

URB3D



Metoda trójwymiarowego modelowania obszarów urbanistycznych z wykorzystaniem metod fotogrametrii



Autorzy: Daniel Borkowski · Julia Farganus^{ID} · Rafał Mielniczuk · Katarzyna Wochal^{ID}

Opiekun: Marek Krótkiewicz

Streszczenie

Celem pracy jest wykonanie aplikacji, która wykorzystuje metody fotogrametrii do modelowania miejskich scen 3D. Dane wejściowe stanowią zdjęcia obszarów miejskich, które są przetwarzane w celu stworzenia modelu 3D, a następnie segmentowane na obiekty przestrzeni miejskiej, takie jak budynki, tereny zielone, itp.. Aplikacja będzie wizualizować model oraz wyniki segmentacji semantycznej.

Innowacyjność tego projektu polega na połączeniu, adaptacji i udoskonaleniu najlepszych dostępnych rozwiązań, takich jak Gaussian Splatting i PointNet, aby stworzyć nowy, kompleksowy produkt.

Przetwarzanie dużych scen miejskich jest wyzwaniem dla obecnie istniejących rozwiązań, które skupiają się głównie na pojedynczych obiektach lub zamkniętych scenach. Typowa scena miejska natomiast może obejmować setki zdjęć, a powstała chmura może zawierać miliony punktów. Dodatkowym wyzwaniem jest niebalansowana reprezentacja kategorii semantycznych. Nasze rozwiązanie ma na celu efektywne przetwarzanie dużych zbiorów danych przy rozsądnym zużyciu zasobów czasowych i pamięciowych.

Zastosowania biznesowe otrzymywanych w ten sposób modeli 3D są szerokie: od gier wideo, przez architekturę, robotykę, pojazdy autonomiczne, po modelowanie urbanistyczne.

1 WPROWADZENIE

Rekonstrukcja, klasyfikacja i wizualizacja scen urbanistycznych to dynamicznie rozwijające się zagadnienie, które zyskało na znaczeniu dzięki rosnącej dostępności nowoczesnych technologii, takich jak LiDAR, oraz postępowi w dziedzinie sztucznej inteligencji. Kluczowym wyzwaniem pozostaje jednak efektywne przetwarzanie ogromnych zbiorów danych – chmur punktów, które nierzadko obejmują miliony elementów. Choć na rynku istnieją liczne programy i algorytmy wspierające tego typu analizy, ich skuteczne wykorzystanie w praktyce bywa problematyczne, głównie z uwagi na skalę i złożoność danych urbanistycznych.

Rozwiązanie będzie umożliwiało przeprowadzenie rekonstrukcji do modelu trójwymiarowego na podstawie odpowiednio przygotowanego zbioru zdjęć, klasyfikację otrzymanej sceny na zbiór pre-definiowanych klas istotnych w kontekście urbanistycznym, oraz wizualizację wykonanych obliczeń.

Poniżej znajdują się cele, które zostaną zrealizowane w przedsięwzięciu:

1. skomponowanie własnego zbioru danych,
2. wykorzystanie algorytmu Gaussian Splatting do rekonstrukcji sceny 3D,
3. filtracja chmury punktów przy użyciu różnych technik,
4. zastosowanie architektur sieci neuronowych takich jak PointNet do klasyfikacji chmury punktów,
5. adaptacja istniejących bibliotek do wizualizacji wyników,
6. implementacja własnego algorytmu do renderowania gaussianów,

2 STAN WIEDZY

Unikalność projektu wynika z połączenia wielu rozwiązań które istnieją samodzielnie na rynku. Algorytm Structure-from-Motion [8] jest popularną fotogrametryczną techniką uzyskiwania chmury punktów ze zbioru zdjęć i jego implementacja oferowana jest m. in. przez oprogramowanie COLMAP.

W przypadku modelu 3D często stosowaną techniką są siatki, ale ich wadą jest niekompatybilność z algorytmami sztucznej inteligencji. Popularne są też rozwiązania wykorzystujące sieci neuronowe jak

np. NeRF [5], jednak długi czas trenowania, osiągający nawet parę dni, jest nieefektywny. Z tego powodu zdecydowano się na wykorzystanie algorytmu Gaussian Splatting [2], który buduje model sceny z tzw. gaussianów, które można interpretować jako punkty rozmyte. Istniejące adaptacje do skali urbanistycznej tego algorytmu to np. CityGaussian [4].

Ważnym krokiem jest również filtracja chmury punktów [1] w celu usunięcia odstających punktów lub tych nieistotnych dla wyników klasyfikacji. W tym obszarze znajdują się np. techniki statystyczne czy oparte na sąsiedztwie.

Istniejące architektury sieci neuronowych dla zadania segmentacji są głównie przeznaczone dla scen zamkniętych lub pojedynczych obiektów. Popularnym rozwiązaniem jest PointNet [6] oraz jego następnik PointNet++ [7] oparte na wielowarstwowym perceptronie, jak i również bardziej skomplikowane rozwiązania jak KPConv [9] wykorzystujące konwolucje. Wyzwaniem dla projektu jest dostosowanie takich architektur do chmury punktów o wielkości rzędu milionów punktów.

W przypadku renderowania istnieją rozwiązania przeznaczone zarówno do chmur punktów jak i do splatów, zaimplementowane często przy pomocy WebGL, jak [3]. Wyzwanie stanowi jednak wydajne i efektywne przedstawienie milionów elementów, co wymaga skorzystania z GPU i niskopoziomowego pisania kodu.

3 WYNIKI

3.1 Wytworzony oprogramowanie

Modularność projektu sprawia, że wygodniej jest omówić niezależnie każdą z części. Dla jasności, przebieg całego procesu jest przedstawiony na schemacie 1. Każda funkcjonalność, jak i wizualizacje wyników poszczególnych zadań są dostępne poprzez interfejs użytkownika.

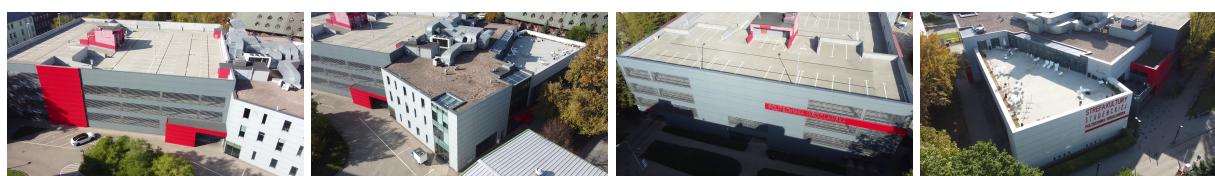


Rysunek 1: Przebieg procesu

3.2 Akwizycja danych

W projekcie założono wykorzystanie metod fotogrametrycznych do tworzenia trójwymiarowych modeli obszarów urbanistycznych. Za część projektu przyjęto z tego względu również pozyskanie własnych zestawów danych fotograficznych (fotogramów), które spełniałyby wymogi techniczne, umożliwiające późniejszą rekonstrukcję 3D. Niezbędne było wykonanie dużej liczby ujęć, obejmujących wiele kątów i perspektyw oraz zapewnienie odpowiedniego nakładania się zdjęć dla poprawnego działania oprogramowania fotogrametrycznego, które identyfikuje i dopasowuje wspólne punkty widoczne na wielu zdjęciach.

Akwizycję zrealizowano na kampusie **Politechniki Wrocławskiej**, koncentrując się na budynkach **C5**, **C7** oraz Strefie Kultury Studenckiej (**SKS**) z i pozyskując zdjęcia zarówno z lotów bezzałogowym statkiem powietrznym, jak i z poziomu gruntu. Stanowią one kompletne zbiory danych, które spełniły wymogi jakościowe i posłużyły do budowy testowych modeli.



Rysunek 2: Przykładowe zdjęcia z akwizycji danych przedstawiające SKS

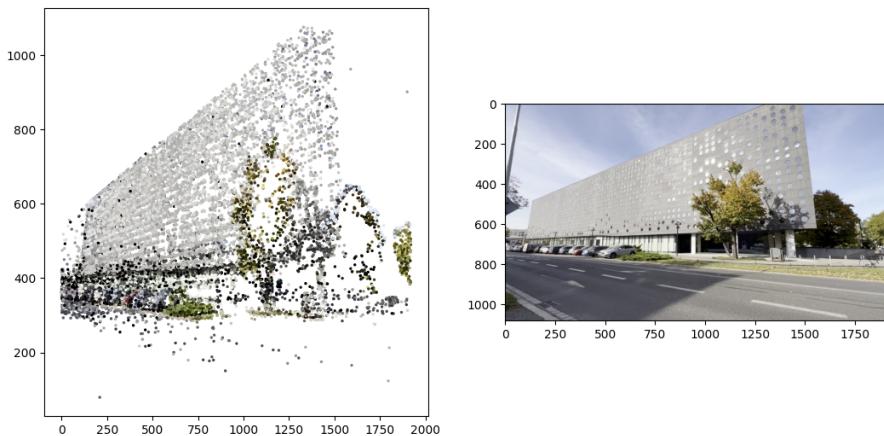
3.3 Structure from motion

Kolejnym etapem projektu było wykorzystanie techniki *Structure from Motion* (SfM) do wyznaczania struktur przestrzennych scen na podstawie dobranych zestawów zdjęć dwuwymiarowych.

Algorytmy SfM, identyfikując i łącząc punkty wspólne między zdjęciami, ustalają zarówno rozmieszczenie tych punktów w przestrzeni, jak i pozycje i orientacje kamer, z których wykonano zdjęcia. Proces ten pozwala na oszacowanie struktury trójwymiarowej sfotografowanego obszaru, czyli wygenerowanie chmury punktów odwzorowującej scenę w postaci zbioru punktów 3D o przypisanych kolorach, tak jak to pokazuje ilustracja 3.

Do realizacji tego zadania użyto popularnego narzędzia COLMAP, a konkretnie jego wersji w formie biblioteki *pycolmap*, oferującej funkcjonalności m.in. do wykrywania charakterystycznych cech na obrazach, łączenia punktów wspólnych na zdjęciach czy prowadzenia rekonstrukcji sceny 3D na podstawie dopasowań między nimi.

Uzyskana chmura punktów jest dodatkowo poddawana filtracji z wykorzystaniem metod opartych na analizie sąsiedztwa każdego punktu. Proces ten pozwala na eliminację szumów poprzez usunięcie punktów, które nie wpisują się w lokalne wzorce przestrzenne. Tak przygotowana trójwymiarowa reprezentacja sceny służy za podstawę do modelowania z zastosowaniem algorytmu Gaussian Splatting.



Rysunek 3: Projekcja przykładowej chmury punktów na płaszczyznę porównana do zdjęcia

3.4 Gaussian Splatting

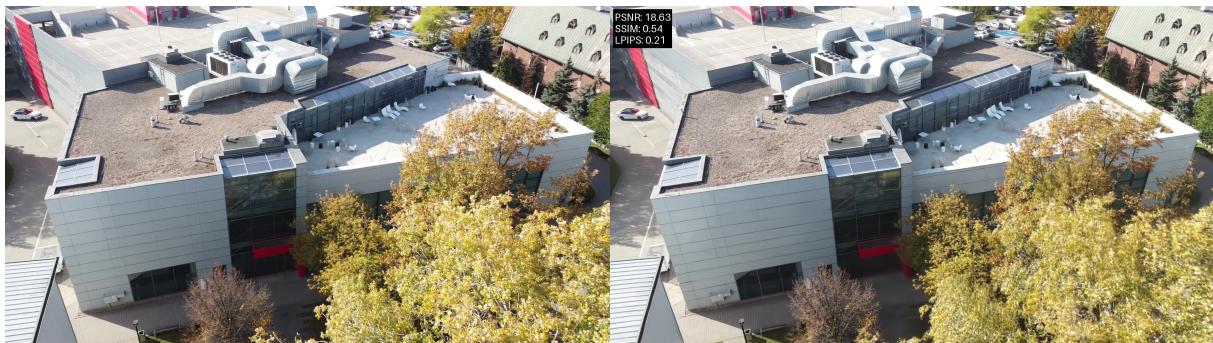
Przy pomocy biblioteki *gsplat* [10] zawierającej implementację *Gaussian Splatting* w Pythonie wykonane zostały eksperymenty polegające na uruchomieniu algorytmu dla różnych wartości hiperparametrów w celu znalezienia wartości które prowadzą do jak najbardziej optymalnego procesu trenowania w kontekście czasu trwania i wykorzystania pamięci.

Na wejściu algorytmu podawana jest chmura punktów, która jest bazą do dalszego dzielenia i powstawania gaussianów, a ich parametry: pozycja, kolor, skala i rotacja są optymalizowane przy pomocy metody spadku wzdłuż gradientu. Metryki przyjęte do oceny jakości to **SSIM** (Structural Similarity Index Measure), **PSNR** (Peak Signal-to-Noise Ratio) oraz **LPIPS** (Learned Perceptual Image Patch Similarity).

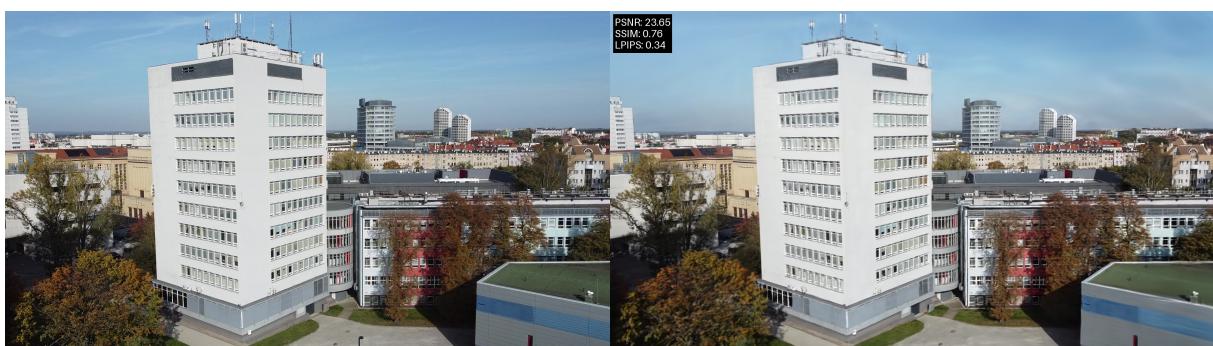
Wykonane eksperymenty pokazały, że najważniejsze parametry dla przebiegu procesu to:

1. liczba Gaussianów: w przypadku scen urbanistycznych w celu oddania odpowiedniej szczegółowości potrzebne jest kilka milionów Gaussianów,
2. strategia i częstość adaptacji: określają w jaki sposób oraz jak często dodawane i usuwane są Gaussiany,
3. liczba iteracji: zwykle im dłużej trenowana jest scena tym wyniki są lepsze, jednak zależy to również od przyjętej strategii. Liczba ta wpływa bezpośrednio na czas trenowania, powinna wynieść nie mniej niż kilkanaście tysięcy,
4. stopień zmiennych harmonicznych: wyrażają one kolor, im większy stopień tym lepsza jakość sceny, ale też zwiększone zużycie pamięci i wydłużony czas trenowania.

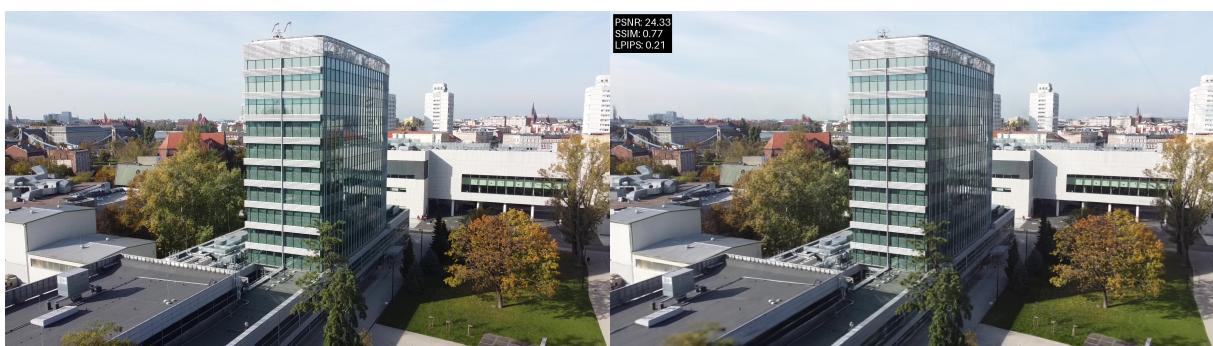
Ilustracje 4, 5 oraz 6 przedstawiają przykładowe wizualizacje (od lewej do prawej: prawdziwe zdjęcie i widok modelu), a tabela 1 zawiera wybrane wielkości dla scen testowych. Renderowania zostały wykonane przy pomocy biblioteki *nerfview* która służy do wizualizacji splatów.



Rysunek 4: Scena SKS



Rysunek 5: Scena C5



Rysunek 6: Scena C7

scena	PSNR	SSIM	LPIPS	liczba gaussianów	czas trenowania	pamięć pliku (MB)
SKS	22.03	0.71	0.25	2,937,549	2h53m	661
C5	21.98	0.71	0.26	4,564,464	10h40m	675
C7	22.63	0.72	0.29	3,000,000	15h15m	675

Tabela 1: Całościowe metryki dla testowych scen. Otrzymane wartości PSNR, SSIM oraz LPIPS zwykle świadczą o dobrej jakości scenie, która oddaje wystarczające szczegóły i wygładzone artefakty.

3.5 Segmentacja

3.6 Wizualizacja

Wizualizacja modeli 3D stanowi wyzwanie dla użytkownika końcowego, głównie z powodu braku spójnych platform umożliwiających realizację całego procesu – od wgrania plików wejściowych po interakcję

z modelem – w ramach jednej aplikacji. Dostępne na rynku rozwiązania wymagają korzystania z aplikacji trzecich i pewnej wiedzy technicznej, co prowadzi do problemów z integracją i spójnością działania.

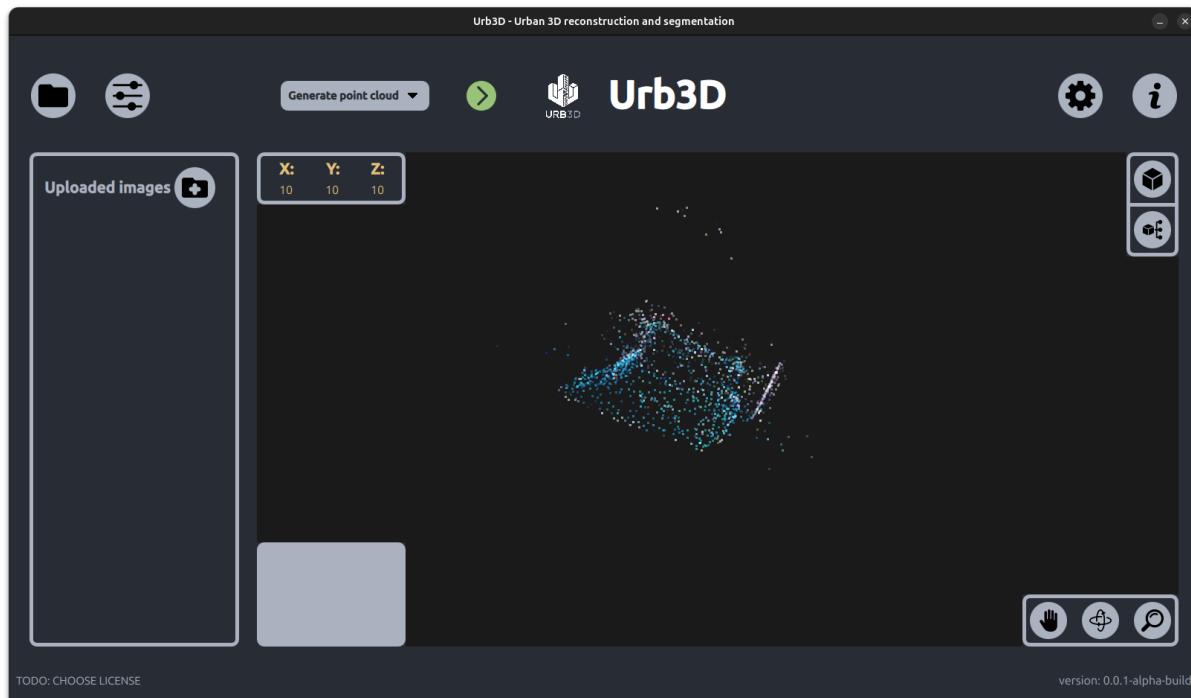
Celem projektu było stworzenie intuicyjnego, dynamicznego i responsywnego **interfejsu** zintegrowanego z wydajnym **renderingiem** GPU. Aplikacja umożliwia użytkownikowi końcowemu realizację pełnego procesu wizualizacji – od tworzenia i modyfikacji modelu po jego segmentację i wyświetlanie – w jednej aplikacji. Projekt rozwiązuje problem fragmentarycznej funkcjonalności dostępnych aplikacji, oferując spójne środowisko do obsługi modeli 3D.

Korzyści z realizacji interfejsu

- zwiększa wydajność dzięki GPU,
- eliminacja konieczności korzystania z wielu narzędzi,
- uproszczony proces użytkowania, co zwiększa dostępność aplikacji dla mniej zaawansowanych użytkowników.

Rendering wykorzystuje plik .ply jako dane wejściowe do wczytania splatów. Są one renderowane jako sześciiany, w których wnętrzu generowane są shadery, bazujące na skalowaniu i rotacji splatów. Takie podejście umożliwia abstrakcyjne przedstawienie splatów przy jednoczesnym zachowaniu wysokiej dokładności wizualnej.

Interfejs widoczny na ilustracji 7 został zaimplementowany w QML, PyQt, natomiast rendering w technologiach C, OpenGL oraz OpenCL, przedstawiony na zdjęciu 8. Tymczasową wizualizację chmury wykonywaliśmy przy pomocy biblioteki VisPy.



Rysunek 7: Zrzut ekranu przedstawiający główny widok aplikacji

4 PODSUMOWANIE

Rekonstrukcja i klasyfikacja krajobrazów urbanistycznych ma wiele potencjalnych zastosowań w dziedzinach takich jak *Smart City* czy też *Virtual Reality*. Projekt "urb3d" pokazał, że wykonanie takiego oprogramowania jest możliwe przy pomocy integracji istniejących rozwiązań i ich adaptacji.

4.1 Wnioski

Zaprojektowane oprogramowanie zapewnia intuicyjne korzystanie z funkcjonalności takich jak dostosowywanie parametrów, wczytywanie zdjęć, uruchamianie poszczególnych etapów oraz przeglądanie rezultatów. Odpowiednio dobrane algorytmy zapewniają jakościowe wyniki, które mogą być dalej wykorzystane razem lub oddzielnie.



Rysunek 8: Zrzut ekranu przedstawiający własny renderer

4.2 Kierunki rozwoju

Większe obszary / Rekonstrukcji? W przypadku algorytmu Gaussian Splatting możliwe by było zastosowanie technik kompresji zmniejszających końcową liczbę gaussianów, lub trenowanie na różnych poziomach szczegółowości (ang. *LoD*). Segmentacja? Rendering?

4.3 Podziękowania

??

LITERATURA

- [1] Xian-Feng Han, Jesse S. Jin, Ming-Jie Wang, Wei Jiang, Lei Gao, and Liping Xiao. A review of algorithms for filtering the 3d point cloud. *Signal Processing: Image Communication*, 57:103–112, 2017.
- [2] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023.
- [3] Kevin Kwok. Splat viewer.
- [4] Yang Liu, He Guan, Chuanchen Luo, Lue Fan, Junran Peng, and Zhaoxiang Zhang. Citygaussian: Real-time high-quality large-scale scene rendering with gaussians, 2024.
- [5] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.
- [6] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2016.
- [7] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017.
- [8] Johannes L. Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [9] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds, 2019.

- [10] Vickie Ye, Ruilong Li, Justin Kerr, Matias Turkulainen, Brent Yi, Zhuoyang Pan, Otto Seiskari, Jianbo Ye, Jeffrey Hu, Matthew Tancik, and Angjoo Kanazawa. gsplat: An open-source library for Gaussian splatting. *arXiv preprint arXiv:2409.06765*, 2024.