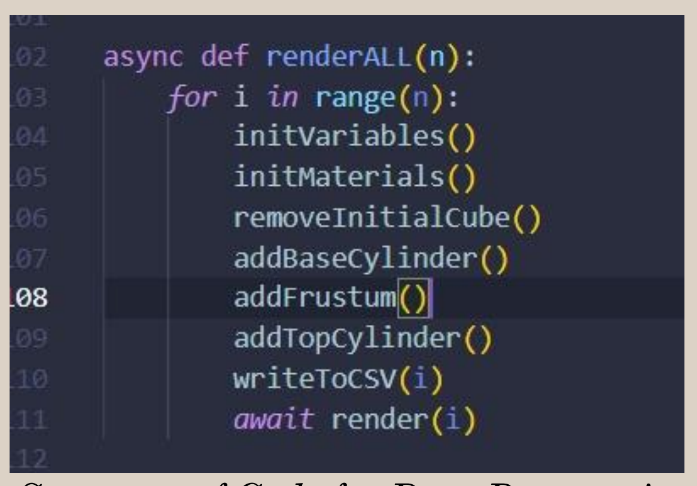
Developing a Novel Method to Reconstruct 3D Models from 2D Images.

Aniket Rajnish, Progyan Das, Shanmuganathan Raman, Shruhrid Banthia | IIT Gandhinagar

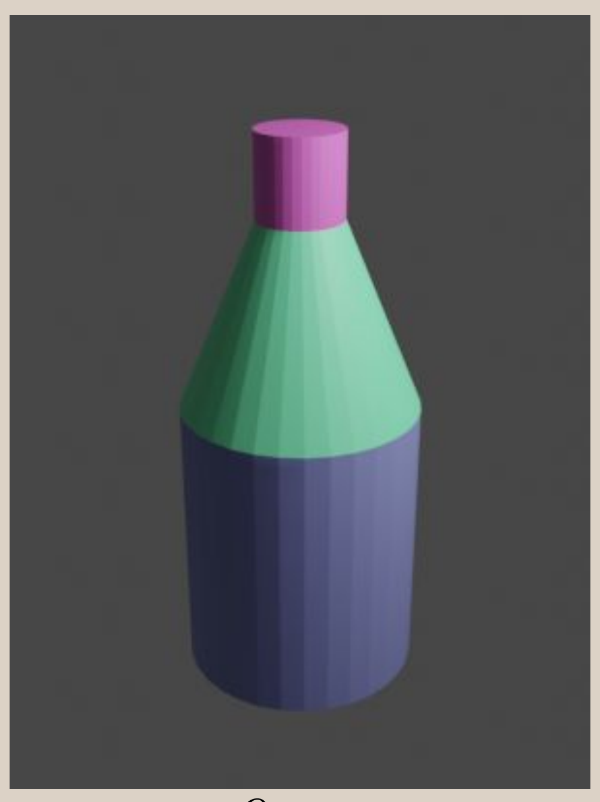
01 Custom Dataset

- We used Blender to prepare the dataset of bottles of different shapes, sizes, and colours.
- Random gaussians were used to generate the dimensions of the bottles.
- The dimensions of the bottles along with the information of color were normalized before being fed to the model in the subsequent steps.
- The images and the information regarding the dimensions of the bottles were saved.
- Each bottle comprised two cylinders and one frustum. A total of 17 metadata entries, along with every image, were stored to reconstruct the model. These entries correspond to the presence, shape, and color of the sub-parts of the bottles.
- Four hundred distinct data points were generated and used.

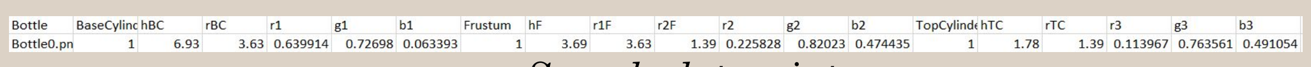


```
async def renderALL(n):
    for i in range(n):
        initVariables()
        initMaterials()
        removeInitialCube()
        addBaseCylinder()
        addFrustum()
        addTopCylinder()
        writeToCSV(i)
        await render(i)
```

Structure of Code for Data Preparation



Output

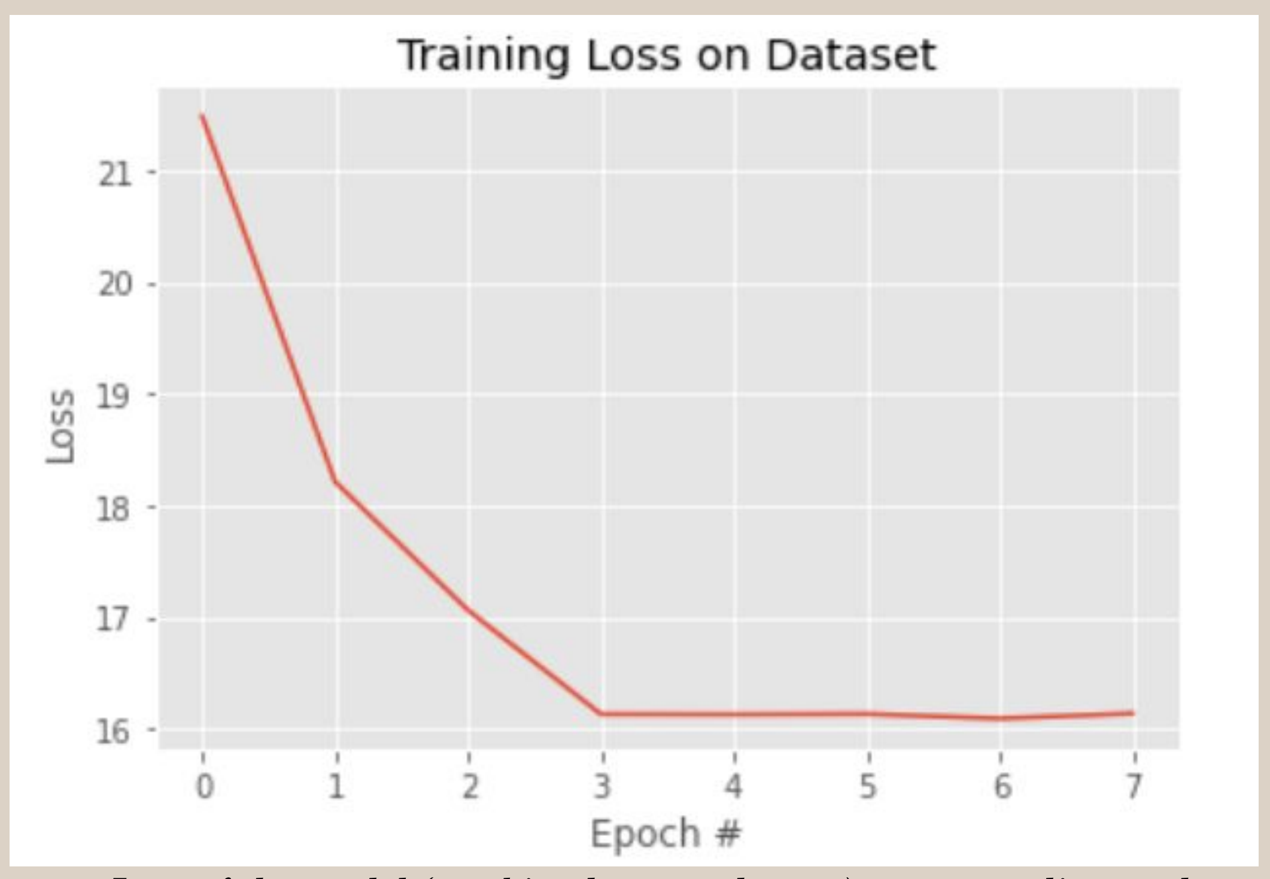


Bottle	BaseCylinc	hBC	rBC	r1	g1	b1	Frustum	hF	r1F	r2F	r2	g2	b2	TopCylinde	hTC	rTC	r3	g3	b3
Bottle0.pn	1	6.93	3.63	0.639914	0.72698	0.063393	1	3.69	3.63	1.39	0.225828	0.82023	0.474435	1	1.78	1.39	0.113967	0.763561	0.491054

Sample datapoint

02 Neural Network

- We have used a Deep Learning Model with architecture inspired by Alex-Net to model the characteristics of the bottle dataset.
- The only input provided to the model is the image itself while we try to extract the 3D characteristics of the bottles from the given image.
- Our results were satisfactory but needed to be more precise.
- This is where we started exploring other possibilities, such as Neural Radiance Fields, and later, Plenoxels and the use of plenoptic functions to model 3D characteristics.
- Link to the notebook – https://github.com/aniketrajnish/CS499-SDFNet/tree/main/Notebooks

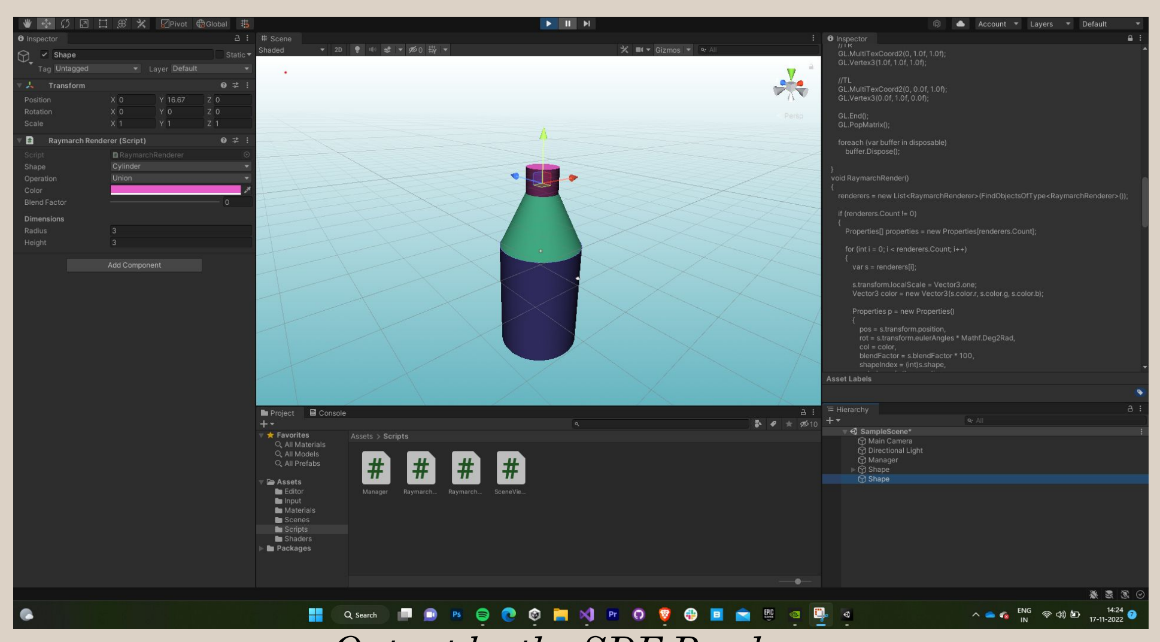


Loss of the model (combined squared error) corresponding to the epoch number

03 Signed Distance Function Renderer

INTRODUCTION

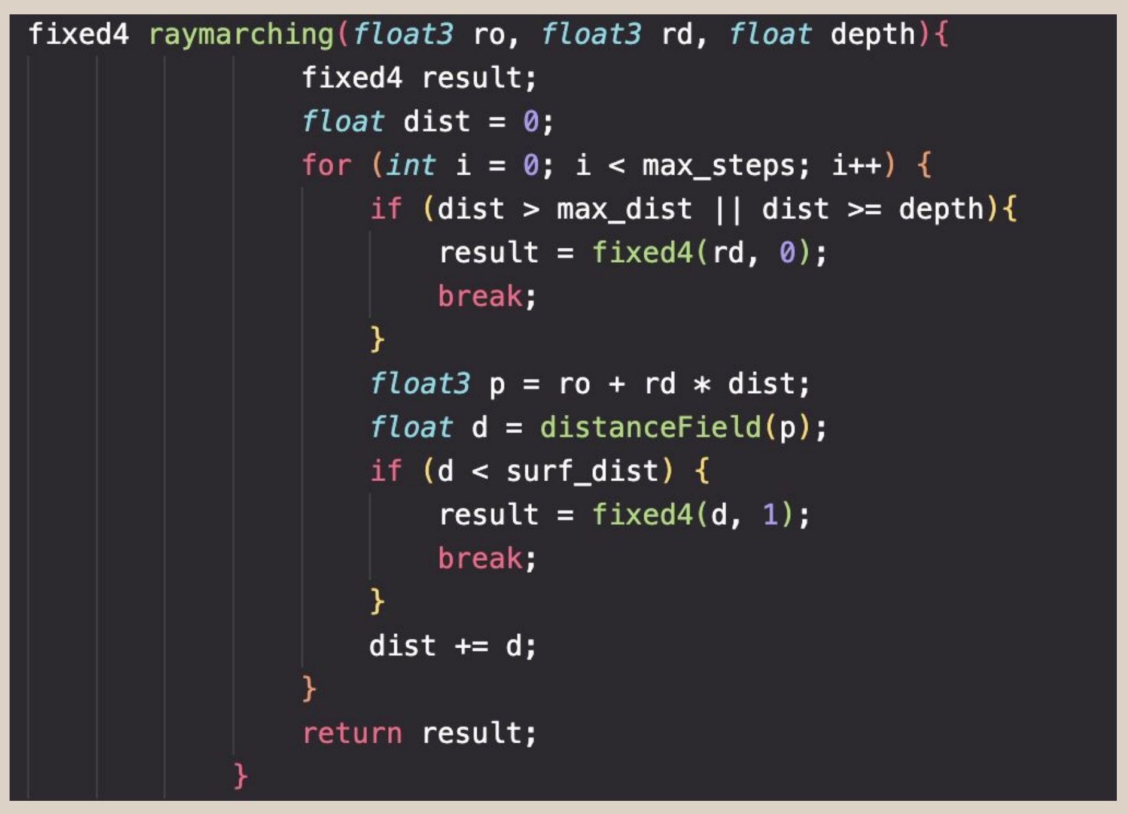
- We decided our model to use surface rendering over volume rendering to be more flexible for our choice of the objects to be rendered.
- We developed an SDF Renderer in the Unity Game Engine with support for over thirty primitives, three operations (Union, Intersection, and Subtraction), and RGB color values (along with shadows). It can be found here – https://github.com/aniketrajnish/CS499-SDFNet/tree/main/Renderer
- The pipeline that we created renders the objects based on the output that our model provides. This includes:
  - ***Shape Index***
  - ***Shape Position (Relative)***
  - ***Shape Orientation (Quaternion/Euler Angles) (Relative)***
  - ***Shape Dimensions (Relative)***
  - ***The RGB values***
- The model uses signed-distance-functions (SDFs) to render individual shapes using the shape index, dimension data, and the RGB values.
- Further it knits them together into the screen space using their relative position and relative dimensions.
- The shapes can further be fine tuned to match the initial image by using the custom editor that we developed inside Unity.



Output by the SDF Renderer

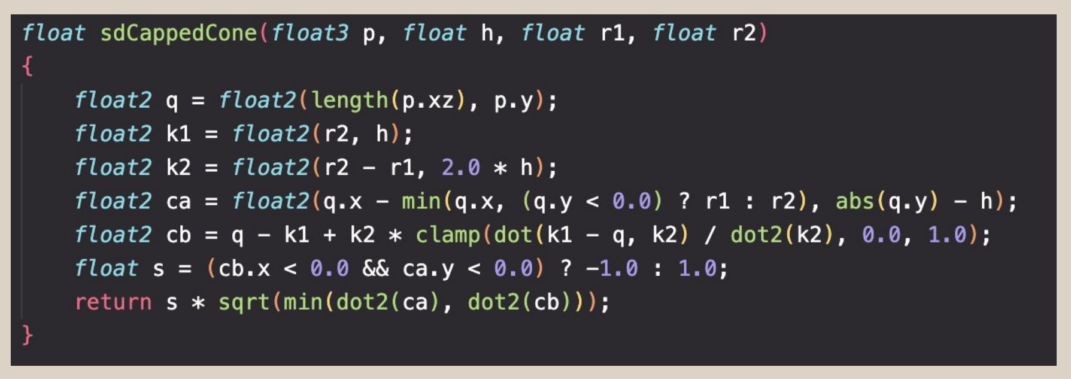
RENDERING THE SHAPES

- We wrote a Image Effect shader to render objects directly in the screen space instead of creating instances of individual objects.
- We formulated a raymarching loop in the shader to render these shapes using their individual signed distance functions.
- All the parameters were taken from our model in a CSV file and were communicated to the shader from a C# script using Compute Buffers.
- The dimensional parameters were stored in custom class of vector12 with 12 fields (maximum dimensional inputs that any shape can take) for floats as the Shader language doesn't support dynamic arrays. So these parameters were communicated in the following way:
  - ***dimensions[0] = new vector12(cyl.r, cyl.h, 0,0,0,0,0,0,0,0,0,0);***
  - ***dimensions[1] = new vector12(cap.r1, cap.r2, cap.h, 0, 0, 0, 0, 0, 0, 0, 0,0);***
- Computer buffers were also used to communicate other information like the number of shapes to be rendered, and the blend factor for the operations for each shape.
- All the shapes are being rendered on render texture in front of the camera, the dimensions of which are communicated to shader.



```
fixed4 raymarching(float3 ro, float3 rd, float depth){
            fixed4 result;
            float dist = 0;
            for (int i = 0; i < max_steps; i++) {
                if (dist > max_dist || dist >= depth){
                    result = fixed4(rd, 0);
                    break;
                }
                float3 p = ro + rd * dist;
                float d = distanceField(p);
                if (d < surf_dist) {
                    result = fixed4(d, 1);
                    break;
                }
                dist += d;
            }
            return result;
        }
```

The Raymarching Loop we formulated

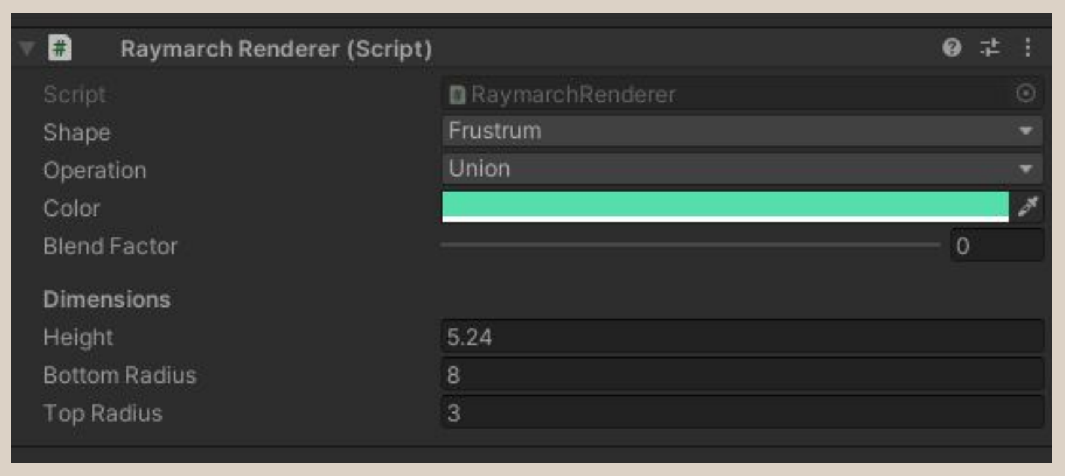


```
float sdCappedCone(float3 p, float h, float r1, float r2)
{
    float2 q = float2(length(p.xz), p.y);
    float2 k1 = float2(r2, h);
    float2 k2 = float2(r2 - r1, 2.0 * h);
    float2 ca = float2(q.x - min(q.x, (q.y < 0.0) ? r1 : r2), abs(q.y) - h);
    float2 cb = q - k1 + k2 * clamp(dot(k1 - q, k2) / dot2(k2), 0.0, 1.0);
    float s = (cb.x < 0.0 && ca.y < 0.0) ? -1.0 : 1.0;
    return s * sqrt(min(dot2(ca), dot2(cb)));
}
```

Example of a Distance Function

CUSTOM EDITOR

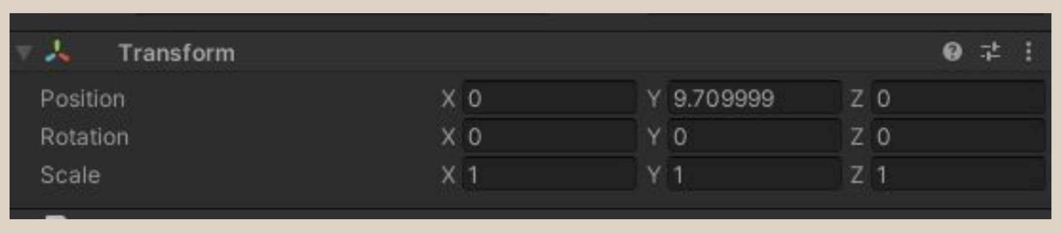
- A custom editor was developed in Unity to aid the need to fine tune the objects rendered on the screen
- The editor could be accessed using the Unity's inspector.
- The following parameters were governed using the Custom Editor-
  - ***Shape Index***
  - ***Shape Operation***
  - ***Color***
  - ***Blend Factor***
  - ***Dimension Factor***
- The following spatial parameters were governed by the Unity's inspector component-
  - ***Shape Position***
  - ***Shape Orientation (Quaternion/Euler Angles)***



Raymarch Renderer (Script)

Script	RaymarchRenderer
Shape	Frustrum
Operation	Union
Color	🟩
Blend Factor	0
Dimensions	
Height	5.24
Bottom Radius	8
Top Radius	3

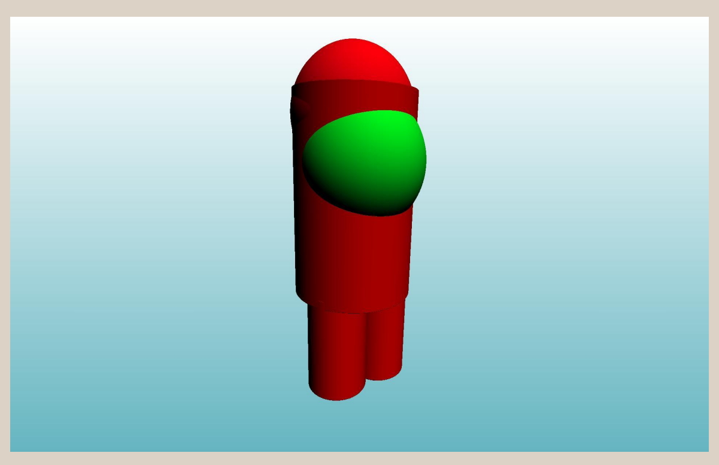
Custom Inspector



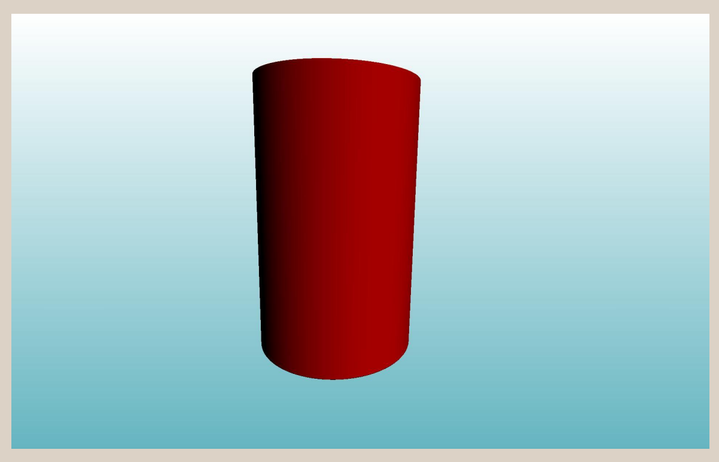
Transform

Position	X	0	Y	9.709999	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

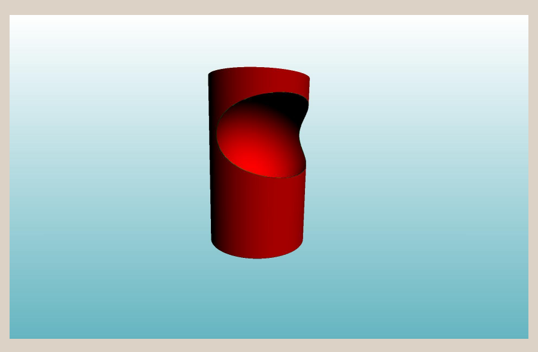
Transform Component



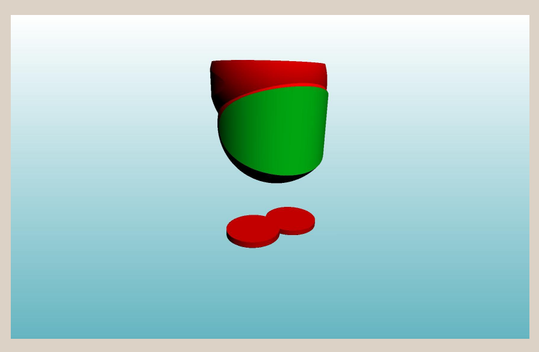
Union



Intersection



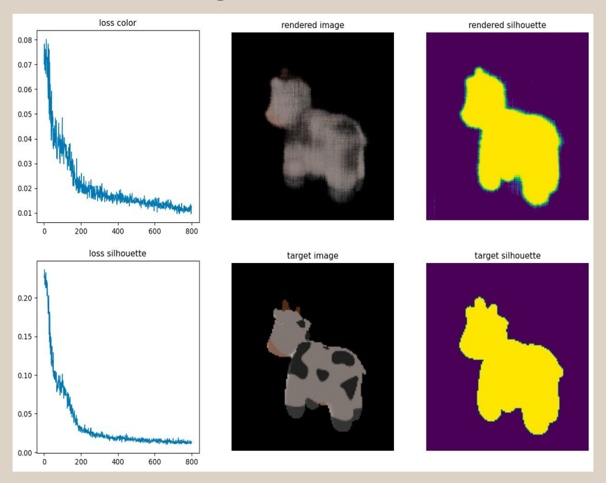
Subtract

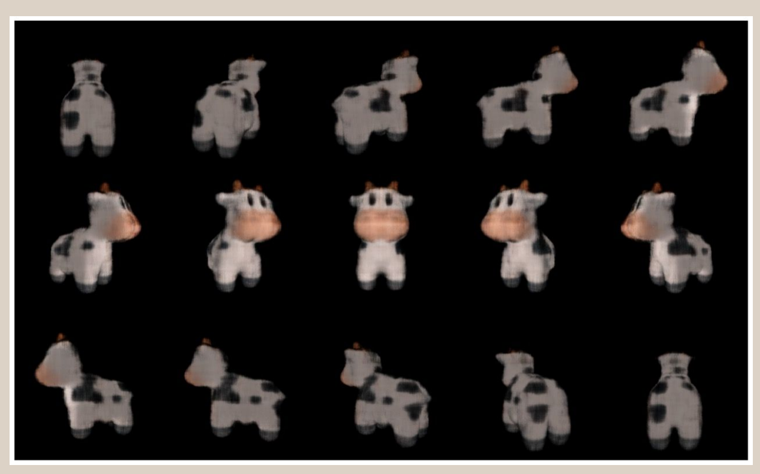


Subtract + Intersect

04 Neural Radiance Field through Raymarching

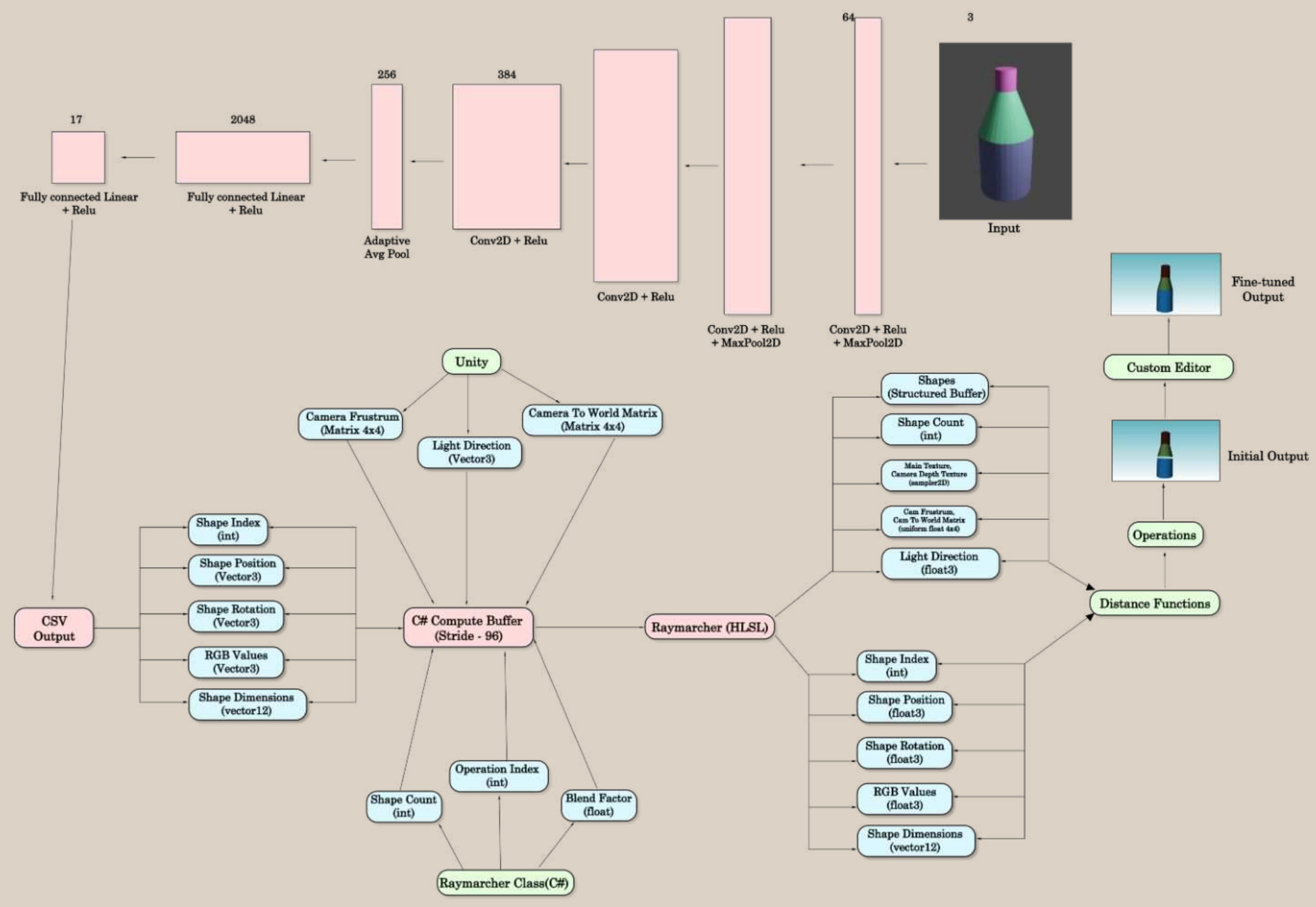
- We trained a Neural Radiance Field in ***pytorch3D***, based on the 2020 ECCV paper to reconstruct novel views for standardized datasets, consisting of batch and silhouette arrays + 3D camera coordinates.
- The input is a number of images of the target, from different angles and their corresponding cameras, and our network attempts to build a scalar field that allows us to generate views from any other angle.
- To fit the radiance field, we render it from the viewpoints of the target cameras, and we compare the results with the observed target images and target silhouettes.





side-by-side: Huber loss during training, views generated from a trained NeRF

- Loss is calculated as the **mean huber loss** (related to **smooth-L1 loss**) between rendered colours and the sampled target images, predicted masks and sampled target silhouettes.



REFERENCES...

- Quilez, Inigo. "Inigo Quilez." Inigo Quilez :: computer graphics, mathematics, shaders, fractals, demoscene and more. Accessed November 18, 2022. https://iquilezles.org.
- Shahrabi, Shahriar. "Raymarching in Unity." Medium, September 20, 2019. https://shahriyarshahrabi.medium.com/raymarching-in-unity-59c72664252a.
- Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." arXiv.org, March 19, 2020. https://arxiv.org/abs/2003.08934v2.
- PyTorch3D · A library for deep learning with 3D data. "PyTorch3D · A Library for Deep Learning with 3D Data." Accessed November 18, 2022. https://pytorch3d.org/.