Guide for Installing Open3D-ML with CUDA Capabilities

System Configuration Used:

- Operating System: Windows 11 (using Windows Subsystem for Linux)
- RAM: 70 GB
- GPU: Nvidia Quadro 4000

Task Overview: The objective was to successfully install and configure Open3D-ML with CUDA-enabled capabilities on a Windows 11 system. The installation process was carried out using Windows Subsystem for Linux (WSL), though it can also be done directly on Ubuntu 20.04.

Key Considerations: While version details of the various tools and dependencies may not be critically important, maintaining a correct and consistent environment is essential due to the complexities encountered during the setup. Ensuring that the dependencies and system configurations are aligned will help avoid potential issues.

Environment Setup: To facilitate a smooth installation process, we created multiple Conda environments, named `op3d_cpu` and `op3d_gpu`. These environments are configured for CPU-based and GPU-backed functionalities, respectively.

Within Windows, using WSL, multiple Conda environments are available. For easy replication of the setup, we have provided a GitHub link that contains both the `.yml` file and `requirements.txt`. These files outline the necessary dependencies for each environment.

Note: Before proceeding, ensure that Conda is installed and running on your system.

Guide to Set Up Conda Environments

1. **Clone the Repository:**

Open your terminal (within WSL or directly in Ubuntu) and clone the repository:

```
git clone https://github.com/AdityaPatel1068/open3d.git
cd open3d
```

2. **You can Install the op3d_cpu Environment:**

   You can create the CPU environment using the `op3d_cpu.yml` file:

```
conda env create -f op3d_cpu.yml
conda activate op3d_cpu
pip install -r requirements.txt
conda deactivate
```

3. **Install the op3d_gpu Environment:**

For the GPU environment, use the `op3d_gpu.yml` file:

```
conda env create -f op3d_gpu.yml
conda activate op3d_gpu
pip install -r requirements.txt
conda deactivate
```

4. **Verify the Installation:**

After setting up both environments, you can check that everything is installed correctly by listing the packages:

```
conda list
```

To use the CPU-based setup:

```
conda activate op3d_cpu
```

To use the GPU-based setup:

```
conda activate op3d_gpu
```

Verifying CUDA Compatibility

To verify that CUDA is properly configured and available in your environment, you can use the following Python code with PyTorch:

```
python3
import torch >>> torch.cuda.is_available() True >>>
torch.cuda.device_count() 1 >>> torch.cuda.current_device() 0 >>>
torch.cuda.device(0) <torch.cuda.device at 0x7efce0b03be0> >>>
torch.cuda.get_device_name(0) 'GeForce GTX 950M'
```

Rendering Issue

If you encounter rendering issues in Open3D, you can solve them by setting the following OpenGL environment variables:

```
export LIBGL_ALWAYS_INDIRECT=0
export MESA_GL_VERSION_OVERRIDE=4.5
export MESA_GLSL_VERSION_OVERRIDE=450
export LIBGL_ALWAYS_SOFTWARE=1
```

Additional Instructions for Managing WSL Distro

- Listing all WSL Distro

  To list all registered WSL distributions, use the following command:

```
wsl --list --all
```

- To unregister (delete) a specific WSL distribution, use:

```
wsl --unregister <DistributionName>
```

- Replace `<DistributionName>` with the name of the distribution you want to unregister.
- For more detailed instructions on moving WSL to another drive, refer to this guide on Medium.

To access a specific WSL distribution (e.g., Ubuntu 20.04 named `U-20.4-op3d`), use:

```
wsl.exe -d U-20.4-op3d
```

If you are logged in as the root user and need to switch to a different user (e.g., `aditya`), use:

```
su aditya
```

Launching Jupyter Notebook

After accessing your WSL environment and switching to the correct user, you can launch Jupyter Notebook with the following command:

```
jupyter notebook --no-browser --port=8889
```

- You can then access Jupyter Notebook via your web browser by navigating to the provided URL (usually something like `http://localhost:8889`).