Soft Rasterizer: A Differentiable Renderer

Paper Review

Actual Paper: https://arxiv.org/pdf/1904.01786.pdf

Ayush Kumar

4 February 2020

1 Introduction

Non-Differentiability in Image Formation

The paper deals with the issue of non-differentiability inherent in the standard rendering pipline. The common rendering stages used in the industry for rendering 3D scenes first involve applying a couple of affine transformations and a projective/orthographic transform all of which are fully differentiable (for that matter all matrix transforms are differentiable) with respect to the scene parameters. This is then followed by two non-differentiable steps, a discrete sampling and z-buffering operation called rasterization which involves iterating over the pixels and sampling the interpolated color of the vertices of the triangle lying closest to the image plane.

To understand the non-differentiability of the traditional rasterization step, consider the following situation. A sphere, which is just occluded by a plane in front of it, is initially invisible (does not affect the output 2D image). Changing the scene parameters θ by a delta amount, the contribution or the influence of the occluded sphere triangles to the final image suddenly jumps to a non-zero value. Clearly, the gradient of the pixels with respect to the coordinates of the sphere triangles is not defined for this value of scene parameters. This introduces the non-differentiability in the image formation process. Simply stated, the non-differentiability is introduced due to the occluded triangles not contributing in the image formation process.

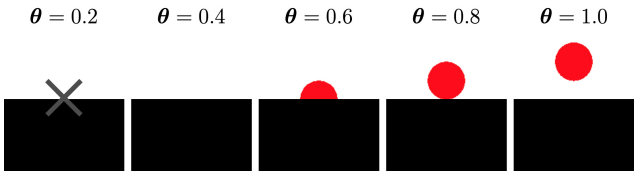

Figure 1: Changes in rendered pixels with θ

Why is differentiability desired?

A good question at this point would be to ask why do we want differentiability in the rendering pipeline. The answer is that we want to integrate the rendering pipeline with deep neural nets to achieve the long sought tasks of 3D Reconstruction from single/multi-view images or Pose Estimation from a given monocular video feed or other similar tasks which involve backpropagation of the gradients from image pixels to 3D scene parameters.

Previous approaches have tackled this problem either by introducing new parametrizations of the scene or by approximating the gradients of the pixel colors using hand-crafted derivative functions.

This paper formulates the process of rasterization and the subsequent z-buffering using a probabilistic approach in which all the triangles of the scene (including occluded ones) contribute to image formation and their contribution is modelled by a smooth sigmoid function $\mathcal{D}$ and a differentiable aggregator $\mathcal{A}_S$ with two tunable parameters σ and γ respectively. This approach seems to be a more general approach to image formation which reduces to the actual rasterization as $\sigma \to 0$ and $\gamma \to 0$. A brief overview of the formulation is described below.

2 Soft Rasterization

Probabilistic Formulation

Consider the scene as a set of triangles with vertex attributes (In fact, any mesh is triangulated before exporting) and the screen as an array of pixels. The influence of triangle j on pixel i or the weight (w_j^i) should be controlled by two factors,

1. **Planar Influence:** The planar or the projected "distance" of the triangle j from pixel i is evaluated by the distance metric $d(i, j)$ and the influence is modelled by the $\mathcal{D}_j^i$ term.
2. **Depth Influence:** The influence due to depth of the triangle i.e the z-coordinate in the NDC (Normalized Device Coordinates) which describes how far the triangle is from the image plane. This term is estimated by $exp(\frac{Z_{far} - Z_j^i}{\gamma \cdot (Z_{far} - Z_{near})})$, where Z_j^i is the distance of the pixel from its projection on the triangle and γ is a parameter to tune this influence.

Hence, the weight should be proportional to the product of these two factors,

$$w_j^i \propto \mathcal{D}_j^i \cdot exp(z_j^i/\gamma)$$

where $z_j^i = \frac{Z_{far} - Z_j^i}{Z_{far} - Z_{near}}$. Also, we need to assign some weight w_b^i to the background color for which the term $\mathcal{D}_j^i$ makes no sense, so $w_b^i \propto exp(\epsilon/\gamma)$ where ϵ is a constant that can be tweaked around. Now, as is always the case with weights, $\sum w_j^i + w_b^i = 1$. Hence we normalize the weights by dividing by their sum,

$$w_j^i = \frac{\mathcal{D}_j^i exp(z_j^i/\gamma)}{\sum_k \mathcal{D}_k^i exp(z_k^i/\gamma) + exp(\epsilon/\gamma)}$$

$$w_b^i = \frac{exp(\epsilon/\gamma)}{\sum_k \mathcal{D}_k^i exp(z_k^i/\gamma) + exp(\epsilon/\gamma)}$$

This looks a bit similar to the softmax function. All that is left now, is to estimate the $\mathcal{D}_j^i$ term. Intuitively, the influence should be inversely proportional to the "distance" if the pixel lies outside the triangle (the triangle projected onto the image plane), and proportional to the distance when the pixel lies inside. Putting these abstract concepts together, $\mathcal{D}_j^i$ is best modelled by,

$$\mathcal{D}_j^i = sigmoid(\delta_j^i \cdot \frac{d^2(i,j)}{\sigma})$$

where $\delta_j^i = +1$ or -1 when the pixel is inside or outside the triangle respectively. Also $d(i,j)$ is any distance metric and σ is another tunable factor. For $d(i,j)$, the authors preferred to use the minimum euclidean distance from the pixel center to any one of the sides of the triangle projected onto the image plane. It should be noted that these ideas are coherent with our intuition. When the pixel is outside the triangle ($\delta_j^i = -1$), the influence decays almost exponentially with distance squared, while when the pixel is inside ($\delta_j^i = +1$), the influence grows with distance (the farthest the pixel can be from the sides of the projected triangle, when inside it, is when the pixel is at its incentre in which case the planar influence should be maximum).

Now, we simply take the weighted sum of the interpolated colors of the vertices of each triangle in the scene and the background color. This is the aggregator function $\mathcal{A}_S$,

$$I^i = \mathcal{A}_S(\{C_j\}) = \sum_j w_j^i \cdot C_j^i + w_b^i \cdot C_b$$

Silhouette Aggregation Function

Additionally, the authors have also defined a silhouette aggregation function $\mathcal{A}_O$ for modelling shadows. The silhouette of pixel i is (I_s^i) which denotes the probability of the pixel being covered by at least one triangle. Since we can think of $\mathcal{D}_j^i$ as a probability distribution over all triangles modelling the probability of the pixel being covered by that triangle, the silhouette aggregation function can be estimated by,

$$I_s^i = \mathcal{A}_O\{\mathcal{D}_j^i\} = 1 - \prod_j (1 - \mathcal{D}_j^i)$$

3 Unsupervised 3D Reconstruction

The differentiable rendering function, call it $R_{soft}(m, \phi)$ where m is the mesh and ϕ the viewpoint or camera, could be attached with a task-specific Loss Function and then could be used for various 3D reasoning tasks such as reconstruction, pose-estimation and others. Hiroharu et. al. [**1**] have demonstrated how to employ a differentiable renderer to perform 3D mesh reconstruction from single/multi-view images. The reconstruction can be performed by deforming a template mesh of 642 vertices using an Encoder-Decoder architecture. The Encoder consists of a feature extractor (3 Conv layers followed by 3 FC layers) which takes an input image and gives a latent representation (of 512 dimensions) of the mesh. The decoder is a shape generator (3 FC layers with input and output sizes as 512 and 642×3 respectively) or a color generator. The feature extractor combined with the shape generator gives the Mesh generator function $G(x)$. We try to learn $G(x)$ by minimizing the loss between the input images $\{s_i\}$ and the rendered images $\{\hat{s}_i = R_{soft}(G(s_i), \phi_i)\}$ considering the viewpoint (or 'eye') to be known for the input images. For the purpose of reconstruction, the authors have designed the Laplacian or the loss function as a weighted sum of three losses: Color Loss $\mathcal{L}_c$, Silhouette Loss $\mathcal{L}_s$ and Geometric Loss $\mathcal{L}_g$. $\mathcal{L}_g$ acts as a regularizer which ensures that $G(x)$ generates a "smooth" mesh.

$$\mathcal{L} = \mathcal{L}_s + \lambda \cdot \mathcal{L}_c + \mu \cdot \mathcal{L}_g$$

$\mathcal{L}_c$ is simply the L1 norm of difference between the rendered and the input images. Though the paper doesn't mention how the geometric loss $\mathcal{L}_g$ is evaluated, it seems equivalent to the smoothness loss $\mathcal{L}_{sm}$ used by Hiroharu et. al. [**1**]. It tries to ensure that intersection angles of all faces are close to 180 degrees.

$$\mathcal{L}_c(\hat{I}_c, I_c) = ||\hat{I}_c - I_c||_1$$

$$\mathcal{L}_s(\hat{I}_s, I_s) = 1 - \frac{||\hat{I}_s \otimes I_s||_1}{||\hat{I}_s \oplus I_s - \hat{I}_s \otimes I_s||_1}$$

$$\mathcal{L}_{sm}(s_i) = \sum_{\theta_i \in E} (cos\theta_i + 1)^2$$

where E is the edge set of $G(s_i)$ and θ_i is the angle between the two faces which contain the ith edge in E.

References

[1] H. Kato, Y. Ushiku, and T. Harada. Neural 3d mesh renderer. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3907–3916, 2018. paper