DIFFERENTIABLE RENDERING AND ADVERSARIAL LEARNING

E. Meloni

February 10, 2021

SAILab, University of Siena

TABLE OF CONTENTS

1. Introduction
2. Applications
3. Algorithms
4. 3D Adversarial Learning
5. Conclusions

Introduction

WHAT IS RENDERING?

Rendering is the process that takes a 3D scene as input and generates a 2D image as output.

A 3D scene is defined by :

1. geometry of the contained 3D objects
2. material properties of each object (colors, reflectivity, etc...)
3. lighting conditions
4. position and orientation of the camera

Rendering is a complex process which has not a straight-forward differentiation.

WHY DIFFERENTIABLE RENDERING?

Neural Models can be efficiently used for 2D and 3D reasoning.

However, most 3D estimation models rely on supervised training, which comes with costly annotations of all observable 3D properties.

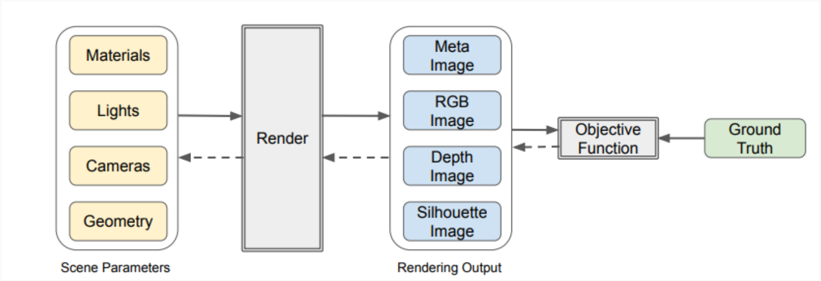
WHY DIFFERENTIABLE RENDERING?

Differentiable rendering enables 2D images to be used as supervisions for the 3D properties of the scene.

This enables the use of available datasets and cheaper annotations.

Differentiable rendering allows to seamlessly insert the rendering process in the learning pipeline and let gradients flow through it.

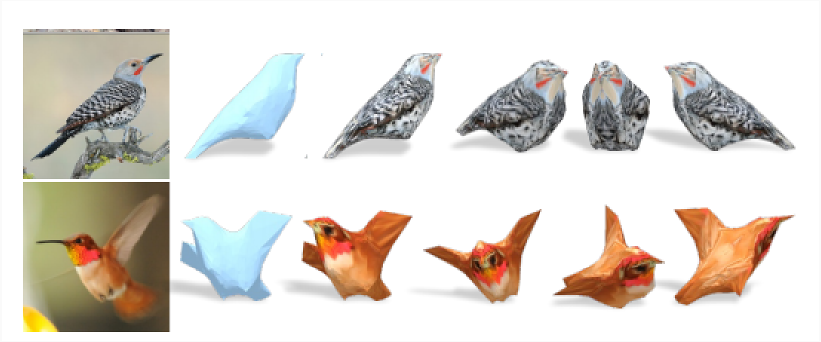
DIFFERENTIABLE RENDERING



Applications

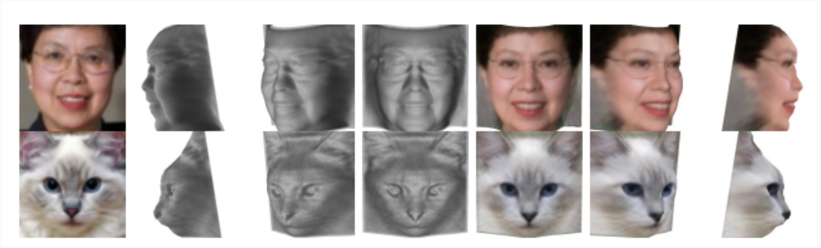
OBJECT RECONSTRUCTION

Using 2D Images as supervisions, a Differentiable Renderer can be used to learn the geometry and materials property of an object.



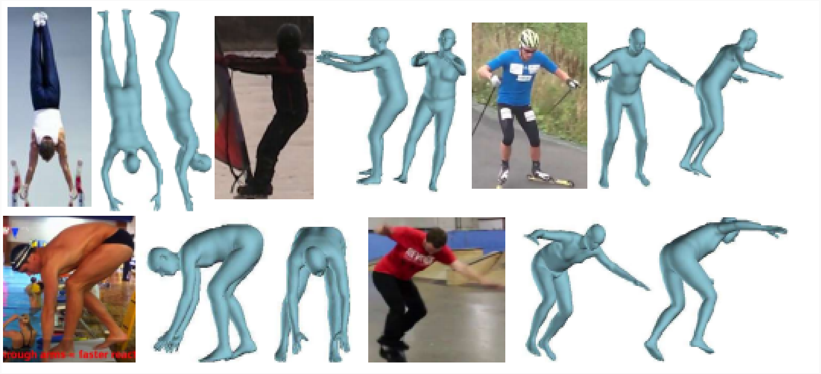
FACE RECONSTRUCTION

A more specialized application is using 2D Images as supervisions to learn the geometry and materials property of a face.



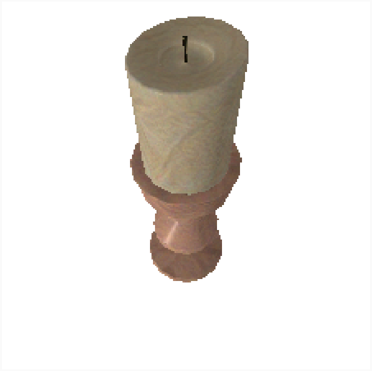
BODY POSE ESTIMATION

Differentiable Rendering can also be used to learn and optimize 3D body models to estimate human poses from 2D Images



3D ADVERSARIAL LEARNING

Differently from 2D Adversarial Learning, which attacks the 2D image input, 3D Adversarial Learning attacks the 3D object properties, such as geometry and materials, to induce a classification error in neural networks.



Algorithms

RENDERING: A MORE PRECISE FORMULATION

Φ_s: Shape parameters (geometry of the object)

Φ_m: Material parameters (colors, reflectivity, etc...)

Φ_l: Lighting parameters

Φ_c: Camera parameters (position, orientation, FOV, etc...)

I_c: RGB Image space

$\mathcal{R} : \Phi_s, \Phi_m, \Phi_l, \Phi_c \rightarrow I_c$: Rendering Function

A Differentiable Renderer computes the gradients $\partial I / \partial \Phi$ that optimize a specific loss function on the rendered images.

RENDERING PROCESS

Given a pixel in the image, the renderer (1) associates it with a face of the mesh and (2) computes its color based on material, camera and lighting parameters.

Step 1 (rasterization) is done by projecting the 3D scene onto the 2D camera space and selecting the enclosing face closest to the camera. This step yields a discrete face identifier and it is thus undifferentiable.

RENDERING PROCESS

Step 2 is instead differentiable, however the derivative of the image pixels with respect to vertex positions is always zero, therefore analytical derivatives are of little help for optimizing the geometry.

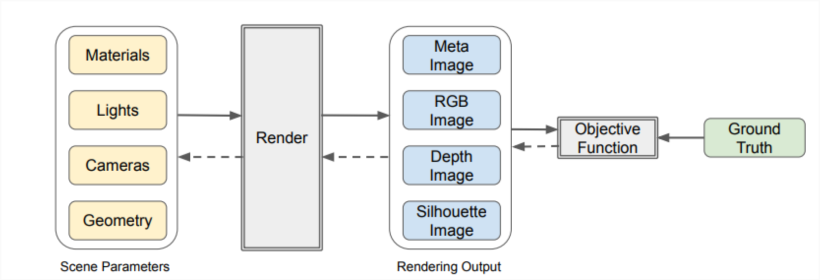
Gradients can be computed by: approximating the gradients themselves or by approximating the rendering (in particular the rasterization step).

3D Adversarial Learning

TASK DETAILS

We want to fool a generic image classifier, changing the input so to make it misclassify the rendered 3D object.

The classifier receives the 2D image output by the differentiable renderer and classifies it.



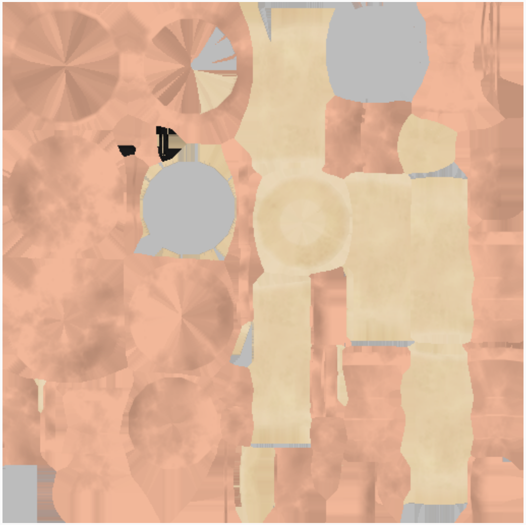
DIFFERENCES FROM 2D ADVERSARIAL LEARNING

Differently from 2D Adversarial Learning, we do not perturbate the direct input to the classifier.

Instead, we use the gradient that flowed through the differential renderer to perturbate the 3D Objects properties, for example its texture.

A texture can be seen as a sticker that is applied onto a 3D object whole surface. It defines the object colors and can approximate other physical properties such as roughness.

EXAMPLE OF A TEXTURE



TEXTURE SALIENCY

Gradient flowing to the material properties enables the estimation of **Texture Saliency**, which are Saliency Maps computed on a material texture.

Texture Saliency Maps highlight the pixels in the texture that most influenced the decision of the classifier.

They can be used to pick the pixels that will change the prediction the most when perturbated.

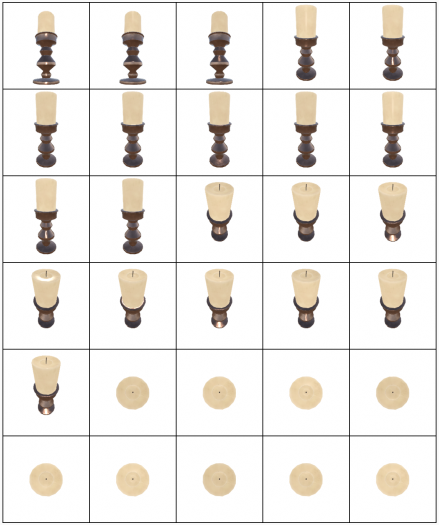
TEXTURE SALIENCY

PGD-TEXTURE ATTACK

We are studying a variation of the PGD attack directed to the texture of the 3D Object.

For each 3D object, we choose several viewpoints from which to render it. This allows us to attack the object when viewed from many directions.

MULTI-VIEW RENDERING



PGD-TEXTURE ATTACK

T: Original texture (fixed)
δ: perturbation (learnable)
ϵ: max L2-norm of the perturbation
α: rate of update (similar to learning rate in GD)

$\mathcal{R}(T, \delta) \rightarrow I_c$: Given the texture T and the perturbation δ, outputs the rendering from all viewpoints.
$\mathcal{C}(I_c) \rightarrow c$: Given an image, classifies the image by assigning it a score for each possible class.

PGD-TEXTURE ATTACK

Every epoch, we compute $c = \mathcal{C}(\mathcal{R}(T, \delta))$

An object of class $\hat{c}$ is correctly classified if $\arg\max c = \hat{c}$

$L = CrossEntropyLoss(c, \hat{c})$ is our loss, it ignores wrong classifications.

We backpropagate the loss L back to δ, obtaining $\nabla\delta$.

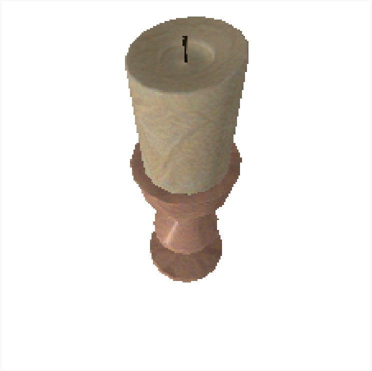
PGD-TEXTURE ATTACK

Then we update δ as follows:

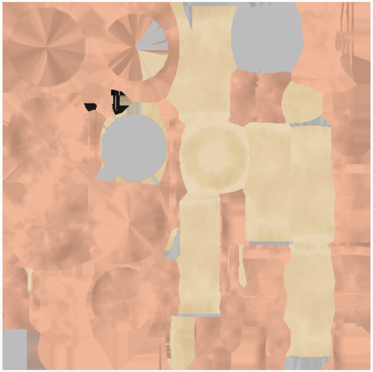
1. $\delta = \delta + \alpha \cdot \nabla\delta$: perturbate with gradient
2. clamp $T + \delta$ between 0 and 1
3. clamp δ to have norm at most ϵ

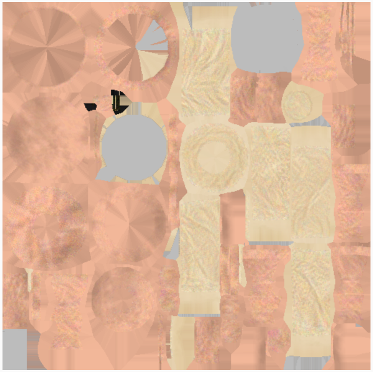
We then pass to the next epoch.

ATTACKED OBJECT EXAMPLE



ATTACKED OBJECT EXAMPLE





3D ADVERSARIAL ATTACK TRANSFER

Differentiable Renderers are still far from photorealism.

We want to verify that the attacks made with differentiable renderers can transfer on photorealistic (non differentiable) renderers.

3D ADVERSARIAL ATTACK TRANSFER

We can use Virtual Environments, such as SAILenv, to render photorealistic versions of the attacked 3D Object, before and after the attack.

The same classifier is then used to compute the accuracy on classifications and evaluate how the attack has transferred to the photorealistic renderer.

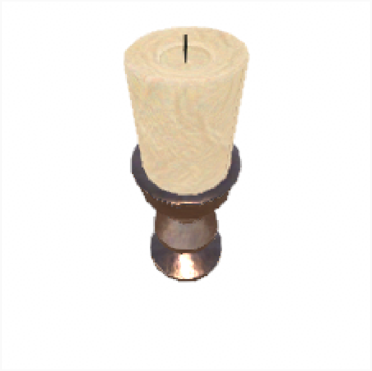
WIP: TEXTURE SALIENCY TRANSFER

Given a viewpoint, A, B are the images rendered respectively by the differentiable renderer and the non differentiable one.

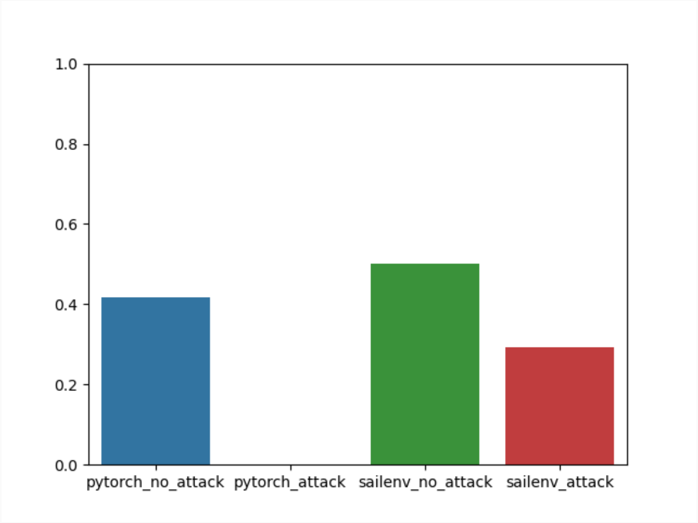
The pixel A_i, A_j corresponds to the texture pixel $T_{\hat{i}}, T_{\hat{j}}$. Since the viewpoint is the same, we can also associate that texture pixel to the pixel B_i, B_j in the same position of the image.

This is only an approximation, as it depends on how the non differentiable renderer samples the texture.

ATTACKED OBJECT TRANSFER EXAMPLE



ATTACKED OBJECT TRANSFER EXAMPLE



Conclusions

OPEN PROBLEMS

- **Little support for embedded environments**: currently hardware and processing power requirements are out of reach.
- **There is no easy and standard support for extensions**: each library need new algorithms to be implemented from scratch
- **Limited functionalities**: each library has several limitations with respect to non differentiable renderers, such as faces primitives other than triangles, output formats, animation etc.

OPEN PROBLEMS

- **Little support for light and material models**: Currently all libraries offer only non-pbr rendering, supporting only color textures and neglecting the impact of light reflected on objects.
- **No benchmarking or debugging**: There is currently no tool that allows for spotting inaccuracies in rendering and gradient estimation.
- **No model sharing**: each library has its own format and it is difficult to export a model for another library

CONCLUSIONS

Differentiable rendering is a novel field, but is quickly maturing.

It allows researcher to develop neural model that understand the 3D world through 2D images.

It will help reducing the need for costly 3D data collection and annotation.

Hopefully it will reach real-time performances in the near future.

CONCLUSIONS

3D Adversarial Learning is a promising approach to transfer Adversarial Attacks to practical real world scenarios.

It is very efficient to attack the 3D objects when rendered by the differentiable renderer.

We are making progress on transferring it to photo-realistic non differentiable renderers.

Thank you for listening!