Special Topics in A2: Neural Geometry

- Marching Cubes
- NERFs
- Gaussian Splats

Technically not neural…
But still a powerful algorithm

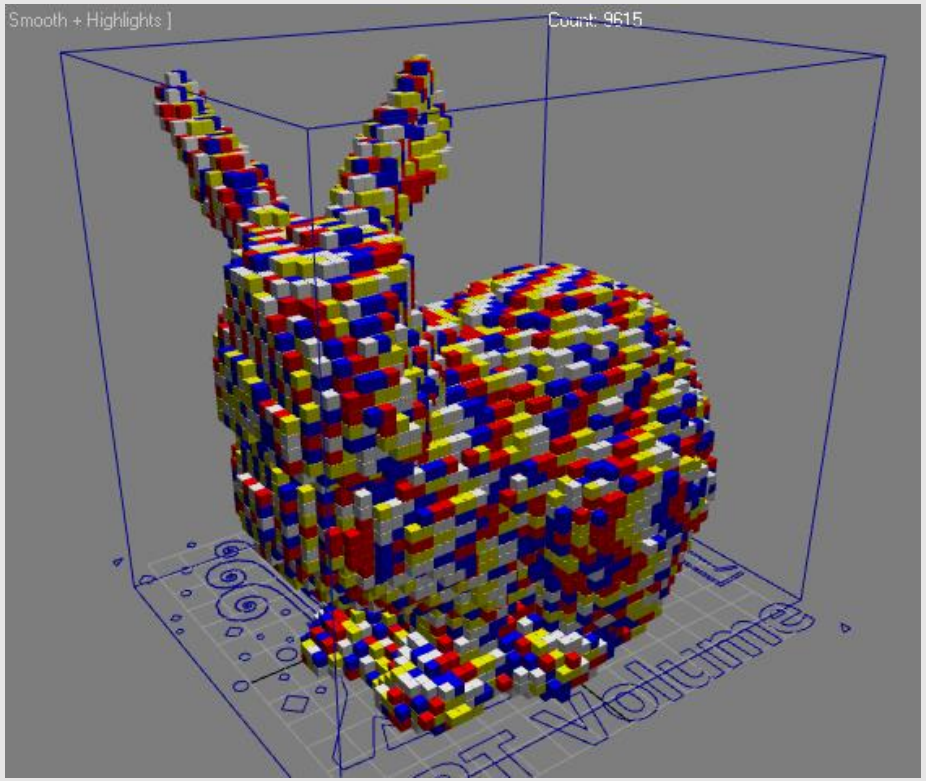
How do we convert implicit geometry to explicit geometry?

Voxel Grid

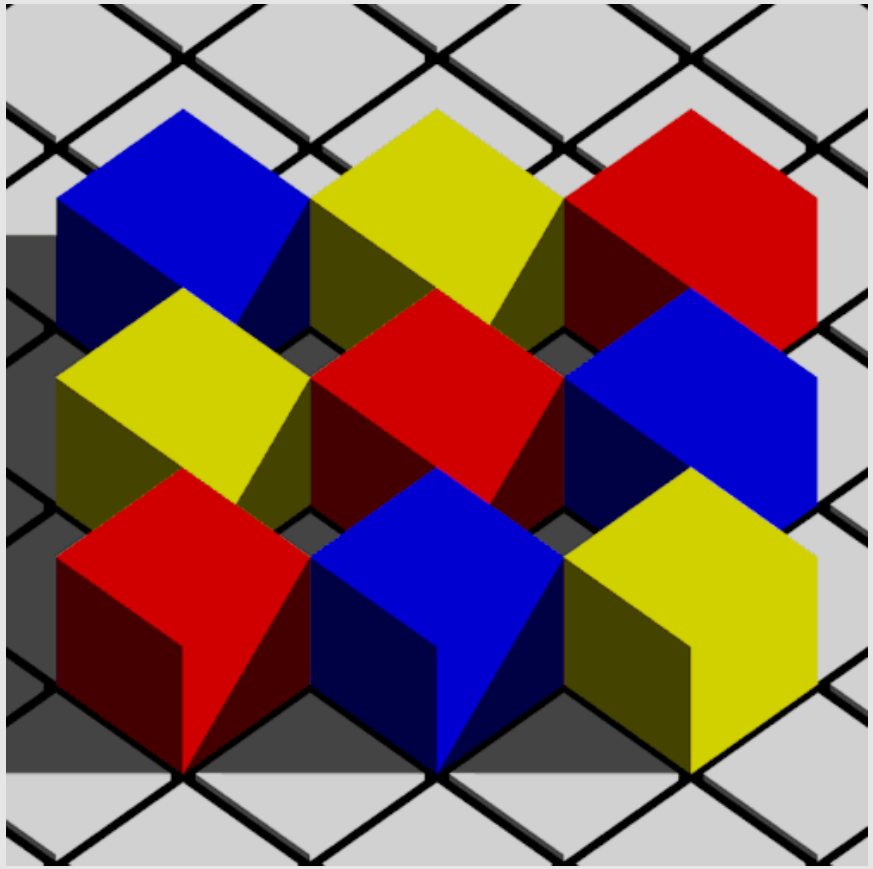
- **Idea:** for an implicit function $f(x, y, z)$, sample points uniformly along the function's domain
  - Plot points where $f(x, y, z) = 0$
  - Result is a point cloud

- **Issue:** how many samples to take
  - More samples lead to higher precision, but are more expensive and memory intensive

- **Issue:** no info on connectivity
  - Difficult to interpolate data



Marching Cubes



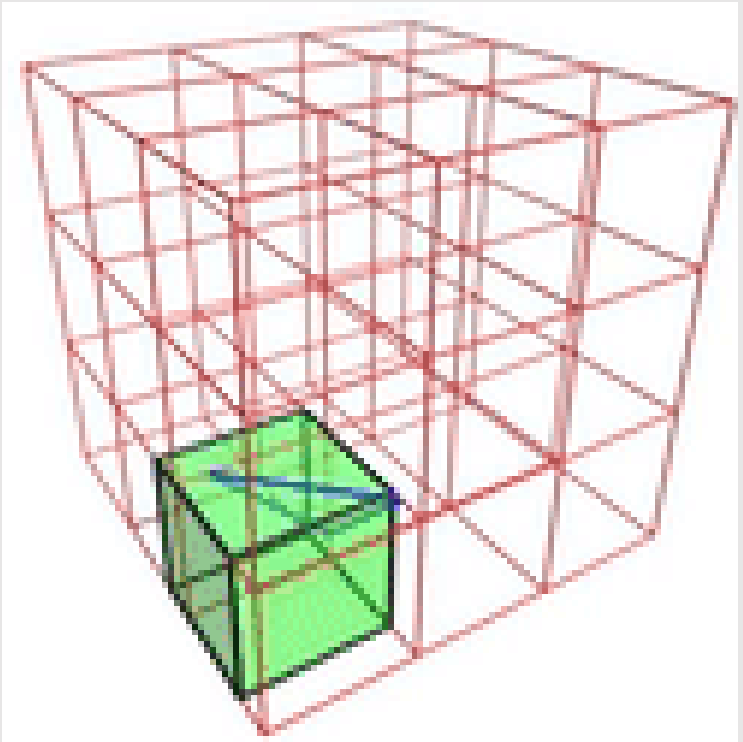
This is not the marching cubes algorithm.
This is literally cubes marching.

- Marching cubes is an algorithm for converting implicit geometry to explicit
  - Adds both positional (vertices) and connectivity (edges)

Marching Cubes

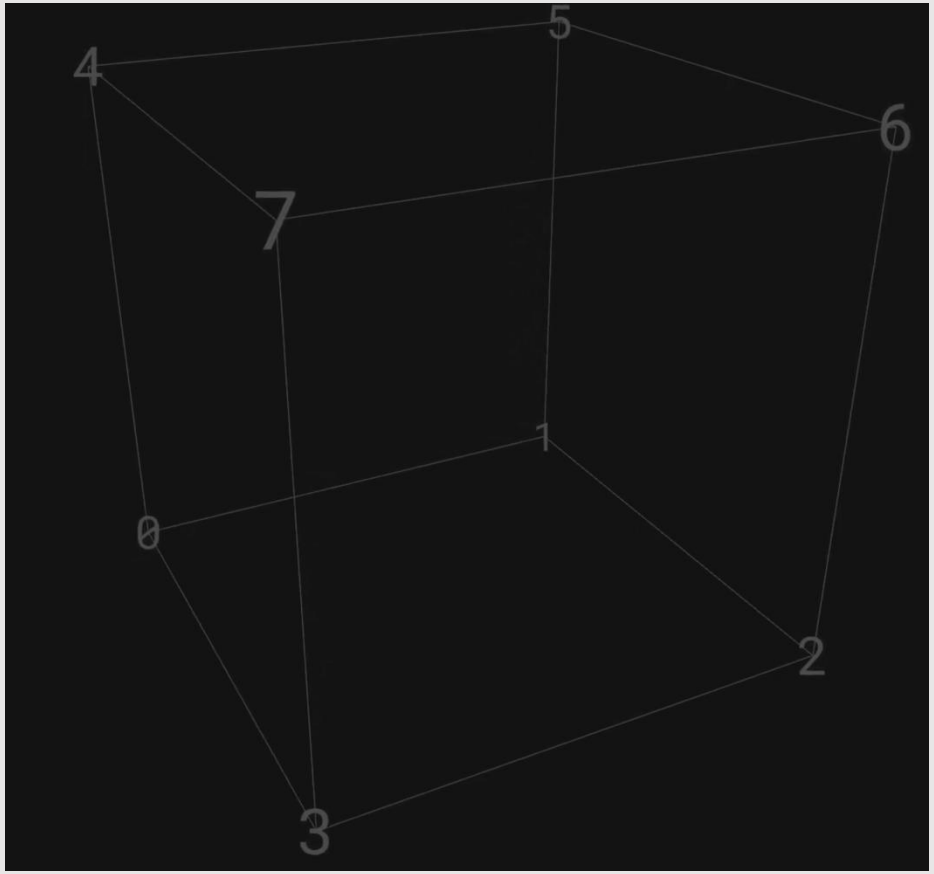
- **Idea:** march a cube though the scene, checking if the cube vertices lie inside or outside the implicit function $f(x, y, z)$
  - 8 vertices, 8 checks
  - Encoded as an 8-bit number
  - Generate geometry where inside vertices are enclosed by the geometry and outside vertices are kept out

- **Issue:** how big of a cube to use
  - A smaller cube leads to finer details
  - A smaller cube requires more samples



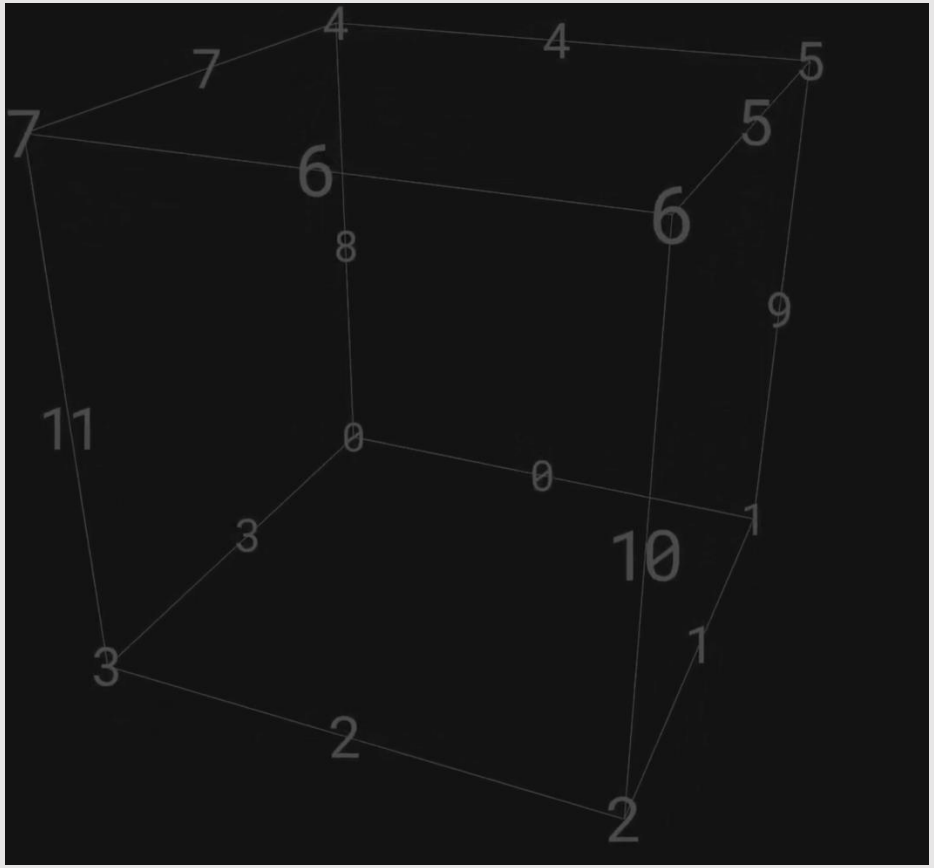
Marching Cubes Vertices

- Each cube has 8 vertices
- Check if each vertex lies inside or outside the implicit function $f(x, y, z)$
  - Can be encoded as an 8-bit number
    - 1 – inside
    - 0 - outside



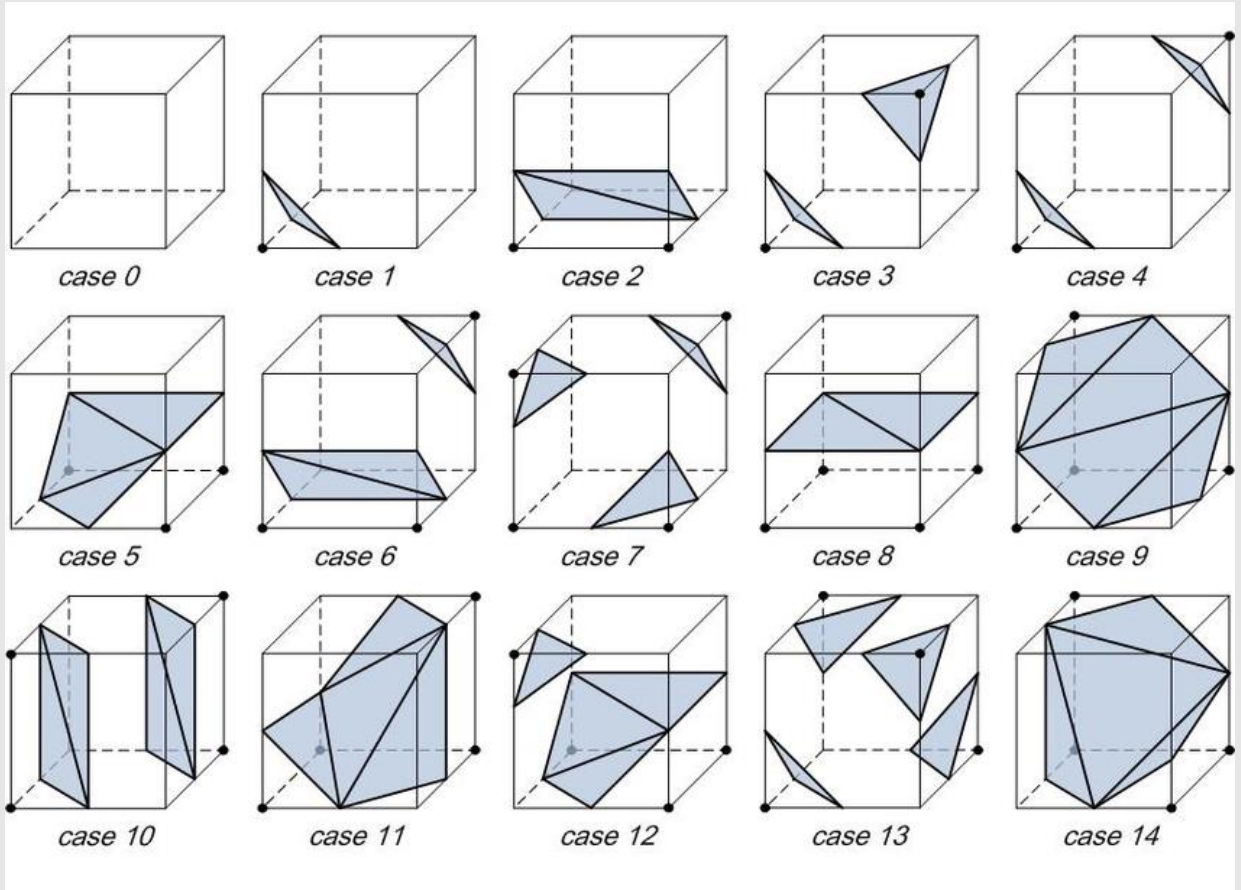
Marching Cubes Edges

- Each cube has 12 edges
- **Goal:** create geometry with vertices along the edges of the cube that enclose inside vertices and excludes outside vertices



Marching Cubes Geometry

- $2^8 = 256$ possible configurations
- How do we know which one to use?



Marching Cubes Lookup Table

```
// For each MC case, a list of triangles, specified as triples of edge indices, terminated by -1
const TriangleTable = [
        [ -1 ],
        [ 0, 3, 8, -1 ],
        [ 0, 9, 1, -1 ],
        [ 3, 8, 1, 1, 8, 9, -1 ],
        [ 2, 11, 3, -1 ],
        [ 8, 0, 11, 11, 0, 2, -1 ],
        [ 3, 2, 11, 1, 0, 9, -1 ],
        [ 11, 1, 2, 11, 9, 1, 11, 8, 9, -1 ],
        [ 1, 10, 2, -1 ],
        [ 0, 3, 8, 2, 1, 10, -1 ],
        [ 10, 2, 9, 9, 2, 0, -1 ],
        [ 8, 2, 3, 8, 10, 2, 8, 9, 10, -1 ],
        [ 11, 3, 10, 10, 3, 1, -1 ],
        [ 10, 0, 1, 10, 8, 0, 10, 11, 8, -1 ],
        [ 9, 3, 0, 9, 11, 3, 9, 10, 11, -1 ],
        [ 8, 9, 11, 11, 9, 10, -1 ],
        [ 4, 8, 7, -1 ],
        [ 7, 4, 3, 3, 4, 0, -1 ],
        [ 4, 8, 7, 0, 9, 1, -1 ],
        [ 1, 4, 9, 1, 7, 4, 1, 3, 7, -1 ],
        [ 8, 7, 4, 11, 3, 2, -1 ],
        [ 4, 11, 7, 4, 2, 11, 4, 0, 2, -1 ],
        [ 0, 9, 1, 8, 7, 4, 11, 3, 2, -1 ],
        [ 7, 4, 11, 11, 4, 2, 2, 4, 9, 2, 9, 1, -1 ],
        [ 4, 8, 7, 2, 1, 10, -1 ],
        [ 7, 4, 3, 3, 4, 0, 10, 2, 1, -1 ],
        [ 10, 2, 9, 9, 2, 0, 7, 4, 8, -1 ],
        [ 10, 2, 3, 10, 3, 4, 3, 7, 4, 9, 10, 4, -1 ],
        [ 1, 10, 3, 3, 10, 11, 4, 8, 7, -1 ],
        [ 10, 11, 1, 11, 7, 4, 1, 11, 4, 1, 4, 0, -1 ],
        [ 7, 4, 8, 9, 3, 0, 9, 11, 3, 9, 10, 11, -1 ],
        [ 7, 4, 11, 4, 9, 11, 9, 10, 11, -1 ],
        [ 9, 4, 5, -1 ],
        [ 9, 4, 5, 8, 0, 3, -1 ],
        [ 4, 5, 0, 0, 5, 1, -1 ],
        [ 5, 8, 4, 5, 3, 8, 5, 1, 3, -1 ],
```

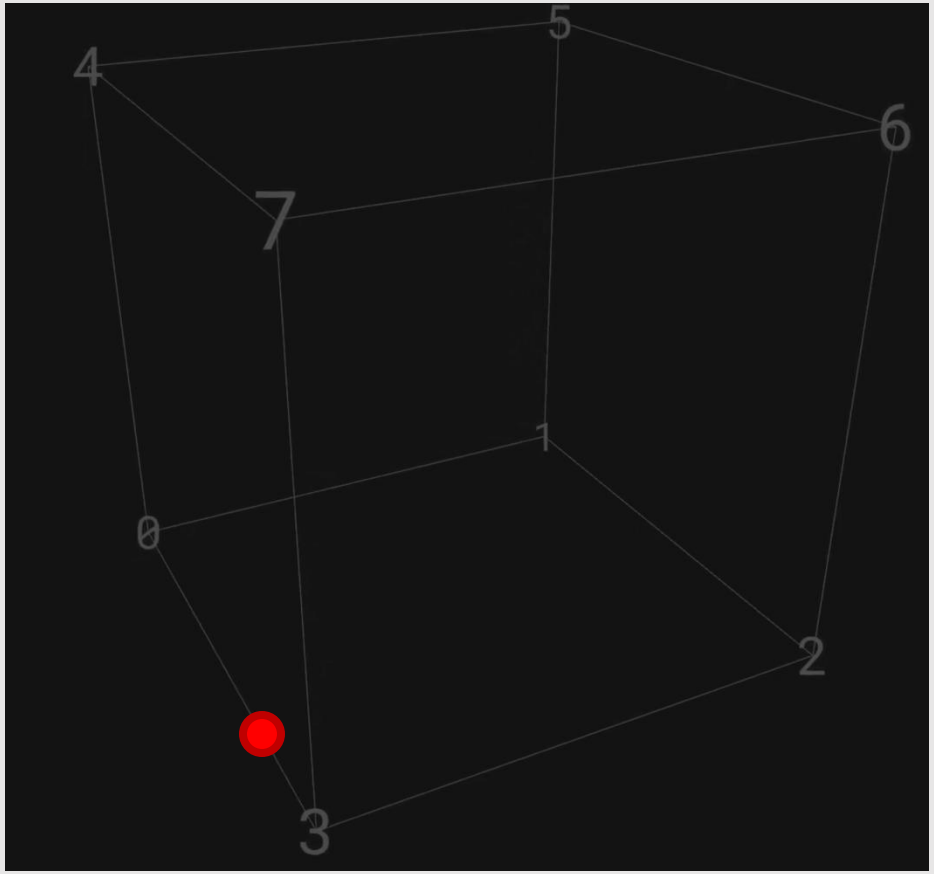
- Just use a lookup table!

Marching Cubes Linear Interpolation

- **Issue:** lookup table only tells us on what edges to place vertices and how to connect them
  - Does not say the location of vertices

- Linearly interpolate them on the edges depending on the evaluated values on the cube vertices

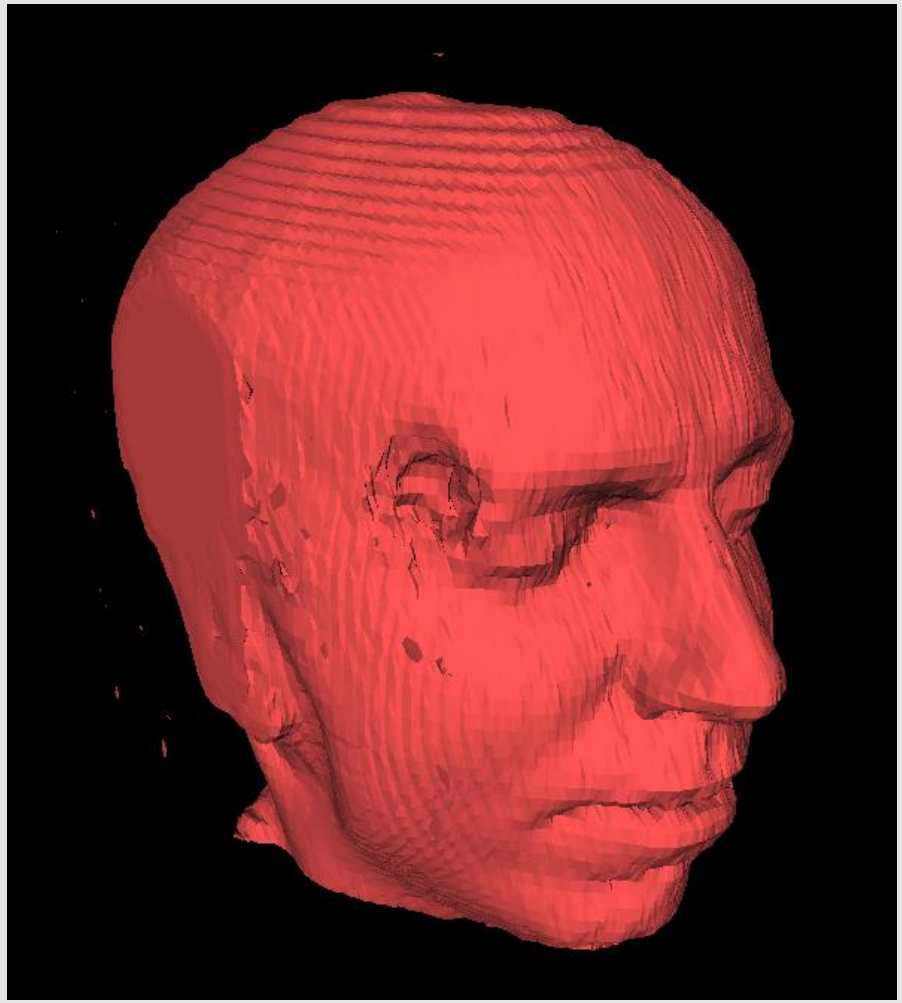
- Example:
  - $f(x_0, y_0, z_0) = -0.75$
  - $f(x_3, y_3, z_3) = +0.25$
    - Vertex is placed ¼ distance away from corner 3, ¾ distance from corner 0



Marching Cubes Examples

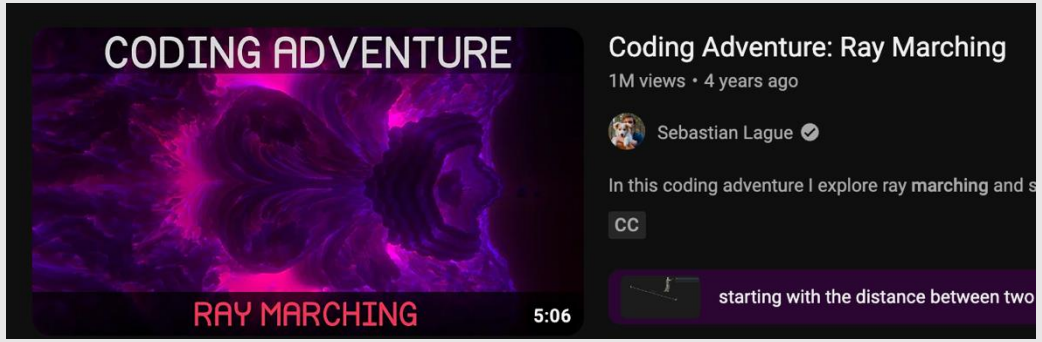
- **Issue:** very cube-like
  - Easy to see cube artifacts

- How to fix?
  - Run refinement
  - Run denoising
  - Run remeshing



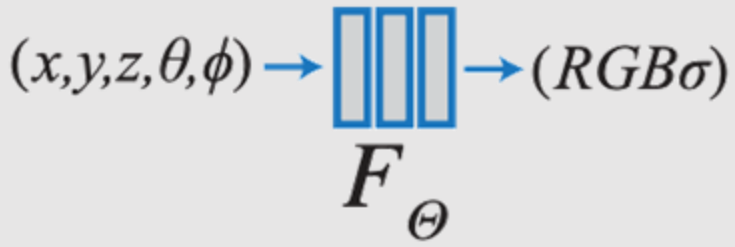
Marching Cubes Application

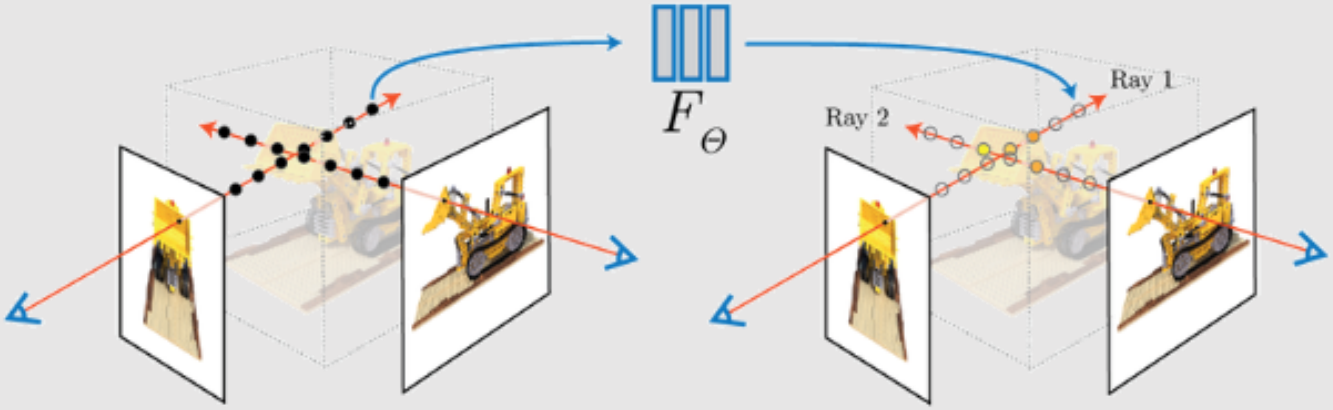
- Terrain generation
- Implicitly generate terrain with algebraic surfaces and noise
  - Convert to explicit mesh for easy rendering



- ~~Marching Cubes~~
- NERFs
- Gaussian Splats

Neural Radiance Fields



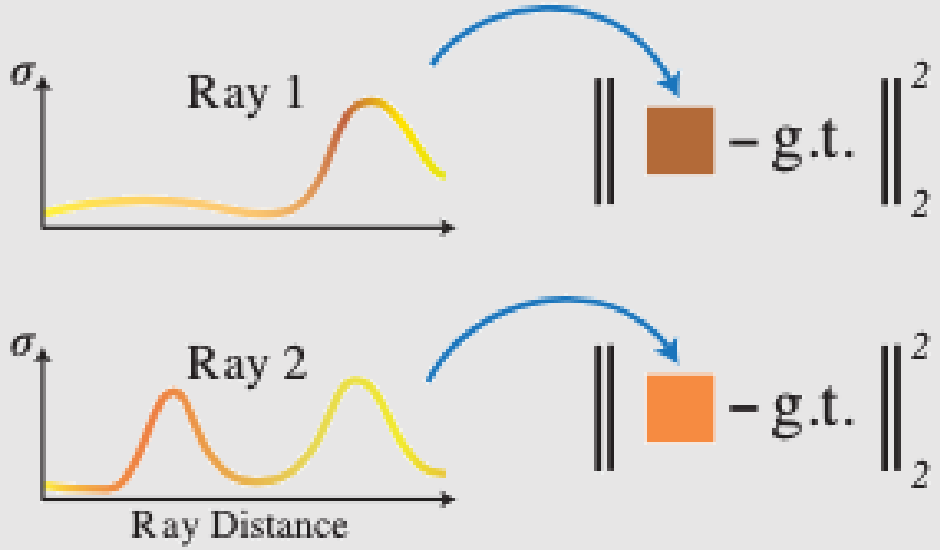


NERF (2020) B. Mildenhall

- Train neural network F on multiple images
  - Training data: (x,y,z) of pixel + view angle + RGB
    - Need depth data!
  - Test data: (x,y,z) of requested pixel + view angle
    - Outputs RGB
    - What if we don't know the z we want?
      - Just grab the nearest z
      - 0-bounce ray tracing
  - Training requires multiple images from different view angles around a fixed origin
    - Fixed origin reduces DOFs
    - Hence only 2 DOF with view angle

Neural Radiance Fields

- We are building a field of radiance values
  - In case that wasn't clear :)
- Output RGB depends on depth
  - Model uses known (depth, RGB) pairs along a ray to learn interpolation of unseen RGB values
- **Key Assumption:** lighting should remain constant between scenes
  - **Q: What if lighting is not constant?**



NERF (2020) B. Mildenhall

View-Dependent Appearances

- Some changes in lighting are inevitable
  - Mirror reflections
  - Glass refractions
  - Anisotropic materials
- **Idea:** treat these as normal pixels
  - View-dependent lighting will get baked into the model
- **Ex:** specular highlights
  - When requesting RGB at specific angles, we get back those specular components

NERF (2020) B. Mildenhall

Extra Features

NERF (2020) B. Mildenhall

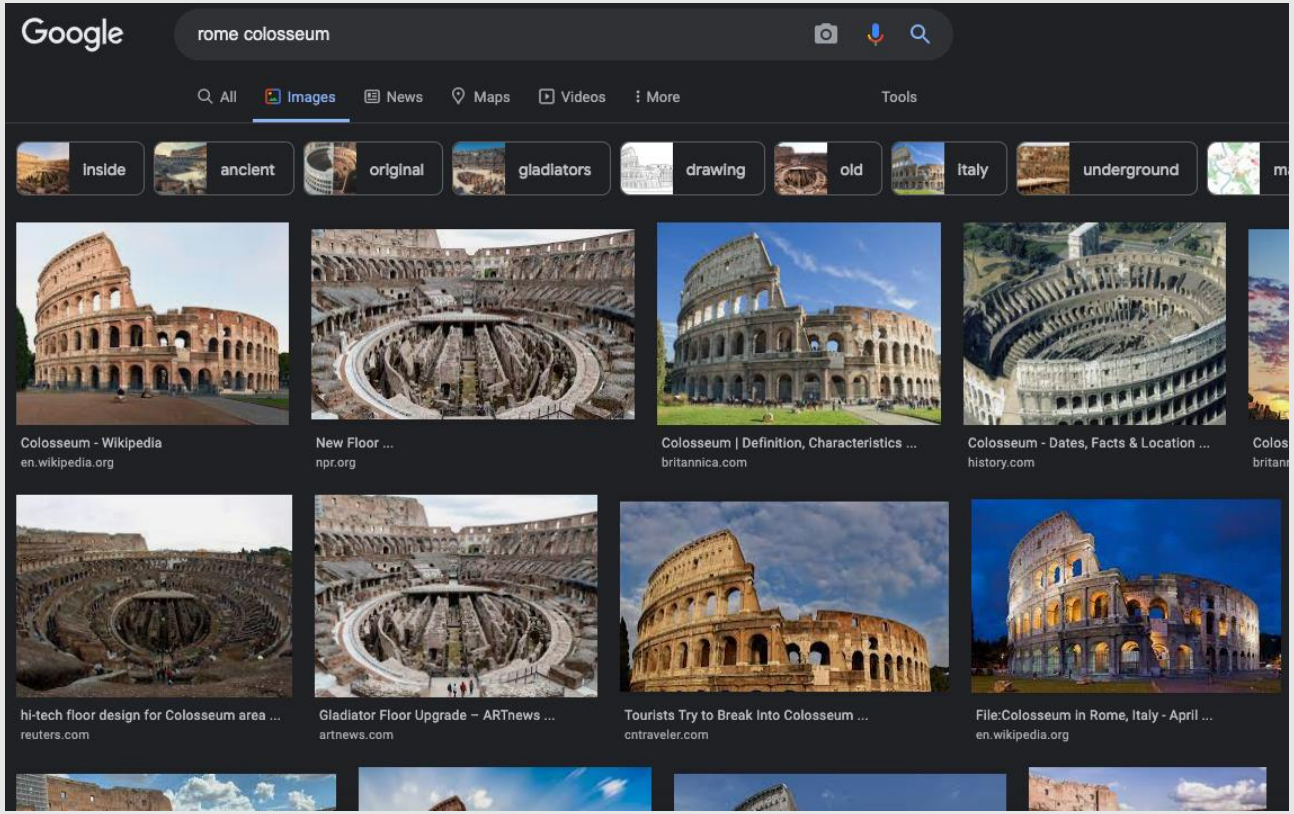
- What can we use NERFs for?
  - Depth Maps
  - Image Relighting
  - Object insertion

https://www.matthewtancik.com/nerf

NeRF In The Wild

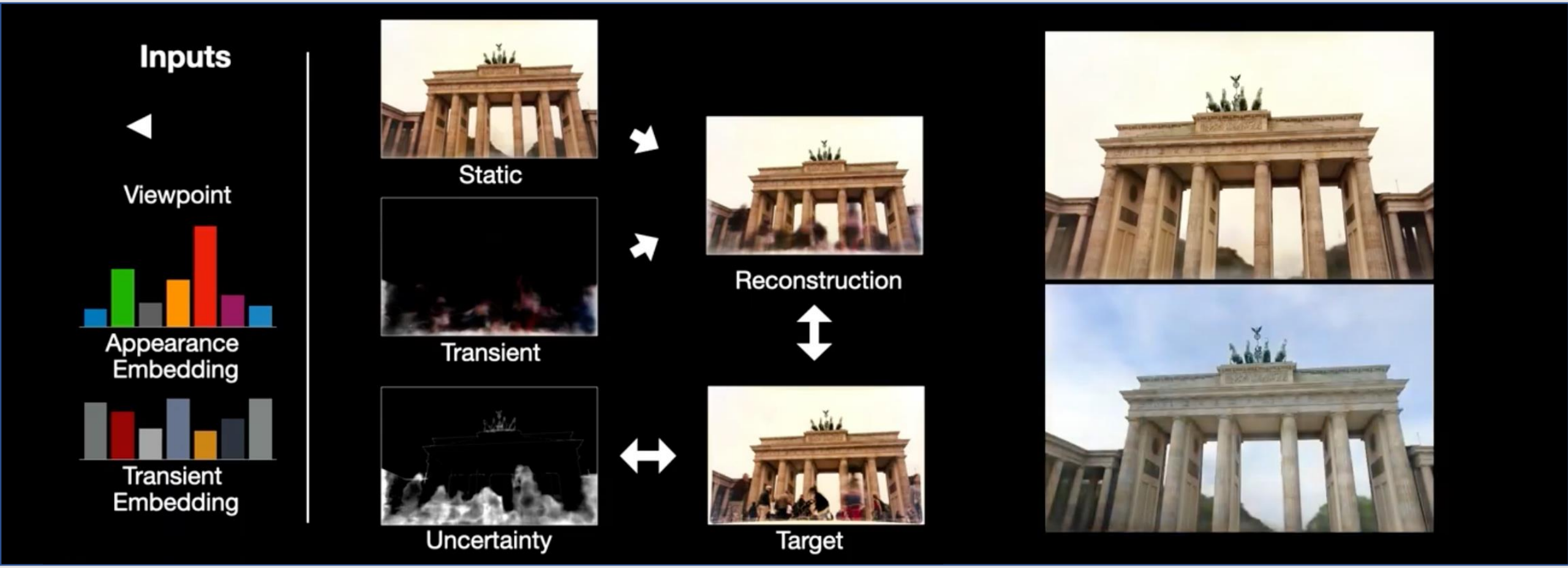
R. Martin-Brualla (2021) Google

Image Databases



- **Key Idea:** photos for scene reconstruction are not always taken by the same user in the same conditions
  - Large database of image contain both **static properties** and **transient properties**
    - Transient properties:
      - People and object occlusions
      - Weather
      - Time of day

Removing Transient Features



Learn static (appearance) and transient embedding
so that transient features can be removed

- ~~Marching Cubes~~
- ~~NERFs~~
- Gaussian Splats

Original Paper

3D Gaussian Splatting for Real-Time Radiance Field Rendering

BERNHARD KERBL*, Inria, Université Côte d'Azur, France
GEORGIOS KOPANAS*, Inria, Université Côte d'Azur, France
THOMAS LEIMKÜHLER, Max-Planck-Institut für Informatik, Germany
GEORGE DRETTAKIS, Inria, Université Côte d'Azur, France

Fig. 1. Our method achieves real-time rendering of radiance fields with quality that equals the previous method with the best quality [Barron et al. 2022], while only requiring optimization times competitive with the fastest previous methods [Fridovich-Keil and Yu et al. 2022; Müller et al. 2022]. Key to this performance is a novel 3D Gaussian scene representation coupled with a real-time differentiable renderer, which offers significant speedup to both scene optimization and novel view synthesis. Note that for comparable training times to InstantNGP [Müller et al. 2022], we achieve similar quality to theirs; while this is the maximum quality they reach, by training for 51min we achieve state-of-the-art quality, even slightly better than Mip-NeRF360 [Barron et al. 2022].

Radiance Field methods have recently revolutionized novel-view synthesis of scenes captured with multiple photos or videos. However, achieving high visual quality still requires neural networks that are costly to train and render, while recent faster methods inevitably trade off speed for quality. For unbounded and complete scenes (rather than isolated objects) and 1080p resolution rendering, no current method can achieve real-time display rates. We introduce three key elements that allow us to achieve state-of-the-art visual quality while maintaining competitive training times and importantly allow high-quality real-time (≥ 30 fps) novel-view synthesis at 1080p resolution. First, starting from sparse points produced during camera calibration, we represent the scene with 3D Gaussians that preserve desirable properties of continuous volumetric radiance fields for scene optimization while avoiding unnecessary computation in empty space; Second, we perform interleaved optimization/density control of the 3D Gaussians, notably optimizing anisotropic covariance to achieve an accurate representation of the scene; Third, we develop a fast visibility-aware rendering algorithm that supports anisotropic splatting and both accelerates training and allows real-

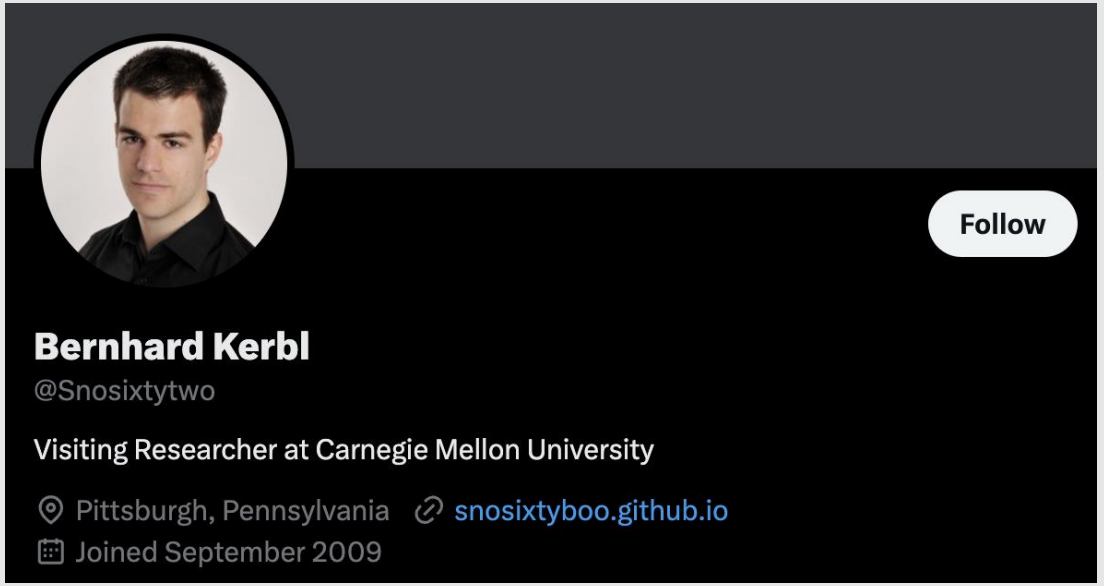
Additional Key Words and Phrases: novel view synthesis, radiance fields, 3D gaussians, real-time rendering

ACM Reference Format:
Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.* 42, 4, Article 1 (August 2023), 14 pages. https://doi.org/10.1145/3592433

1 INTRODUCTION

Meshes and points are the most common 3D scene representations because they are explicit and are a good fit for fast GPU/CUDA-based rasterization. In contrast, recent Neural Radiance Field (NeRF) methods build on continuous scene representations, typically optimizing a Multi-Layer Perceptron (MLP) using volumetric ray-marching for novel-view synthesis of captured scenes. Similarly, the most efficient

One of the 1st authors goes to CMU!



Algorithm

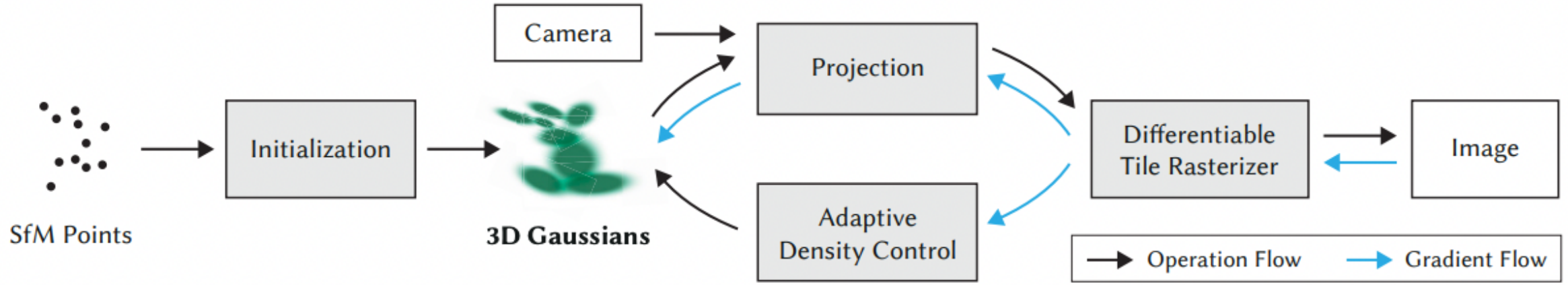
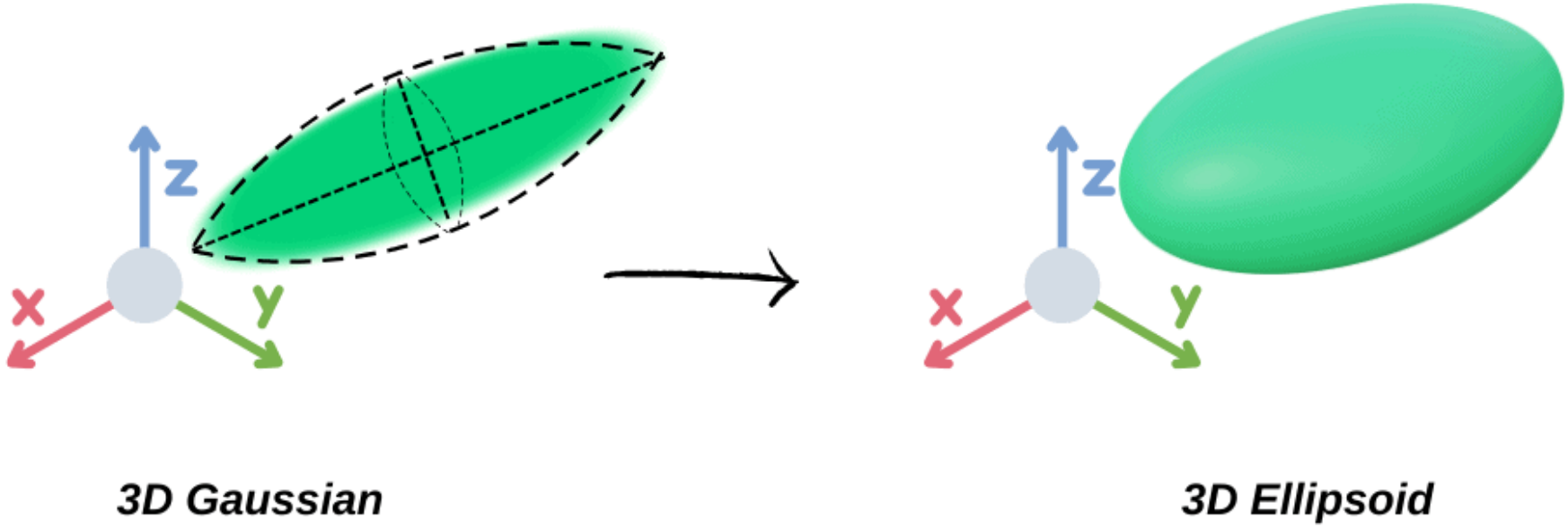


Fig. 2. Optimization starts with the sparse SfM point cloud and creates a set of 3D Gaussians. We then optimize and adaptively control the density of this set of Gaussians. During optimization we use our fast tile-based renderer, allowing competitive training times compared to SOTA fast radiance field methods. Once trained, our renderer allows real-time navigation for a wide variety of scenes.

3D Gaussian Splatting (2023) B. Kerbl et al.

3D Gaussians



3D Gaussian Splatting Introduction (2024) Learn OpenCV

3D gaussian is like an ellipsoid
Stores RGBA data

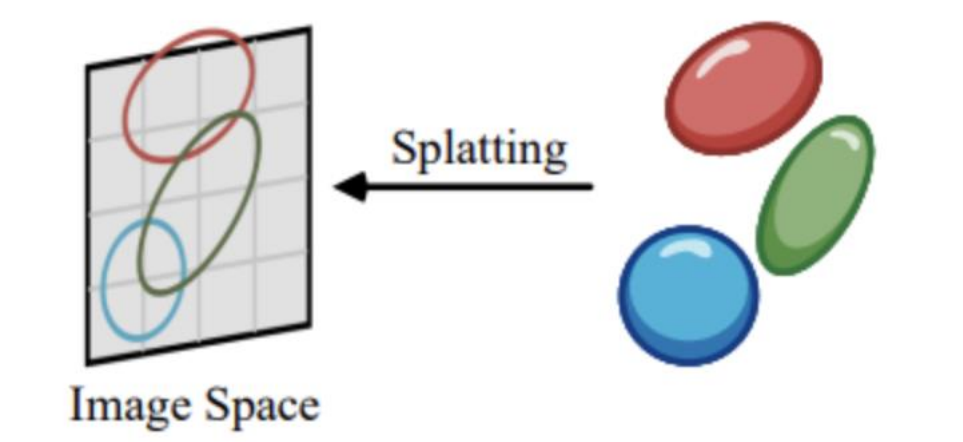
3D Gaussian Scene

What is 3D Gaussian Splatting? (2023) Bad Decisions Studio

Kinda looks like a point cloud…

Rasterization

- 3D gaussian data is perspective projected (“splat”) into 2D
  - Rasterization performed for final image
- All gaussians have RGBA, need to perform depth-based alpha blending

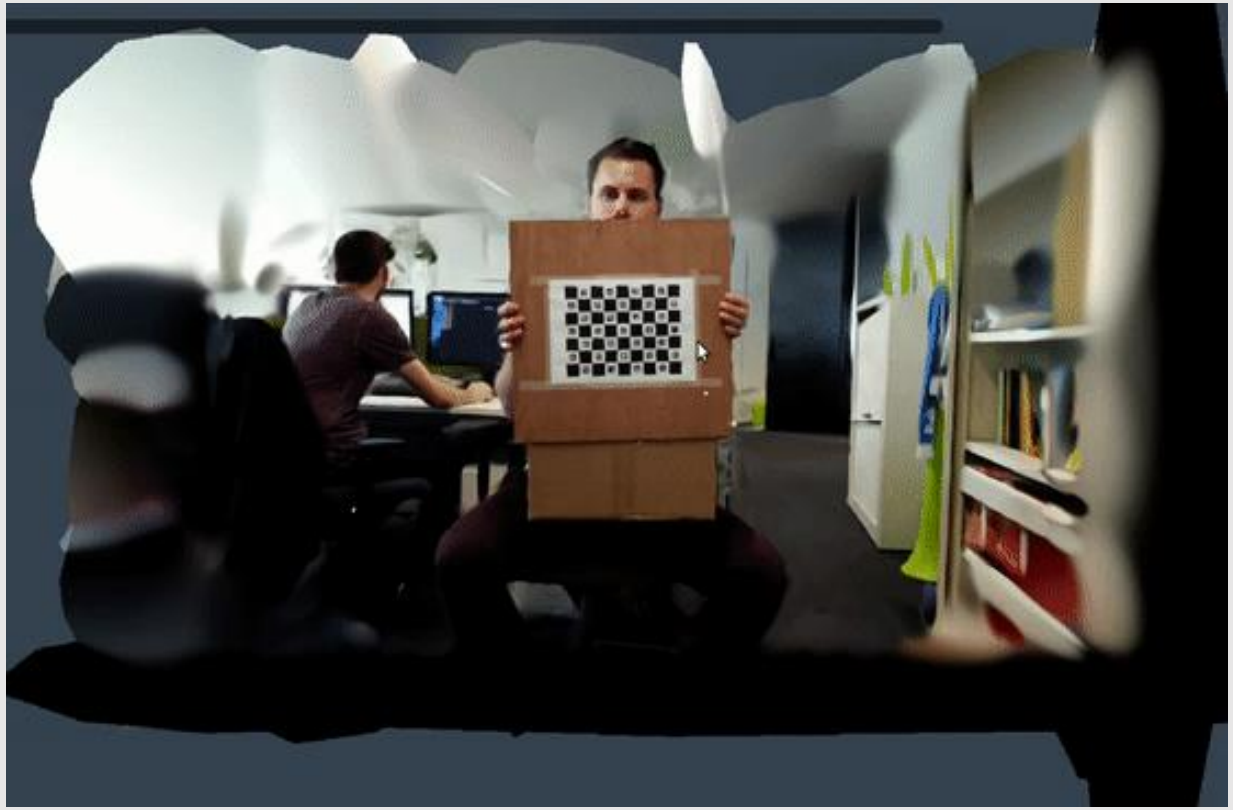


Survey on 3D Gaussian Splatting (2024) Chen & Wang

Rise of 3D Gaussian Splatting (2024) Alessio Regalbuto

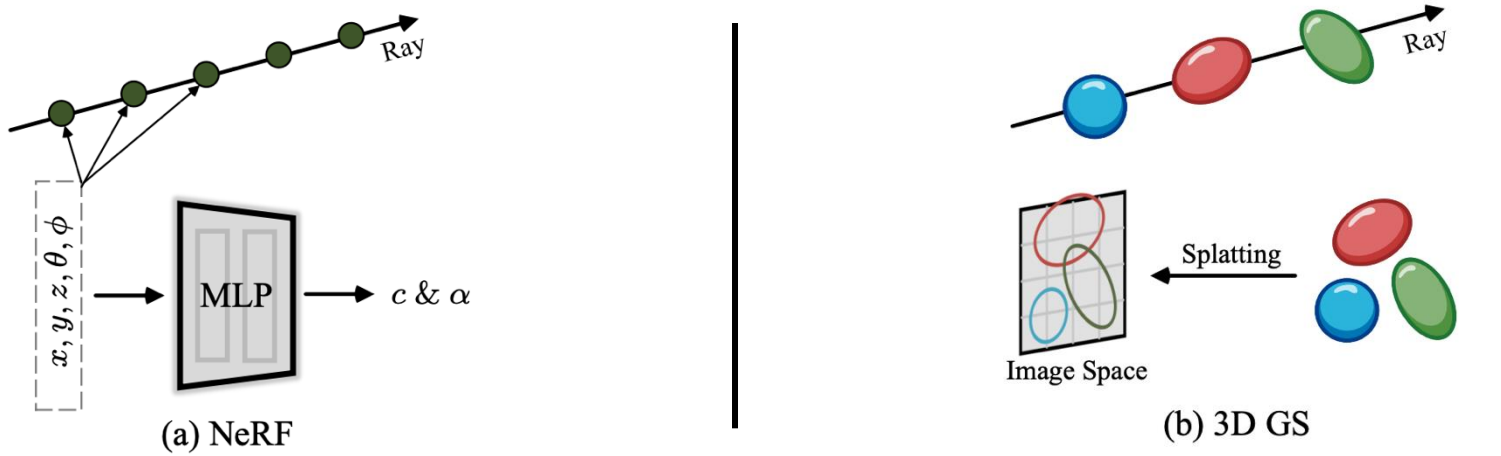
Training & Testing

- **Training:** input images + camera views
  - Minimize difference between true image and GSplat rendering at camera view
- **Testing:** new camera view
  - Outputs new image via rasterization
- Super fast!
  - Training is like a one-layer neural network
  - Testing only requires rasterization
- **Q:** can it model mirrors?



AI-Driven Breakthroughs in Image-Based Rendering (2024) Ruben Verhack

NERF vs Splats



Survey on 3D Gaussian Splatting (2024) Chen & Wang

NERFs

- [+] Implicit Geometry
- [+] Raytracing Based
- [+] Supports Reflection/Refraction
- [-] Longer to Generate
- [-] Longer To Render
- [-] Data Intense
- [-] Harder To Edit

Splats

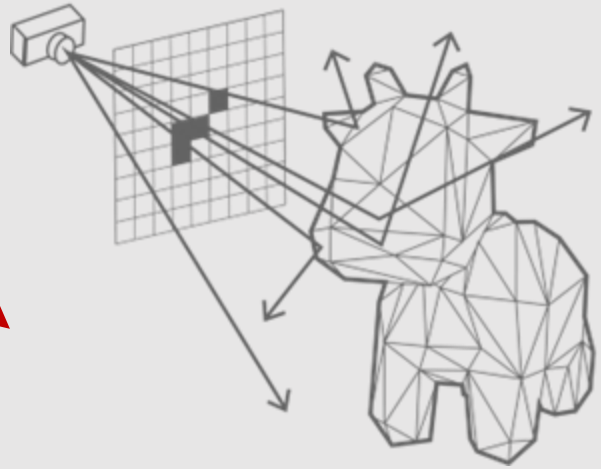
- [+] Explicit Geometry
- [+] Rasterization-Based
- [+] Real-Time Rendering
- [+] Quick To Generate
- [+] Lightweight
- [+] Easier To Edit
- [-] No Native Reflection/Refraction Support
- [-] Aliasing Artifacts

Course Roadmap

[A1: Rasterization]

[A2: MeshEdit]

Next Time



[A3: PathTracer]

[A4: Animation]