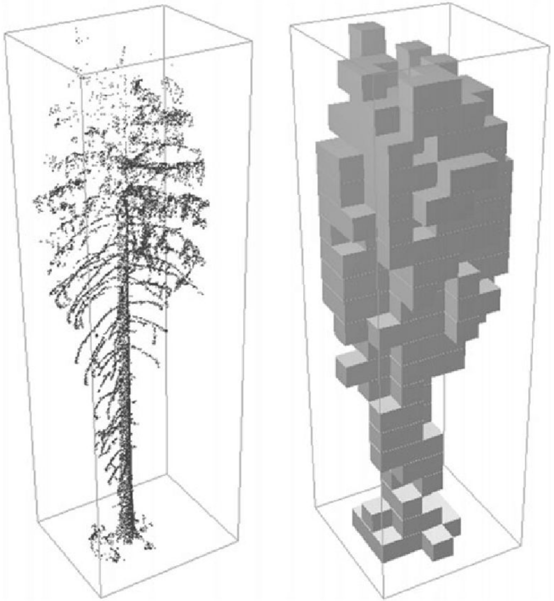
جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

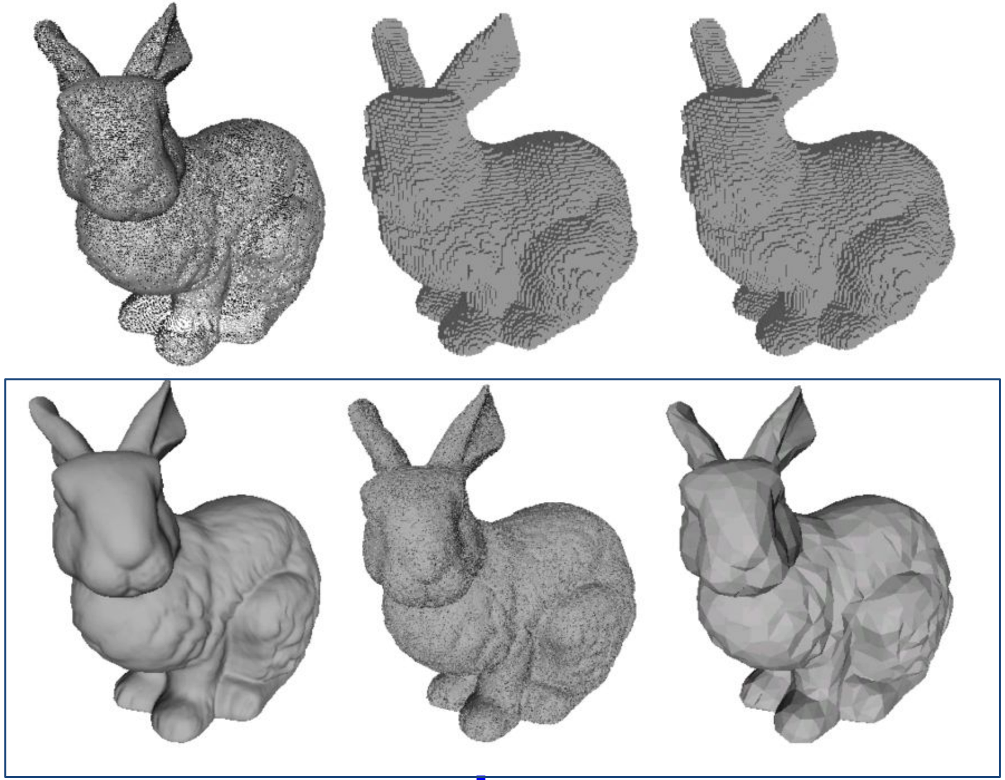
Differentiable Rendering

Abdullah Hamdi , IVUL group meeting

3D representation



point cloud vs Voxels



mesh

Yanir Kleiman , 2012

3D representation

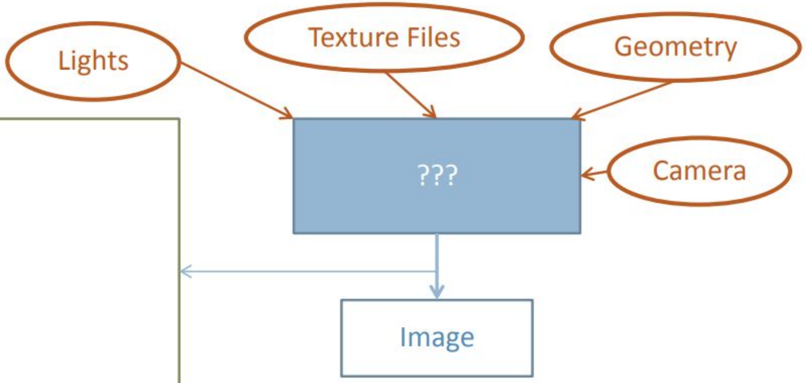
A Mesh is compact & useful for shading

Yanir Kleiman , 2012

Rendering in nutshill

What is Rendering?

- Consider:
  - Perspective
  - Occlusion
  - Color / Texture
  - Lighting
  - Shadows
  - Reflections / Refractions
  - Indirect illumination
  - Sampling / Antialiasing



Yanir Kleiman , 2012

Rendering in nutshill

Two Approaches

- Start from **geometry**
  - For each polygon / triangle:
    - Is it visible?
    - Where is it?
    - What color is it?

Rasterization

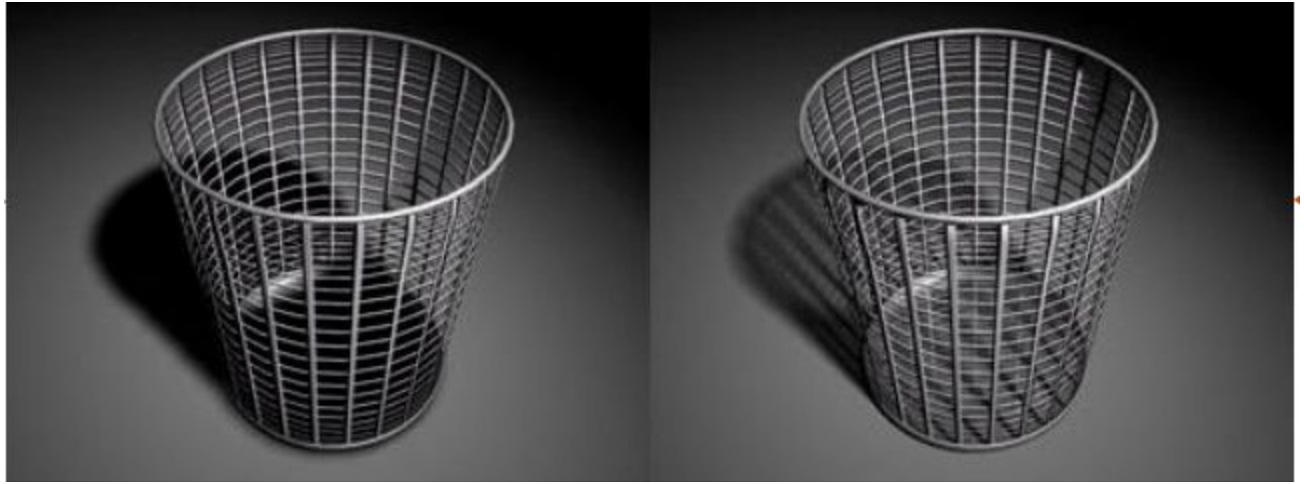
- Start from **pixels**
  - For each pixel in the final image:
    - Which object is visible at this pixel?
    - What color is it?

Ray Tracing

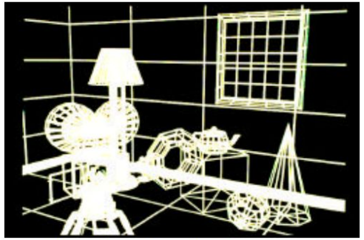
Yanir Kleiman , 2012

Shadows

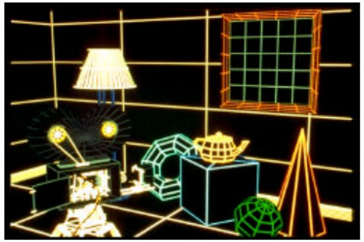
Rasterization vs ray tracing



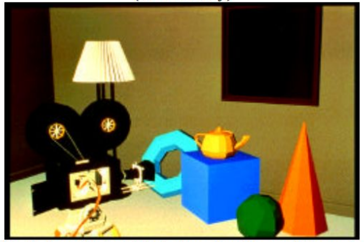
Degrees of realism



Polygonal model rendered in wire-frame
(no visibility)



With visibility.



Shaded rendering. Note how the faces of the cube and cone have different intensities depending on their orientation relative to the light source.

Smooth patches and shading including highlights.

Texture-Mapping improves the appearance of surfaces (a better lighting is used too).

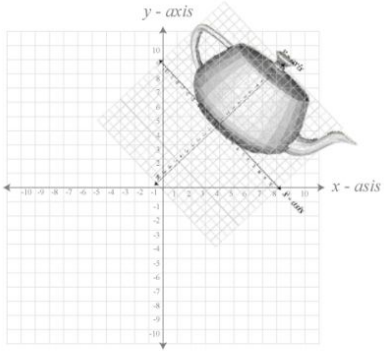
Shadows.

Rendering in nutshell

Rasterization – Graphics Pipeline

Model Transformation → Lighting → Projection → Clipping → Scan Conversion → Image

- Transform each triangle from **object space** to **world space**
- Local space -> Global space



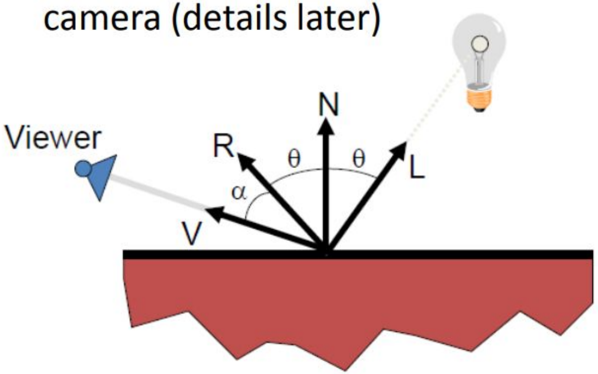
Yanir Kleiman , 2012

Rendering in nutshell

Rasterization – Graphics Pipeline

Model Transformation
↓
Lighting
↓
Projection
↓
Clipping
↓
Scan Conversion
↓
Image

- Computation is based on angles between light source, object and camera (details later)

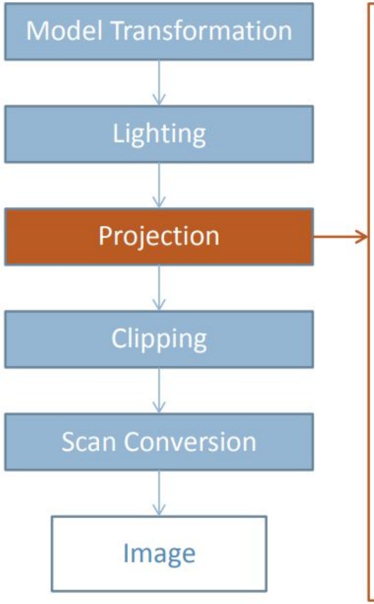


- Backface culling

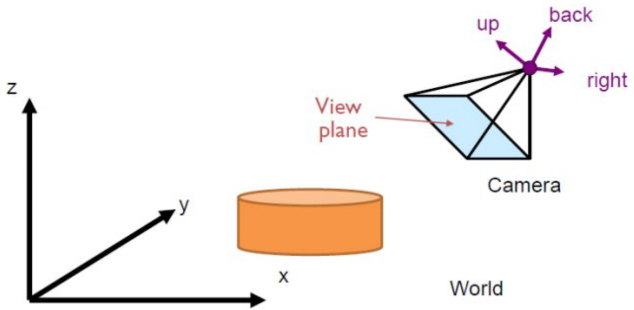
Yanir Kleiman , 2012

Rendering in nutshell

Rasterization – Graphics Pipeline



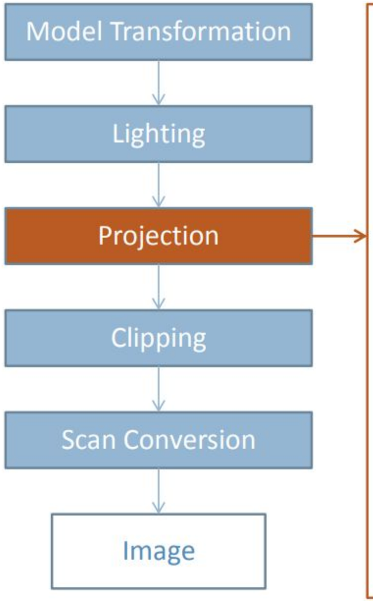
- Step 1: Transform triangles from **world space** to **camera space** (orthogonal transformation)



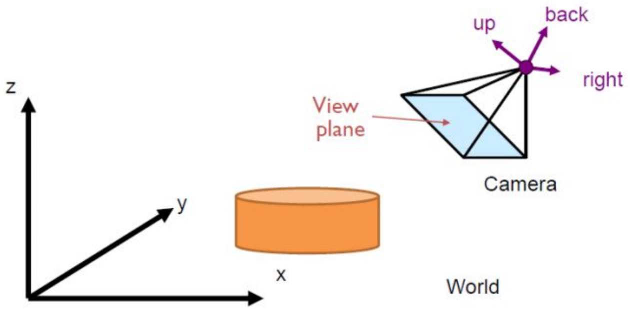
Yanir Kleiman , 2012

Rendering in nutshell

Rasterization – Graphics Pipeline



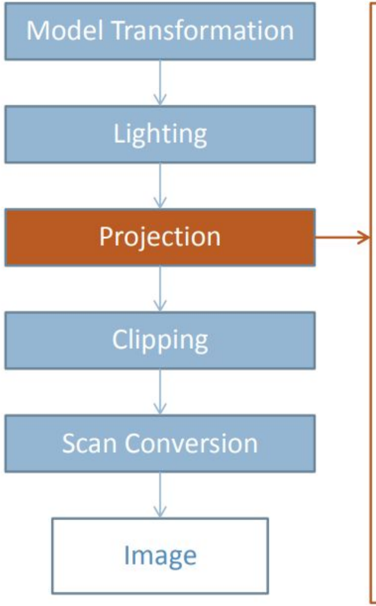
- Step 1: Transform triangles from **world space** to **camera space** (orthogonal transformation)



Yanir Kleiman , 2012

Rendering in nutshell

Rasterization – Graphics Pipeline



- Step 1: Transform triangles from **world space** to **camera space** (orthogonal transformation)
- Camera is at (0, 0, 0)
- X axis is right vector
- Y axis is up vector
- Z axis is "back vector" (away from camera)

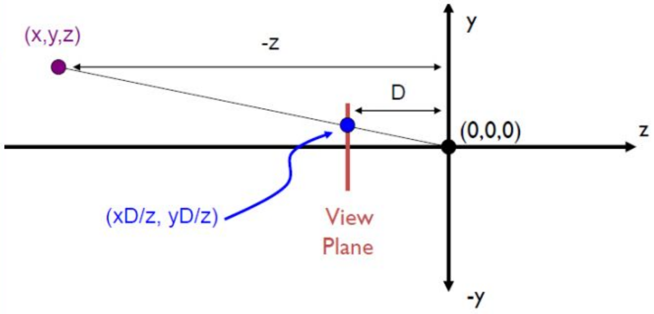
Yanir Kleiman , 2012

Rendering in nutshell

Rasterization – Graphics Pipeline

- Model Transformation
- Lighting
- **Projection**
- Clipping
- Scan Conversion
- Image

- Step 2: Perspective Projection
- Depends on focal length (D)



- Calculate Z-Buffer

Yanir Kleiman , 2012

Rendering in nutshell

Projection equation

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}, \quad \mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Rendering in nutshell

Rasterization – Graphics Pipeline

Model Transformation → Lighting → Projection → **Clipping** → Scan Conversion → Image

- Remove triangles that fall outside the clipping plane
- Determine boundaries of triangles partially within the clipping plane

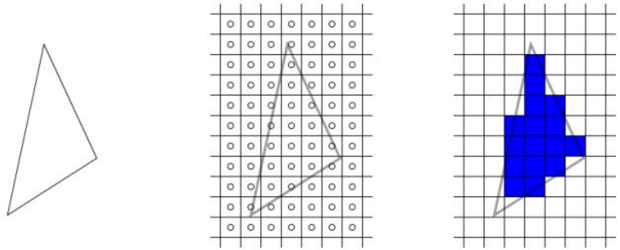
Yanir Kleiman , 2012

Rendering in nutshell

Rasterization – Graphics Pipeline

- Model Transformation
- Lighting
- Projection
- Clipping
- Scan Conversion
- Image

- Drawing the triangles in 2D
- Scanning horizontal scan lines for each triangle

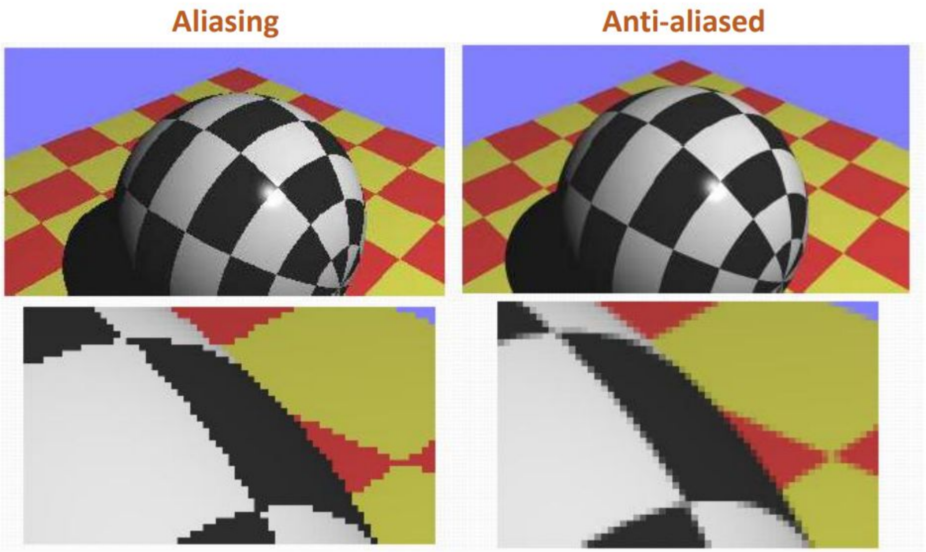


Yanir Kleiman , 2012

Rendering in nutshell

Rasterization – Antialiasing

- Aliasing examples

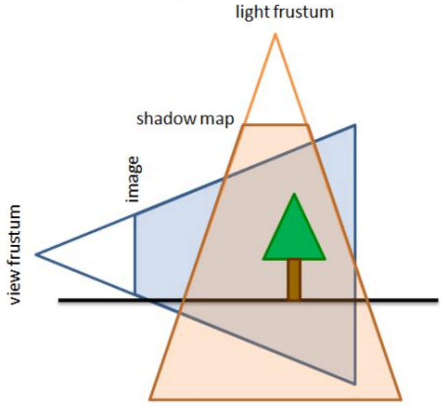


Yanir Kleiman , 2012

Rendering in nutshell

Rasterization – Shadow Maps

- Render an image from the light's point of view (the light is the camera)] **Shadow map**
- Keep "depth" from light of every pixel in the map
- During image render:
  Calculate position and depth on the **shadow map** for each **pixel** in the **final image** (not vertex!)
- If **pixel depth** > **shadow map depth** the pixel will not receive light from this source



Yanir Kleiman , 2012

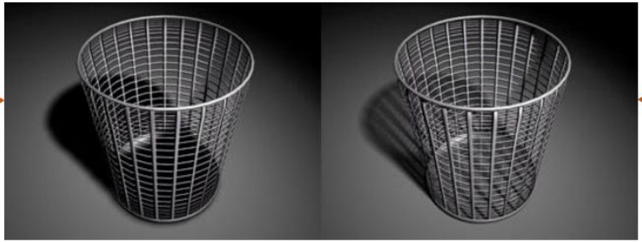
Rendering in nutshell

Rasterization – Shadow Maps

- This solution is not optimal
- Shadow map resolution is not correlated to render resolution – one shadow map pixel can span a lot of rendered pixels!
- Shadow aliasing
- Only allows sharp shadows
- Semi-transparent objects

Various hacks and complex solutions

Blurred hard shadows (shadow map)



True soft shadows (ray tracing)

Yanir Kleiman , 2012

Neural 3D Mesh Renderer (CVPR'18)

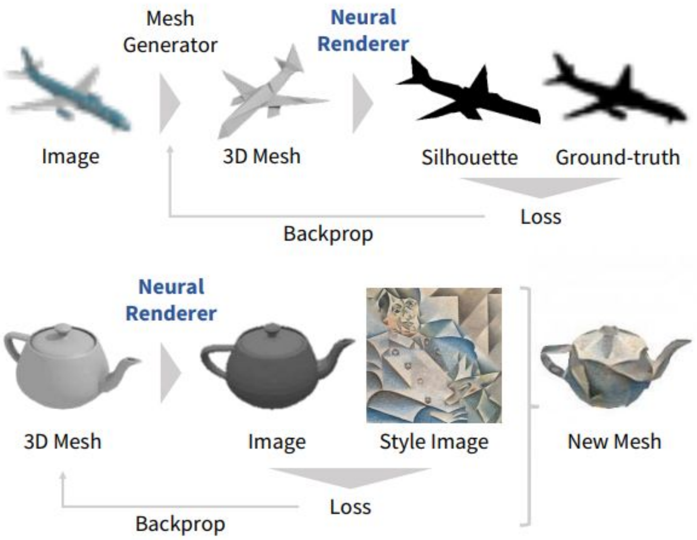
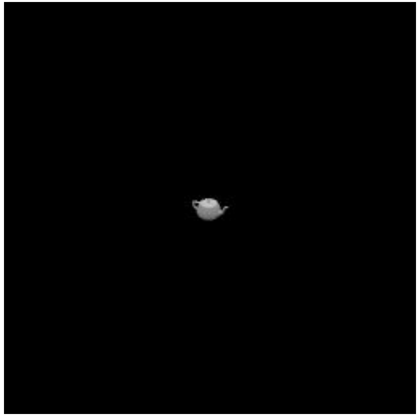


Figure 1. Pipelines for single-image 3D mesh reconstruction (upper) and 2D-to-3D style transfer (lower).



Neural 3D Mesh Renderer (CVPR’18)

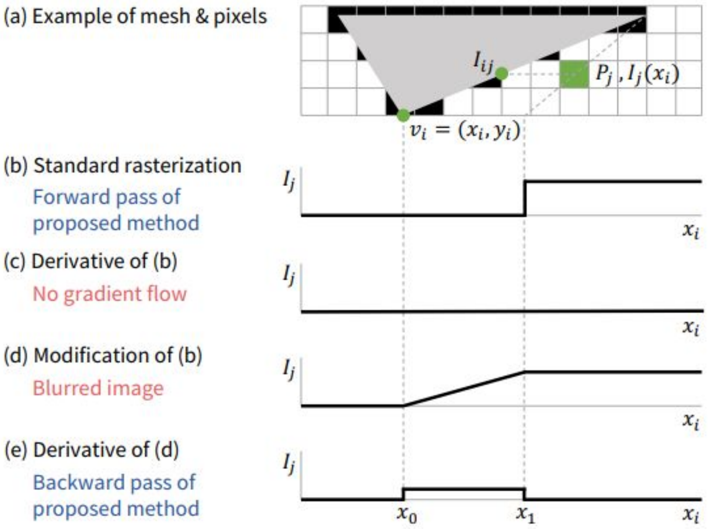


Figure 2. Illustration of our method. $\boldsymbol{v}_i = \{x_i, y_i\}$ is one vertex of the face. I_j is the color of pixel P_j. The current position of x_i is x_0. x_1 is the location of x_i where an edge of the face collides with the center of P_j when x_i moves to the right. I_j becomes I_{ij} when $x_i = x_1$.

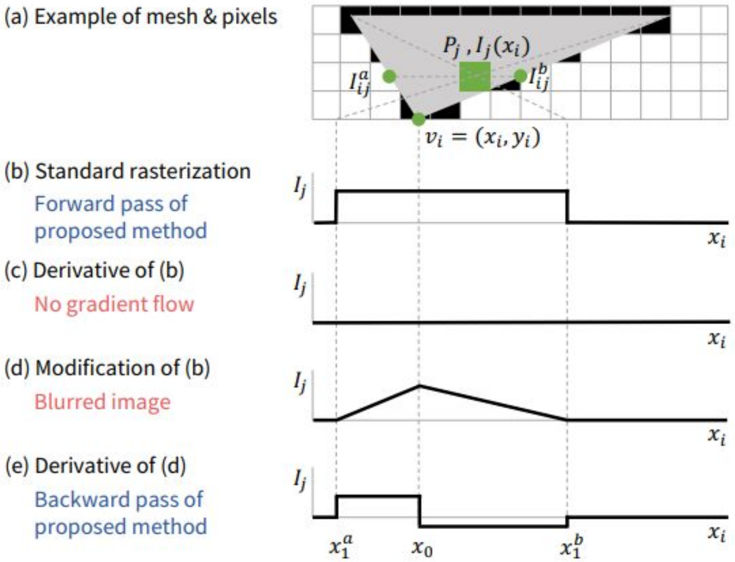


Figure 3. Illustration of our method in the case where P_j is inside the face. I_j changes when x_i moves to the right or left.

Neural 3D Mesh Renderer (CVPR'18)

$$\left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0} = \begin{cases} \frac{\delta_j^I}{\delta_i^x}; & \delta_j^P \delta_j^I < 0. \\ 0; & \delta_j^P \delta_j^I \geq 0. \end{cases}$$

$$\left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0} = \left.\frac{\partial I_j(x_i)}{\partial x_i}\right|^a_{x_i=x_0} + \left.\frac{\partial I_j(x_i)}{\partial x_i}\right|^b_{x_i=x_0}.$$

$$\left.\frac{\partial I_j(x_i)}{\partial x_i}\right|^a_{x_i=x_0} = \begin{cases} \frac{\delta_j^{I^a}}{\delta_x^a}; & \delta_j^P \delta_j^{I^a} < 0. \\ 0; & \delta_j^P \delta_j^{I^a} \geq 0. \end{cases}$$

$$\left.\frac{\partial I_j(x_i)}{\partial x_i}\right|^b_{x_i=x_0} = \begin{cases} \frac{\delta_j^{I^b}}{\delta_x^b}; & \delta_j^P \delta_j^{I^b} < 0. \\ 0; & \delta_j^P \delta_j^{I^a} \geq 0. \end{cases}$$

Neural 3D Mesh Renderer (CVPR'18)

$$I_j^l = \left(l^a + \left(\boldsymbol{n}^d \cdot \boldsymbol{n}_j\right) l^d\right) I_j.$$

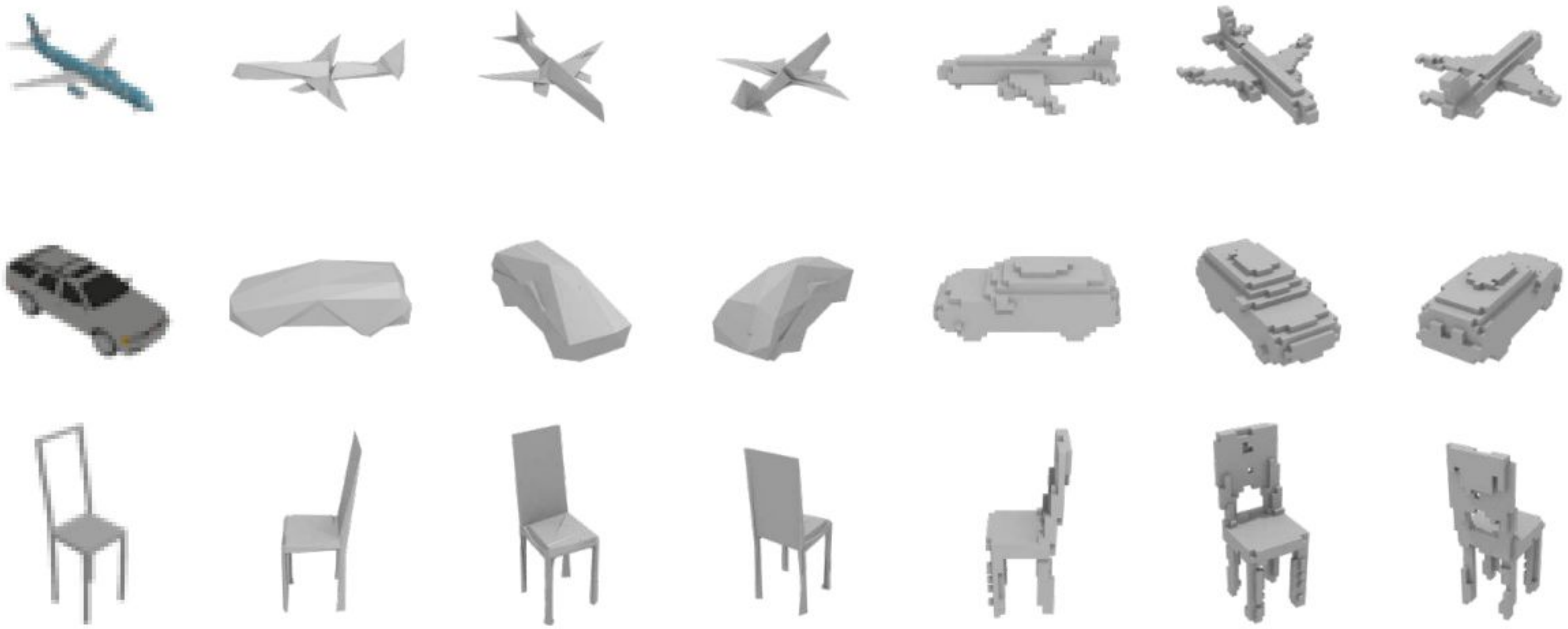
$$\begin{aligned}\mathcal{L}_{\mathrm{sl}}(x|\phi_i, s_i) &= -\frac{|\hat{s}_i \odot s_i|_1}{|\hat{s}_i + s_i - \hat{s}_i \odot s_i|_1}. \\ \mathcal{L}_{\mathrm{sm}}(x) &= \sum_{\theta_i \in \mathcal{E}} (\cos\theta_i + 1)^2.\end{aligned}$$

Neural 3D Mesh Renderer (CVPR'18)

	airplane	bench	dresser	car	chair	display	lamp	
Retrieval [36]	0.5564	0.4875	0.5713	0.6519	0.3512	0.3958	0.2905	
Voxel-based [36]	0.5556	0.4924	0.6823	**0.7123**	0.4494	0.5395	**0.4223**	
Mesh-based (ours)	**0.6172**	**0.4998**	**0.7143**	0.7095	**0.4990**	**0.5831**	0.4126	
	loudspeaker	rifle	sofa	table	telephone	vessel		mean
Retrieval [36]	0.4600	0.5133	0.5314	0.3097	0.6696	0.4078		0.4766
Voxel-based [36]	0.5868	0.5987	0.6221	**0.4938**	0.7504	0.5507		0.5736
Mesh-based (ours)	**0.6536**	**0.6322**	**0.6735**	0.4829	**0.7777**	**0.5645**		**0.6016**

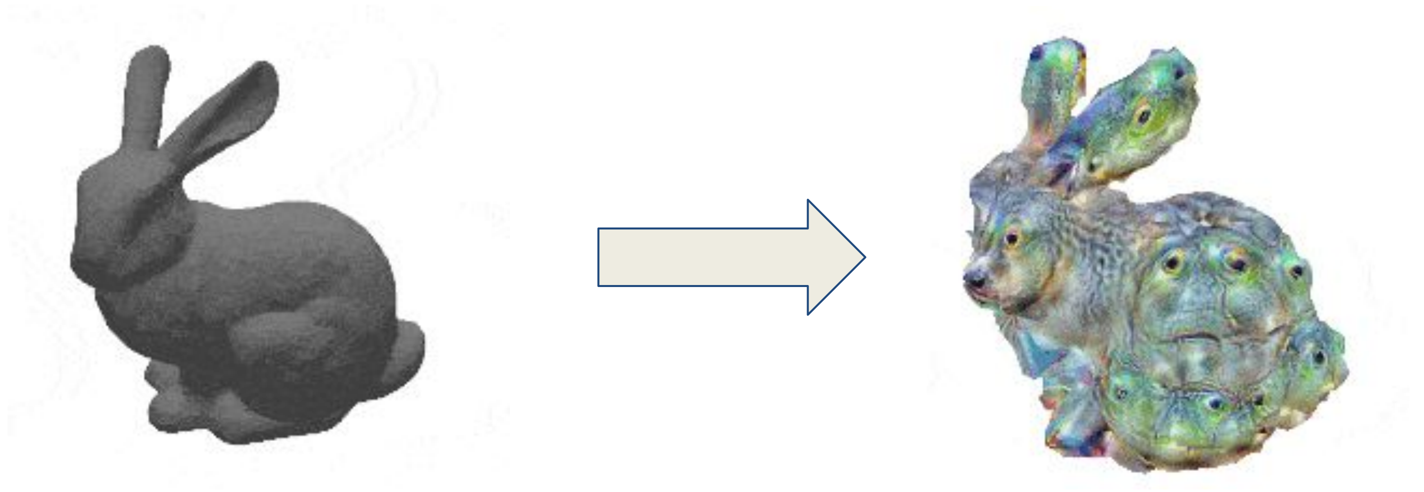
Table 1. Reconstruction accuracy measured by voxel IoU. Higher is better. Our mesh-based approach outperforms the voxel-based approach [36] in 10 out of 13 categories.

Neural 3D Mesh Renderer (CVPR’18)



applications

Deep dream

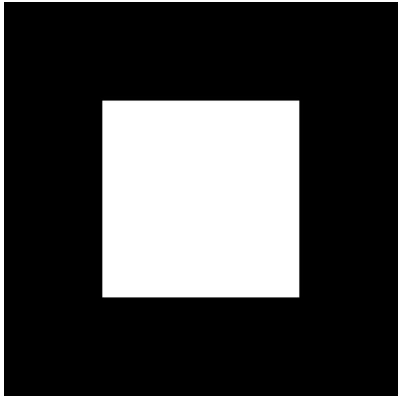


applications

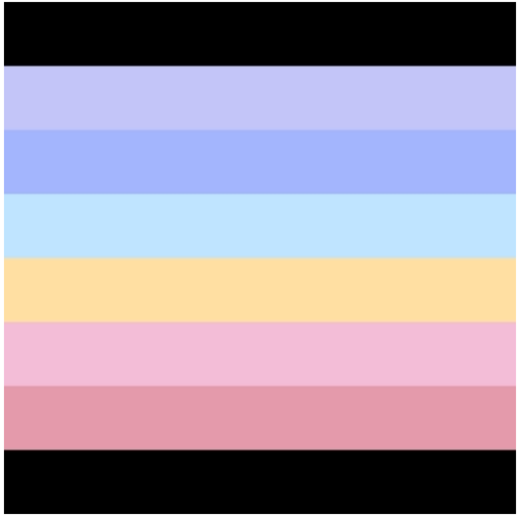
Style tansfer

applications

Optimizing mesh to match a projection

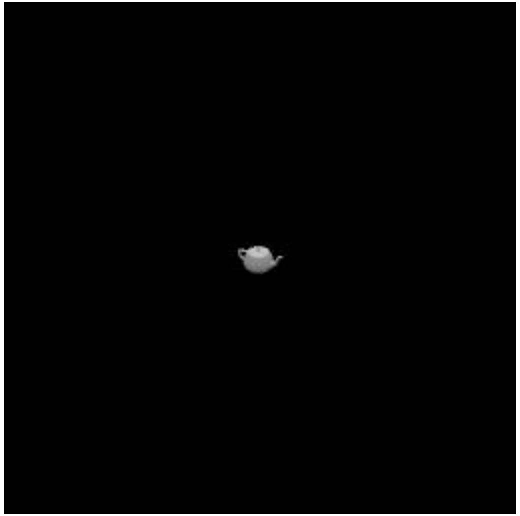



Optimizing texture



applications

Optimizing view point



Jupyter Notebook !!

Let’s all do it !!!

Neaural_render ●

Markdown Python [conda env:mytorch]

Neural Mesh Renderer in pytorch

[Kato etal (CVPR18')]

implements four examples to use Neaural Renderer with pytorch

--

go to the repo here and hit

```
sudo python setup.py install
```

importing relevent modules

In [1]:
```
from __future__ import division
import os
import argparse
import glob

import torch
import torch.nn as nn
import numpy as np
from skimage.io import imread, imsave
from tqdm import tqdm_notebook as tqdm

import imageio
from IPython.display import display, HTML

import neural_renderer as nr

current_dir = os.getcwd()
data_dir = os.path.join(current_dir, 'data/3d_renderer')
```

supporting functions

In [2]:
```
def make_gif(filename):
    with imageio.get_writer(filename, mode='I') as writer:
        for filename in sorted(glob.glob('/tmp/_tmp_*.png')):
            writer.append_data(imread(filename))
```