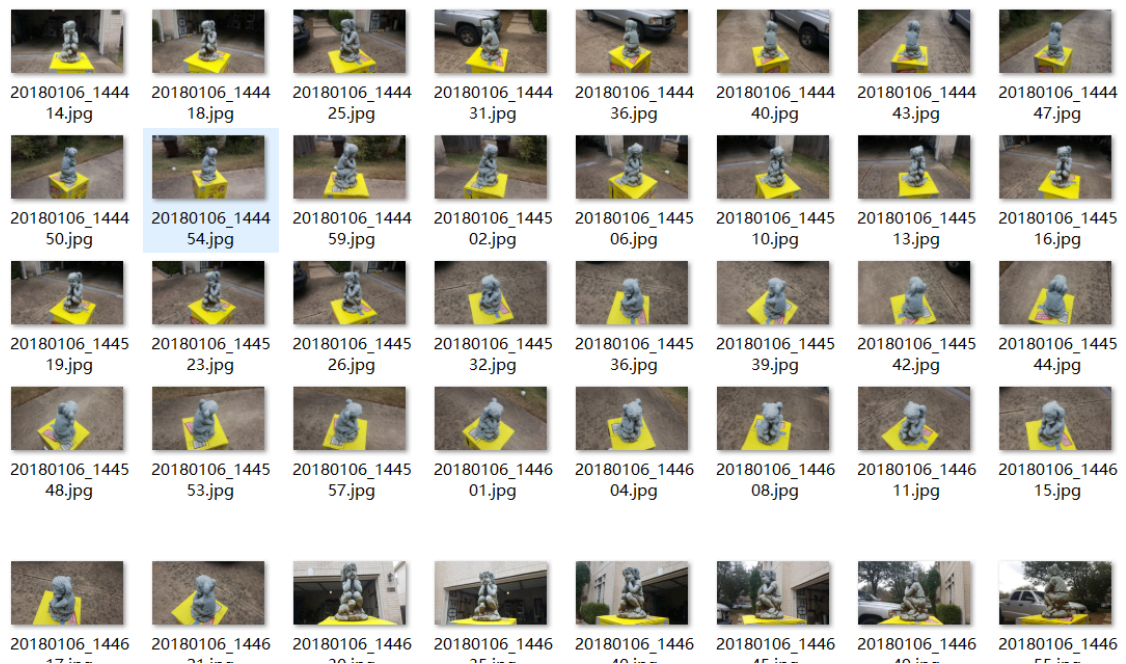


```

C:\WINDOWS\system32\cmd.exe
Stream #0:0 -> #0:0 (hevc (native) -> png (native))
Press [q] to stop, [?] for help
Output #0, image2, to '_frames\frame%03d.png':
Metadata:
  major_brand      : qt
  minor_version    : 0
  compatible_brands: qt
  com.apple.quicktime.creationdate: 2024-03-29T22:39:54+0800
  com.apple.quicktime.location.accuracy.horizontal: 22.708547
  com.apple.quicktime.location.ISO6709: +38.8832+121.5300+032.045/
  com.apple.quicktime.make: Apple
  com.apple.quicktime.model: iPhone 13
  com.apple.quicktime.software: 15.4.1
  encoder          : Lavf61.1.100
Stream #0:0(und): Video: png, rgb48be(pc, gbr/bt2020/arib-std-b67, progressive), 1920x1080, q=2-31, 200 kb/s, 1 fps, 1
tbn (default)
Metadata:
  creation_time     : 2024-03-29T14:39:54.000000Z
  handler_name      : Core Media Video
  vendor_id         : [0][0][0][0]
  encoder           : Lavc61.3.100 png
Side data:
  DOVI configuration record: version: 1.0, profile: 8, level: 4, rpu flag: 1, el flag: 0, bl flag: 1, compatibility
id: 4
Ambient Viewing Environment, ambient_illuminance=314.000000, ambient_light_x=0.312700, ambient_light_y=0.329000
[out#0/image2 @ 0000025e376da000] video:492441KiB audio:0KiB subtitle:0KiB other streams:0KiB global headers:0KiB muxing
overhead: unknown
frame= 56 fps=4.8 q=-0.0 Lsize=N/A time=00:00:56.00 bitrate=N/A speed=4.81x
抽帧完成，帧图像存放在: _frames
Press any key to continue . . .

```

- 生成的照片储存在脚本目录的 `_frames`，新的视频抽帧生成的文件夹会覆盖之前的文件！



2. 通过拍照或者视频抽帧获取的图片要尽量清晰，尤其注意要避免**运动模糊**，模糊的照片会对稀疏重建算法造成较大影响，尽可能挨个检查一下，删除质量较差的照片。

使用Colmap获取相机位姿

使用Colmap生成OpenMVS需要的文件

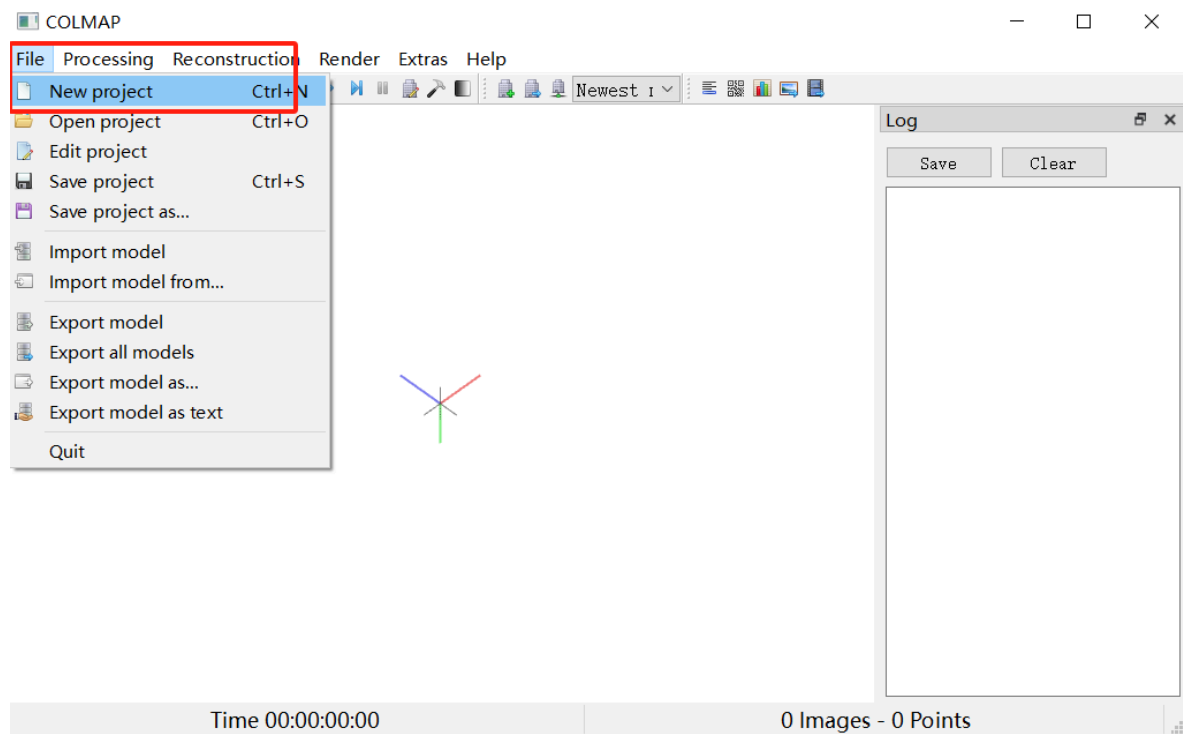
1. 新建文件夹，目录结构：

```
toy/  
  images/  
    toy1.jpg;  
    toy2.jpg;  
    ....  
  sparse/  
    toy.db  
    xxx.txt  
    ...  
    xxx.ini  
    xxx.nvm  
    ...
```

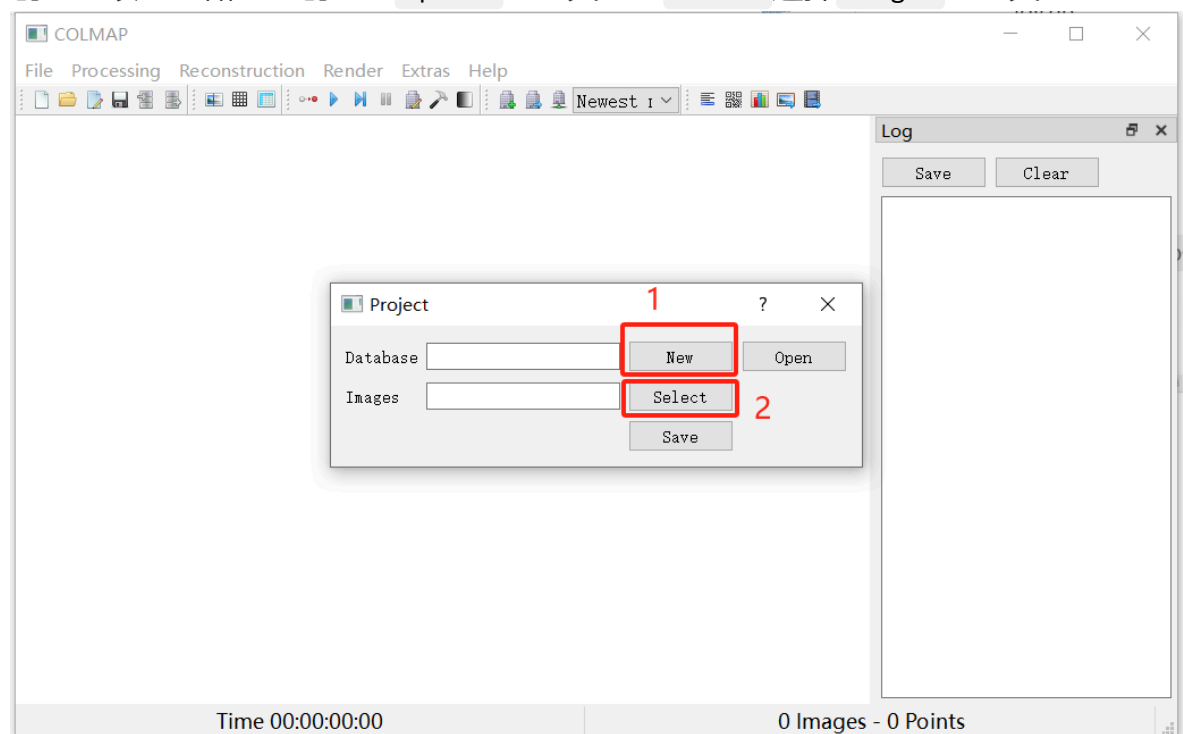
这里的 `toy` 可以自己随意命名,在这里用其举例说明

注意一定要将储存colmap信息的文件夹名命名为 `sparse`

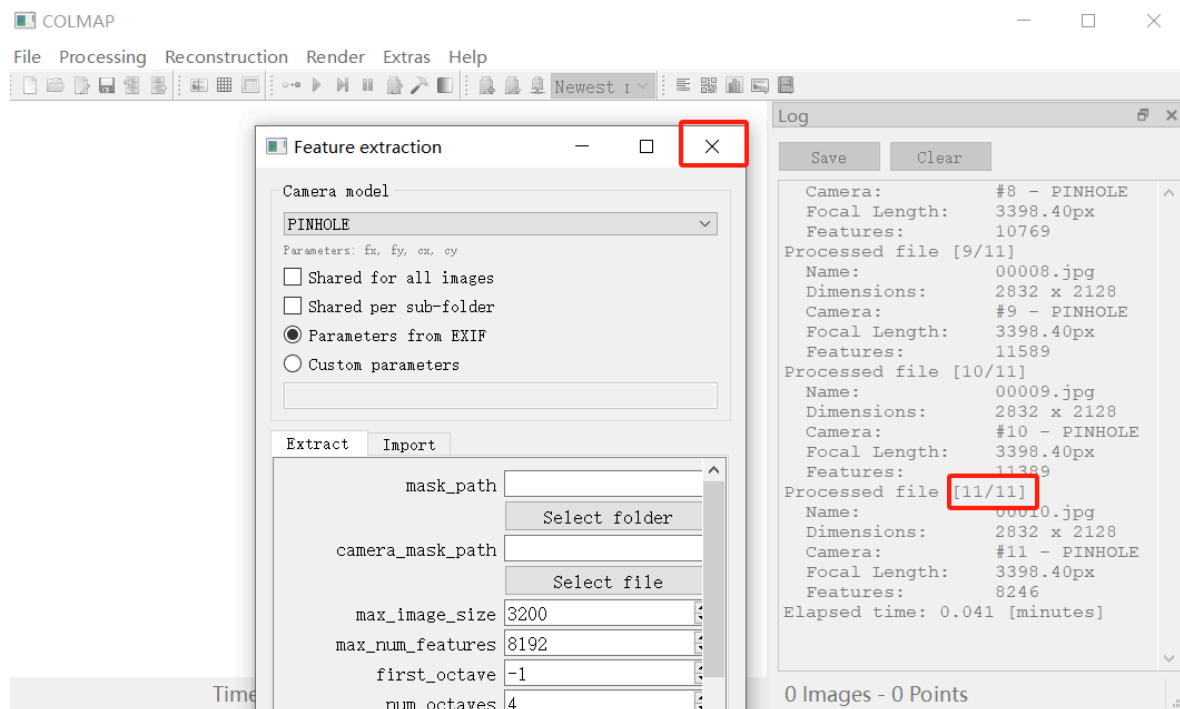
2. 打开 Colmap 图形界面， `File->New project` (如果目录包含中文则会创建不成功)



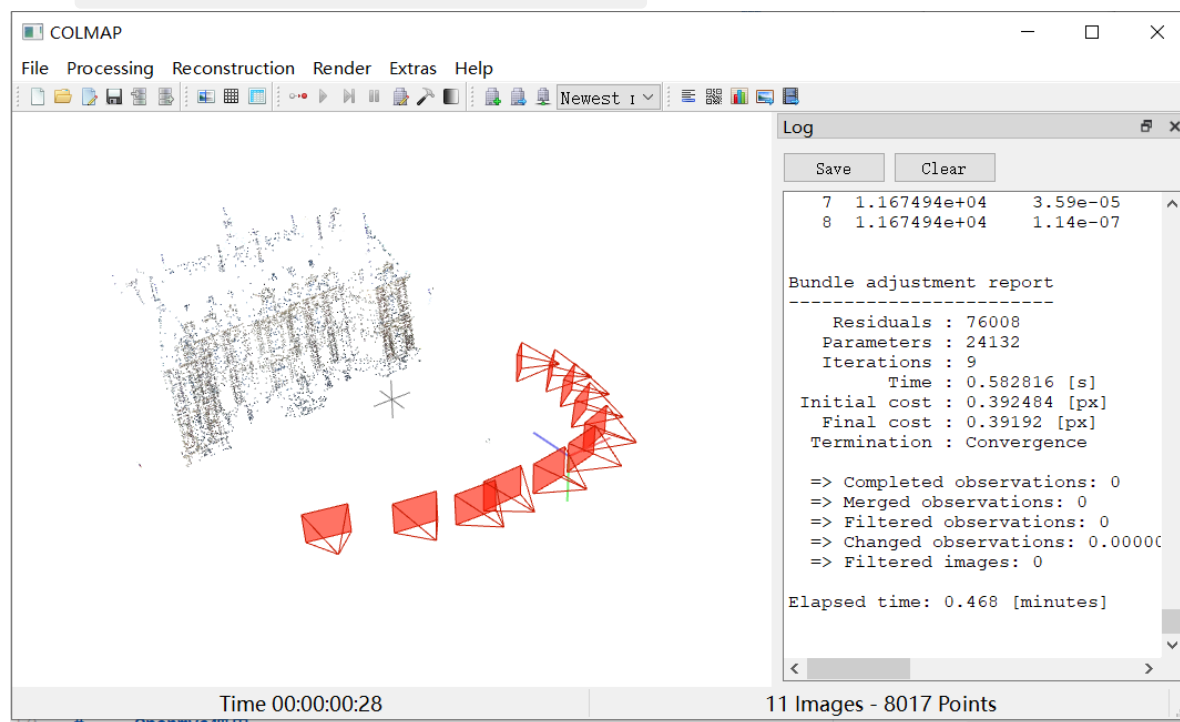
3. 创建新项目，储存在创建的 `sparse` 文件夹下，`Select` 选择 `images` 文件夹



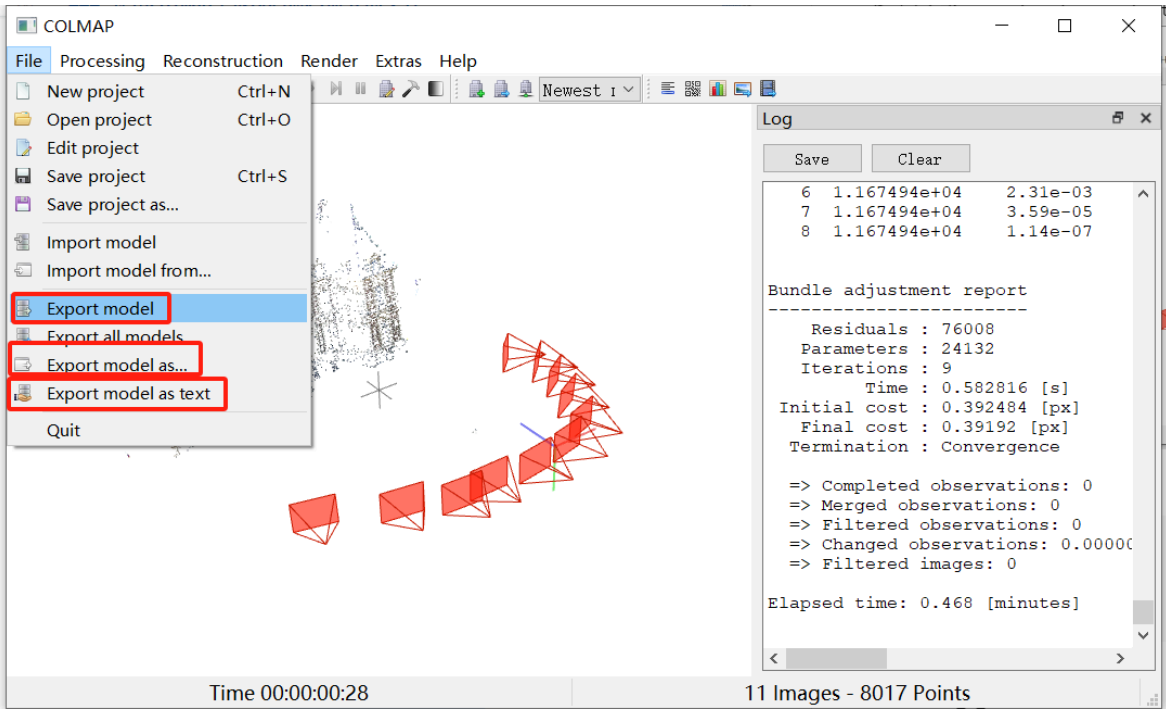
4. 点击 `Processing`->`Feature extraction` 相机模式设置为 `PINHOLE`，其他参数默认。点击 `Extract` 进行提取，完成后点击关闭。



5. 点击 Processing->Feature matching 参数默认直接点击 run，完成后点击关闭
6. 点击 Resconstruction->start reconstruction，等待重建完成。



7. 稀疏重建后导出到 sparse 文件夹内：



Export model
Export model as...
Export model as text

保存后的 sparse 文件夹应当包含以下文件

名称	修改日期	类型	大小
2.db	2024/5/7 23:13	ANSYS v150 .db ...	18,264 KB
cameras.bin	2024/5/7 23:17	BIN 文件	1 KB
images.bin	2024/5/7 23:17	BIN 文件	2,682 KB
points3D.bin	2024/5/7 23:17	BIN 文件	697 KB
nvm.nvm	2024/5/7 23:17	NVM 文件	1,413 KB
project.ini	2024/5/7 23:17	配置设置	5 KB
cameras.txt	2024/5/7 23:17	文本文档	1 KB
images.txt	2024/5/7 23:17	文本文档	4,406 KB
points3D.txt	2024/5/7 23:17	文本文档	1,009 KB

如果你包含这些文件，恭喜你 colmap 任务完成，保存关闭即可。

二、OpenMVS使用

1. 将toy文件夹拖动到 InterfaceCOLMAP.exe 上打开（简单快捷），或者使用命令

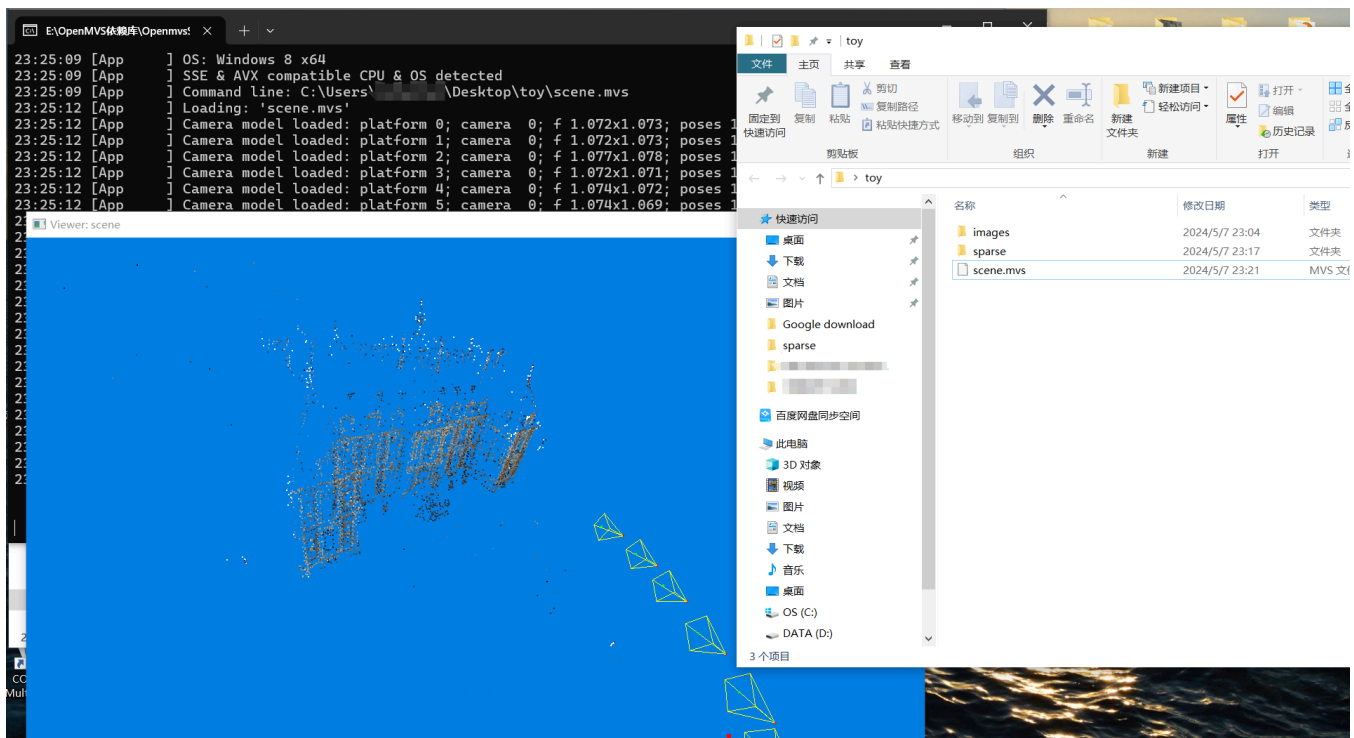
```
InterfaceCOLMAP.exe -i ...你的路径/toy
```

其他使用方法可以查看官网，或者双击 InterfaceCOLMAP.exe 查看命令含义（在文件夹下生成的.log文件内查看）

Generic options:

-h [--help]	produce this help message
-w [--working-folder] arg	working directory (default current directory)
-c [--config-file] arg (=InterfaceCOLMAP.cfg)	file name containing program options
...	

程序运行完毕后会在 toy 文件夹下生成 scene.mvs 文件，可以使用 openmvsSamples 文件夹下的 Viewer.exe 查看（将 .mvs 文件拖动到 exe 文件上）



2. 将生成的 `scene.mvs` 拖动到 `DensifyPointCloud.exe` 生成稠密点云。
3. 完成后将生成的 `scene_dense.mvs` 拖动到 `ReconstructMesh.exe` 上重建网格。
4. 完成后将生成的 `scene_dense_mesh.mvs` 拖动到 `RefineMesh.exe` 上优化网格。
5. 完成后将生成的 `scene_dense_mesh_refine.mvs` 拖动到 `TextureMesh.exe` 上贴图。
6. 生成的文件为 `.ply` 格式，可以使用 `meshlab` 查看。

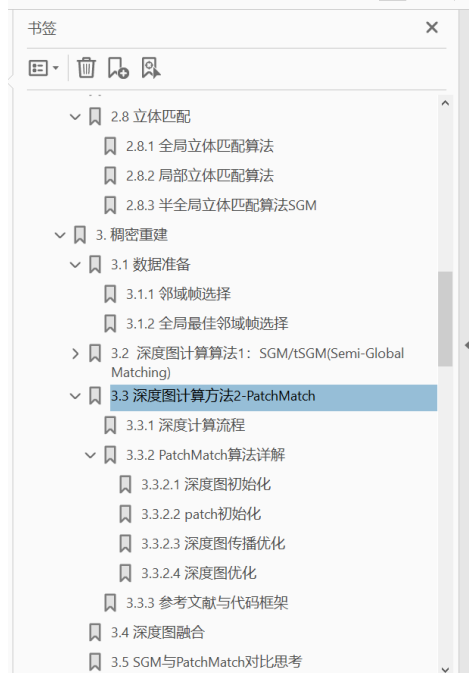
OpenMVS解析文档

联系作者提供OpenMVS原理详解 + 逐行源码解析文档

```

{
    ASSERT(!in_facets.empty());
    static const int facet_vertex_order[] = {2,1,3,2,2,3,0,2,0,3,1,0,0,1,2,0};
    int coplanar[3];
    const REAL prevDist(inter.dist);
    for (const facet_t& in_facet: in_facets) {
        ASSERT(!Tr.is_infinite(in_facet));
        const int nb_coplanar(intersect(Tr.triangle(in_facet), seg, coplanar));
        if (nb_coplanar >= 0) {
            // skip this cell if the intersection is not in the desired direction
            // 计算in_facet所在的平面到点的距离，如果比上个大小则跳过。期望的方向是所穿过的facet距离点的越来越近
            const REAL interDist(inter.ray.IntersectsDist(getFacetPlane(in_facet)));
            if ((interDist > prevDist) != inter.bigger)
                continue;
            // vertices of facet i: j = 4 * i, vertices = facet_vertex_order[j,j+1,j+2] negative orientation
            // 每个facet有三个顶点，顶点顺序是逆时针方向。
            inter.facet = in_facet;
            inter.dist = interDist;
            switch (nb_coplanar) {
            case 0: { // 相交于面上
                // face intersection
                inter.type = intersection_t::FACET;
                // now find next facets to be checked as
                // the three faces in the neighbor cell different than the origin face
                // 在邻域cell中不同于相交的这个face的其它三个face放入out_facets参与下一次求相交计算
                out_facets.clear();
                const cell_handle_t nc(inter.facet.first->neighbor(inter.facet.second));
                ASSERT(!Tr.is_infinite(nc));
                for (int i=0; i<4; ++i)
            }
            }
        }
    }
}

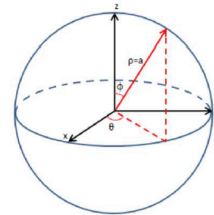
```



法向量 n_i 计算:

$$n_i = \begin{bmatrix} \cos \theta \sin \phi \\ \sin \theta \sin \phi \\ \cos \phi \end{bmatrix} \quad (36)$$

上式中，是用以 C_i 为中心的球坐标系计算。



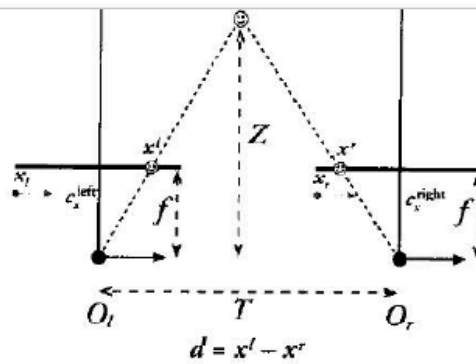
$$m(p, f) = 1 - \rho_p(x, y) \quad (37)$$

$$w(p, q) = e^{-\left(\frac{(p_x - q_x)^2}{\sigma_x^2} + \frac{(p_y - q_y)^2}{\sigma_y^2}\right)} \quad (38)$$

$$\rho_p(x, y) = \frac{\frac{1}{\text{norm}(w)} \sum_{i=1}^m \sum_{j=1}^n w(i, j) (S_{xg}(i, j) - \bar{S}_{xg}) (g(i, j) - \bar{g})}{\sqrt{\frac{1}{\text{norm}(w)} \sum_{i=1}^m \sum_{j=1}^n w(i, j) (S_{xg}(i, j) - \bar{S}_{xg})^2} \sqrt{\frac{1}{\text{norm}(w)} \sum_{i=1}^m \sum_{j=1}^n w(i, j) (g(i, j) - \bar{g})^2}} \quad (39)$$

代价计算时，减少计算量的小技巧推导:

$$H = \begin{bmatrix} H[0] & H[1] & H[2] \\ H[3] & H[4] & H[5] \\ H[6] & H[7] & H[8] \end{bmatrix} \quad (40)$$



利用三角形关系我们很容易推出Z值:

$$\frac{T - (x^l - c_x^{left} + c_x^{right} - x^r)}{Z - f} = \frac{T}{Z} \Rightarrow Z = \frac{fT}{x^l - x^r + c_x^{right} - c_x^{left}} \quad (8)$$

如果主点 c_x^l, c_x^r 坐标相同则可简化为:

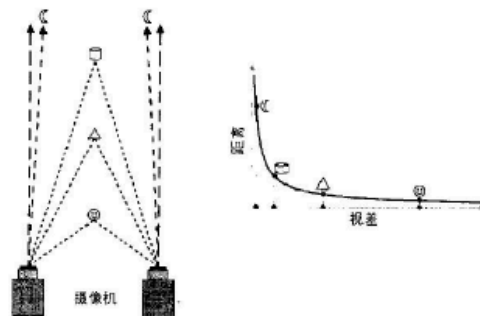
$$\frac{T - (x^l - x^r)}{Z - f} = \frac{T}{Z} \Rightarrow Z = \frac{fT}{x^l - x^r} \quad (9)$$

因为视差 $d = x^l - x^r$,且 Z 为我们的深度值 $depth$,故:

$$depth = \frac{fT}{d} \quad (10)$$

• 视差与深度图关系:

- 视差与深度成反比, 视差接近0时, 微小的视差变化会产生较大的深度变化
- 当视差较大时, 微小的视差变化几乎不会引起深度多大的变化
- 因此, 立体视觉系统仅物体距离相机较近时具有较高的深度精度



• 极线校正

- 校正过程: 将相机在数学上对准到同一观察平面上, 使得相机上像素行是严格对齐的