

Guía para la instalación y el uso de NVIDIA instant-ngp en Ubuntu 18.04

Esta guía ha sido desarrollada para usar NeRF gracias a instant-ngp en Ubuntu 18.04 con una tarjeta gráfica NVIDIA RTX 2060 de 12GB. Fecha de actualización: febrero de 2023.

Esta guía es para la versión de CUDA 11.8, la cual a fecha de redacción de la misma es la última compatible con todos los programas necesarios para el uso de instant-ngp con y la creación de dataset propios. En el caso de requerir bajar de una versión más reciente a una anterior usar la guía correspondiente para ello.

A fecha de hoy la última versión de CUDA, la 12.0 no es compatible con Colmap, pero sí con NeRF, así que si no se van a crear dataset propios sino usar unos ya existentes se puede usar dicha versión.

Capítulos de este manual:

- 1. CONFIGURACIÓN PREVIA**
- 2. DESCARGA Y COMPILACIÓN DE instant-ngp**
- 3. CREACIÓN Y USO DE UN DATASET PROPIO**
- 4. LISTA DE COMANDOS ÚTILES**
- ANEXO. RECOPIACIÓN DE POSIBLES FALLOS**

1. CONFIGURACIÓN PREVIA

Antes de empezar con NeRF hay que realizar la configuración del equipo indicada tanto en la página de gitHub: <https://github.com/NVlabs/instant-ngp>

Como en la página del tutorial de NVIDIA:

<https://developer.nvidia.com/blog/getting-started-with-nvidia-instant-nerfs/>

Doc extra → Paper: <https://nvlabs.github.io/instant-ngp/>

Requiere de los siguientes pasos previos (algunos opcionales):

- 1.1. Instalación de Visual Studio 2019 o GCC/G++ 7.5+**
- 1.2. Instalación de CUDA ToolKit 10.2 o superior**
- 1.3. Instalación de Python3.9**
- 1.4. Instalación de CMake 3.22+**
- 1.5. OptiX 7.3 o superior (opcional)**
- 1.6. Vulkan SDK (opcional)**
- 1.7. Librerías extra (opcional)**
- 1.8. Añadir CUDA a los PATH**
- 1.9. Instalación NVIDIA DLSS SDK**

1.1. Instalación de Visual Studio 2019 o GCC/G++ 7.5 o superior

→ Si usas Windows: Instala Visual Studio 2019:

<https://visualstudio.microsoft.com/es/vs/older-downloads/>

Por ejemplo, en mi caso está disponible Visual Studio Professional 2019 (version 16.11) con mi cuenta de Microsoft de la UPM.

→ Si usas Linux: Instala GCC/G++ en la versión 7.5 o superior. En Ubuntu 18.04 se recomienda la 8 por compatibilidad con la librería EGL:

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
sudo apt-get install gcc-8 g++-8
gcc-8 --version
```

Para tener gcc redireccionado automáticamente a la versión 8:

```
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-7 700 --slave /usr/bin/g++ g++ /usr/bin/g++-7
(el comando anterior es una única línea)
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-8 800 --slave /usr/bin/g++ g++ /usr/bin/g++-8
(el comando anterior es una única línea)
```

Después para chequear la versión por defecto usar:

```
sudo update-alternatives --config gcc
```

1.2. Instalación CUDA Toolkit (instalación laboriosa)

Dan problemas, sobre todo: MLNX_OFED y el comando: “`sudo apt-get -y install cuda`”

Nota: antes de la instalación yo ya tenía los drivers de NVIDIA para mi tarjeta GTX 1060 instalados y actualizados a la última versión. Esto es así porque los drivers nouveau que vienen por defecto con la instalación de linux, antes de intentar esta instalación, ya los había borrado por un bug de loop en el inicio de sesión de Ubuntu 18.04 tras la instalación del sistema operativo. Si en tu caso sigues teniendo los drivers nouveau que vienen por defecto he leído que dan problemas durante la instalación. Existen dos soluciones previas a la instalación: o instalas los drivers correspondientes a tu tarjeta o apagas la tarjeta y después de la instalación desactivas los nouveau drivers, apagas el ordenador, vuelves a conectar la tarjeta de vídeo y enciendes el ordenador de nuevo. (fuente: <https://forums.fast.ai/t/ubuntu-18-04-05-gtx-1060-6gb-cuda-11-2-clean-install-guide/84861>, step1, optional step -entre step6 y step7- y step7)

En esta guía sólo se explica como seguir los pasos que hacen falta para llevar a cabo la instalación en un ordenador con las características especificadas al principio del documento, incluyendo las acciones recomendadas previas y posteriores que aplican a las instalaciones de los componentes. Para mayor seguridad se recomienda seguir esta guía en paralelo con las fuentes utilizadas:

- <https://docs.nvidia.com/cuda/cuda-installation-guide-linux/>
- <https://forums.fast.ai/t/ubuntu-18-04-05-gtx-1060-6gb-cuda-11-2-clean-install-guide/84861>

NVIDIA CUDA Installation Guide for Linux: Se sigue dicha guía de instalación (IMPORTANTE no saltarse los pre-installation y post-installation steps): <https://docs.nvidia.com/cuda/cuda-installation-guide-linux/>

1.2.1. Pre-installation Actions:

1.2.1.1. Verify You Have a CUDA-Capable GPU:

Chequear que la tarjeta gráfica figura en la lista de GPUs compatibles: <https://developer.nvidia.com/cuda-gpus>

1.2.1.2. Verify You Have a Supported Version of Linux:

Ejecutar

```
uname -m && cat /etc/*release
```

y

```
uname -r
```

Y comprobar que la versión de Linux aparece en la tabla *Table 1. Native Linux Distribution Support in CUDA 11.8* y que la versión del kernel es superior a la mínima necesaria.

1.2.1.3. Verify the System Has gcc Installed:

Ver apartado 1.1.

1.2.1.4. Verify the System has the Correct Kernel Headers and Development Packages Installed:

Instalar los headers y paquetes de desarrollo correspondientes al kernel actual:

```
sudo apt-get install linux-headers-$(uname -r)
```

1.2.1.5. MLNX_OFED Requirements and Installation:

Según el manual los requisitos e instrucciones se pueden encontrar aquí:

<https://docs.nvidia.com/gpudirect-storage/troubleshooting-guide/index.html#mofed-req-install>

Descargar el archivo seleccionando tu sistema operativo en la última versión de MLNX con LTS disponible en siguiente enlace (seleccionar la versión .tgz):

https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/

MLNX_OFED se descarga sólo cuando aceptas los consentimientos al final de la página de descarga.

Descomprimir el .tgz e instalar siguiendo la guía que aparece donde el link del enlace de descarga para la versión que hemos seleccionado (aparece en la misma columna, pero más abajo). Ahí vamos al apartado de instalación y tenemos cuidado de leer todos los recuadros amarillos antes de lanzar ningún comando.

IMPORTANTE: se puede instalar lanzando el comando que aparece en Installation Script. Pero si en vez de eso se requiere una instalación manual entonces ir al apartado Installation Procedure, allí tener cuidado e ir al párrafo en amarillo donde dice: “MLNX_OFED for Ubuntu should be installed with the following flags in chroot environment” e instalar con los flags que ahí especifica, el comando tiene una pinta parecida a esta:

```
sudo ./mlnxofedinstall --without-dkms --add-kernel-support --kernel <kernel version in chroot> --without-fw-update --force
```

(el comando anterior es una única línea)

To obtain current kernel version in chroot type on the command shell:

```
uname -r
```

For example:

```
sudo ./mlnxofedinstall --without-dkms --add-kernel-support --kernel 5.4.0-135-generic --without-fw-update --force
```

(el comando anterior es una única línea)

Note that the path to kernel sources (`--kernel-sources`) should be added if the sources are not in their default location.

1.2.2. Installation:

1.2.2.1. CUDA Toolkit 11.8 Downloads:

Usar el enlace: <https://developer.nvidia.com/cuda-11-8-0-download-archive> y seleccionar la versión correspondiente. Esto mostrará las instrucciones que para Ubuntu 18.04 son las siguientes:

https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/cuda-ubuntu1804.pin

`wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/cuda-ubuntu1804.pin` (el comando anterior es una única línea)

```
sudo mv cuda-ubuntu1804.pin /etc/apt/preferences.d/cuda-repository-pin-600
```

`wget https://developer.download.nvidia.com/compute/cuda/11.8.0/local_installers/cuda-repo-ubuntu1804-11-8-local_11.8.0-520.61.05-1_amd64.deb` (el comando anterior es una única línea)

```
sudo dpkg -i cuda-repo-ubuntu1804-11-8-local_11.8.0-520.61.05-1_amd64.deb
```

```
sudo cp /var/cuda-repo-ubuntu1804-11-8-local/cuda-*-keyring.gpg /usr/share/keyrings/
```

(Donde * se sustituye por lo aportado en el comando inmediatamente anterior)

```
sudo apt-get update
```

```
sudo apt-get -y install cuda
```

El comando `install` toma su tiempo.

(Referencia:

[https://developer.nvidia.com/cuda-11-8-0-download-](https://developer.nvidia.com/cuda-11-8-0-download-archive?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=11.8.0&target_type=deb_local)

[archive?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=11.8.0&target_type=deb_local](https://developer.nvidia.com/cuda-11-8-0-download-archive?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=11.8.0&target_type=deb_local))

Si al instalar CUDA Toolkit (última línea: “`sudo apt-get -y install cuda`”) sale el mensaje:

The following packages have unmet dependencies:

cuda : Depends: cuda-11-8 (>= 11.8.0) but it is not going to be installed

E: Unable to correct problems, you have held broken packages.

Ejecutar:

```
sudo apt clean
```

```
sudo apt update
```

```
sudo apt purge nvidia-*
```

```
sudo apt autoremove
```

```
sudo apt install -y cuda
```

Probablemente se cambien los drivers de NVIDIA, yo tenía instalados para GTX 1060 la versión de los drivers número 367 y al instalar CUDA ahora tengo la versión 520.

Después de la instalación puede desconfigurarse ROS. Esto se debe a que algunos de los paquetes instalados pueden haber necesitado instalar o actualizar CMake del cual depende ROS. En tal caso aparecerá este mensaje al abrir la terminal:

```
bash: /opt/ros/melodic/setup.bash: No existe el archivo o el directorio
```

Cuando se finalicen todos los pasos de esta guía se puede volver a instalar ROS sin problema. No se recomienda hacerlo en este momento salvo que sea necesario ya que en pasos posteriores también puede desconfigurarse y/o desinstalarse de nuevo.

1.2.3. Post-installation Actions:

1.2.3.1. Environment Setup:

Ojo, si no quieres **actualizar** los PATH cada vez que abras un terminal nuevo, hay que modificar de forma permanente **PATH** y **LD_LIBRARY_PATH**, no sólo hacer los export como indica en la documentación.

Para PATH: usando “`sudo nano /etc/environment`” añadir “`:/usr/local/cuda-<version>/bin`” al final del PATH (donde <version> es la versión de CUDA instalada)

Si hay alguna duda sobre la ruta de instalación de Cuda lanzar:

```
which nvcc
```

Y de ahí se obtiene la ruta hasta cuda-11.8 (se borra la parte de “/bin/nvcc”)

Para LD_LIBRARY_PATH: añadir el export completo al final del archivo .bashrc que se encuentra en la carpeta \home (ojo esto produce cambios sólo a nivel de usuario, no de sistema):

```
export LD_LIBRARY_PATH=/usr/local/cuda-11.8/lib64\
${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

(el comando anterior es una única línea)

También se puede añadir el PATH en .bashrc, pero es necesario añadirlo al archivo environment en cualquier caso:

```
export PATH="/usr/local/cuda-11.8/bin:$PATH"
```

1.2.3.2. POWER9 Setup:

Saltarse el apartado “13.1.2. POWER9 Setup”. Este sólo vale para este tipo de superordenadores IBM Newell POWER9 basados en la nube, si no sabes lo que es un POWER9 es que no lo estás usando.

De todas formas, más información sobre qué hacen los comandos en la parte de disable udev rule, que no queda claro en la documentación, leer step10): <https://forums.fast.ai/t/ubuntu-18-04-05-gtx-1060-6gb-cuda-11-2-clean-install-guide/84861>

+info adicional sobre udev rules (no es necesario mirarse esto, pero sirve como material de consulta): <https://linuxconfig.org/tutorial-on-how-to-write-basic-udev-rules-in-linux>

1.2.3.3. Install Persistence Daemon:

Para evitar problemas de persistencia de estado de los drivers entre diferentes usos de CUDA hay que lanzar este comando con cada inicio de sesión:

```
/usr/bin/nvidia-persistenced --verbose
```

Para automatizar el proceso añadir a /etc/rc.local. Si se trata de una instalación de Ubuntu reciente, como es mi caso, es posible que no exista, entonces hay que crear el archivo:

```
sudo touch /etc/rc.local
```

```
sudo vim /etc/rc.local
```

y añadir:

```
/usr/bin/nvidia-persistenced --verbose
```

Si no se dispone del editor Vim instalado hacer previamente:

```
sudo apt install vim
```

Dentro del editor pulsar la tecla “insert” para añadir texto. Cuando haya finalizado pulsar la tecla “esc” y escribir “:wq” y a continuación pulsar “enter” para guardar y salir.

En este punto se recomienda reiniciar el ordenador para poder verificar la correcta instalación de los componentes.

Verificar la instalación:

```
cat /proc/driver/nvidia/version
```

Se obtiene la versión de los drivers de nvidia.

Para conocer la versión actual de CUDA e información más detallada usar:

```
nvidia-smi
```

Para comprobar que todo funciona correctamente se recomienda la descarga y ejecución de los samples de CUDA.

Descarga desde el repo git <https://github.com/nvidia/cuda-samples> :

```
git clone https://github.com/NVIDIA/cuda-samples.git
```

Se pueden descargar en cualquier sitio, pero se recomienda que sea dentro de una carpeta llamada cuda-samples

1.2.3.4 Instalación de librerías:

Para que funcionen hay que instalar todas las librerías que hay en la lista. Para la versión 11.8 de CUDA son los siguientes:

1.2.3.4.1. FreeImage:

```
sudo apt-get install libfreeimage3 libfreeimage-dev
```

1.2.3.4.2. Message Passing Interface (MPI):

CAUTION: INSTALL ONLY IF THERE IS NO VERSION OF OpenMPI ALREADY INSTALLED ON THE COMPUTER

¿Cómo comprobar si hay una versión ya instalada?: lanzar el comando “`mpirun`” en el terminal. Si sale un mensaje del tipo:

```
-----  
mpirun could not find anything to do.
```

```
It is possible that you forgot to specify how many processes to run  
via the "-np" argument.  
-----
```

Es que está instalado.

Puede que esté instalado pero los PATH no estén actualizados: puede que la carpeta se encuentre en `home/.openmpi` o como en mi caso en `/usr/mpi/gcc/openmpi-4.1.5a1/`

añadir a `.bashrc`:

```
export PATH="$PATH:/usr/mpi/gcc/openmpi-4.1.5a1/bin"  
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/mpi/gcc/openmpi-4.1.5a1/lib"
```

Y ya después probar “`mpirun`”.

Si no está instalado entonces seguir los siguientes pasos y luego actualizar los PATH

```
sudo apt-get install gcc  
sudo apt-get install openmpi-bin openmpi-common libopenmpi-dev libgtk2.0-dev  
(el comando anterior es una única línea)  
sudo wget https://www.open-mpi.org/software/ompi/v4.1/downloads/openmpi-4.1.4.tar.gz  
(el comando anterior es una única línea)  
sudo tar xvfz openmpi-4.1.4.tar.gz  
cd openmpi-4.1.4/  
  
sudo ./configure --prefix="/home/mpiuser/.openmpi"  
sudo make  
sudo make install
```

Nota: con la doc oficial me hice un jaleo: <https://www.open-mpi.org/> (instrucciones en <https://www.open-mpi.org/faq/?category=building>)

1.2.3.4.3. DirectX:

DirectX es para Windows

1.2.3.4.4. DirectX12:

Ídem que DirectX

1.2.3.4.5. OpenGL:

Si el sistema es compatible se debería instalar automáticamente junto con CUDA. Para chequear la versión:

```
glxinfo | grep -i opengl
```

Si por lo que fuera no se ha instalado, hacer:

```
sudo apt install mesa-utils
```

1.2.3.4.6. OpenGL ES:

Se instala junto a OpenGL

Para chequear la versión:

```
glxinfo | grep -i opengl
```

1.2.3.4.7. Vulkan SDK:

En los sistemas que admiten Vulkan, la implementación de Vulkan de NVIDIA se instala automáticamente con el controlador CUDA. Pero para crear y ejecutar aplicaciones Vulkan, es necesario instalar Vulkan SDK.

Check Vulkan version: `vulkaninfo`, pero para que nos devuelva la información que buscamos hace falta instalar: `sudo apt install vulkan-utils`. Según la versión puede ser `sudo apt install vulkan-tools`

Instalacion Vulkan SDK: <https://vulkan.lunarg.com/sdk/home#linux> (ir a Linux, pestaña de Ubuntu Packages y hacer click en documentation section). En la documentación para Ubuntu se especifican las instrucciones para instalar correctamente VulkanSDK, en el caso de Ubuntu 18.04 son de la forma:

```
wget -qO - https://packages.lunarg.com/lunarg-signing-key-pub.asc | sudo apt-key add -  
(el comando anterior es una única línea)  
sudo wget -qO /etc/apt/sources.list.d/lunarg-vulkan-1.3.224-bionic.list  
https://packages.lunarg.com/vulkan/1.3.224/lunarg-vulkan-1.3.224-bionic.list  
(el comando anterior es una única línea)  
sudo apt update  
sudo apt install vulkan-sdk
```

1.2.3.4.8. OpenMP:

OpenMP es una API para la programación de multiprocesamiento. OpenMP se puede instalar utilizando el sistema de administración de paquetes de distribución de Linux. Suele venir preinstalado con GCC. Para chequear si hay una versión de OpenMP instalada se puede usar:

```
gcc --version
```

Y buscar si viene instalado con la versión de GCC en <http://gcc.gnu.org/wiki/openmp>. O usar:

```
echo | cpp -fopenmp -dM | grep -i open
```

Que devuelve el año y el mes de la versión, si está instalada claro.

1.2.3.4.9. Screen:

Solo para QNX operating system, para software embebido.

1.2.3.4.10. X11:

X11 se puede instalar usando el administrador de paquetes de su distribución de Linux. Comprobar si está instalada:

```
dpkg -l | grep xorg
```

ó

```
echo $XDG_SESSION_TYPE
```

1.2.3.4.11. EGL:

Para la compatibilidad con Mesa EGL hace falta tener gcc8 o superior instalado (ver apartado 1.1).

También tener Clang 5.0 o superior:

```
sudo apt install clang
```

También Python 3.5 o superior (para NeRF hará falta 3.9). Comprobar la versión instalada con:

```
python3 -V
```

(al ejecutar python -V sale la versión a la que apunta el sistema, aunque haya otras instaladas)

Si la versión de Python3 es insuficiente para este paso, ir al apartado 1.3. Instalación de Python 3.9.

También Python Mako module:

```
sudo pip install Mako
```

Comprobar que se dispone de la última versión de pip:

```
sudo python -m pip install --upgrade pip
```

Si no se dispone de pip instalar:

```
sudo apt install python3-pip
```

Para evitar errores con las versiones instaladas por defecto en Ubuntu se recomienda instalar la última versión de pip descargando el ejecutable de la página de pip: <https://bootstrap.pypa.io/get-pip.py> (El enlace actualizado aparece en la siguiente documentación <https://pip.pypa.io/en/stable/installation/>)

Abrir un terminal en la carpeta donde se descargó y hacer:

```
sudo python get-pip.py
```

También Lex and Yacc:

```
sudo apt-get install bison flex
```

Una vez hecho lo anterior lanzar la siguiente línea de comandos:

```
sudo apt-get install freeglut3-dev build-essential libx11-dev libxmu-dev \ libxi-dev mesa-utils libgl1-mesa-glx  
libegl1-mesa libglu1-mesa \ libglu1-mesa-dev libglfw3-dev libgles2-mesa-dev libwayland-dev libxrandr-dev libxt-dev  
(el comando anterior es una única línea)
```

1.2.3.4.12. EGLOutput:

Debería haberse instalado automáticamente junto con EGL

1.2.3.4.13. EGLSync:

Debería haberse instalado automáticamente junto con EGL

1.2.3.4.14. NVSCI:

El paquete NvStreams que contiene esta librería es un producto de NVIDIA Automotive SW y parece no estar disponible por el momento. Cuando se lance la compilación de las demos, las que dependan de este paquete darán error, así que en la carpeta cuda-samples buscamos el Makefile y editamos la línea 40 (donde está el FILTER_OUT) para que quede tal que así:

```
FILTER_OUT := 0_Simple/cudaNvSci/Makefile
```

1.2.3.4.15. NvMedia:

Relacionado con NVSCI y NvStreams. No es relevante que funcione dado que las samples que prueban estas funcionalidades no deberían interesarnos.

1.2.3.5. Comprobación de la instalación de CUDA ToolKit – Lanzamiento de los cuda-samples:

Como comentaba antes de la instalación de las librerías estamos siguiendo un poco el tutorial <https://forums.fast.ai/t/ubuntu-18-04-05-gtx-1060-6gb-cuda-11-2-clean-install-guide/84861>, continuamos por la parte de después de la instalación de las librerías.

Así pues, una vez instaladas todas las librerías correspondientes ya se puede compilar dentro de la carpeta cuda-samples (donde se encuentra el Makefile). Para compilar las samples y probar que CUDA ha sido instalado y funciona correctamente:

```
make -k
```

“-k flag is needed to continue even after some errors. Nvidia guys says that errors are completely normal thing right now and they are there because something wrong with samples. You can carry on without worrying about those. (as of january 2021)”

Aunque se termine la compilación con un mensaje de que no se ha podido hacer build de all no tiene por qué significar que la compilación no ha sido correcta, precisamente por el disclaimer del párrafo anterior.

Dentro del directorio de las Samples entrar a 1_Uilities y entrar a deviceQuery. Entonces lanzar y comprobar que el resultado es PASS:

```
./deviceQuery
```

Dentro del directorio de las Samples entrar a 1_Uilities y entrar a bandwidthTest. Entonces lanzar y comprobar que el resultado es PASS:

```
./bandwidthTest
```

1.3. Instalación/actualización de Python3 a la versión 3.9

Comprobar si hay alguna versión de python3 instalada:

```
python3 -V
```

Añadir el repositorio y update:

```
sudo add-apt-repository ppa:deadsnakes/ppa  
(aceptar la adición del repositorio)  
sudo apt-get update
```

Comprobar que se está disponible el paquete de python3.9

```
apt list | grep python3.9
```

En realidad, como se muestra en la siguiente imagen al hacer el paso de añadir el repositorio ya nos indicó qué versiones están disponibles. Como se puede ver, para Ubuntu 18.04 están disponibles de la 3.7 a la 3.11:

```
Supported Ubuntu and Python Versions  
=====
```

- Ubuntu 18.04 (bionic)	Python2.3 - Python 2.6, Python 3.1 - Python 3.5, Python3.7 - Python3.11
- Ubuntu 20.04 (focal)	Python3.5 - Python3.7, Python3.9 - Python3.11
- Ubuntu 22.04 (jammy)	Python3.7 - Python3.9, Python3.11

- Note: Python2.7 (all), Python 3.6 (bionic), Python 3.8 (focal), Python 3.10 (jammy) are not provided by deadsnakes as upstream ubuntu provides those packages.

Instalarlo:

```
sudo apt-get install python3.9
```

Edit: mejor no hacer estos pasos:

Añadir Python y las versiones anteriormente instaladas a las alternativas de instalación (ordenar de 1 a n las versiones de menor a mayor):

```
sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.x 1  
(el comando anterior es una única línea)  
sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.y 2  
...  
(el comando anterior es una única línea)  
sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.9 n  
(el comando anterior es una única línea)
```

OJO: EL TERMINAL DEJA DE FUNCIONAR SI SE APUNTA A UNA VERSIÓN DE PYTHON3 DIFERENTE DE LA PRESENTE POR DEFECTO EN EL SISTEMA

CONCLUSIÓN ¡¡NO EJECUTAR EL PASO SIGUIENTE!!

Actualizar python3 para apuntar a la versión 3.9:

```
sudo update-alternatives --config python3
```

Es más igual nos podemos ahorrar toda la parte de la configuración de las update-alternatives

Se mostrará una tabla similar a la siguiente captura. Seleccionar la versión correspondiente a la 3.9, en mi caso la 2:

```
robcib@robcib-B250-Gaming-Infinite-A-MS-B915:~$ sudo update-alternatives --config python3
Existen 2 opciones para la alternativa python3 (que provee /usr/bin/python3).

  Selección  Ruta                Prioridad  Estado
-----
* 0          /usr/bin/python3.11  2          modo automático
 1          /usr/bin/python3.11  2          modo manual
 2          /usr/bin/python3.9   1          modo manual

Pulse <Intro> para mantener el valor por omisión [*] o pulse un número de selección: 2
```

Comprobar que se ha actualizado bien:

```
python3 -V
```

Debería devolver

```
Python 3.9.x
```

1.4. Instalación de CMake 3.22 o superior

Es probable que en algún momento de los 3 pasos anteriores previos a la instalación de NVIDIA Instant NeRFs se haya desinstalado Cmake produciendo la desinstalación de paquetes que dependan de este último, como es el caso de ROS, así que posteriormente a finalizar todos los pasos habrá que reinstalar esos paquetes.

Para instalar CMake 3.22 o superior es necesario disponer de la librería OpenSSL:

```
sudo apt-get install libssl-dev
```

Comprobar versión actual de CMake:

```
cmake --version
```

También se puede usar:

```
apt-cache policy cmake
```

ó:

```
dpkg --get-selections | grep cmake
```

Chequear cuál es la última versión de CMake disponible para nuestro sistema operativo haciendo:

```
sudo snap info cmake
```

En mi caso la última versión disponible de CMake 3.22 es la 3.22.6 .

Si se da la opción, se puede instalar directamente lanzando el comando (si no o si diera problemas a la hora de compilar instant-ngp o colmap en pasos posteriores de esta guía, mirar unos párrafos más adelante):

```
sudo snap install cmake
```

Es posible que se requiera de la flag adicional:

```
--classic
```

Si no se puede instalar directamente con el comando de snap o la instalación con snap no fuera compatible, lanzar uno por uno esperando a que termine cada paso:

```
sudo wget https://github.com/Kitware/CMake/releases/download/v3.22.6/cmake-3.22.6.tar.gz
```

(el comando anterior es una única línea)

```
sudo tar -zxvf cmake-3.22.6.tar.gz
```

```
cd cmake-3.22.6
```

```
sudo ./bootstrap
```

```
sudo make -j8
```

Referencia:

https://gist.github.com/bmegli/4049b7394f9cfa016c24ed67e5041930?permalink_comment_id=3965623

En mi caso he descargado y compilado el paquete en la carpeta `/usr/local/cmake` , por eso he necesitado usar el comando `sudo` antes de cada acción, pero sino no sería necesario.

Una vez construido el ejecutable instalarlo usando `checkinstall`, para que así sea más fácilmente eliminable si es necesario. Instalar `checkinstall`:

```
sudo apt-get install checkinstall
```

(Guía de uso de `checkinstall`:

<https://manpages.ubuntu.com/manpages/bionic/en/man8/checkinstall.8.html>)

Por defecto `checkinstall` lleva a cabo la instalación del paquete `.deb` en `/home/<user>/<pkgname>` , pero nosotros vamos a elegir que se instale en `bin` con el resto de paquetes de Ubuntu.

```
sudo checkinstall --pkgname=cmake --pkgversion="3.22-custom" --pakdir=/usr/local/cmake --default
```

(el comando anterior es una única línea)

Nota: el paso “Copiando los archivos al directorio temporal...” toma su tiempo y parece que el ordenador no está haciendo nada, pero tener paciencia. Y dejarlo hacer tranquilamente.

Si todo ha ido bien el resultado de la instalación debería parecerse a esto:

```
*****
Done. The new package has been installed and saved to
/bin/cmake/cmake_3.22-custom-1_amd64.deb
You can remove it from your system anytime using:

    dpkg -r cmake
*****
```

Verificar instalación:

`hash -r` (para resetear la caché del terminal para actualizar los paths o abrir nueva ventana)
`apt-cache policy cmake` (devuelve si hay algún paquete con el nombre de Cmake aunque no se hubiera instalado correctamente)
`cmake --version` (si la instalación ha sido correcta y el paquete se ha generado correctamente, devuelve la versión instalada de CMake)
`which cmake` (devuelve la ubicación de Cmake)

Si el paso “`cmake --version`” no devolviera la versión, entonces es que no se ha instalado correctamente el paquete. En tal caso hacer:

```
sudo dpkg -r cmake
```

IMPORTANTE: no hagas la acción anterior si hay más versiones de cmake instaladas, podrías borrar alguna que no fuera la que tratamos de instalar. Investiga antes cómo borrar de forma segura el paquete.

Si en el futuro algún día se quiere desinstalar → ¡CUIDADO!:

No desinstales directamente, plantéate si cambiar de versión. Desinstalar Cmake puede causar que se desinstalen otros paquetes que dependen de CMake como ROS.

(https://gist.github.com/bmegli/4049b7394f9cfa016c24ed67e5041930?permalink_comment_id=3965623)

Identifica la versión of de interés:

```
apt-cache policy cmake
```

Cambia a otra versión de la lista anterior:

```
apt-get install cmake=VERSION
```

Si no lo haces así tendrías que volver a instalar todos los paquetes que se hayan desinstalado.

1.5. OptiX 7.3 o superior (opcional)

Para comprobar si ya encuentra instalado en el equipo buscar la carpeta de optix. En mi caso se encuentra en `usr/local`

Si no está presente en el equipo se recomienda su instalación ya que ofrece mejores prestaciones en el trazado de rayos en la GPU para el renderizado de gráficos.

<https://developer.nvidia.com/designworks/optix/download>

Para llevar a cabo la descarga es necesario formar parte de NVIDIA Developers Program, es un registro gratuito. Has de tener primero un usuario de NVIDIA, yo me he registrado con la cuenta de alumnos.upm y luego me he unido al programa.

Elegir la versión de OptiX compatible con los drivers instalados con CUDA. Igualmente chequear el Foro de NVIDIA Developers para más información:

<https://forums.developer.nvidia.com/>

Apartado Gaming and Visualization Technologies → Visualization → OptiX

Para chequear la versión de los drivers recordamos que hay que lanzar el comando:

```
nvidia-smi
```

Una vez descargado el paquete de OptiX se instala como cualquier .sh

OJO: se instalará en la carpeta donde tengas el .sh, se recomienda /usr/local/:

```
sudo chmod +x NVIDIA-OptiX-SDK-{version}.sh
```

```
sudo bash NVIDIA-OptiX-SDK-{version}.sh
```

Aceptar la licencia: pulsar enter hasta llegar al final (~punto 8.11) y aceptar cuando lo requiera

Para compilar los samples de OptiX seguir las instrucciones del archivo INSTALL_LINUX.txt en la carpeta SDK del módulo de OptiX. Al no tener instalada una versión de CMake superior a 3.24.3, usar \$ cmake <path> en lugar de ccmake. La configuración se hará automáticamente.

Es posible que haga falta instalar las librerías de Xinerama:

```
sudo apt install libxcb-xinerama0 libxcb-xinerama0-dev libxinerama-dev libxinerama1 libxcursor-dev
```

1.6. Vulkan SDK (opcional)

En teoría opcional, pero es necesario para que funcionen los samples que comprueban el correcto funcionamiento de CUDA ToolKit. Ver apartado 1.2.3.7.

1.7. Librerías extra (opcional)

Para acabar y como recomendación extra se recomienda la instalación de las siguientes librerías:

```
sudo apt-get install build-essential git python3-dev python3-pip libopenexr-dev libxi-dev libglfw3-dev libglew-dev libomp-dev libxinerama-dev libxcursor-dev
```

(el comando anterior es una única línea)

1.8. Añadir CUDA a los PATH

Ver apartado 1.2.3.1. Environment Setup.

Actualizar los PATH. Para ello lanzar:

```
which nvcc
```

De ahí se obtiene la ruta de instalación de CUDA

Por ejemplo, si se ha instalado CUDA 11.8, añadir lo siguiente al final del archivo ~/.bashrc :

```
export PATH="/usr/local/cuda-11.8/bin:$PATH"  
export LD_LIBRARY_PATH="/usr/local/cuda-11.8/lib64:$LD_LIBRARY_PATH"
```

1.9 Instalación NVIDIA DLSS SDK

Antes de nada: hace falta que la tarjeta gráfica sea compatible (del tipo RTX) para poder instalar DLSS. Por ejemplo, la tarjeta antigua de mi ordenador (NVIDIA GTX 1060 6GB) no era compatible.

El paquete DLSS básico viene incluido en los drivers nvidia de la tarjeta gráfica si es compatible, por lo que no es necesaria su instalación.

Sin embargo, para el desarrollo de aplicaciones gráficas que usen el trazado de rayos se recomienda instalar el SDK para mejorar el rendimiento del renderizado con la interpolación de píxeles de DLSS durante la ejecución de NeRF.

1.9.1. Pre-Requirements

1.9.1.1. Linux kernel: 2.6.32 and newer

Check kernel version:

```
uname -r
```

1.9.1.2. glibc 2.11 or newer

Comprobar si se tiene instalado glibc 2.11 o posterior:

```
ldd --version ldd
```

Ldd viene instalado con glibc y debería de dar la versión correspondiente de glibc

Si no seguir las siguientes instrucciones:

Instalar glibc en la carpeta /opt

Página de glibc, donde encontrar la última versión oficial: <https://www.gnu.org/software/libc/>

Elegir la última versión y descargar:

```
wget -c https://ftp.gnu.org/gnu/glibc/glibc-{version}.tar.gz
```

En mi caso la versión es la 2.36:

```
wget -c https://ftp.gnu.org/gnu/glibc/glibc-2.36.tar.gz
```

Descomprimir:

```
tar -zxvf glibc-{version}.tar.gz
```

Antes de instalar y/o de realizar la configuración de instalación para glibc hay que instalar los headers del kernel de linux en un directorio aparte, que luego especificaremos durante la configuración de glibc. Para ello usamos la siguiente documentación:

<https://www.tecmint.com/install-kernel-headers-in-ubuntu-and-debian/>

https://docs.kernel.org/kbuild/headers_install.html

Para conocer si tenemos los headers instalados:

```
uname -r  
apt search linux-headers-$(uname -r)
```

Para conocer su ubicación:

```
ls -l /usr/src/linux-headers-$(uname -r)
```

Si no estuviesen instalados el último comando daría error, entonces habría que ejecutar:

```
sudo apt update  
sudo apt install linux-headers-$(uname -r)
```

Si están instalados el output de la terminal debería ser del estilo de:

```
robci@robci-NeRF: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
(base) robci@robci-NeRF:~$ uname -r  
5.4.0-135-generic  
(base) robci@robci-NeRF:~$ apt search linux-headers-$(uname -r)  
Ordenando... Hecho  
Buscar en todo el texto... Hecho  
linux-headers-5.4.0-135-generic/bionic-updates,bionic-security,now 5.4.0-135.152~18.04.2 amd64 [instalado]  
Linux kernel headers for version 5.4.0 on 64 bit x86 SMP  
  
(base) robci@robci-NeRF:~$ ls -l /usr/src/linux-headers-$(uname -r)  
total 1616  
drwxr-xr-x 3 root root 4096 dic 14 21:11 arch  
lrwxrwxrwx 1 root root 40 nov 28 22:10 block -> ../linux-hwe-5.4-headers-5.4.0-135/block  
lrwxrwxrwx 1 root root 40 nov 28 22:10 certs -> ../linux-hwe-5.4-headers-5.4.0-135/certs  
lrwxrwxrwx 1 root root 41 nov 28 22:10 crypto -> ../linux-hwe-5.4-headers-5.4.0-135/crypto  
lrwxrwxrwx 1 root root 48 nov 28 22:10 Documentation -> ../linux-hwe-5.4-headers-5.4.0-135/Documentation  
lrwxrwxrwx 1 root root 42 nov 28 22:10 drivers -> ../linux-hwe-5.4-headers-5.4.0-135/drivers  
drwxr-xr-x 2 root root 4096 dic 16 20:56 dummy  
lrwxrwxrwx 1 root root 37 nov 28 22:10 fs -> ../linux-hwe-5.4-headers-5.4.0-135/fs  
drwxr-xr-x 4 root root 4096 dic 14 21:11 include  
lrwxrwxrwx 1 root root 39 nov 28 22:10 init -> ../linux-hwe-5.4-headers-5.4.0-135/init  
lrwxrwxrwx 1 root root 38 nov 28 22:10 ipc -> ../linux-hwe-5.4-headers-5.4.0-135/ipc  
lrwxrwxrwx 1 root root 41 nov 28 22:10 Kbuild -> ../linux-hwe-5.4-headers-5.4.0-135/Kbuild  
lrwxrwxrwx 1 root root 42 nov 28 22:10 Kconfig -> ../linux-hwe-5.4-headers-5.4.0-135/Kconfig  
drwxr-xr-x 2 root root 4096 dic 14 21:11 kernel  
lrwxrwxrwx 1 root root 38 nov 28 22:10 lib -> ../linux-hwe-5.4-headers-5.4.0-135/lib  
lrwxrwxrwx 1 root root 43 nov 28 22:10 Makefile -> ../linux-hwe-5.4-headers-5.4.0-135/Makefile  
lrwxrwxrwx 1 root root 37 nov 28 22:10 mm -> ../linux-hwe-5.4-headers-5.4.0-135/mm  
-rw-r--r-- 1 root root 1619845 nov 28 22:10 Module.symvers  
lrwxrwxrwx 1 root root 38 nov 28 22:10 net -> ../linux-hwe-5.4-headers-5.4.0-135/net  
lrwxrwxrwx 1 root root 42 nov 28 22:10 samples -> ../linux-hwe-5.4-headers-5.4.0-135/samples  
drwxr-xr-x 8 root root 12288 dic 14 21:11 scripts  
lrwxrwxrwx 1 root root 43 nov 28 22:10 security -> ../linux-hwe-5.4-headers-5.4.0-135/security  
lrwxrwxrwx 1 root root 40 nov 28 22:10 sound -> ../linux-hwe-5.4-headers-5.4.0-135/sound  
drwxr-xr-x 3 root root 4096 dic 14 21:11 tools  
lrwxrwxrwx 1 root root 41 nov 28 22:10 ubuntu -> ../linux-hwe-5.4-headers-5.4.0-135/ubuntu  
lrwxrwxrwx 1 root root 38 nov 28 22:10 usr -> ../linux-hwe-5.4-headers-5.4.0-135/usr  
lrwxrwxrwx 1 root root 39 nov 28 22:10 virt -> ../linux-hwe-5.4-headers-5.4.0-135/virt  
(base) robci@robci-NeRF:~$
```

Una vez localizados, en el directorio de los headers (en el ejemplo /usr/src/linux-hwe-5.4-headers-5.4.0-135), lanzar el comando indicando la ubicación deseada para INSTALL_HDR_PATH:

```
sudo make headers_install INSTALL_HDR_PATH=/opt/glibc/include
```

A veces es necesario especificar la arquitectura del kernel:

```
sudo make headers_install ARCH=x86_64 INSTALL_HDR_PATH=/opt/glibc/include
```

Para conocer la arquitectura del kernel usar:

```
uname -m
```

para continuar necesitamos realizar los pasos de configuración que especifica en el apartado C.4 Specific advice for GNU/Linux systems de:

https://www.gnu.org/software/libc/manual/2.36/html_mono/libc.html#Linux

Para ello seguiremos los pasos de:

<https://www.linuxfromscratch.org/lfs/view/9.0-systemd/chapter05/glibc.html>

```
mkdir -v build
```

```
cd build
../configure \
--prefix=/opt/glibc \
--host=$LFS_TGT \
--build=$(../scripts/config.guess) \
--enable-kernel=3.2 \
--with-headers=/opt/glibc/include
(el comando anterior es una única línea)
sudo make
sudo make install
```

Check installation:

```
echo 'int main(){}' > dummy.c
$LFS_TGT-gcc dummy.c
readelf -l a.out | grep ': /opt/glibc'
```

Clean up test files:

```
rm -v dummy.c a.out
```

1.9.1.3. gcc and g++ 8.4.0 or newer

Ver apartado 1.1 Instalación de Visual Studio 2019 o GCC/G++ 7.5+

1.9.2 Instalación de NVIDIA DLSS SDK

Obtener el descargable y seguir los pasos para NVIDIA DLSS SDK en el siguiente enlace:

<https://developer.nvidia.com/rtx/dlss/get-started#sdk-version>

2. DESCARGA Y COMPILACIÓN DE instant-ngp

Se requiere el uso de Git, si no está instalado en el equipo, por favor hacer:

```
sudo apt-get update
sudo apt-get install git
```

Si se desea configurar el usuario de Git se recomienda el uso de los manuales de referencia oficiales: <https://git-scm.com/doc> aunque no es necesario puesto que solo se va a proceder a la descarga de un repositorio.

Siguiendo el manual en: <https://github.com/NVlabs/instant-ngp>

En el archivo Readme.md: <https://github.com/NVlabs/instant-ngp/blob/master/README.md>

En concreto en el apartado Linux build: <https://github.com/NVlabs/instant-ngp#building-instant-ngp-windows--linux>

Se procede a la descarga del repositorio de instant-ngp. Se recomienda en `/usr/local/` :

```
sudo git clone --recursive https://github.com/nvlab/instant-ngp
cd instant-ngp
```

Antes de instalar los pip requirements, comprobar que se dispone de la última versión de pip:

```
sudo python -m pip install --upgrade pip
```

Para evitar errores con las versiones instaladas por defecto en Ubuntu se recomienda instalar la última versión de pip descargando el ejecutable de la página de pip: <https://bootstrap.pypa.io/get-pip.py> (El enlace actualizado aparece en la siguiente documentación <https://pip.pypa.io/en/stable/installation/>)

Abrir un terminal en la carpeta donde se descargó y hacer:

```
sudo python get-pip.py
```

Cerrar aplicaciones gráficas y navegadores web. Después proceder con la instalación de los requirements:

```
sudo python -m pip install -r requirements.txt
```

La instalación del módulo `opencv-python-headless` puede durar horas. Si no se dispone de tanto tiempo se puede intentar:

```
sudo python -m pip install "opencv-python-headless<4.3"
```

Y si hace falta actualizar posteriormente a una versión más reciente.

Es posible que se requiera de la instalación previa de ciertas librerías para la correcta instalación de los requirements, por ejemplo en mi caso:

```
sudo apt-get install python-opencv python3-opencv libjsoncpp-dev
```

Para terminar compilamos NeRF. Para compilar se añade la regla `-DCMAKE_CUDA_COMPILER` por error de compilación, ver README.md en la carpeta descargada de instant-ngp:

```
sudo cmake . -B build -DCMAKE_CUDA_COMPILER=/usr/local/cuda-<your cuda version>/bin/nvcc
(el comando anterior es una única línea)
```

IMPORTANTE: Si se desea automatizar procesos de NeRF haciendo uso de Python es necesario que no se produzca el siguiente warning durante la configuración:

```
-- Could NOT find Python (missing: Python_INCLUDE_DIRS Python_LIBRARIES Development Development.Module Development.Embed) (found su tab
le version "3.9.16", minimum required is "3.7")
```

Añadir las reglas de compilación correspondientes de forma manual:

```
sudo cmake . -B build -DCMAKE_CUDA_COMPILER=/usr/local/cuda-<your cuda version>/bin/nvcc \
    -DPython_INCLUDE_DIRS=$(python -c "import sysconfig; print(sysconfig.get_path('include'))") \
    -DPython_LIBRARIES=$(python -c "import sysconfig; print(sysconfig.get_config_var('LIBDIR'))")
```

(el comando anterior es una única línea)

o "python3 -c ..." si da problemas

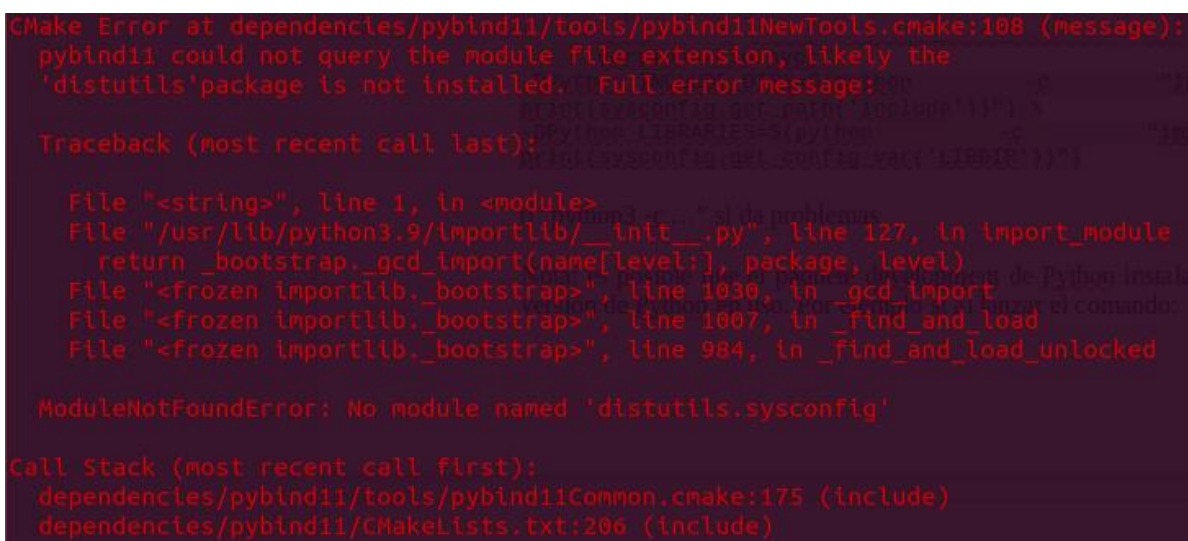
Nota: si hay varias versiones de python instaladas en el equipo es posible que el paquete development de Python instalado no se corresponda con la versión de Python en uso. Por ejemplo si al lanzar el comando:

```
python -c "import sysconfig; print(sysconfig.get_path('include'))"
```

Devuelve la versión pythonx.y verificar que el paquete development correspondiente está instalado:

```
sudo apt-get install pythonx.y-dev
```

Nota2: Si aparece el siguiente mensaje de error durante la configuración puede que falte el paquete distutils de la versión correspondiente de python



```
CMake Error at dependencies/pybind11/tools/pybind11NewTools.cmake:108 (message):
pybind11 could not query the module file extension, likely the
'distutils' package is not installed. Full error message:

Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python3.9/importlib/__init__.py", line 127, in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
  File "<frozen importlib._bootstrap>", line 1030, in _gcd_import
  File "<frozen importlib._bootstrap>", line 1007, in _find_and_load
  File "<frozen importlib._bootstrap>", line 984, in _find_and_load_unlocked

ModuleNotFoundError: No module named 'distutils.sysconfig'

Call Stack (most recent call first):
  dependencies/pybind11/tools/pybind11Common.cmake:175 (include)
  dependencies/pybind11/CMakeLists.txt:206 (include)
```

Instalar haciendo:

```
sudo apt install pythonx.y-distutils
```

Una vez terminado el proceso de configuración lanzar el montaje de la aplicación:

```
sudo cmake --build build --config RelWithDebInfo -j
```

Una vez finalizado el proceso de build, para probar si funciona, usar alguno de los ejemplos provistos, como el de la cabeza disecada de zorro:

```
sudo ./instant-ngp data/nerf/fox
```

Para descargar el resto de demos descargar la siguiente carpeta en /usr/local/instant-ngp/data:
<https://drive.google.com/drive/folders/1JDdLGDruGNXWnM1eqY1FNL9PIStjaKWi>

Y lanzar la demo deseada para el entrenamiento de la red. Por ej., dentro de la carpeta instant-ngp:

```
sudo ./instant-ngp data/nerf_synthetic/lego/transforms_train.json
```

Tips para la configuración de los parámetros de entrenamiento: https://github.com/NVlabs/instant-ngp/blob/master/docs/nerf_dataset_tips.md

Tutorial para crear tu propia animación NeRF: <https://www.youtube.com/watch?v=8GbENSmdVeE>

3. CREACIÓN Y USO DE UN DATASET PROPIO

Para generar la homografía de nuestro propio dataset a partir de una serie de fotos guardadas en `instant-ngp/data/dataset_de_prueba/images` usar el script `colmap2nerf.py` desde `dataset_de_prueba`:

IMPORTANTE: no lanzar el comando hasta haber completado el apartado 3.1

```
sudo python3 ../../scripts/colmap2nerf.py --colmap_matcher exhaustive --run_colmap --aabb_scale 16
```

(el comando anterior es una única línea)

3.1 Requisitos previos a la generación de la homografía de nuestro propio dataset

Para llevar a cabo la ejecución de la anterior línea de comandos se requiere de la aplicación COLMAP y de librerías de Python que es posible no estén instaladas en su equipo.

COLMAP es un software gratuito de fotogrametría que se encarga de, junto con el script `colmap2nerf.py` provisto por `instant-ngp`, de generar la homografía de nuestro dataset.

Instalación de COLMAP (se recomienda usar los comandos que aparecen la referencia actualizada: <https://colmap.github.io/install.html>):

Dependencias de COLMAP:

```
sudo apt-get install \
  git \
  cmake \
  build-essential \
  libboost-program-options-dev \
  libboost-filesystem-dev \
  libboost-graph-dev \
  libboost-system-dev \
  libboost-test-dev \
  libeigen3-dev \
  libsuitesparse-dev \
  libfreeimage-dev \
  libmetis-dev \
  libgoogle-glog-dev \
  libgflags-dev \
  libglew-dev \
  qtbase5-dev \
  libqt5opengl5-dev \
  libcgald \
  libceres-dev
```

(el comando anterior es una única línea)

Para ubuntu 16.04 y 18.04 añadir:

```
sudo apt-get install libcgall-dev
```

A continuación instalar Ceres Solver. Seguiremos la guía: <http://ceres-solver.org/installation.html>

Instalar dependencias de Ceres Solver:

```
sudo apt-get install libgoogle-glog-dev libgflags-dev
sudo apt-get install libatlas-base-dev
sudo apt-get install libeigen3-dev
sudo apt-get install libsuitesparse-dev
```

Descargar la última versión estable en el link que aparece en: <http://ceres-solver.org/installation.html> y en la carpeta donde se ha descargado ejecutar:

```
tar xzf ceres-solver-{release}.tar.gz
mkdir ceres-bin
cd ceres-bin
cmake ../ceres-solver-{release}
make -j3
make test
sudo make install
```

Nota: Durante el proceso de testeo de ceres solver, teniendo la versión 12.0 de CUDA, se produjeron los siguientes test fail:

The following tests FAILED:

- 1 - cuda_memcheck_dense_qr_test (Not Run)
- 2 - cuda_memcheck_dense_cholesky_test (Not Run)
- 26 - cuda_dense_cholesky_test (Failed)

Pueden ser la antesala de problemas posteriores en la instalación de colmap. De hecho hasta la fecha colmap no es compatible con CUDA 12.0: <https://github.com/colmap/colmap/issues/1742>

Antes de continuar con Colmap se requiere la instalación de las siguientes librerías:

Instalar librería FLANN:

```
sudo apt-get update -y
sudo apt-get install -y libflann-dev
```

Instalar LZ4 library usando snapd:

```
sudo apt update
sudo apt install snapd
sudo snap install lz4
```

Dado que hay que usar la herramienta `snapd` para instalar `lz4`, esta no se instala en los directorios por defecto (`/usr`, `/usr/local`, `/opt`, `/opt/local`). Por tanto, cuando se descargue `colmap`, antes de ejecutar el comando “`cmake ..`”, hay que modificar el archivo de búsqueda de la librería `lz4` de `COLMAP` para que sea capaz de encontrarlo. Se trata del archivo `FindLZ4.cmake` ubicado en `../colmap/cmake` . Hay que añadir lo siguiente:

- En la lista `LZ4_CHECK_INCLUDE_DIRS` añadir la ubicación de los includes de `lz4`, por ejemplo: `/snap/lz4/current/usr/local/include`
- En la lista `LZ4_CHECK_LIBRARY_DIRS` añadir la ubicación de las librerías de `lz4`, por ejemplo: `/snap/lz4/current/usr/local/lib`

Si la versión de `cuda` no corresponde con la de `nvidia-cuda-toolkit`, entonces borrarlo:

```
apt-cache policy nvidia-cuda-toolkit
```

Para borrar:

```
sudo apt remove nvidia-cuda-toolkit
```

Una vez hecho esto, ya se puede instalar `COLMAP`:

```
git clone https://github.com/colmap/colmap.git
cd colmap
git checkout dev
mkdir build
cd build
cmake ..
make -j
sudo make install
```

Nota: es posible que no se detecte automáticamente el valor de `CMAKE_CUDA_ARCHITECTURES` durante la fase de “`cmake ..`”. Esto puede deberse a la existencia de varias GPU en el ordenador al mismo tiempo o en una sustitución de la tarjeta gráfica original. Si esto ocurriera hay que añadir la siguiente línea al fichero `CmakeLists.txt` ubicado en la carpeta de `colmap` (`../colmap`):

```
set(CMAKE_CUDA_ARCHITECTURES “arch”)
```

Donde `arch` es la arquitectura de la tarjeta gráfica. Hay que poner el código correspondiente entre comillas dobles (ojo que a veces al copiar-pegar las comillas dobles no son las buenas). Esta línea se puede añadir al principio del archivo, por ejemplo a continuación de las definiciones de `COLMAP_VERSION`.

En el apartado `Compilation` de la guía de instalación de `NeRF` (`Readme.md`) se especifica el código para ciertas familias de tarjetas gráficas de `nvidia`, también se especifica un link a una lista exhaustiva si se requiere más información:

<https://github.com/NVlabs/instant-ngp/blob/master/README.md>

Si lo anterior no soluciona el problema se recomienda el uso de la versión 3.7 de colmap en vez de posteriores. En este caso está comprobado que funciona la versión 3.7 a día 16/11/2022 con SHA: b02e93953ea4b6448e004524f65b82bf5e807c7c

Abrir el repositorio Git en la carpeta de colmap y hacer checkout de la versión anterior.

Para probar si COLMAP funciona, lanzar COLMAP:

```
colmap -h  
colmap gui
```

Además, el script `colmap2nerf.py` tiene ciertas dependencias.

- Para instalar las librerías que faltan de forma manual se puede usar pip:

Se puede instalar haciendo uso de apt-get, pero se recomienda el método que se propone después para asegurar que se instala la última versión.

```
sudo apt-get install python-pip python3-pip
```

Descargar el instalador de pip en el siguiente enlace: <https://bootstrap.pypa.io/get-pip.py>

Abrir un terminal en la ubicación donde se descargó y ejecutar:

```
sudo python get-pip.py
```

Después instalar las librerías necesarias haciendo uso de pip (las dependencias aparecen en la cabecera del archivo `colmap2nerf.py` en la sección de imports):

```
sudo python -m pip install -U {missing library}
```

p ej:

```
sudo python -m pip install -U numpy
```

- O a través de apt-get directamente:

```
sudo apt-get install python-opencv python3-opencv libopencv-dev python-numpy python3-numpy python-dev python3-dev
```

(el comando anterior es una única línea)

Por último hace falta instalar ffmpeg:

```
sudo apt install ffmpeg
```

Para comprobar que la instalación se ha llevado a cabo correctamente:

```
ffmpeg -version
```

Una vez instalada la aplicación de COLMAP y las librerías necesarias, ya se puede lanzar el script `colmap2nerf.py` con la orden especificada al principio de este punto. Al acabar su ejecución se genera un archivo `transforms.json` que contiene el resultado del estudio de homografía de nuestro dataset. Este

se deberá pasar como parámetro del ejecutable testbed que obtuvimos tras la compilación de NeRF, con la siguiente línea de comandos y desde la carpeta instant-ngp:

```
sudo ./instant-ngp data/dataset_de_prueba/transforms.json
```

Con ello se abre la GUI de NeRF instant-ngp y se pone a entrenar la red neuronal sobre nuestro dataset. A los pocos segundos podremos ver el resultado de nuestro dataset con la generación de un modelo virtual en nube de puntos definido por rayos de nuestro dataset.

3.2 Conclusiones primeros experimentos NeRF con instant-ngp:

Antes de lanzar el script que realiza la homografía, es muy importante preparar un correcto dataset. La resolución y la cantidad de imágenes en el dataset afecta enormemente. Se recomienda que el dataset contenga entre 50 y 150 imágenes. Para un set de 250 imágenes la fotogrametría puede tardar en generarse unas 30 veces más que para el mismo dataset pero reducido a 50-75 imágenes (2min | 60 imágenes, 4min | 100 imágenes vs. ~60min | 270 imágenes). Además una resolución menor a 1080x1080 píxeles reduce exponencialmente el tiempo de ejecución de `colmap2nerf.py`. Ojo con no pasarse para que los resultados de la generación del entrono virtual sean aceptables. 720X720 está bien. Si las imágenes son en HD, el incremento de tiempo de generación de la homografía entre por ejemplo 1080 y 1440 no ha sido muy grande para los casos estudiados. A mayor cantidad de imágenes y mayor resolución, la red neuronal de instant-nerf también requerirá mayor tiempo de entrenamiento cuando se lance.

El dataset de imágenes debe estar ubicado en una carpeta de nombre images para que el script la encuentre.

Ejemplo de secuencia de lanzamiento del script que genera la homografía del dataset desde la carpeta que contiene dicho dataset, que a su vez está dentro de la carpeta data de instant-ngp.

```
sudo python3 ../../scripts/colmap2nerf.py --colmap_matcher exhaustive --run_colmap --aabb_scale 4
```

(el comando anterior es una única línea)

El parámetro `--aabb_scale` es el más importante. Define la el tamaño de la escena, escala la distancia media a la que se se encuentran las cámaras posicionadas desde el origen. Los valores son potencias de 2 desde 1, 2, 4, 8, 16 hasta 128.

El parámetro `exhaustive` se usa para especificar si el dataset tiene un origen adhoc (es una serie de imágenes de una escena tomadas independientemente) o si es `sequential` es decir, provenientes de un vídeo y se trata de una secuencia de imágenes ordenadas. Si el dataset es de tipo `sequential` su fotogrametría se genera de forma más eficiente y por tanto mucho más rápida, pudiendo ser del entorno del doble de rápida para una misma cantidad de imágenes. También hay que tener en cuenta que si en el dataset de tipo `exhaustive` la imágenes están muy repartidas a lo largo de la escena la fotogrametría tomará más tiempo en realizarse.

Por tanto y aunque la forma genérica para cualquier tipo de dataset sea la `exhaustive`, se recomienda el paso de un vídeo al script para que este obtenga las imágenes de forma secuencial y por tanto después sea más eficiente a la hora de realizar la fotogrametría.

Otro parámetro que influye notablemente en la generación correcto del modelo virtual es que las imágenes no estén borrosas. En un vídeo siempre aparecerán tomas borrosas, por tanto se debe

realizar de la forma más estable posible. Si se hace la selección de imágenes de forma manual se deberán descartar preferentemente las que estén borrosas.

Ejemplo de secuencia de lanzamiento del script para la obtención del dataset a partir de un vídeo y a su vez generar a continuación la homografía a partir del mismo:

```
sudo python3 ../../scripts/colmap2nerf.py --video_in video_zapatillaUSB.mp4 --video_fps 1 --run_colmap --aabb_scale 4
```

(el comando anterior es una única línea)

El parámetro `--video_in` sirve para especificar el nombre del archivo de vídeo que debe encontrarse en la misma carpeta desde la que se lanza el script y será en la que se cree la carpeta images conteniendo todo el dataset.

El parámetro `--video_fps` sirve para especificar el número de imágenes (frames) que se extraerán de cada segundo del vídeo original, así que si se quiere un dataset de 100 imágenes de un vídeo que dura 1:40 min (100 segundos) pues un frame per second está bien.

El resto de parámetros funcionan igual que en el ejemplo anterior de obtención de la homografía a partir del dataset de imágenes previamente obtenido.

Una vez obtenido el archivo de homografía `transforms.json` se lanza desde la carpeta instant-ngp:

```
sudo ./instant-ngp data/dataset_de_prueba/transforms.json
```

Esto abrirá la GUI de instant-ngp. Dentro de la GUI hay varios parámetros importantes a configurar para que el visualizado sea más fluido. El primero es el parámetro Target FPS, se puede bajar a 5 por ejemplo para conseguir mejor resolución de renderizado. Crop aabb se usa para reducir la escena hasta el centro deseado. Se establece el máximo en el valor especificado al lanzar `colmap2nerf.py` y se puede reducir para evitar ver los límites de la escena que pueden estar ocultando lo que se desea ver. También se puede modificar el parámetro field of view para ampliar el campo de visión para un determinado zoom (esto puede afectar a la deformación de la escena).

Una vez la inteligencia se haya entrenado con el dataset (normalmente basta entre 1 y 2 minutos) se puede usar la tecla T o pulsar el botón stop training. Si durante el primer minuto no se ha generado nada correctamente con casi toda probabilidad no se generará nada aunque se deje pasar el tiempo, así que es mejor revisar el dataset. A veces parece que no se ha generado nada pero en realidad son los límites de la escena que aparecen como una especie de niebla que la envuelve y no dejan ver el interior. Esto se soluciona navegando o haciendo zoom hacia el interior o usando la herramienta crop para apagar la visualización de los puntos más alejados del centro de la escena.

Según la documentación más allá de dos minutos de entrenamiento no van a conseguir grandes mejoras en la generación del modelo.

Bajar la resolución de la GUI en la llamada a run.py mejora notablemente el rendimiento al generar el modelo. Por ejemplo, cuando se lanza el ejecutable para el entrenamiento del modelo se especifica la resolución de la GUI que interpreta el archivo run.py:

```
sudo ./instant-ngp data/dataset_zapatillaUSB/transforms.json --width 1280 --height 720
```

(el comando anterior es una única línea)

```
sudo ./instant-ngp data/dataset_zapatillaUSB/transforms.json --width 960 --height 540
```

(el comando anterior es una única línea)

Editar transforms.json:

Añadir las líneas:

```
"scale": 0.005,  
"offset": [0.5, -0.5, 0.5],
```

Entre las líneas de "aabb_scale": 4, y "frames": [

Funcionamiento de scale: por defecto 0.33 . Escala la escena para que pueda caber dentro del cubo. A más pequeño este valor, el cubo abarca una zona mayor de la escena.

Funcionamiento de offset: por defecto [0.5, 0.5, 0.5] que corresponden a los ejes [z, x, y] de la referencia dentro de NeRF (z y x son el plano horizontal e y la cota vertical). El offset representa la distancia del cubo de generación respecto al centroide de las posiciones de la cámara. A medida que crece el offset el cubo se mueve en la dirección que correspondería a las coordenadas negativas, así que si queremos moverlo hacia las coordenadas positivas habría que darle valores negativos y viceversa.

Funcionamiento de aabb_scale: por defecto 16 . Captura los objetos a una distancia 16 veces el tamaño del cubo. Acepta valores que sean potencias de 2, desde 1 hasta 128.

Para evitar entrenar a la red cada vez, nada más lanzar el proceso se le puede dar a stop training y a continuación a load (en el menú de Snapshot) para cargar el modelo previamente entrenado con ese dataset.

Para poder cargar el entrenamiento de la red para un dataset es necesario haberla entrenado primero y después haberlo guardado. En el menú Snapshot seleccionar save (se puede elegir el nombre del archivo). Se genera un archivo de tipo .msgpack que guarda el resultado del entrenamiento de la red para poder cargarlo directamente tras lanzar el testbed sobre un dataset ya entrenado.

En el menú de Marching Cubes (Mesh) se puede exportar un Mesh para abrirlo con otros motores de renderizado como Blender o Unity. El archivo .obj sirve para exportar un "mesh" de tipo triangular (poliedros y superficies hechas de triángulos (?) creo) y se puede elegir que contenga sólo los poliedros o también información sobre el color. Los archivos .obj estándar no guardan información sobre color, así que la mayoría de motores de renderizado interpretarán un objeto sin color. Para añadir el color hace falta pulsar en Save RGBA PNG sequence. Será un archivo png que contenga las texturas correspondientes al modelo renderizado como mesh después de haber hecho mesh it.

Parece que hay un problema al generar este archivo puesto que la ruta de creación es /transforms.json/rgba_slices, y no se puede crear un directorio con el mismo nombre que un archivo, así que el guardado no se realiza.

```
(base) robclb@robclb-B250-Gaming-Infinite-A-M5-B915:/usr/local/instant-ngp$ sudo ./build/testbed --scene data/dataset_zapatillaUSB/transforms.json --width 960 --height 540
14:22:03 INFO Loading NeRF dataset from
14:22:03 INFO data/dataset_zapatillaUSB/transforms.json
14:22:04 SUCCESS Loaded 73 images after 0s
14:22:04 INFO cam_aabb=[min=[-0.332548,-0.130735,1.25344], max=[1.17137,1.83627,1.63488]]
14:22:04 INFO Loading network config from: configs/nerf/base.json
14:22:04 INFO GridEncoding: Nmin=16 b=1.51572 F=2 T=2^19 L=16
Warning: FullyFusedMLP is not supported for the selected architecture 61. Falling back to CutlassMLP. For maximum performance, raise the target GPU architecture to 75+.
Warning: FullyFusedMLP is not supported for the selected architecture 61. Falling back to CutlassMLP. For maximum performance, raise the target GPU architecture to 75+.
Warning: FullyFusedMLP is not supported for the selected architecture 61. Falling back to CutlassMLP. For maximum performance, raise the target GPU architecture to 75+.
14:22:04 INFO Density model: 3--[HashGrid]-->32--[FullyFusedMLP(neurons=64, layers=3)]-->1
14:22:04 INFO Color model: 3--[Composite]-->16+16--[FullyFusedMLP(neurons=64, layers=4)]-->3
14:22:04 INFO total_encoding_params=13074912 total_network_params=10240
14:22:04 WARNING Vulkan error: loader_validate_device_extensions: Device extension VK_NVX_binary_import not supported by selected physical device or enabled layers.
14:22:04 WARNING Vulkan error: vkCreateDevice: Failed to validate extensions in list
14:22:04 WARNING Could not initialize Vulkan and NGX. DLSS not supported. (/usr/local/instant-ngp/src/dlss.cu:358 vkCreateDevice(vk_physical_device, &device_create_info, nullptr, &vk_device) failed)
14:30:09 INFO #vertices=381231 #triangles=751256
14:30:49 SUCCESS Wrote RGBA PNG sequence to data/dataset_zapatillaUSB/transforms.json/rgba_slices
14:31:27 SUCCESS Wrote RGBA PNG sequence to data/dataset_zapatillaUSB/transforms.json/rgba_slices
14:32:20 SUCCESS Wrote RGBA PNG sequence to data/dataset_zapatillaUSB/transforms.json/rgba_slices
14:32:58 SUCCESS Wrote density PNG to data/dataset_zapatillaUSB/transforms.json.density_slices_144x160x256.png
14:32:58 INFO #lattice points=5898240 #zero-x voxels=360360 (6.10962%) #lattice near zero-x=385855 (6.54187%)
14:33:15 SUCCESS Wrote RGBA PNG sequence to data/dataset_zapatillaUSB/transforms.json/rgba_slices
```

Además esta operación consume mucha memoria así que para poder realizar guardado de mesh se recomienda el uso de la GUI en la mínima resolución posible, tener el entrenamiento parado en el momento del guardado y también desactivar el renderizado puede ayudar.

How to save and load 3D objects from NeRF to 3D environment (Blender and MeshLab)
<https://www.youtube.com/watch?v=Yejdb9l1w2Q> - Create NeRF scene from video or images
<https://www.youtube.com/watch?v=5rLfWVnM-Xc> - Trim neural radiance NeRF model
<https://www.youtube.com/watch?v=55XKtYOIB7Y> - Export 3D Object from NeRF

Cuando hago mesh it de la figura si es grande como el dataset exterior 2, peta por falta de memoria. También peta si haces mesh it directamente de vertex colors, mejor solo normals y luego ya lo cambias después de hacer mesh it. Si es suficientemente pequeña sí se genera el mesh pero al seleccionar la casilla de optimize mesh cuidado, primero activarla y luego aumentar los parámetros de resolución y calidad, y no al revés, porque sino peta.

Por ahora no he conseguido guardar la imagen RGB del modelo :’(

Si le doy a load para evitar tener que entrenarlo cada vez en el de la zapatilla también peta :’(

4. LISTA DE COMANDOS ÚTILES

```
sudo cp -r dataset_de_prueba_2/ ../../usr/local/instant-ngp/data/
```

```
sudo cp -r dataset_exterior_mio/ ../../usr/local/instant-ngp/data/
```

```
../../usr/local/instant-ngp/data/dataset_zapatillaUSB
```

```
../../usr/local/instant-ngp/data/dataset_exterior_051q/
```

```
sudo cp transforms.json ../../usr/local/instant-ngp/data/dataset_exterior_051q/
```

```
sudo rm transforms.json
```

```
sudo cp transforms.json ../../../../usr/local/instant-ngp/data/dataset_exterior_mio/
```

```
sudo cp transforms.json ../../../../usr/local/instant-ngp/data/dataset_exterior_mio_2/
```

```
sudo python3 ../../scripts/colmap2nerf.py --colmap_matcher sequential --run_colmap --  
aabb_scale 16  
(el comando anterior es una única línea)
```

```
sudo python3 ../../scripts/colmap2nerf.py --colmap_matcher exhaustive --run_colmap --  
aabb_scale 4  
(el comando anterior es una única línea)
```

```
sudo python3 ../../scripts/colmap2nerf.py --video_in video_zapatillaUSB.mp4 --video_fps 1 --run_colmap --  
aabb_scale 4  
(el comando anterior es una única línea)
```

```
sudo ./build/testbed --scene data/dataset_de_prueba/transforms.json
```

```
sudo ./build/testbed --scene data/dataset_zapatillaUSB/transforms.json --width 1280 --height 720  
(el comando anterior es una única línea)
```

```
./build/testbed --mode nerf --scene
```

```
sudo python ../../scripts/colmap2nerf.py --video_in flores_amarillas.mp4 --video_fps 2 --run_colmap --aabb_scale 32
```

```
sudo python ../../scripts/colmap2nerf.py --video_in calderas.mp4 --video_fps 2 --run_colmap --aabb_scale 32
```

```
sudo ./instant-ngp data/datasets_seminario/dataset_flores_amarillas/base.ingp
```

```
sudo ./instant-ngp data/datasets_seminario/dataset_calderas/transforms.json
```

Para ver uso de la memoria de la gpu y comprobar si realmente se está haciendo uso de toda la RAM -> Utilizar jtop (hay que instalarlo).

Y al hilo de esto último, mirar el archivo de configuración de cuando se lanza el NeRF para ver si aprovecha toda la RAM de la GPU o está limitado de alguna forma.

ANEXO. RECOPIACIÓN DE POSIBLES FALLOS

Investigar también: cuando lanzas La demo de NeRF o `vulkaninfo` da unos warning:


```
robci@robci-B250-Gaming-Infinite-A-MS-B915: /usr/local/instant-ngp
Archivo Editar Ver Buscar Terminal Ayuda
up vector was [-0.92431898 0.17315428 0.34007649]
computing center of attention...
[-3.48153926 3.6044711 -1.13988727]
avg camera distance from origin 6.114105873260507
13 frames
writing transforms.json
robci@robci-B250-Gaming-Infinite-A-MS-B915: /usr/local/instant-ngp/data/dataset_de_prueba$ sudo python3 ../../scripts/colmap2nerf.py
--colmap_matcher exhaustive --run_colmap --aabb_scale 16
running colmap with:
  db=colmap.db
  images="images"
  sparse=colmap_sparse
  text=colmap_text
warning! folders 'colmap_sparse' and 'colmap_text' will be deleted/replaced. continue? (Y/n)
robci@robci-B250-Gaming-Infinite-A-MS-B915: /usr/local/instant-ngp/data/dataset_de_prueba$ cd ../../
robci@robci-B250-Gaming-Infinite-A-MS-B915: /usr/local/instant-ngp$ sudo ./build/testbed --scene data/dataset_de_prueba/transforms_t
rain.json
14:52:01 ERROR Scene path data/dataset_de_prueba/transforms.train.json does not exist.
robci@robci-B250-Gaming-Infinite-A-MS-B915: /usr/local/instant-ngp$ sudo ./build/testbed --scene data/dataset_de_prueba/transforms.j
son
14:52:48 INFO Loading NeRF dataset from
14:52:48 INFO data/dataset_de_prueba/transforms.json
14:52:49 SUCCESS Loaded 13 images after 0s
14:52:49 INFO cam_aabb=[min=[0.257742,-0.940844,0.136859], max=[2.00011,0.946658,1.30016]]
14:52:49 INFO Loading network config from: configs/nerf/base.json
14:52:49 INFO GridEncoding: Nmin=16 b=1.66248 F=2 T=2^19 L=16
Warning: FullyFusedMLP is not supported for the selected architecture 61. Falling back to CutlassMLP. For maximum performance, raise
the target GPU architecture to 75+.
Warning: FullyFusedMLP is not supported for the selected architecture 61. Falling back to CutlassMLP. For maximum performance, raise
the target GPU architecture to 75+.
Warning: FullyFusedMLP is not supported for the selected architecture 61. Falling back to CutlassMLP. For maximum performance, raise
the target GPU architecture to 75+.
14:52:49 INFO Density model: 3--[HashGrid]-->32--[FullyFusedMLP(neurons=64, layers=3)]-->1
14:52:49 INFO Color model: 3--[Composite]-->16+16--[FullyFusedMLP(neurons=64, layers=4)]-->3
14:52:49 INFO total_encoding_params=13623184 total_network_params=10240
14:52:50 WARNING Vulkan: loader_scanned_icd_add: Driver /usr/lib/x86_64-linux-gnu/libvulkan_radeon.so supports Vulkan 1.2, but only
supports loader interface version 4. Interface version 5 or newer required to support this version of Vulkan (Policy #LDP_DRIVER_7)
14:52:50 WARNING Vulkan: loader_scanned_icd_add: Driver /usr/lib/x86_64-linux-gnu/libvulkan_intel.so supports Vulkan 1.2, but only s
upports loader interface version 4. Interface version 5 or newer required to support this version of Vulkan (Policy #LDP_DRIVER_7)
14:52:50 WARNING Vulkan: loader_icd_scan: Can not find 'ICD' object in ICD JSON file /usr/share/vulkan/icd.d/nvidia_layers.json. Sk
ipping ICD JSON
14:52:51 WARNING Vulkan error: loader_validate_device_extensions: Device extension VK_NVX_binary_import not supported by selected ph
ysical device or enabled layers.
14:52:51 WARNING Vulkan error: vkCreateDevice: Failed to validate extensions in list
14:52:51 WARNING Could not initialize Vulkan and NGX. DLSS not supported. (/usr/local/instant-ngp/src/dlss.cu:358 vkCreateDevice(vk_
physical_device, &device_create_info, nullptr, &vk_device) failed)
Attempting to free memory arena while it is still in use.
Could not free memory arena: /usr/local/instant-ngp/dependencies/tiny-cuda-nn/include/tiny-cuda-nn/gpu_memory.h:486 cuMemRelease(hand
le) failed with error CUDA_ERROR_INVALID_VALUE
14:52:52 ERROR Uncaught exception: /usr/local/instant-ngp/dependencies/tiny-cuda-nn/include/tiny-cuda-nn/gpu_memory.h:584 cuMemCre
ate(&m_handles.back(), n_bytes_to_allocate, &prop, 0) failed with error CUDA_ERROR_OUT_OF_MEMORY
robci@robci-B250-Gaming-Infinite-A-MS-B915: /usr/local/instant-ngp$
```

Attempting to free memory arena while it is still in use. Could not free memory arena: /usr/local/instant-ngp/dependencies/tiny-cuda-nn/include/tiny-cuda-nn/gpu_memory.h:486 cuMemRelease(handle) failed with error CUDA_ERROR_INVALID_VALUE 14:52:52 ERROR Uncaught exception: /usr/local/instant-ngp/dependencies/tiny-cuda-nn/include/tiny-cuda-nn/gpu_memory.h:584 cuMemCreate(&m_handles.back(), n_bytes_to_allocate, &prop, 0) failed with error CUDA_ERROR_OUT_OF_MEMORY

Librerías sugeridas al instalar gcc8:

gcc-8-locales g++-8-multilib gcc-8-doc libstdc++6-8-dbg gcc-8-multilib libgcc1-dbg libgomp1-dbg libitm1-dbg libatomic1-dbg libasan5-dbg liblsan0-dbg libtsan0-dbg libubsan1-dbg libmpx2-dbg libquadmath0-dbg libstdc++-8-doc

Para arreglar el error de guardado de las RGBA slices modificar la siguiente línea:


```

1451
1452     if (m_testbed_mode == ETestbedMode::Nerf) {
1453         ImGui::SameLine();
1454         if (ImGui::ColoredButton("Save RGBA PNG sequence", 0.2f)) {
1455             auto effective_view_dir = flip_y_and_z_axes ? Vector3f{0.0f, 1.0f, 0.0f} : Vector3f{0.0f, 0.0f, 1.0f};
1456             // Depth of 0.01f is arbitrarily chosen to produce a visually interpretable range of alpha values
1457             // Alternatively, if the true transparency of a given voxel is desired, one could use the voxel
1458             // the voxel diagonal, or some form of expected ray length through the voxel, given random direction
1459             GPUmemory<Array4f> rgba = get_rgba_on_grid(res3d, effective_view_dir, true, 0.01f);
1460             auto dir = m_data_path / "rgba_slices";
1461             if (!dir.exists()) {
1462                 fs::create_directory(dir);
1463             }
1464             save_rgba_grid_to_png_sequence(rgba, dir.str().c_str(), res3d, flip_y_and_z_axes);
1465         }
1466         if (ImGui::ColoredButton("Save raw volumes", 0.4f)) {
1467             auto effective_view_dir = flip_y_and_z_axes ? Vector3f{0.0f, 1.0f, 0.0f} : Vector3f{0.0f, 0.0f, 1.0f};
1468             auto old_local = m_render_aabb_to_local;
1469             auto old_aabb = m_render_aabb;
1470             m_render_aabb_to_local = Eigen::Matrix3f::Identity();
1471             auto dir = m_data_path / "volume_raw";
1472             if (!dir.exists()) {
1473                 fs::create_directory(dir);
1474             }
1475             for (int cascade = 0; (1<<cascade)<= m_aabb.diag().x()+0.5f; ++cascade) {
1476                 float radius = (1<<cascade) * 0.5f;
1477                 m_render_aabb = BoundingBox(Eigen::Vector3f::Constant(0.5f-radius), Eigen::Vector3f::Constant(0.5f+radius));
1478                 // Dump raw density values that the user can then convert to alpha as they please.
1479                 GPUmemory<Array4f> rgba = get_rgba_on_grid(res3d, effective_view_dir, true, 0.0f, true);
1480                 save_rgba_grid_to_raw_file(rgba, dir.str().c_str(), res3d, flip_y_and_z_axes, cascade);
1481             }
1482             m_render_aabb_to_local = old_local;
1483             m_render_aabb = old_aabb;
1484         }
1485     }

```

Eliminando el elemento “/” y lo que viene después, de manera que no cree un nuevo directorio con el nombre transforms.json previniendo la creación del mismo puesto que ya existe el archivo. No he encontrado otra manera mejor de hacerlo debido a errores de compilación si se usan otros métodos. Lo malo es que se escriben todas las imágenes en el mismo directorio de trabajo.

```

1452     if (m_testbed_mode == ETestbedMode::Nerf) {
1453         ImGui::SameLine();
1454         if (ImGui::ColoredButton("Save RGBA PNG sequence", 0.2f)) {
1455             auto effective_view_dir = flip_y_and_z_axes ? Vector3f{0.0f, 1.0f, 0.0f} : Vector3f{0.0f, 0.0f, 1.0f};
1456             // Depth of 0.01f is arbitrarily chosen to produce a visually interpretable range of alpha values
1457             // Alternatively, if the true transparency of a given voxel is desired, one could use the voxel
1458             // the voxel diagonal, or some form of expected ray length through the voxel, given random direction
1459             GPUmemory<Array4f> rgba = get_rgba_on_grid(res3d, effective_view_dir, true, 0.01f);
1460             auto dir = m_data_path;
1461             if (!dir.exists()) {
1462                 fs::create_directory(dir);
1463             }
1464             save_rgba_grid_to_png_sequence(rgba, dir.str().c_str(), res3d, flip_y_and_z_axes);
1465         }
1466         if (ImGui::ColoredButton("Save raw volumes", 0.4f)) {
1467             auto effective_view_dir = flip_y_and_z_axes ? Vector3f{0.0f, 1.0f, 0.0f} : Vector3f{0.0f, 0.0f, 1.0f};
1468             auto old_local = m_render_aabb_to_local;
1469             auto old_aabb = m_render_aabb;
1470             m_render_aabb_to_local = Eigen::Matrix3f::Identity();
1471             auto dir = m_data_path / "volume_raw";
1472             if (!dir.exists()) {
1473                 fs::create_directory(dir);
1474             }
1475             for (int cascade = 0; (1<<cascade)<= m_aabb.diag().x()+0.5f; ++cascade) {
1476                 float radius = (1<<cascade) * 0.5f;
1477                 m_render_aabb = BoundingBox(Eigen::Vector3f::Constant(0.5f-radius), Eigen::Vector3f::Constant(0.5f+radius));
1478                 // Dump raw density values that the user can then convert to alpha as they please.
1479                 GPUmemory<Array4f> rgba = get_rgba_on_grid(res3d, effective_view_dir, true, 0.0f, true);
1480                 save_rgba_grid_to_raw_file(rgba, dir.str().c_str(), res3d, flip_y_and_z_axes, cascade);
1481             }
1482             m_render_aabb_to_local = old_local;
1483             m_render_aabb = old_aabb;
1484         }
1485     }

```

No se soluciona. La solución buena en realidad es dejarlo como estaba. Se puede conseguir generar el set de imágenes que contienen la información RGBA del mesh lanzando directamente el archivo que contiene el entrenamiento de la red directamente.

```
sudo ./instant-ngp data/dataset_zapatillaUSB/transforms_base.ingp
```

En este caso la carpeta “rgba_slices” se genera directamente en la carpeta de instant-ngp.