NeRFStudio on Windows for Gaussian Splatting: A Practical Case Study

Abstract

This document reports on the process of setting up and using NeRFStudio on Windows for 3D reconstruction from images, with a particular focus on Gaussian Splatting (splatfacto) and its practical limitations on a Windows toolchain. It documents the full workflow—from environment setup and data preprocessing with COLMAP, through NeRF training with nerfacto, to export and visualization—and analyses the technical obstacles encountered when attempting Gaussian Splatting on Windows with CUDA 11.8 and modern PyTorch/gsplat versions.

1. Objective

The primary goals of this work were:

- **G1**: Install and configure NeRFStudio on Windows using a reproducible Python/Conda environment.
- **G2**: Process a set of captured images into a NeRFStudio dataset using COLMAP.
- **G3**: Train both a NeRF model (nerfacto) and a Gaussian Splat model (splatfacto) on the same dataset.
- **G4**: Export the resulting 3D representation for visualization in Blender.
- **G5**: Evaluate the practical feasibility of Gaussian Splatting on Windows and identify more sustainable alternatives.

2. Environment and Toolchain

Operating system: Windows 10

Package manager: Conda (via Miniconda/Anaconda Prompt)

Python environment: python=3.10 in a dedicated Conda env nerfstudio

Core libraries:

- PyTorch with CUDA 11.8 wheels (cu118)
- NeRFStudio
- gsplat (for Gaussian Splatting, later found to be problematic)

External tools:

- ffmpeg (for video/image handling in ns-process-data)
- COLMAP (SfM/MVS, installed via conda-forge)
- NVIDIA CUDA Toolkit 11.8
- Microsoft Visual Studio Build Tools (2022; 2026 was initially installed but incompatible with CUDA 11.8)

Target applications:

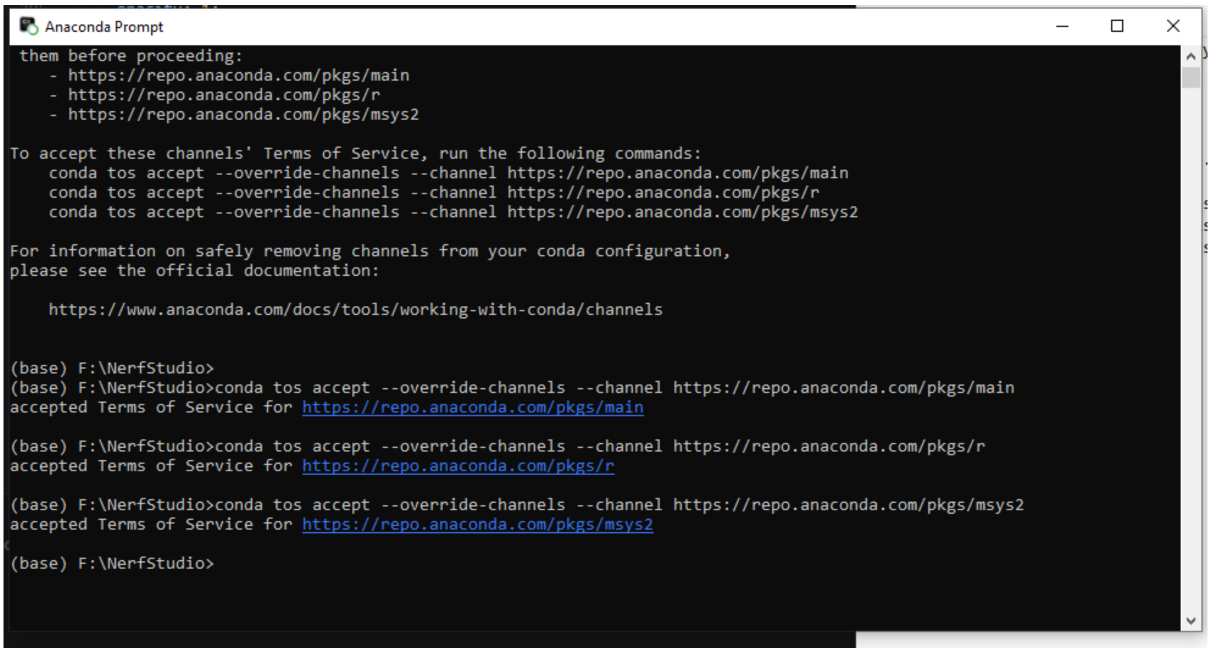
- NeRFStudio (ns-* CLI tools)
- Blender (for point cloud / mesh visualization)

3. NeRFStudio Installation

3.1 Conda environment setup

1. Install Miniconda / Anaconda Prompt.
2. Due to initial issues with default channels, Conda license terms were accepted explicitly and main channels configured:

```
conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/main

conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/r

conda tos accept --override-channels –channel
https://repo.anaconda.com/pkgs/msys2
```

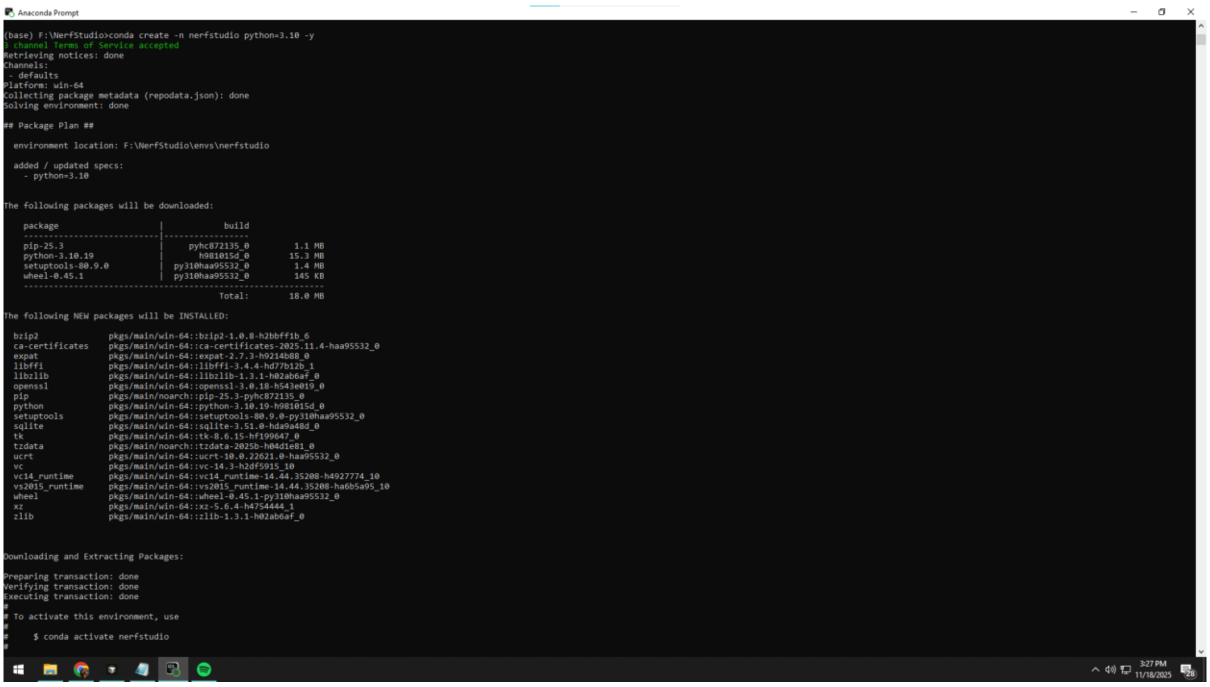


```
them before proceeding:
   - https://repo.anaconda.com/pkgs/main
   - https://repo.anaconda.com/pkgs/r
   - https://repo.anaconda.com/pkgs/msys2

To accept these channels' Terms of Service, run the following commands:
    conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/main
    conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/r
    conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/msys2

For information on safely removing channels from your conda configuration,
please see the official documentation:

    https://www.anaconda.com/docs/tools/working-with-conda/channels


(base) F:\NerfStudio>
(base) F:\NerfStudio>conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/main
accepted Terms of Service for https://repo.anaconda.com/pkgs/main

(base) F:\NerfStudio>conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/r
accepted Terms of Service for https://repo.anaconda.com/pkgs/r

(base) F:\NerfStudio>conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/msys2
accepted Terms of Service for https://repo.anaconda.com/pkgs/msys2

(base) F:\NerfStudio>
```

3. Create and activate the NeRFStudio environment:

conda create -n nerfstudio python=3.10 -y

conda activate nerfstudio



```
(base) F:\NerfStudio>conda create -n nerfstudio python=3.10 -y
3 channel Terms of Service accepted
Retrieving notices: done
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: F:\NerfStudio\envs\nerfstudio

  added / updated specs:
    - python=3.10


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    pip-25.3                   |     pyhc872135_0         1.1 MB
    python-3.10.19             |       h981015d_0        15.3 MB
    setuptools-80.9.0          |  py310haa95532_0         1.4 MB
    wheel-0.45.1               |  py310haa95532_0         145 KB
    ------------------------------------------------------------
                                           Total:        18.0 MB

The following NEW packages will be INSTALLED:

  bzip2              pkgs/main/win-64::bzip2-1.0.8-h2bbff1b_6
  ca-certificates    pkgs/main/win-64::ca-certificates-2025.11.4-haa95532_0
  expat              pkgs/main/win-64::expat-2.7.3-h9214b88_0
  libffi             pkgs/main/win-64::libffi-3.4.4-hd77b12b_1
  libzlib            pkgs/main/win-64::libzlib-1.3.1-h02ab6af_0
  openssl            pkgs/main/win-64::openssl-3.0.18-h543e019_0
  pip                pkgs/main/noarch::pip-25.3-pyhc872135_0
  python             pkgs/main/win-64::python-3.10.19-h981015d_0
  setuptools         pkgs/main/win-64::setuptools-80.9.0-py310haa95532_0
  sqlite             pkgs/main/win-64::sqlite-3.51.0-hda9a48d_0
  tk                 pkgs/main/win-64::tk-8.6.15-hf199647_0
  tzdata             pkgs/main/noarch::tzdata-2025b-h04d1e81_0
  ucrt               pkgs/main/win-64::ucrt-10.0.22621.0-haa95532_0
  vc                 pkgs/main/win-64::vc-14.3-h2df5915_10
  vc14_runtime       pkgs/main/win-64::vc14_runtime-14.44.35208-h4927774_10
  vs2015_runtime     pkgs/main/win-64::vs2015_runtime-14.44.35208-ha6b5a95_10
  wheel              pkgs/main/win-64::wheel-0.45.1-py310haa95532_0
  xz                 pkgs/main/win-64::xz-5.6.4-h4754444_1
  zlib               pkgs/main/win-64::zlib-1.3.1-h02ab6af_0



Downloading and Extracting Packages:

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate nerfstudio
#
```

3.2 PyTorch + CUDA 11.8

PyTorch with CUDA 11.8 was installed from the official wheel index:

pip install torch torchvision --index-url https://download.pytorch.org/whl/cu118

3.3 NeRFStudio and dependencies

NeRFStudio was then installed via pip:

pip install nerfstudio

Later, for Gaussian Splatting experiments, gsplat was added:

pip install -U nerfstudio gsplat

4. Data Preprocessing with COLMAP

4.1 Dataset

- Raw images directory: F:\NerfData\church_raw
- Processed NeRFStudio dataset target: F:\NerfData\church_processed

Initial attempts with .cr3 files failed due to NeRFStudio reporting **"No usable images in the data folder."** The pipeline succeeded after converting or using .JPG images.

4.2 Required tools: ffmpeg and COLMAP

NeRFStudio's ns-process-data reported missing external dependencies:

1. **ffmpeg**

- Downloaded from https://www.gyan.dev/ffmpeg/builds/
- Added ffmpeg to the system PATH.

2. **COLMAP**

- Initial manual installs (e.g., 3.13, 3.8, 3.9) conflicted with NeRFStudio's expectations, specifically:
- colmap feature_extractor ... --SiftExtraction.use_gpu 1
- Older COLMAP binaries did not recognize --SiftExtraction.use_gpu.

Final working approach:

```
conda install -c conda-forge colmap=3.8
```

COLMAP was verified via:

```
colmap -h
```

4.3 Running ns-process-data

With dependencies available, the dataset was successfully processed:

```
ns-process-data images \
  --data "F:\NerfData\church_raw" \
  --output-dir "F:\NerfData\church_processed"
```

The logs confirmed successful COLMAP feature extraction, matching, bundle adjustment, and intrinsics refinement, with poses found for all images.

5. Attempted Gaussian Splatting (splatfacto)

5.1 Initial training attempt

The Gaussian Splatting pipeline was started as:

```
ns-train splatfacto --data "F:\NerfData\church_processed"
```

Initial errors:

- **"gsplat: No CUDA toolkit found. gsplat will be disabled."**

→ Resolved by installing CUDA Toolkit 11.8.

- After installing CUDA, gsplat attempted to JIT-compile CUDA extensions but failed with errors around:
- Missing cl.exe (Microsoft compiler)
- PyTorch's C++ extension tools unable to locate a supported Visual Studio toolchain

5.2 Visual Studio Build Tools and compiler setup

To provide cl.exe:

1. Install **Visual Studio Build Tools** (2026 initially, then 2022).

2. Initialize the environment in the same shell as the Conda env:

conda activate nerfstudio

call "C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\VC\Auxiliary\Build\vcvars64.bat"

where cl

This allowed cl.exe to be found.

5.3 CUDA 11.8 vs compiler version

Despite the above, builds still failed during gsplat CUDA compilation with errors like:

- #error: -- unsupported Microsoft Visual Studio version! Only the versions between 2017 and 2022 (inclusive) are supported!
- Later, with newer toolchains and gsplat versions:
- namespace "cooperative_groups" has no member "labeled_partition"
- static assertion failed with "error STL1002: Unexpected compiler version, expected CUDA 12.4 or newer."

Key implications:

- The installed **gsplat version targets CUDA ≥ 12.4** and uses APIs not available in CUDA 11.8.
- MSVC's STL explicitly expects a **CUDA 12.4+** toolchain for this combination.
- On Windows, official PyTorch wheels at the time target CUDA 11.8 (cu118), creating a version mismatch with gsplat's expectations.

5.4 Workarounds attempted

Several workarounds were tried:

- Setting environment variables:

  set TORCH_NVCC_FLAGS=-allow-unsupported-compiler

  set NVCC_FLAGS=-allow-unsupported-compiler

  set TORCH_CUDA_ARCH_LIST=6.1

- Manually editing CUDA's host_config.h to comment out the Visual Studio version checks.

Even with these aggressive steps, gsplat compilation still failed due to deeper incompatibilities (missing CUDA APIs, version assumptions in gsplat, and STL checks).

5.5 Conclusion on splatfacto on Windows

Result: Gaussian Splatting via ns-train splatfacto did not reach a stable, working state on this Windows + CUDA 11.8 + PyTorch cu118 stack.**Root cause**:
A fundamental version mismatch:

- Windows PyTorch officially supports CUDA 11.8 for the used configuration.
- Modern gsplat expects CUDA ≥ 12.4 and a specific MSVC/STL combination.
- Forcing the combination via hacks leads to fragile, failing builds rather than a reliable research setup.

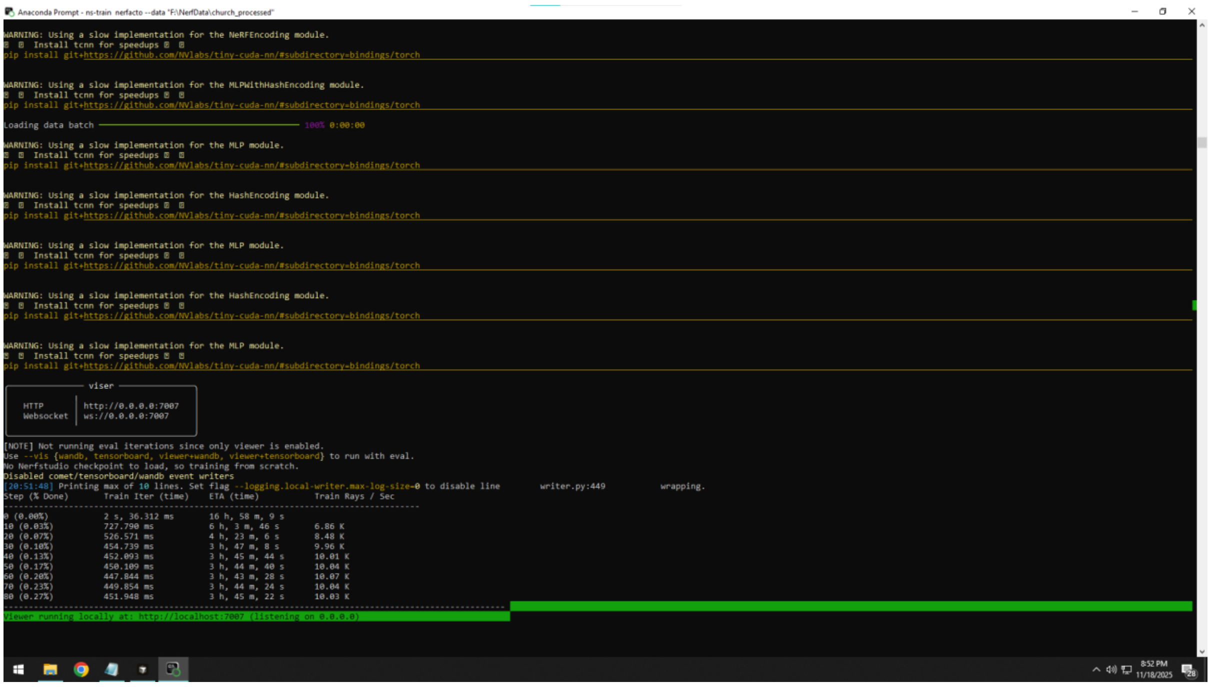
6. NeRF Baseline: nerfacto Training

Given the gsplat limitations, a NeRF baseline was successfully trained using nerfacto:

```
ns-train nerfacto --data "F:\NerfData\church_processed"
```

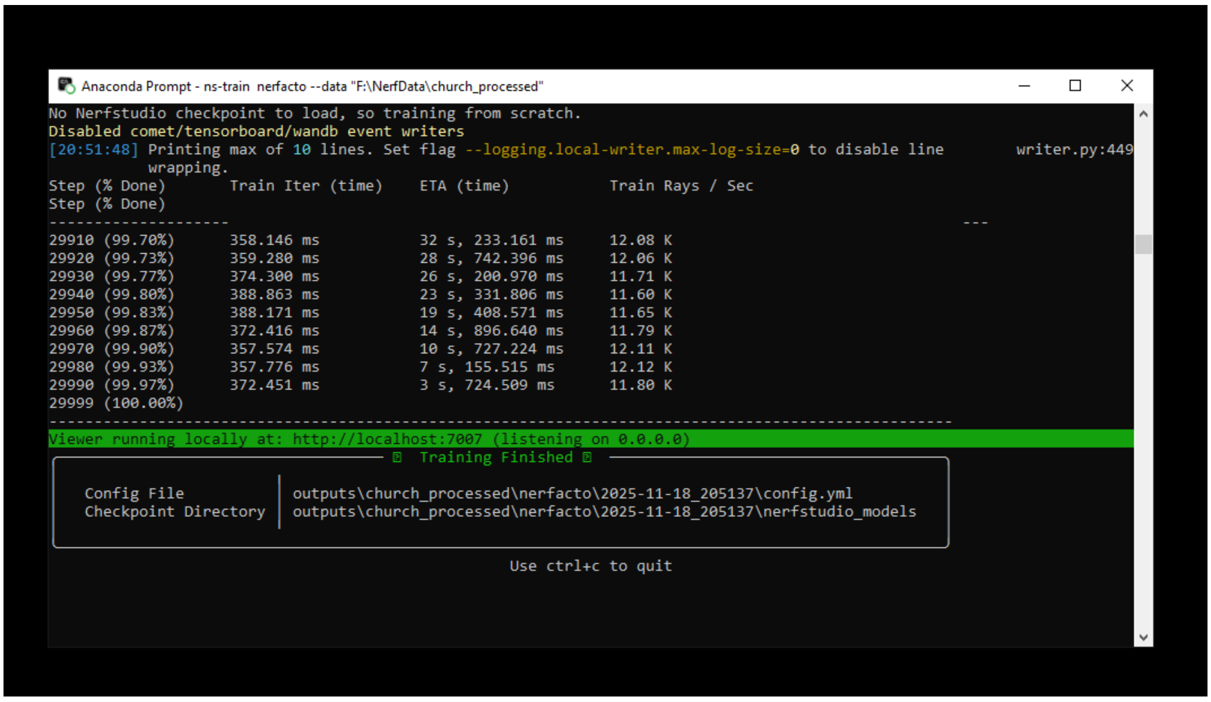
- Training completed successfully (approx. 3.5 hours).
- Checkpoints were stored under:
- outputs\church_processed\nerfacto\2025-11-18_205137\...

This established a working NeRF pipeline on the same dataset, confirming that the core NeRFStudio installation and data preprocessing were correct;
the problems were localized to the Gaussian Splatting extension.



Anaconda Prompt - ns-train nerfacto --data "F:\NerfData\church_processed"

```
WARNING: Using a slow implementation for the NeRFEncoding module.
□ □  Install tcnn for speedups □  □
pip install git+https://github.com/NVlabs/tiny-cuda-nn/#subdirectory=bindings/torch


WARNING: Using a slow implementation for the MLPWithHashEncoding module.
□ □  Install tcnn for speedups □  □
pip install git+https://github.com/NVlabs/tiny-cuda-nn/#subdirectory=bindings/torch

Loading data batch ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00

WARNING: Using a slow implementation for the MLP module.
□ □  Install tcnn for speedups □  □
pip install git+https://github.com/NVlabs/tiny-cuda-nn/#subdirectory=bindings/torch


WARNING: Using a slow implementation for the HashEncoding module.
□ □  Install tcnn for speedups □  □
pip install git+https://github.com/NVlabs/tiny-cuda-nn/#subdirectory=bindings/torch


WARNING: Using a slow implementation for the MLP module.
□ □  Install tcnn for speedups □  □
pip install git+https://github.com/NVlabs/tiny-cuda-nn/#subdirectory=bindings/torch


WARNING: Using a slow implementation for the HashEncoding module.
□ □  Install tcnn for speedups □  □
pip install git+https://github.com/NVlabs/tiny-cuda-nn/#subdirectory=bindings/torch


WARNING: Using a slow implementation for the MLP module.
□ □  Install tcnn for speedups □  □
pip install git+https://github.com/NVlabs/tiny-cuda-nn/#subdirectory=bindings/torch

╭─────────── viser ───────────╮
│                             │
│   HTTP      │ http://0.0.0.0:7007 │
│   Websocket │ ws://0.0.0.0:7007   │
│                             │
╰─────────────────────────────╯
[NOTE] Not running eval iterations since only viewer is enabled.
Use --vis {wandb, tensorboard, viewer+wandb, viewer+tensorboard} to run with eval.
No Nerfstudio checkpoint to load, so training from scratch.
Disabled comet/tensorboard/wandb event writers
[20:51:48] Printing max of 10 lines. Set flag --logging.local-writer.max-log-size=0 to disable line       writer.py:449          wrapping.
Step (% Done)       Train Iter (time)    ETA (time)           Train Rays / Sec
---------------------------------------------------------------------------
0 (0.00%)           2 s, 36.312 ms       16 h, 58 m, 9 s
10 (0.03%)          727.790 ms           6 h, 3 m, 46 s       6.86 K
20 (0.07%)          526.571 ms           4 h, 23 m, 6 s       8.48 K
30 (0.10%)          454.739 ms           3 h, 47 m, 8 s       9.96 K
40 (0.13%)          452.093 ms           3 h, 45 m, 44 s      10.01 K
50 (0.17%)          450.109 ms           3 h, 44 m, 40 s      10.04 K
60 (0.20%)          447.844 ms           3 h, 43 m, 28 s      10.07 K
70 (0.23%)          449.854 ms           3 h, 44 m, 24 s      10.04 K
80 (0.27%)          451.948 ms           3 h, 45 m, 22 s      10.03 K
---------------------------------------------------------------------------
Viewer running locally at: http://localhost:7007 (listening on 0.0.0.0)
```



Anaconda Prompt - ns-train nerfacto --data "F:\NerfData\church_processed"

```
No Nerfstudio checkpoint to load, so training from scratch.
Disabled comet/tensorboard/wandb event writers
[20:51:48] Printing max of 10 lines. Set flag --logging.local-writer.max-log-size=0 to disable line       writer.py:449
           wrapping.
Step (% Done)       Train Iter (time)    ETA (time)           Train Rays / Sec
Step (% Done)
--------------------                                                                            ---
29910 (99.70%)      358.146 ms           32 s, 233.161 ms     12.08 K
29920 (99.73%)      359.280 ms           28 s, 742.396 ms     12.06 K
29930 (99.77%)      374.300 ms           26 s, 200.970 ms     11.71 K
29940 (99.80%)      388.863 ms           23 s, 331.806 ms     11.60 K
29950 (99.83%)      388.171 ms           19 s, 408.571 ms     11.65 K
29960 (99.87%)      372.416 ms           14 s, 896.640 ms     11.79 K
29970 (99.90%)      357.574 ms           10 s, 727.224 ms     12.11 K
29980 (99.93%)      357.776 ms           7 s, 155.515 ms      12.12 K
29990 (99.97%)      372.451 ms           3 s, 724.509 ms      11.80 K
29999 (100.00%)
---------------------------------------------------------------------------------------------------
Viewer running locally at: http://localhost:7007 (listening on 0.0.0.0)
╭──────────────────────────────── □ Training Finished □ ────────────────────────────────╮
│                                                                                        │
│   Config File          │ outputs\church_processed\nerfacto\2025-11-18_205137\config.yml          │
│   Checkpoint Directory │ outputs\church_processed\nerfacto\2025-11-18_205137\nerfstudio_models   │
│                                                                                        │
╰────────────────────────────────────────────────────────────────────────────────────────╯
                                     Use ctrl+c to quit
```

7. Export and Visualization

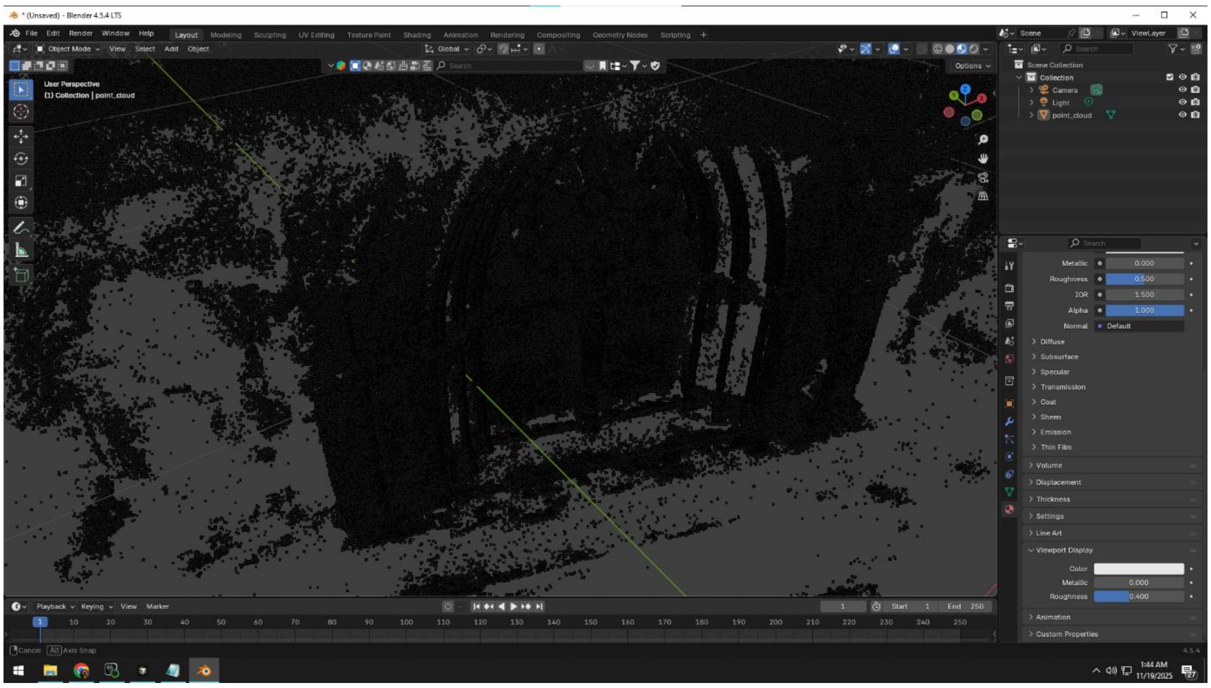
7.1 Gaussian Splat export (not usable)

An attempt was made to export Gaussian splats from the nerfacto model:

```
ns-export gaussian-splat \
  --load-config "outputs\church_processed\nerfacto\2025-11-18_205137\config.yml" \
  --output-dir "F:\NerfData\church_export"
```

This failed because the Gaussian Splatting components were not successfully built/available in the environment.

7.2 Point cloud export



Metallic 0.000
Roughness 0.500
IOR 1.500
Alpha 1.000
Normal Default
Diffuse
Subsurface
Specular
Transmission
Coat
Sheen
Emission
Thin Film
Volume
Displacement
Thickness
Settings
Line Art
Viewport Display
Color
Metallic 0.000
Roughness 0.400
Animation
Custom Properties

Instead, the model was exported as a colored point cloud:

```
ns-export pointcloud \
  --load-config "outputs\church_processed\nerfacto\2025-11-18_205137\config.yml" \
  --output-dir "F:\NerfData\church_export" \
  --num-points 2000000 \
  --normal-method open3d
```

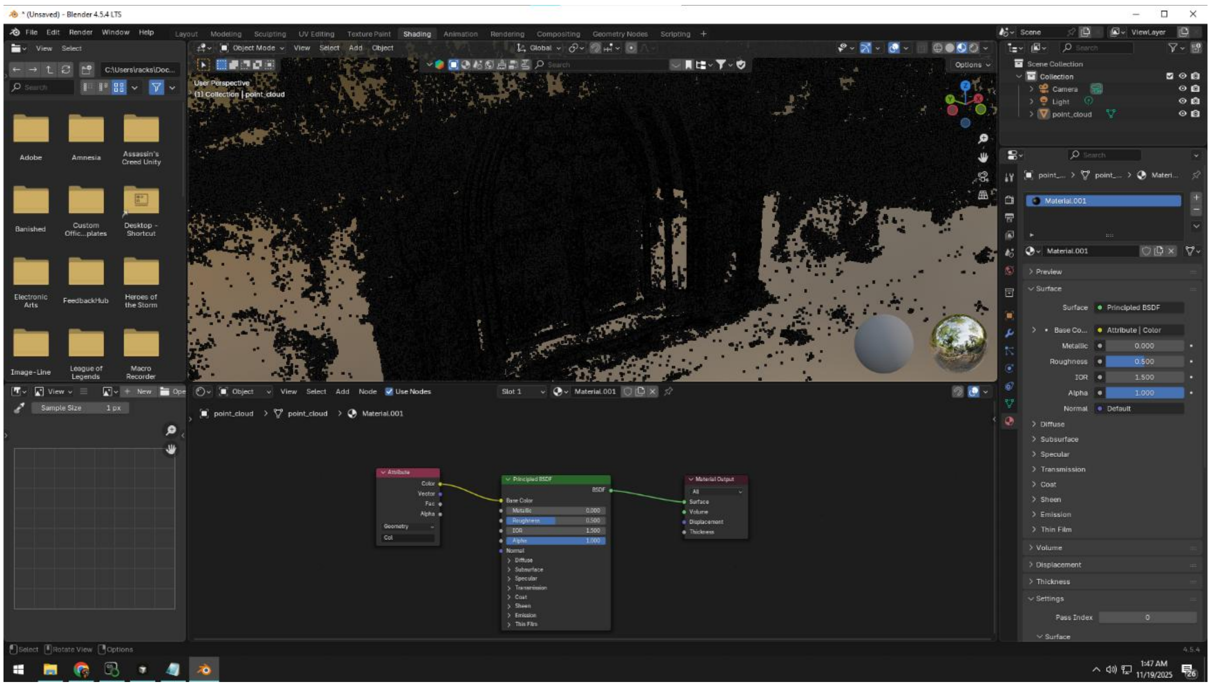
Notes:

- Initial run without --normal-method failed due to missing normal outputs in the pipeline.
- Using --normal-method open3d allowed export with estimated normals.

7.3 Visualization in Blender

- The exported .ply was imported into Blender.
- By default, Blender treats the data as a **point cloud with vertex colors**, not a mesh:

- Materials in Blender primarily shade faces,
  so imported points appear as sparse colored dots
  unless specialized point-cloud rendering or geometry node tricks are used.
- Without additional add-ons or heavy geometry node setups (e.g., instancing tiny spheres per point), the result is visually far from a clean, textured mesh.


* (Unsaved) - Blender 4.5.4 LTS
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Geometry Nodes Scripting
User Perspective
(1) Collection | point_cloud
Scene Collection
Collection
Camera
Light
point_cloud
Material.001
Attribute
Principled BSDF
Material Output
Slot 1
Use Nodes

Surface: Principled BSDF
Base Color: Attribute | Color
Metallic: 0.000
Roughness: 0.500
IOR: 1.500
Alpha: 1.000
Normal: Default

Conclusion:

The NeRFStudio → pointcloud → Blender pipeline works, but by design produces a **dense colored point cloud**, not a production-quality textured mesh.

8. Analysis and Lessons Learned

Version coupling is critical:

- NeRFStudio + PyTorch + CUDA + gsplat + MSVC must all be mutually compatible.
- On Windows, PyTorch's official CUDA 11.8 support does **not** align
  with the gsplat versions that expect CUDA 12.4+.

Windows is a second-class citizen for cutting-edge GS research code:

- Many Gaussian Splatting implementations assume a Linux/Ubuntu + CUDA 12.x toolchain.
- Attempting to replicate this tooling on Windows leads
  to extensive compiler/CUDA friction.

NeRFStudio itself is stable on Windows:

- ns-process-data (with COLMAP 3.8 via conda-forge) and ns-train nerfacto worked reliably.
- The failure points are tied specifically to JIT-compiled gsplat extensions and their dependencies on newer CUDA/toolchains.

Export expectations matter:

- NeRFStudio's pointcloud export is well-suited to research visualization and analysis.
- For a "game-ready" textured mesh, point clouds need further processing (meshing, texturing) or a different toolchain entirely.

9. Alternative Paths to Gaussian Splatting

Given the difficulties with gsplat on Windows, several practical alternatives were identified.

9.1 Desktop Applications (Free, Local Processing)

Postshot by Jawset:

- Cross-platform desktop application (Windows/Mac/Linux) specifically designed for photogrammetry and Gaussian Splatting.
- Workflow: Import images → automatic processing → view and export GS models.
- Free tier available: Provides core Gaussian Splatting functionality without cost.
- Advantage: User-friendly GUI designed for accessibility; runs locally without cloud dependency; significantly easier setup than research-grade tools.
- Website: https://jawset.com/

OpenSplat:

- Open-source Gaussian Splatting tool with improved usability compared to the original Inria implementation.
- Platform support: Windows, Linux, macOS.
- GUI options: Community-developed interfaces and launchers available.

- Advantage: Free and open source;
  more accessible than raw research code while maintaining local control and privacy; does not require cloud services.
- Repository: https://github.com/pierotofy/OpenSplat

9.2 Cloud or Browser-Based Gaussian Splatting

Splattr (web-based):

- Upload a video or image set.
- Their servers run the full Gaussian Splatting reconstruction.
- Exports .ply or similar GS formats that can be explored in dedicated GS viewers or imported into DCC tools with appropriate add-ons.
- Advantage: Completely avoids local CUDA, MSVC, and PyTorch compatibility issues.
- Note: Paid service; pricing varies by project complexity.
- Solenvo3D (web, GS-only, beta):
- Similar concept: upload photos → get a GS scene + viewer and exports.
- Useful when Splattr is unavailable or if a second tool is desired for comparison.

9.3 Hybrid App + Add-On Workflow

KIRI Engine – 3D Gaussian Splatting mode:

- Capture images or video in KIRI (mobile/desktop).
- Choose the dedicated 3D Gaussian Splat pipeline.
- Export GS models compatible with a dedicated Blender add-on.
- Advantage: Provides an end-to-end capture → GS → Blender pipeline without exposing the user to raw CUDA or compiler setup.
- Note: 3D Gaussian Splatting requires a Pro subscription (~$17.99/month or $79.99/year); basic photogrammetry is free.

Luma AI:

- Mobile and web app for NeRF and Gaussian Splatting-like reconstructions.
- Free tier available: Basic captures and exports included.
- Workflow: Record video on phone → cloud processing → view/export.
- Advantage: Very low barrier to entry; no local hardware requirements.

Polycam:

- Mobile/web photogrammetry and 3D scanning app.
- Free tier available: Basic reconstruction and limited exports.
- Advantage: Accessible for quick tests; integrates well with mobile LiDAR on supported devices.

9.4 Future Linux / WSL2 Route

If a more research-oriented, open-source workflow is desired in the future:

- Use WSL2 + Ubuntu + CUDA 12.x (or a native Linux install).
- Then run:
- The official Gaussian Splatting reference implementation (Inria), or
- NeRFStudio + splatfacto built against a CUDA 12.x + compatible PyTorch stack.

This aligns with the environment assumed by most research code and significantly reduces the kind of toolchain conflicts experienced on Windows.

10. Conclusion

This case study demonstrates that:

- **NeRFStudio + COLMAP + nerfacto on Windows** is feasible and can successfully reconstruct and export a dense colored point cloud from a set of images.
- **Gaussian Splatting via splatfacto on Windows with CUDA 11.8** proved **impractical**, primarily due to version conflicts with gsplat's reliance on CUDA 12.4+ and specific MSVC/STL combinations.
- For students or practitioners who want **Gaussian Splatting without deep toolchain work**, the most realistic current options are:
- **Desktop applications**: **Postshot** (free, GUI-based, cross-platform) or **OpenSplat** (open-source, more accessible than research code) for local processing without cloud dependency.
- **Cloud/browser solutions**: **Splattr**, **Solenvo3D**, **Luma AI**, or **Polycam** for offloading computation entirely.
- **Integrated mobile/desktop solutions**: **KIRI Engine 3DGS + Blender add-on** (requires Pro subscription).

- For long-term, **open-source research workflows**, migrating to a **Linux/WSL2 + CUDA 12.x** stack is recommended to access tools like the official Inria Gaussian Splatting implementation or NeRFStudio + splatfacto without version conflicts.

Overall, the project successfully achieved NeRF reconstruction and export, and documented in detail why Gaussian Splatting on this Windows stack remains fragile. More importantly, it identified that **accessible, free desktop tools (Postshot, OpenSplat) exist and should be the first choice** for practitioners seeking a balance between ease of use, local control, and cost, before resorting to either cloud services or complex research toolchains.