NEURAL COMPUTER GRAPHICS

Antonio D’Orazio

OUTLINE

1. Background: Computer Graphics
    1. Applications
    2. Terminology
    3. Rendering algorithms
    4. Ray tracing
    5. Representing shapes
    6. Implicit representations
2. Neural Computer Graphics
    1. Neural implicit representations
    2. Training a neural shape
    3. Adding details
    4. Final pipeline
3. My research

BACKGROUND: COMPUTER GRAPHICS

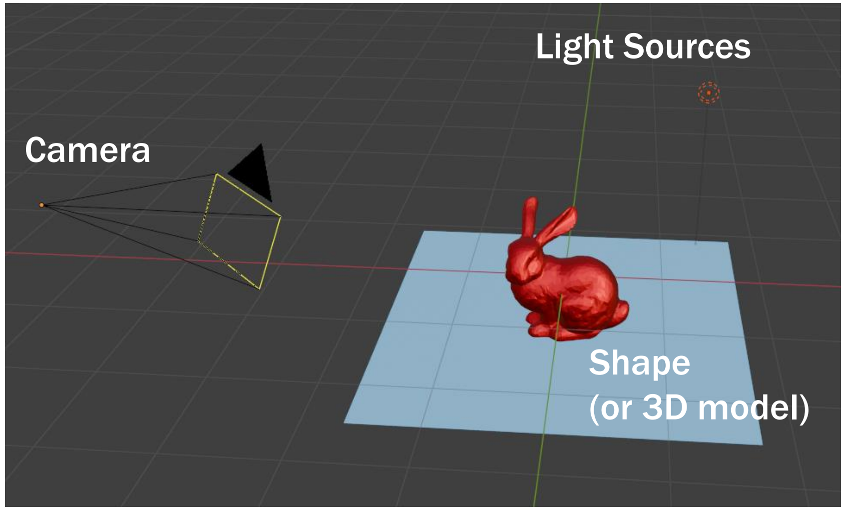
APPLICATIONS

Videogames

Movies

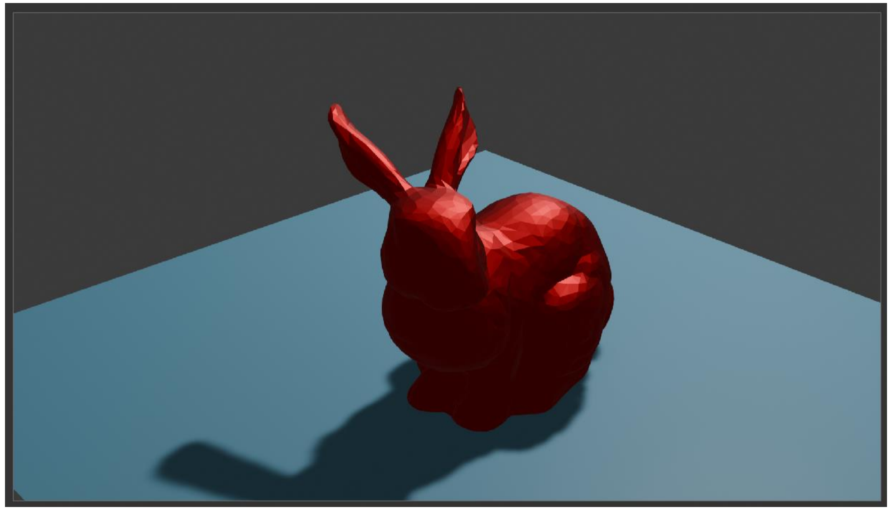
Architecture

TERMINOLOGY



3D Scene

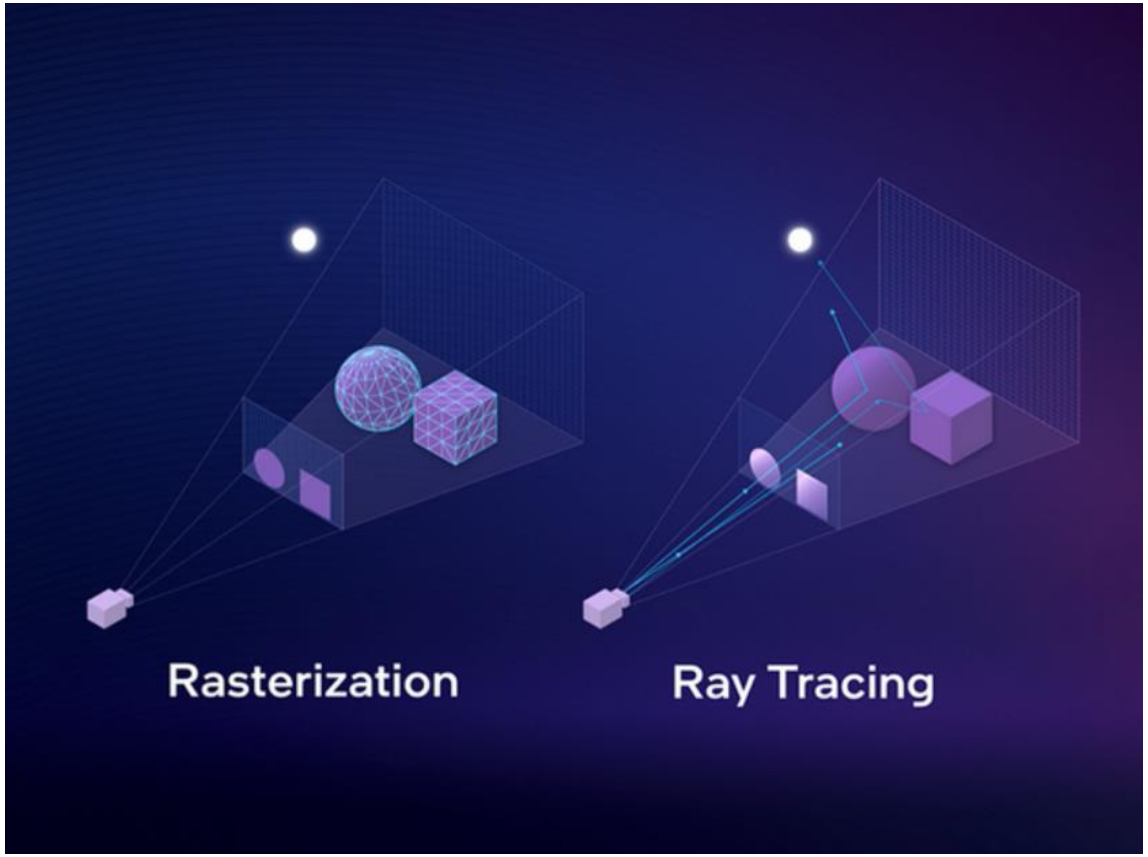
Rendering



Final 2D Image

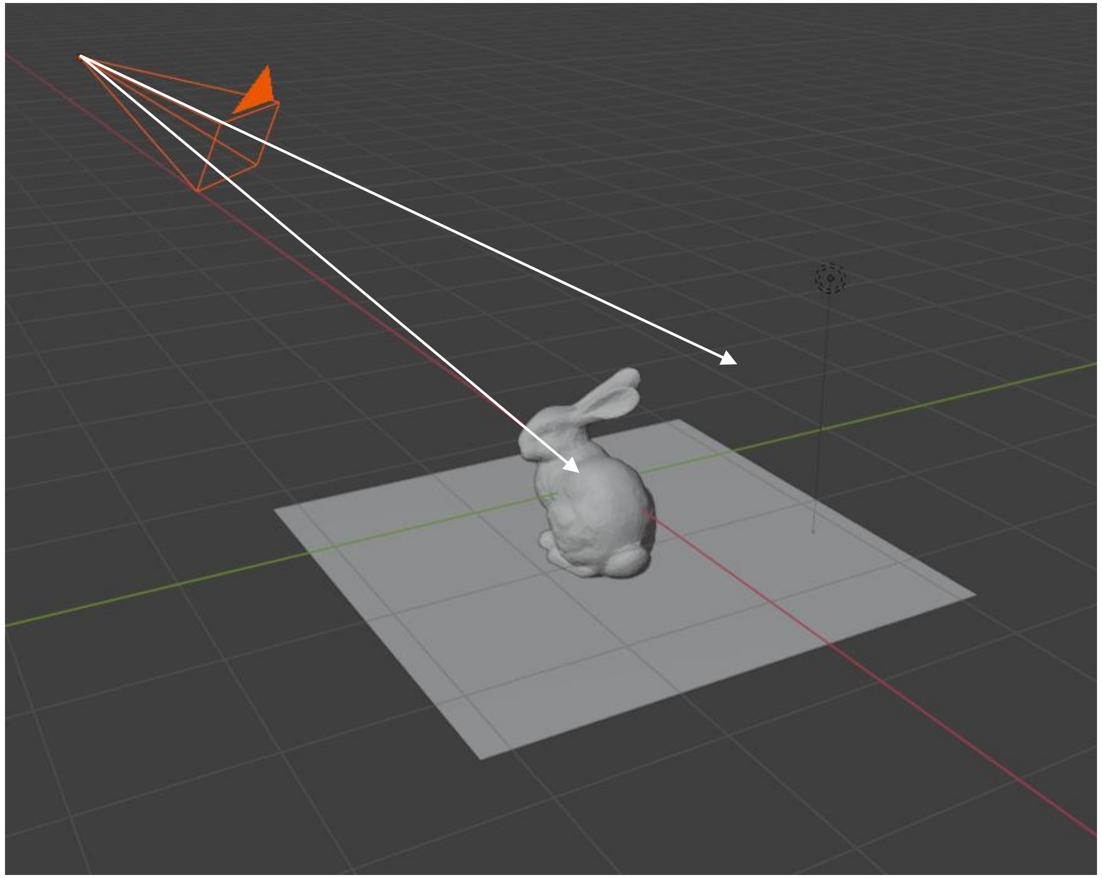
RENDERING ALGORITHMS

- Rasterization
  - Fast, approximated
- Ray Tracing
  - Slow, photorealistic



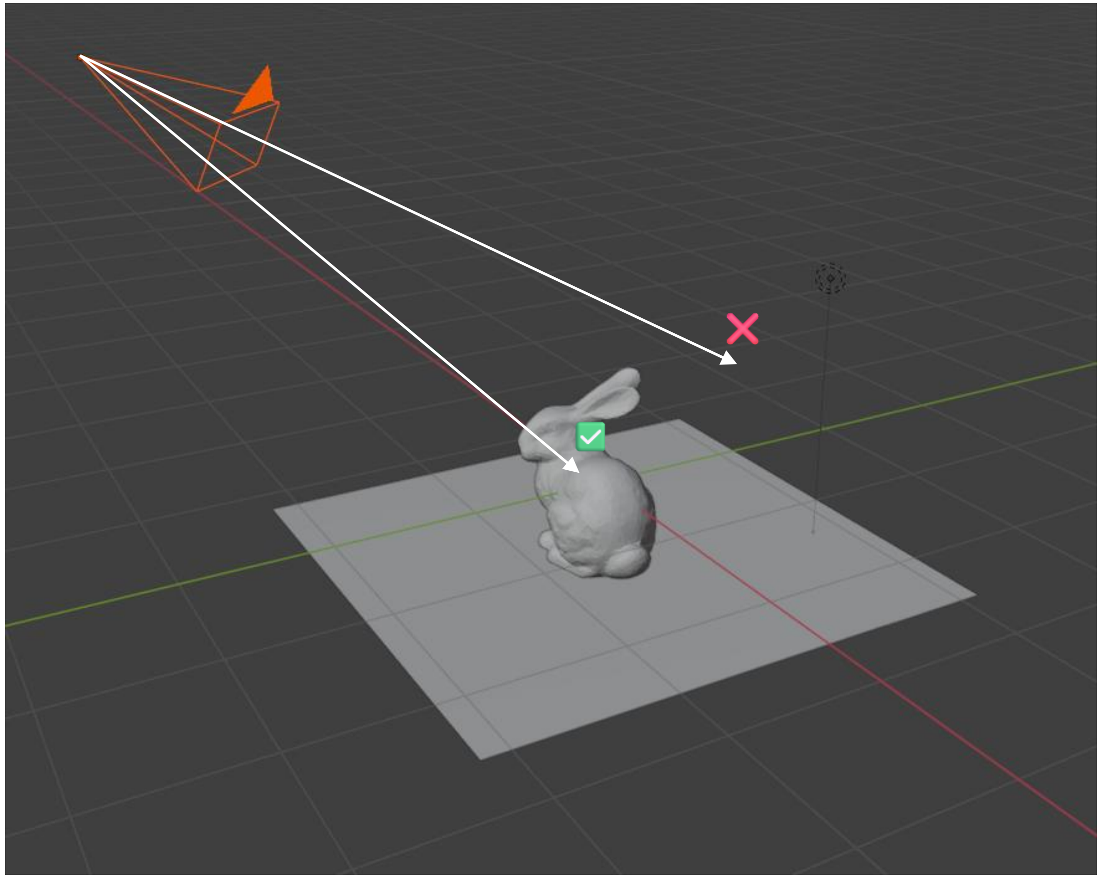
RAY TRACING

- **Cast rays from the camera through each pixel**



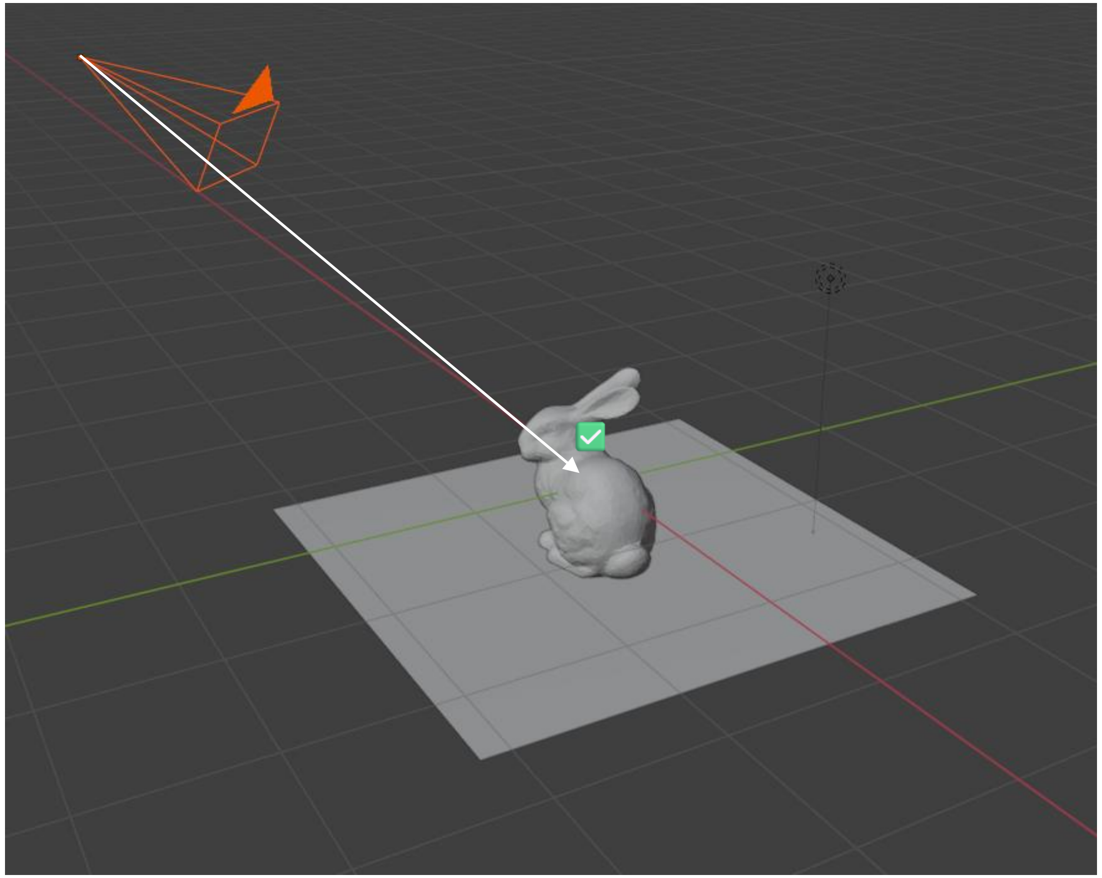
RAY TRACING

- Cast rays from the camera through each pixel
- Find intersections (slow)



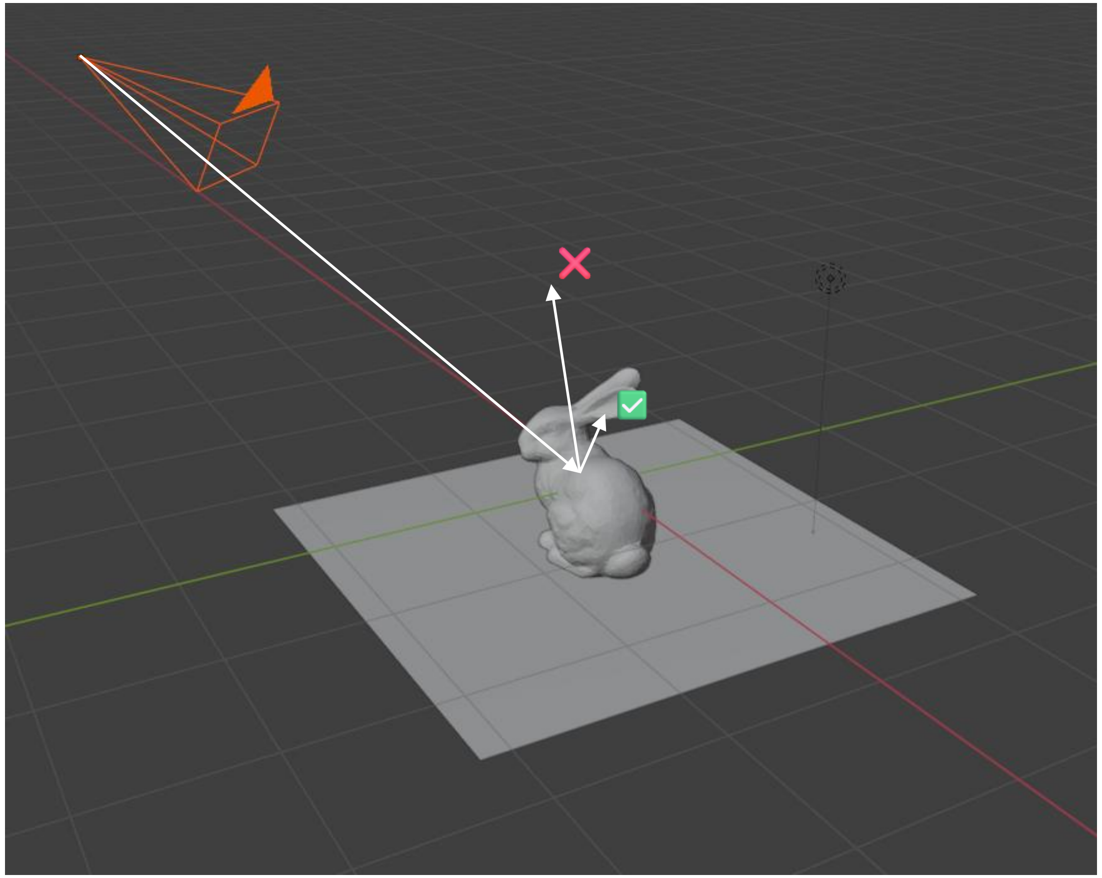
RAY TRACING

- Cast rays from the camera through each pixel
- Find intersections (slow)
- Sum the color to the corresponding pixel



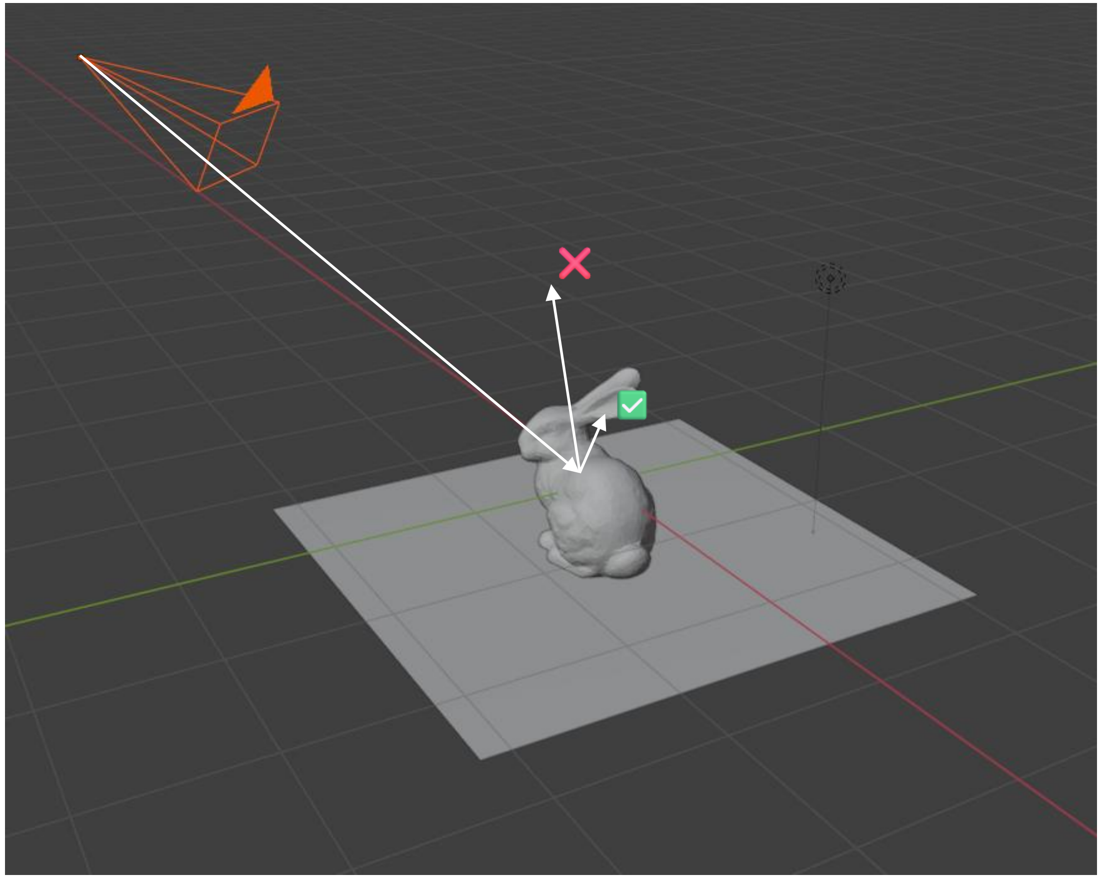
RAY TRACING

- Cast rays from the camera through each pixel
- Find intersections (slow)
- Sum the color to the corresponding pixel
- Compute the next bounce(s)

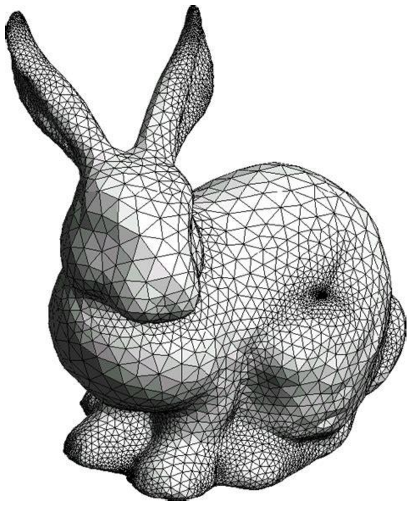


RAY TRACING

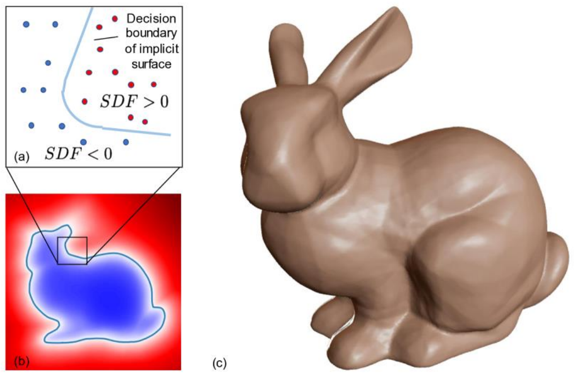
- Cast rays from the camera through each pixel
- Find intersections (slow)
- Sum the color to the corresponding pixel
- Compute the next bounce(s)
- Sum shadow and reflection colors to the pixel



REPRESENTING SHAPES



Explicit: mesh



Implicit: signed distance functions (SDFs)

Shape representation: Explicit vs Implicit

- Explicit representations: the 3D model «as is»
  - Graph $G = \{V, E\}$, Set of edges and vertices
  - Hard to manipulate
- Implicit representations: we only know how far the model is from us.
  - Signed Distance Functions: $d = f(x, y, z)$
  - When rendering, we use the distances to discover its properties.

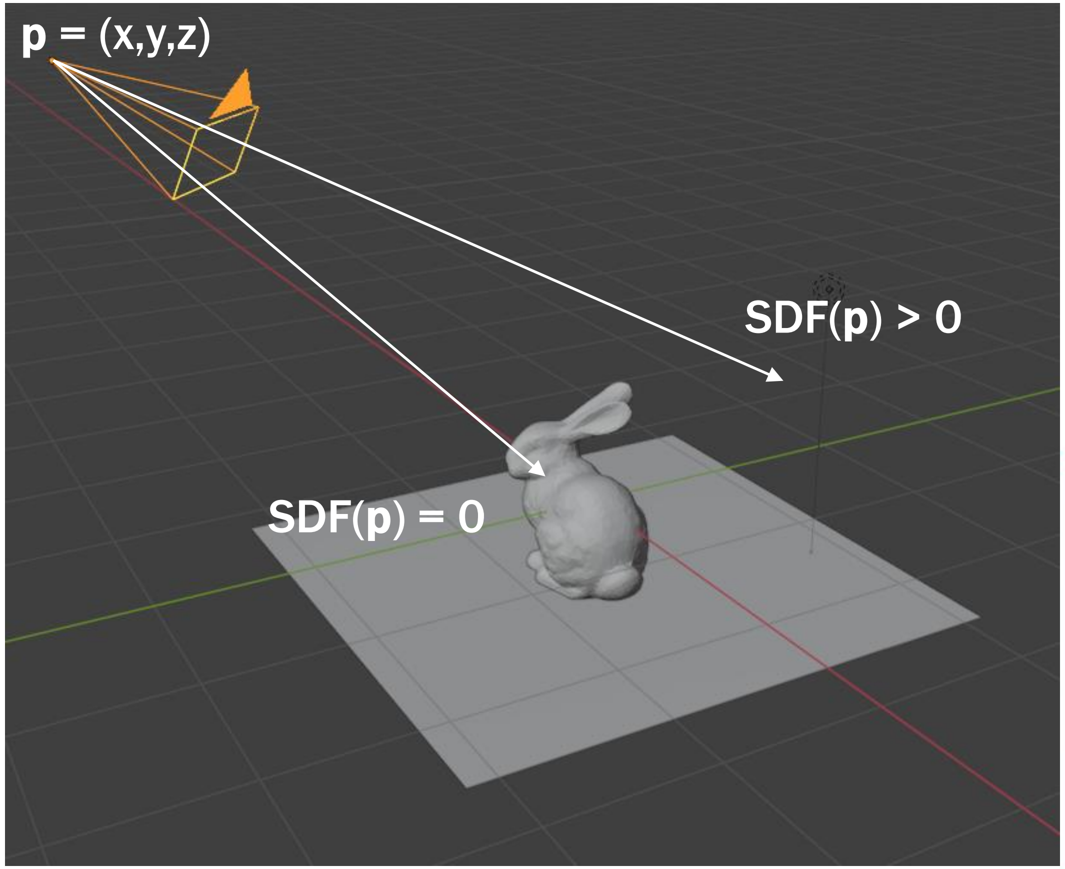
IMPLICIT REPRESENTATIONS

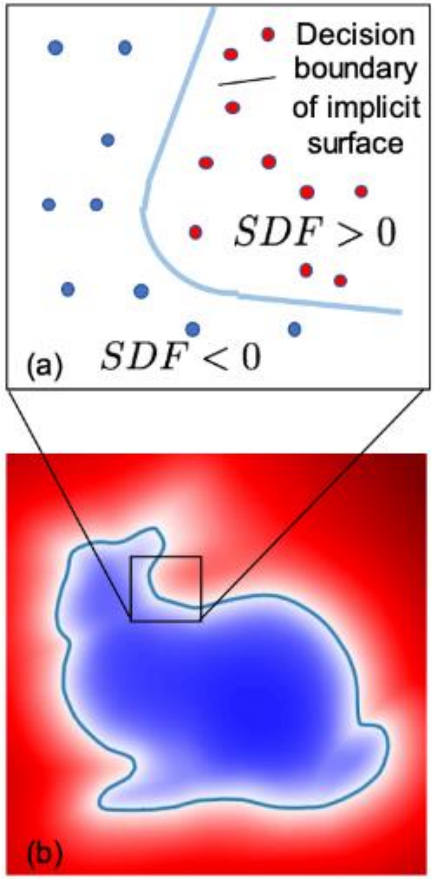
Work well with ray tracing

Q: How do we detect intersections?
A: If SDF(**p**) = 0

We only need to evaluate a function

https://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/

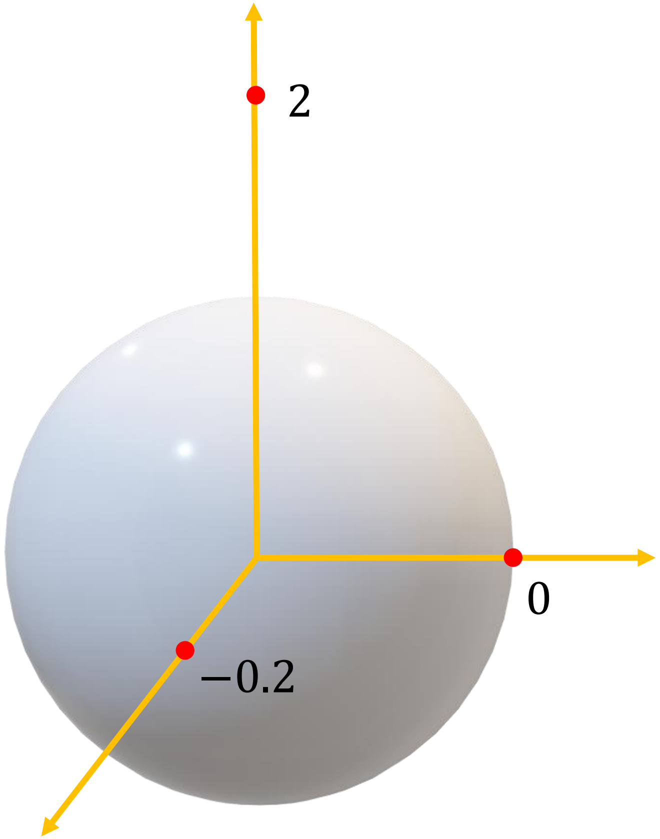


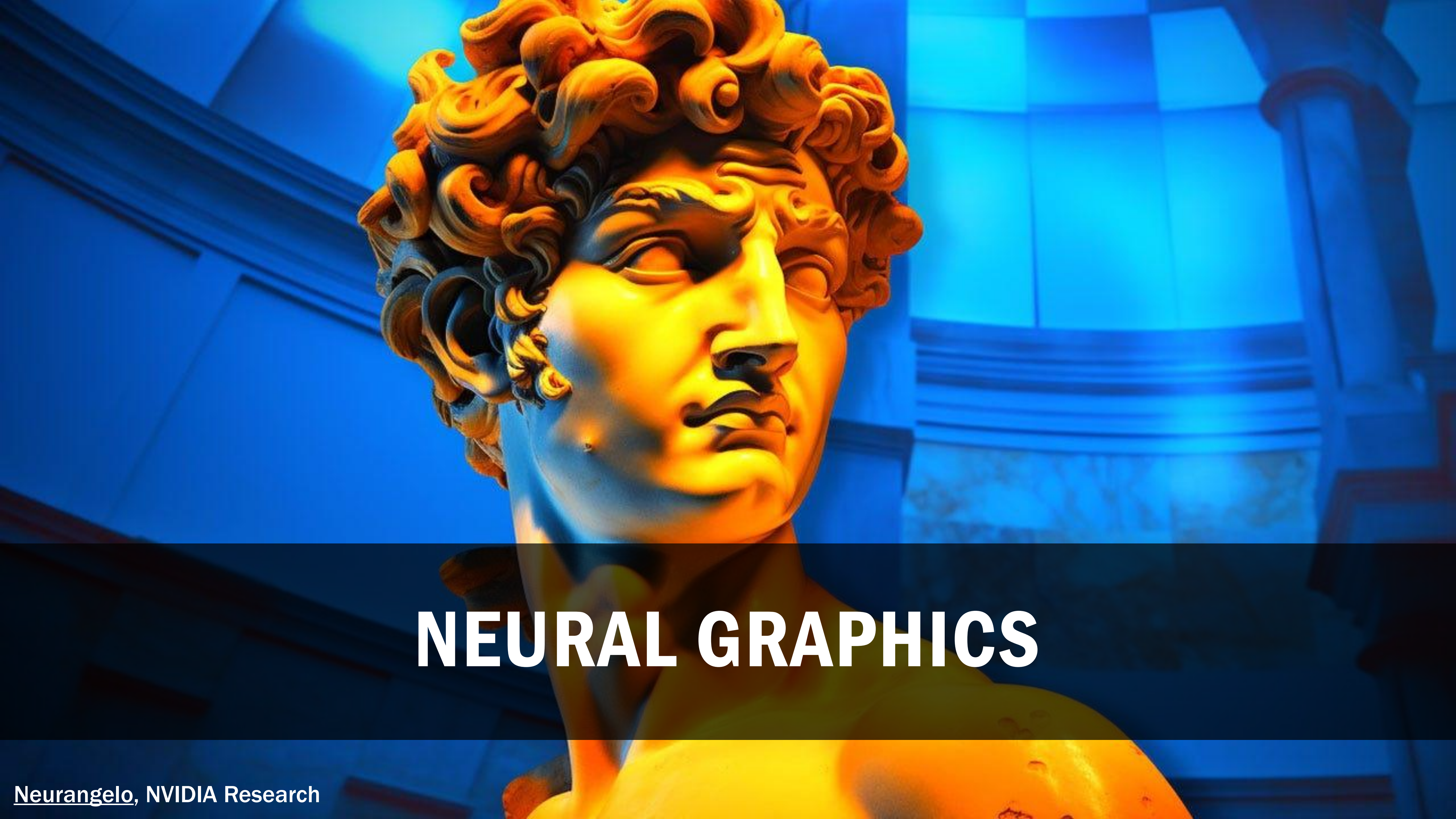


EXAMPLE: IMPLICIT SPHERE

$$SDF(x, y, z) = \sqrt{x^2 + y^2 + z^2} - 1$$

$SDF(1,0,0) = 0$	*On the surface*
$SDF(0,0,0.5) = -0.5$	*Inside*
$SDF(0,3,0) = 2$	*Outside*





NEURAL GRAPHICS

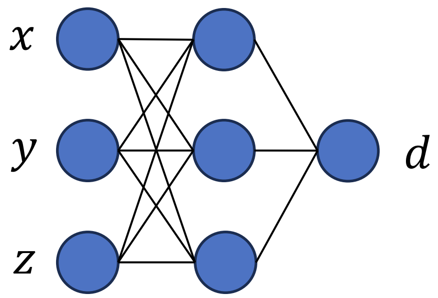
Neurangelo, NVIDIA Research

NEURAL IMPLICIT REPRESENTATIONS

SDFs are functions!

- We can **learn** them with a neural network

$$SDF_{sphere}(x, y, z) = \sqrt{x^2 + y^2 + z^2} - 1 \quad \approx \quad MLP_\theta(x, y, z)$$



- We can learn **very complex** SDFs

NEURAL IMPLICIT REPRESENTATIONS

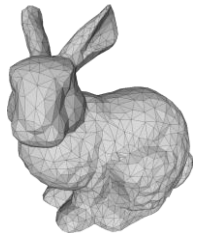
SDFs are functions!

- We can **learn** them with a neural network

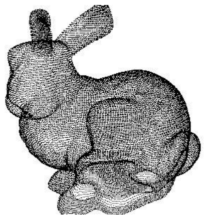
$$SDF_{sphere}(x, y, z) = \sqrt{x^2 + y^2 + z^2} - 1 \quad \approx \quad MLP_{\theta}(x, y, z)$$

- We can learn **very complex** SDFs

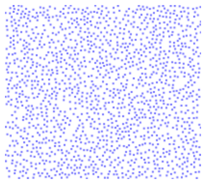
TRAINING A NEURAL SHAPE



Input mesh



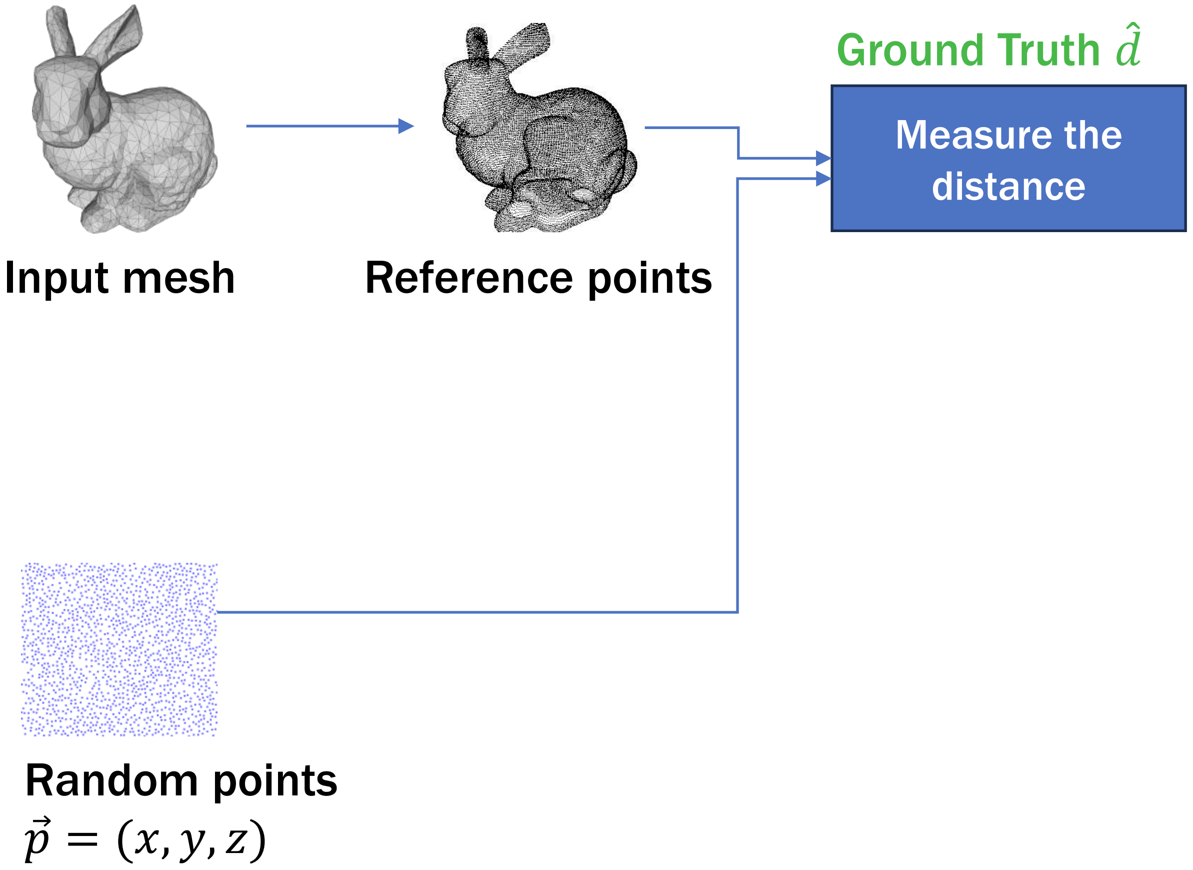
Reference points



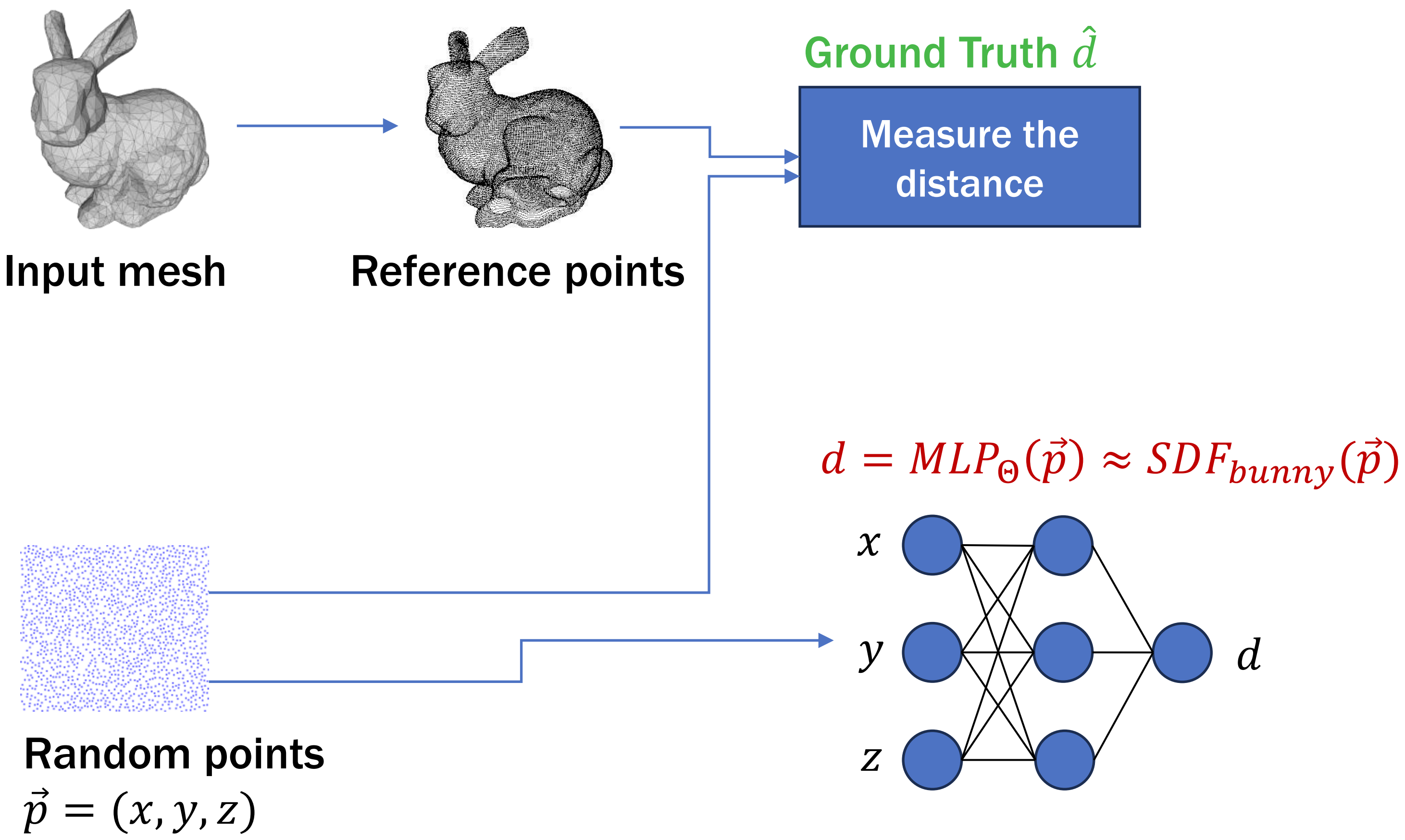
Random points

$\vec{p} = (x, y, z)$

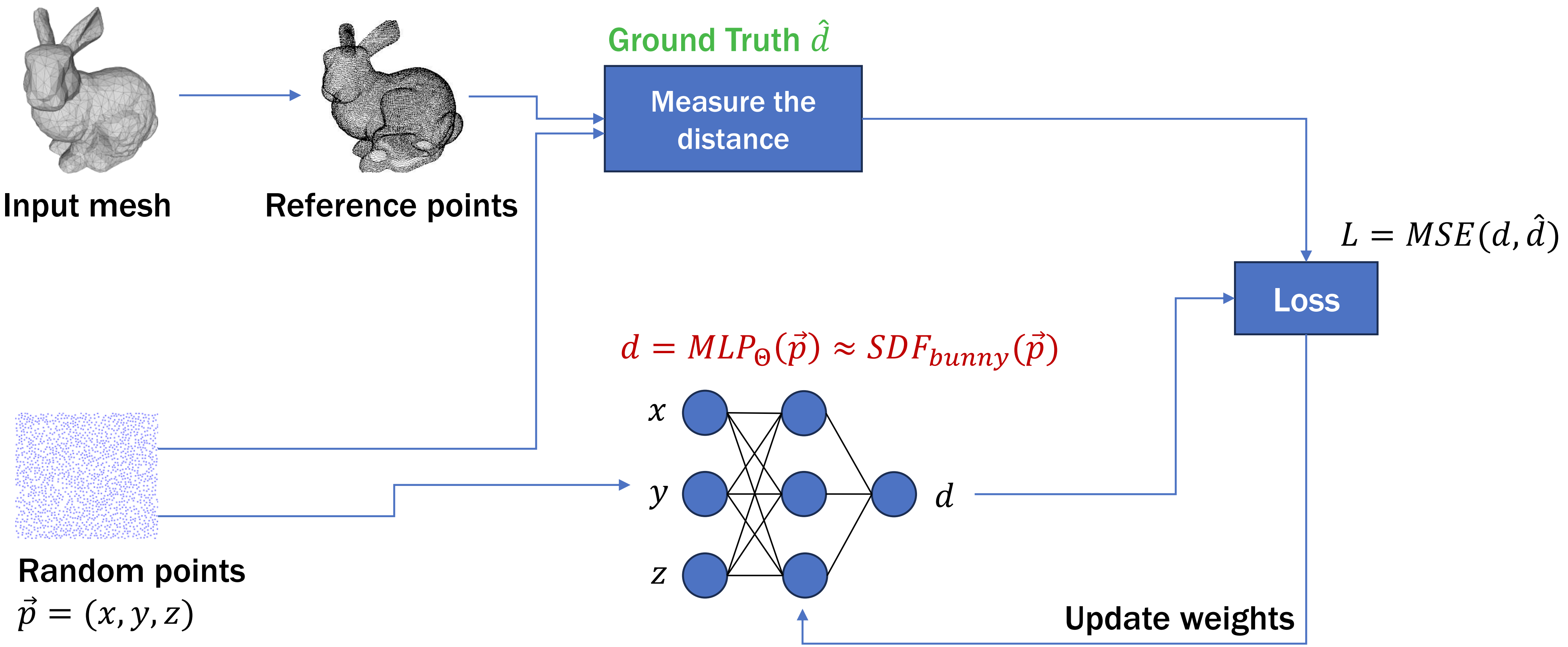
TRAINING A NEURAL SHAPE



TRAINING A NEURAL SHAPE



TRAINING A NEURAL SHAPE



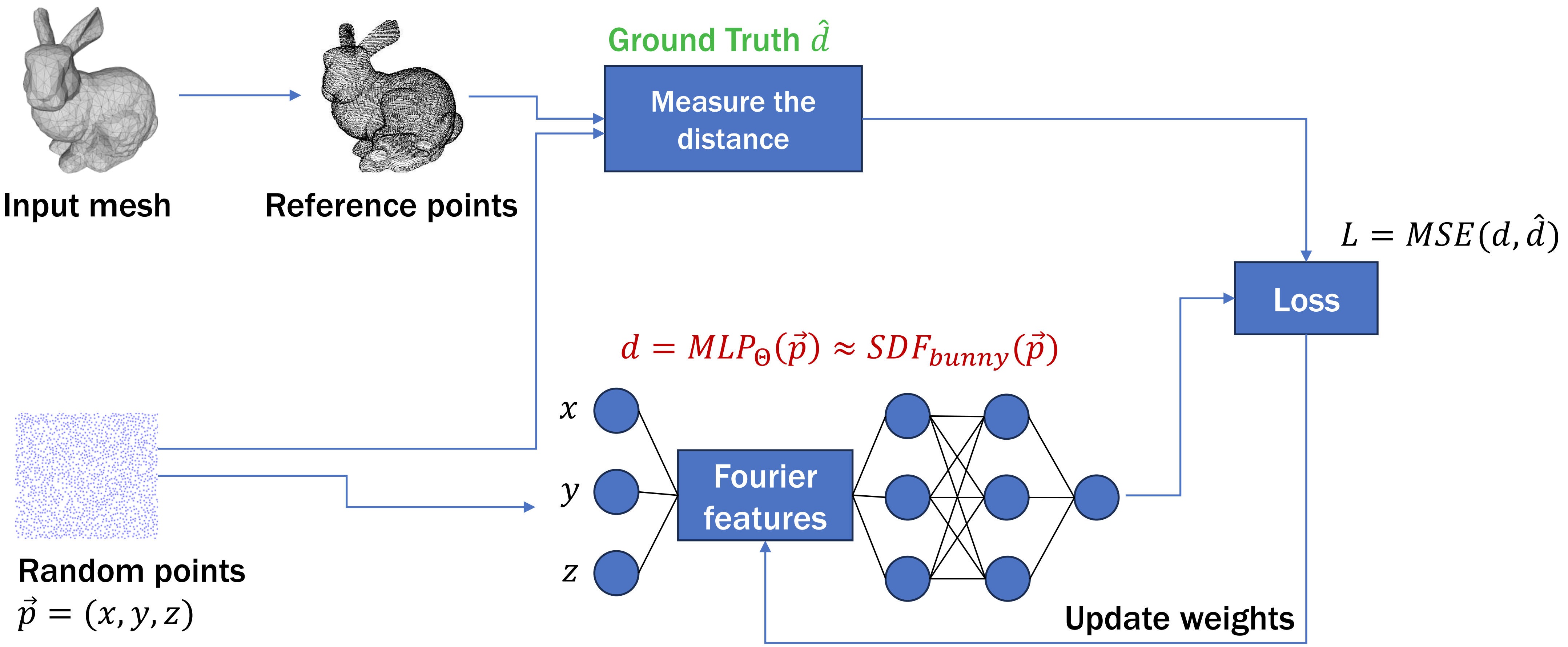
ADDING DETAILS

- We decompose the input vector into a set of frequencies (cosine and sine)
- $w_1, \dots w_n$ are learned
- The model is forced to extract the high frequencies
    - High frequencies encode fine details (also in images)

$$\phi(\vec{p}) = [\cos(2\pi \mathrm{w}_1 \vec{p}), \sin(2\pi \mathrm{w}_1 \vec{p}), \dots, \cos(2\pi \mathrm{w}_\mathrm{n} \vec{p}), \sin(2\pi \mathrm{w}_n \vec{p})]^\mathrm{T}$$

- Those features are called **Fourier features**
- https://arxiv.org/abs/2006.10739

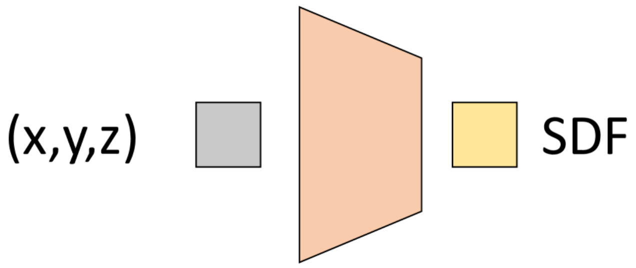
FINAL PIPELINE



MY RESEARCH

EXPLAINABLE GENERATIVE MODELS

Current 3D generative models are **not explainable**



3D artists want to manipulate the generated shapes

- «I want a taller bottle»
- «I want a thicker cup»

APPLYING OPERATORS TO THE SHAPES

Operators

- Implicit representations offer trivial way to combine primitives

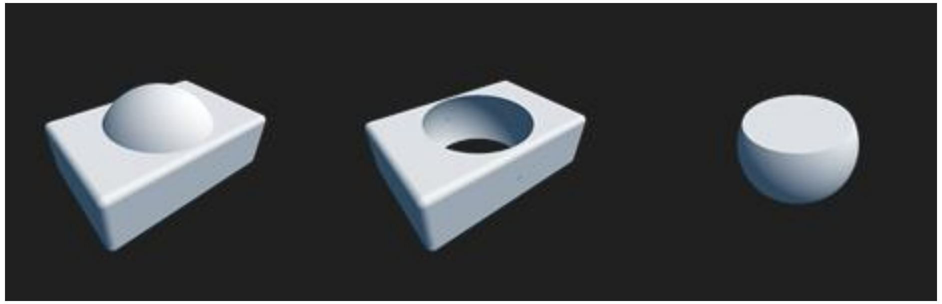
$$s_1(p,c,r) = dist(p,c) - r$$

$$s_1(p,c,r) = dist(p,c) - r$$

$$s_1 \cup s_2 = min(s_1, s_2)$$

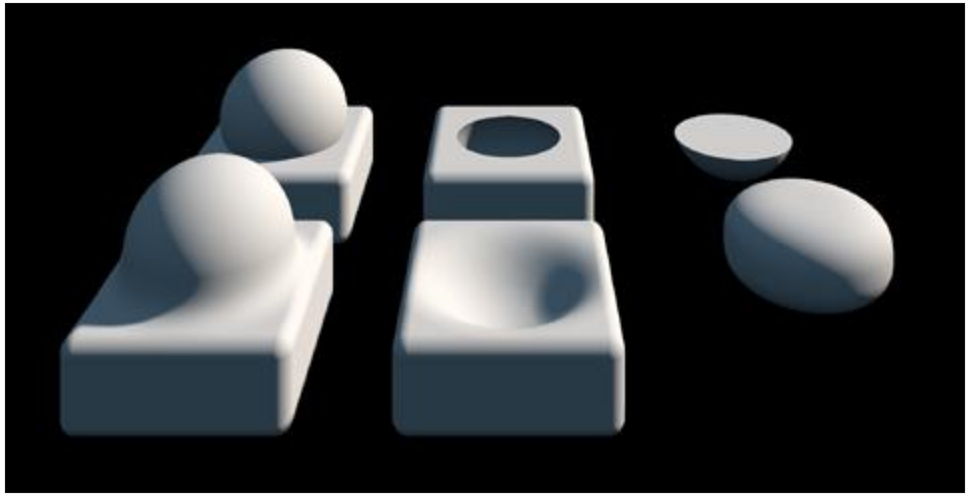
$$s_1 \cap s_2 = max(s_1, s_2)$$

$$s_1 \cup s_2 = min(-s_1, s_2)$$



MORE COMPLEX OPERATIONS

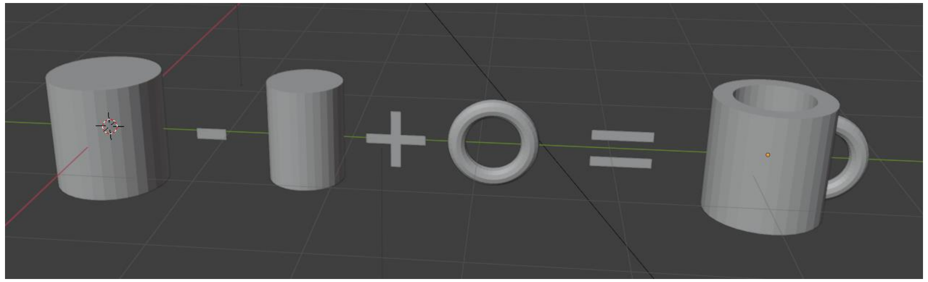
We can blend primitives to get more detail

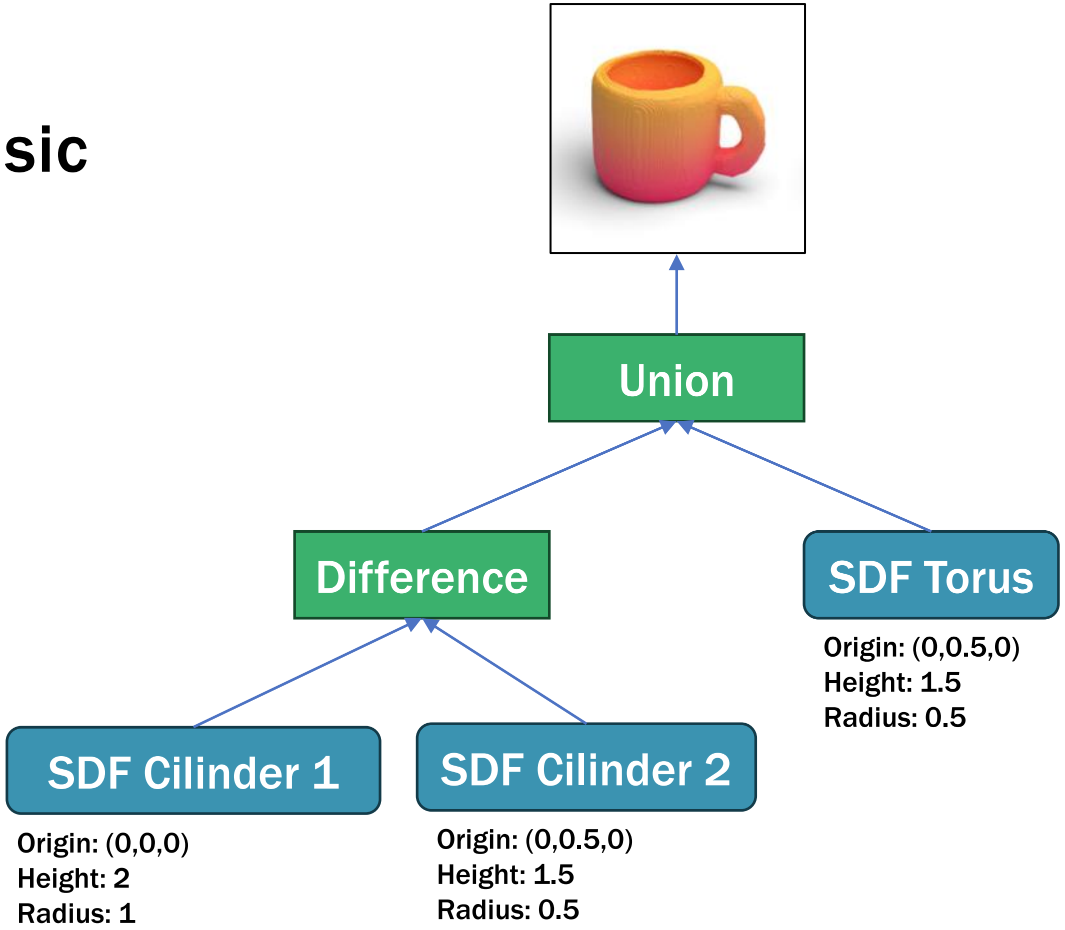


```
float opSmoothUnion( float d1, float d2, float k )
{
    float h = clamp( 0.5 + 0.5*(d2-d1)/k, 0.0, 1.0 );
    return mix( d2, d1, h ) - k*h*(1.0-h);
}

float opSmoothSubtraction( float d1, float d2, float k )
{
    float h = clamp( 0.5 - 0.5*(d2+d1)/k, 0.0, 1.0 );
    return mix( d2, -d1, h ) + k*h*(1.0-h);
}

float opSmoothIntersection( float d1, float d2, float k )
{
    float h = clamp( 0.5 - 0.5*(d2-d1)/k, 0.0, 1.0 );
    return mix( d2, d1, h ) + k*h*(1.0-h);
}
```

CONSTRUCTIVE SOLID GEOMETRY

Create a detailed 3D model using basic building blocks



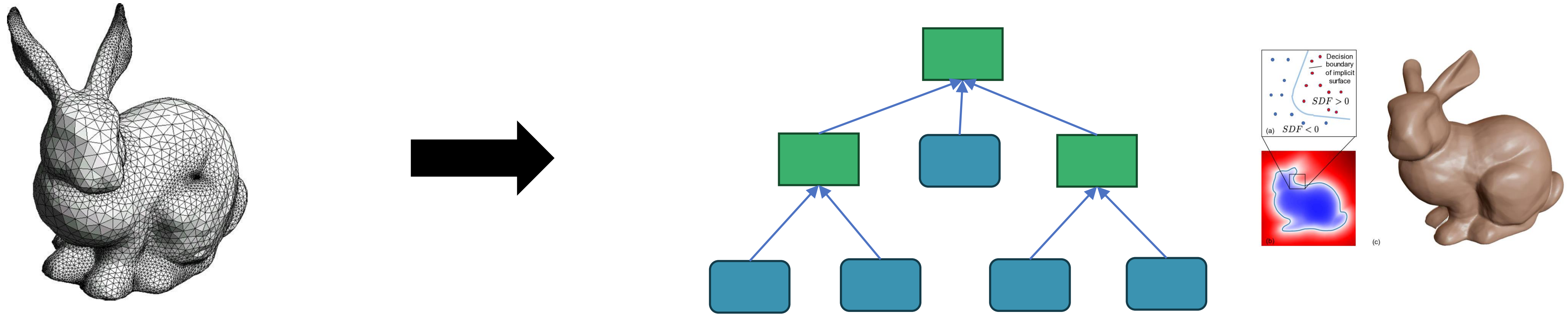


EXAMPLE TASK: MODEL RECONSTRUCTION

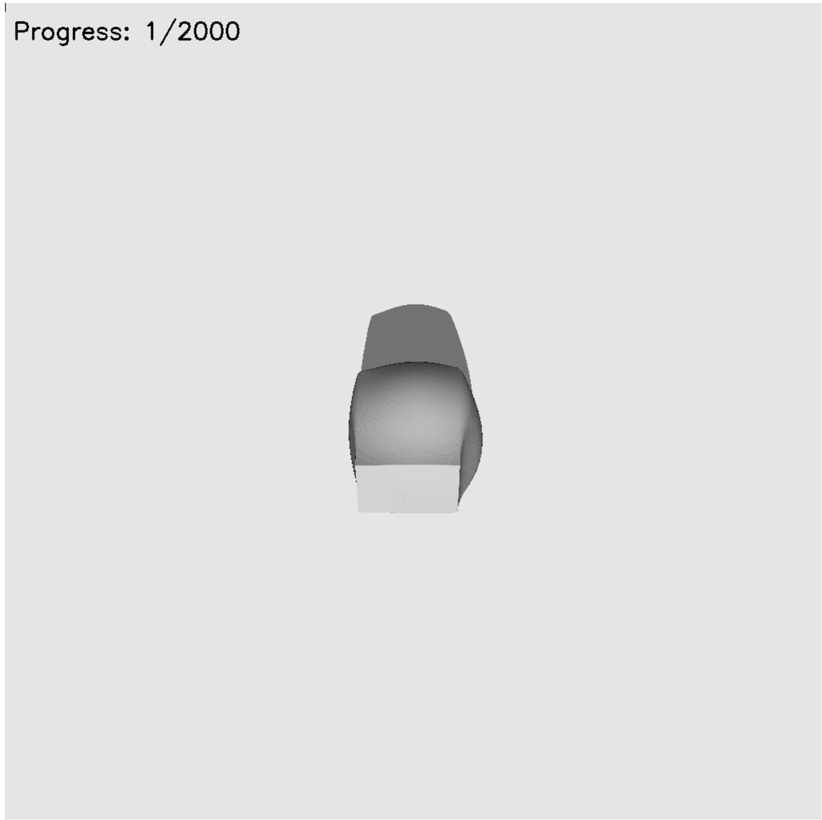
From a given 3D mesh (explicit representation),

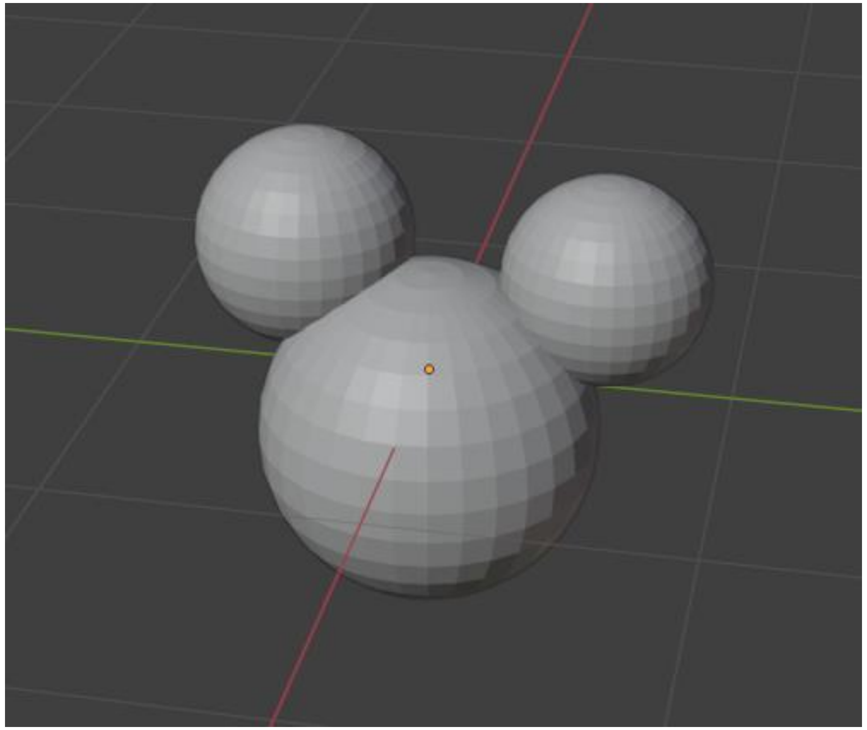
predict the constructive solid geometry tree.

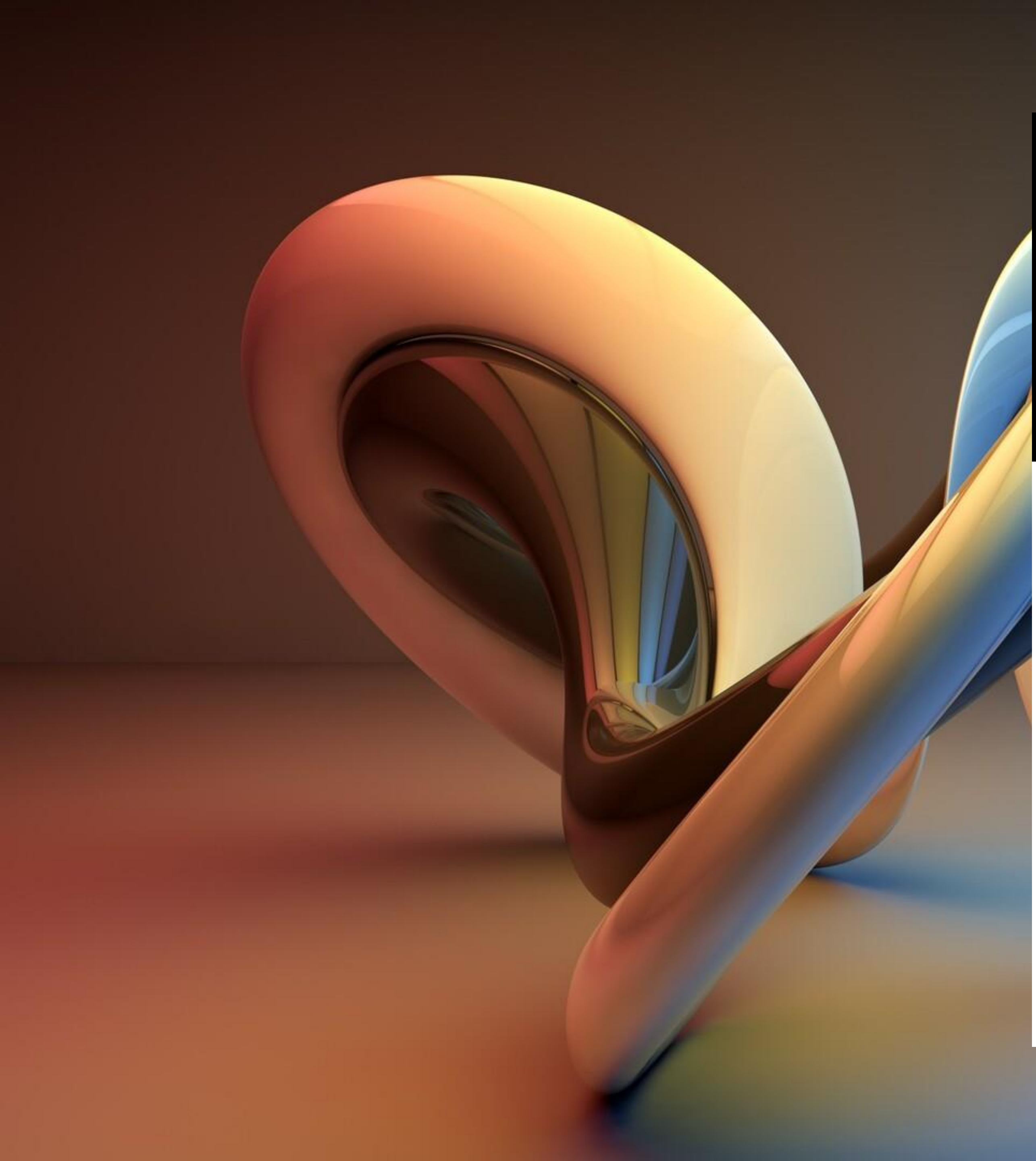
The root node is still an SDF, but it's **explainable.**



RESULTS







NOTEBOOK

https://colab.research.google.com/drive/109t1n8Nv7SiepqEKx8tHJyydF_baxF8H