Differentiable rendering

D.A. Forsyth, UIUC
Rendering and differentiability

- NeRF rendering model is differentiable in density
  - desirable feature for inference - you could do inference by descent
    - as in NeRF - figure out density that produces images
      - by gradient descent
- We wish to differentiate rendered image
  - with respect to surface parameters, illumination, etc.
  - three issues
    - interaction between renderer and representation
    - intrinsic failures of differentiability
    - sampling
- Applications
  - “inverse graphics” - vision by inference
  - improved rendering - “learn” to render
We have seen one DR

• The NeRF rendering process is differentiable
  • in sigma, color
  • and so in parameters
• BUT
  • many gradients might be zero, or close to it
    • large sigma=small contribution=small gradient
  • there is variance in the estimate of the colors
    • and so in gradient estimates
Some very simple rendering problems - I

- One triangle
  - Diff wrt vertices in 3D
    - projection is clearly differentiable, so issue is 2D
  - Constant color, no shading
    - gradients only at boundary
    - sampling complicates the issue slightly
    - filter
Think of image as function of $x$, $y$.

Whether this function has a derivative or not depends on what kinds of derivatives you allow. You can think of derivative as delta-functions. As soon as you smooth the image very slightly, derivative is unequivocal, but is concentrated on the boundary.
In this sampling model, every gradient is zero for this triangle (and almost every triangle, almost all the time)

Sampling: pixel value is given by value at center of pixel

But the image has to be discretized….

Other problems: jaggies, etc
All due to aliasing!
Better sampling model - subdivide the pixel into fine grid, report weighted sum of samples in that grid. This approximates filtering the image before sampling it, so reducing aliasing effects. The pixel value will have a gradient wrt triangle vertices.
Some very simple rendering problems -II

- One triangle
  - Diff wrt vertices, shading, texture
Standard procedure:

at each vertex, we know:

- a shading value;
- texture coordinates.

for each location in triangle, compute a value by:

- interpolate shading values (bilinear);
- interpolate texture coordinates (bilinear);

\[
\text{value} = (\text{texture value at interpolated coords}) \times (\text{interpolated shading})
\]

value is diff. wrt texture (if properly filtered), shading at verts
pixel is diff. wrt vertex positions; non-zero if properly filtered
Some very simple rendering problems -III

• Two triangles
  • Constant color
  • Diff wrt vertices
We now have a discontinuity caused by occlusion. You could think of derivative of pixel value wrt depth as a delta function, but this isn’t particularly helpful.

Clearly, we must smooth the pixel value over depth in some way.

Q: How?

One common A:

\[
\text{Pixel} = \sum_i \frac{e^{-d_i}}{\sum_j e^{-d_j}} \text{triang}_i
\]
The basic difficulty with smoothing....

- You get at least one, maybe both of:
  - a bad rendering
  - highly concentrated gradients in image space
The basic difficulty with smoothing....

- You get at least one, maybe both of:
  - a bad rendering
  - highly concentrated gradients in image space
    - relatively few pixels are affected by a set of vertices
    - this is predictable
      - meshes have small triangles so they look curved...
      - equivalently
        - meshes are quite good for rendering,
          - but not great at encoding shape
            - too many parameters that have little significant effect
Variations here address three issues:
- More complex illumination models (expensive, but straightforward)
- Different smoothing procedures
- Correct but unhelpful gradients
Variations here address three issues:

- More complex illumination models (expensive, but straightforward)
- Different smoothing procedures
- Correct but unhelpful gradients

TABLE 1
Overview of the representative differentiable rendering methods. They are classified by the four main underlying data representations.

<table>
<thead>
<tr>
<th>Data Repr</th>
<th>Type</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mesh</td>
<td>Analytical derivative</td>
<td>Local minima in geometry optimization</td>
</tr>
<tr>
<td></td>
<td>Approx. gradient</td>
<td>Handcrafting gradient calculation</td>
</tr>
<tr>
<td></td>
<td>Approx. rendering</td>
<td>Not precise rendering result</td>
</tr>
<tr>
<td></td>
<td>Global illumination</td>
<td>Computationally too expensive [28]</td>
</tr>
</tbody>
</table>

Kato et al 20
Implicit surfaces

• Advantage:
  • representation is at “long scale”
  • parameters represent overall shape

• Disadvantage:
  • gradients might be hard to work with
Quiz: what could go wrong?

Rendering of sphere, constant color, no shading, using ray tracer.

Q1: what is the gradient of pixel value wrt sphere center pos’n, radius “like”?

Q2: How is this affected by sampling?
Quiz: what could go wrong?

Rendering of sphere, constant color, no shading, using ray tracer.
Quiz: what could go wrong?

Rendering of sphere, constant color, no shading, using ray tracer.
Quiz: what happens if I shade the sphere?

Q1: what is the gradient of pixel value wrt sphere center pos’n, radius, lighting “like”?

Q2: How is this affected by sampling?
Quiz: shade +texture the sphere?

Rendering of sphere, shading, textured using ray tracer.

Q1: what is the gradient of pixel value wrt sphere center pos’n, radius, texture coordinates, lighting “like”?

Q2: How is this affected by sampling?
Quiz: two spheres?

Rendering of sphere, constant color, using ray tracer.

Q1: what is the gradient of pixel value wrt sphere center pos’n, radius, “like”?

Q2: How is this affected by sampling?
Point clouds

- **Point based rendering**
  - large cloud of points in 3D
  - each point has an associated color
  - render by
    - accumulating point influence on pixels
      - image plane easy,
      - obviously differentiable if weights OK
        - depth is a problem
Depth

- At a pixel, estimate a key depth
  - closest point
  - closest k points
- use this to weight point colors while smoothing
  - weight down points that are further away
Representations of geometry

<table>
<thead>
<tr>
<th>Representation</th>
<th>Type</th>
<th>Strengths</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mesh</td>
<td>Analytical derivative</td>
<td>Usage of advanced forward rendering</td>
<td>Local minima in geometry optimization</td>
</tr>
<tr>
<td></td>
<td>Approximated gradient</td>
<td>Usage of advanced forward rendering</td>
<td>Handcrafting gradient calculation</td>
</tr>
<tr>
<td></td>
<td>Approximated rendering</td>
<td>Auto-diff support</td>
<td>Not precise rendering result</td>
</tr>
<tr>
<td></td>
<td>Global illumination</td>
<td>Realistic rendering outcome</td>
<td>Computationally too expensive</td>
</tr>
<tr>
<td>Voxel</td>
<td>Occupancy / transparency</td>
<td>Simple, easy to optimize</td>
<td>Excessive memory consumption</td>
</tr>
<tr>
<td></td>
<td>Signed distance function</td>
<td>Efficient ray trace</td>
<td>Not suitable for transparent volume</td>
</tr>
<tr>
<td>Point Cloud</td>
<td>Point cloud</td>
<td>Easy to render and differentiate</td>
<td>Pseudo-sizing, lack of surface</td>
</tr>
<tr>
<td></td>
<td>RGBD image</td>
<td>Ordered point cloud by default</td>
<td>Point density reduces with the distance</td>
</tr>
<tr>
<td>Implicit</td>
<td>Occupancy / transparency</td>
<td>Simple, easy to optimize</td>
<td>Inside and outside is ambiguous</td>
</tr>
<tr>
<td></td>
<td>Level set</td>
<td>Clear object boundary</td>
<td>Difficult to optimize</td>
</tr>
</tbody>
</table>

TABLE 3
Comparison between different algorithm types.

Kato et al 20