65 ideas from “Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations” (2019) by Vincent Sitzmann, Michael Zollhöfer, Gordon Wetzstein

Actual paper: https://arxiv.org/abs/1906.01618

1. Key idea: represent a scene as a function ϕ that maps a spatial location x to a feature representation v of learned scene properties at that spatial location…
2. ϕ: R^3 -> R^n such that x |–> phi(x) = v
3. The feature representation v may encode visual information such as surface color or reflectance, or higher-order information such as the signed distance of x to the closest scene surface.
4. This continuous formulation can be interpreted as a generalization of discrete neural scene representations.
5. For example, voxel grids discretize R^3 and store features in the resulting 3D grid
6. Also, point clouds, which may contain points at any position in R^3, but only sparsely sample surface properties of the scene.
7. In contrast, phi densely models scene properties and can, in theory, model arbitrary spatial resolutions, since it is continuous over R^3 and can be sampled with arbitrary resolution.
8. In practice, phi is represented as a multi-layer perception, and thus spatial resolution is limited by the capacity of the MLP.
9. Since the input to phi are world coordinates, phi is explicitly aware of 3D structure.
10. This allows interacting with phi via the toolbox of multi-view and perspective geometry that the physical world obeys,...
11. … only using learning to approximate the unknown properties of the scene itself.
12. We introduce a neural rendering algorithm **ϴ**: X x R^(3x4) x R^(3x3) → R^(HxWx3) such that (phi, E, K) |--> theta(phi, E, K) = I
13. The key complication in rendering a scene represented by phi is that geometry is represented implicitly.
14. For example, the surface of a wooden table is defined as the subspace of R^3 where phi undergoes a change from a feature vector representing free space to one representing wood.
15. To render a single pixel in the image observed by a virtual camera, 2 subproblems have to be solved:
16. (1) find the world coordinates of the intersections of the respective camera rays with scene geometry
17. (2) mapping the feature representation v at that spatial coordinate to a color
18. First, we will propose a neural ray marching algorithm with learned, adaptive step size to find ray intersections with scene geometry.
19. Second, we discuss the architecture of the pixel generator network that learns the feature-to-color mapping.

20. Intersection testing, intuitively, is the solving of an optimization problem, where the point along each camera ray is sought that minimizes the distance to the surface of the scene.
21. To model this problem, we parameterize the points along each ray, identified with the coordinates (u,v) of the respective pixel, with their distance d to the camera:
22. $r_{u,v}(d) = R^T (K^{-1} (u,v,d)^T - t), d > 0$
23. For each ray we aim to solve: arg min d such that $r_{u,v}(d) \in \omega$, $d>0$ where omega is the set of all points that lie on the surface of the scene
24. Sphere tracing belongs to the class of ray marching algorithms that solve this optimization problem by starting at a distance close to the camera and stepping along the ray until scene geometry is intersected.
25. Sphere tracing is defined by a special choice of this step length, where each step has a length equal to the signed distance to the closest surface point of the scene.
26. Since this distance is only 0 on the surface of the scene, the algorithm takes non-zero steps until it has arrived at the surface.
27. A major downside of sphere tracing is its weak convergence guarantee:
28. … sphere tracing is only guaranteed to converge for an infinite number of steps.
29. Extensions of sphere tracing propose heuristics to modify the step length to speed up convergence
30. We introduce a ray marching LSTM that maps the feature vector $\phi(x_i) = x_i$ at the current estimate of the ray intersection x_i to the length of the next ray marching step.
31. Given our current estimate d_i, compute world coordinates $x_i = r_{u,v}(d_i)$
32. Via the formula I said earlier… $r_{u,v}(d) = R^T (K^{-1} (u,v,d)^T - t), d > 0$
33. Compute $\phi(x_i)$ to obtain a feature vector v_i, which we expect to encode information about nearby scene surfaces.
34. Compute the step length delta via the RM-LSTM as $(\delta, h_{i+1}, c_{i+1}) = \mathrm{LSTM}(v_i, h_i, c_i)$ where h and c are the output and cell states
35. Increment d_i by delta
36. We iterate this process for a constant number of steps: 1 – calculate world coordinates, 2 – extract feature vector, 3 – predict step length using ray marching LSTM, 4 – update d
37. This is critical, because a dynamic termination criterion would have no guarantee for convergence…
38. … in the beginning of the training, where both phi and the ray marching LSTM are initialized at random.
39. The z-coordinates of running and final estimates of intersections in camera coordinates yield depth maps, which visualize every step of the ray marcher.
40. This makes the ray marcher interpretable, as failures in geometry estimation show as inconsistencies in the depth map.
41. Note that depth maps are differentiable with respect to all model parameters, but are not required for training phi.
42. We choose as a generator architecture a per-pixel MLP that maps a single feature vector v to a single RGB vector.
43. This is equivalent to a CNN with only 1x1 convolutions
44. Pro’s and con’s of formulating the generator without 2D convolutions:

45. Pro’s: the generator will always map the same (x,y,z) coordinates to the same color value.
46. This implies the rendering is trivially multi-view consistent,
47. …assuming that the ray-marching algorithm finds the correct intersection.
48. 2D convolutions come with no guarantee of multi-view consistency…
49. Since when transforming the camera in 3D, the 2D neighborhood of a feature may change.
50. With our per-pixel formulation, the rendering function theta operates independently on all pixels, allowing images to be generated with arbitrary resolutions and poses.
51. Also, the per-pixel formulation requires the ray-marching, the SRNs, and the pixel generator to operate on the same (potentially high) resolution, requiring a significant memory budget.
52. We reason about the set of function {phi_j}j = 1 to M that represent instances of objects belonging to the same class.
53. Represent phi_j, parameterized as an MLP, with its vector of parameters lowercase phi_j in R^l.
54. Assume scenes of the same class have common shape and appearance properties
55. …that can be fully characterized by a set of latent variables z in R^k where k < l.
56. Equivalently, assume that all parameters lowercase phi_j are in a k-dimensional subspace of R^l.
57. Define a mapping psi: R^k to R^l where z_j, a latent vector |--> lowercase phi_j of the corresponding phi_j.
58. Now, parameterize psi as an MLP with parameters lowercase psi.
59. This architecture was previously introduced as a Hypernetwork, a neural network that regresses the parameters of another neural network.
60. We share the parameters of the rendering function across scenes.
61. We note that assuming a low-dimensional embedding manifold has so far mainly been empirically demonstrated for classes of single objects.
62. Here, we also only demonstrate generalization over classes of single objects.
63. We follow an auto-decoder framework to find the latent code vectors z_j.
64. So, each object instance C_j is represented by its own latent code z_j.
65. The z_j are free variables and are optimized jointly with the parameters of the hypernetwork and the neural renderer.