Mesh Extraction for Unbounded Scenes Using Camera-Aware Octrees

Zeyu Ma, Alexander Raistrick, Lahav Lipson*, Jia Deng
Department of Computer Science, Princeton University
{zeyum, araistrick, llipson, jiadeng}@princeton.edu

Abstract

Mesh extraction from occupancy functions is a useful tool in creating synthetic datasets for computer vision. However, existing mesh extraction methods have artifacts or performance profiles that limit their use. We propose OcMesher, a mesh extractor that efficiently handles high-detail unbounded scenes with perfect view consistency, with easy export to downstream real-time engines. The main novelty is an algorithm to construct an octree based on a given occupancy function and multiple camera views. We performed extensive experiments, and demonstrate OcMesher's usefulness for synthetic training & benchmark datasets, generating real-time environments for embodied AI and mesh extraction from depthmaps or novel view synthesis methods.

1. Introduction

We tackle the problem of camera-aware mesh extraction: given an occupancy function and a set of camera poses, we produce a 3D mesh that is high-detail when viewed from any camera but has minimum overall cost. An occupancy function is any mapping $f : \mathbf{R}^3 \mapsto \{0, 1\}$ which indicates whether any point in 3D space is occupied (i.e. contained within a solid object) or is unobstructed free-space. Occupancy functions can represent any closed 3D shape with arbitrary precision, and can flexibly represent unbounded scenes, which have visible geometry at arbitrary distances. However, converting an occupancy function to an efficient but high-detail mesh remains an unsolved problem.

Mesh extraction from occupancy functions is a useful tool in creating synthetic datasets for computer vision. Synthetic data from traditional graphics is widely used in computer vision [2, 4, 8, 17, 21, 24, 26–28, 31, 35, 44, 52, 58, 59]. Most recent is Infinigen [42], which proposed to create assets through procedural generation, i.e through randomized mathematical rules. In particular, they use a procedural occupancy function (i.e. an occupancy function defined using randomized rules and noise functions) to generate natural terrain in unbounded scenes, which must be converted to a mesh for use in data generation.

Mesh extraction is also applicable to works in novel view synthesis that represent scene geometry as a density function [3, 38], signed distance function (SDF) [29, 53], or any other representation that can be thresholded to produce occupancy. Mesh-based novel view synthesis is also a growing research area [5, 43], and we expect work on mesh extraction to aid these efforts.

Extracting a mesh is important step in many computer graphics pipelines. Major game engines [10], embodied simulators [25, 36, 45], graphics packages [6] and more all focus on meshes as input. Sphere tracing [19] can directly render implicit functions, but are not as performant, and typically require a true SDF not just occupancy. Mesh based graphics is widely adopted, efficient, photorealistic, and can be easily deployed to mobile hardware [5].

However, existing solutions for mesh extraction are unsatisfactory, especially in unbounded scenes. The most widely used is Marching Cubes [34], which extracts a mesh from a uniform grid of evaluated occupancy. This poses two main issues. First is that it is uniform in resolution: if one part of the scene needs to be high detail, the entire scene must use a small grid size, which is slow to evaluate and produces an overly expensive mesh. Second, it requires the user to provide a bounding box for the scene to determine the extents of the grid. The combination of these two issues is especially troubling, since an unbounded scene with any detailed foreground object must create a very large, very detailed grid. Unbounded scenes are important in synthetic data [42] and are a growing interest in novel view synthesis [3].

A workaround is to perform marching cubes on a warped or non-uniform grid, but this also has issues. Infinigen [42] creates a spherical grid centered on the camera location, which constructs an efficient mesh for any one camera view. However, most applications require meshes that are valid from many views, and merging many spherical grids is impossible. One can re-evaluate new meshes as the camera moves (as performed in Infinigen), but this is pro-

*work done while a student at Princeton University

Figure 1. Infinigen's spherical marching cubes has low poly artifacts if the camera changes its location significantly. OcMesher produces meshes with no artifacts for a wide range of camera views.

hibitively expensive, diffcult to integrate with downstream applications, and introduces flickering and low-detail artifacts when cameras move beyond the valid range (Fig. 1). Similar issues hold in applying existing solutions to scenes from NeRF and other methods: either the mesh must be prohibitively detailed everywhere, or if detail is non-uniform it is difficult to ensure the mesh is valid for an arbitrary set of camera views.

In this paper, we propose OcMesher, a camera-aware mesh extraction algorithm for unbounded scenes that avoids the pitfalls of existing solutions. OcMesher can generate a single mesh that represents unbounded geometry efficiently and can be rendered without artifacts across any pre-defined range of camera views. Our solution achieves a *combination* of capabilities that was not possible with existing solutions:

- *Unbounded scenes:* Our solution can efficiently handle unbounded scenes that include arbitrarily distant geometry, without intractable memory requirements.
- *Real-time rendering:* Our solution generates a single mesh that can be directly used by a real-time rendering engine like Unreal Engine [10]. The mesh renders well across a range of camera views, as long as the camera stays within an area predefined by the user.
- *View consistency:* Because only a single mesh is needed, our solution has perfect view consistency, with zero flickering artifacts across the rendered video frames, regardless of the poly count of the mesh. Our perfect view consistency improves data quality.

Our solution constructs an *octree* [37] instead of a regular grid. An octree is irregular, multi-resolution grid. Given a user-defined set of camera views and an occupancy function, we construct an octree that is high resolution around surfaces close to any camera view, but low resolution for locations in empty space or far away from all camera views. Our algorithm seeks to construct an *efficient* octree, minimizing the number of total cells used while maintaining the visual quality of the renders. Once the octree is constructed, we perform dual contouring [23] to extract a mesh. Fig. 2 illustrates the idea in comparison with existing solutions.

Compared to Marching Cubes, OcMesher requires a set or range of intended camera poses & intrinsics as an extra input, and specifies target mesh resolution as a desired *angular resolution* (i.e. how big is the biggest projected face size in any camera?) rather than a grid cell size. This is well-suited to our downstream applications. In synthetic dataset generators, the camera trajectory is typically an arbitrary path generated by the user [42, 54], which can be directly provided to our system. In games or embodied simulators, the user can provide the intended playable area (i.e. where is the agent allowed to walk, and what height(s) will the camera be?) and produce a mesh that is valid for any reachable location. Angular resolution is also a more intuitive control of mesh detail.

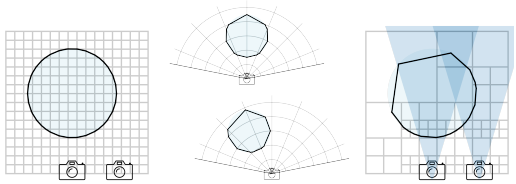


Figure 2. (a) Uniform marching cubes mesh extraction cannot efficiently represent high-detail unbounded scenes; (b) Infinigen's solution requires different meshes for new views, so introduces artifacts; (c) Our algorithm solves these issues by constructing an *efficient* octree and extracts the mesh from it to solve the issues.

The main novelty of our solution lies in our algorithm to handle *many* views *efficiently* with a given occupancy function, which existing approaches cannot do. Our heuristics are a nontrivial adaptation and combination of existing techniques. For example, the original marching cubes algorithm evaluates the occupancy and grows regions for fine-grained uniform grids, but we evaluate the occupancy in intermediate-level non-uniform grids, and we grow regions based on both occupancy and visibility.

We perform extensive experiments to validate the effectiveness of our solution. Experiments show that our solution improves the quality of synthetic data from Infinigen, and that our solution can generate unbounded environments that can be rendered at 50 FPS in Unreal Engine, seamlessly and without artifacts even with large view changes, a capability not available from existing solutions. Moreover, we show our solution can extract low-cost high-quality meshes

from BakedSDF[60] or a set of input depthmaps.

We encourage the reader to view the video here for more qualitative results and please visit https://github.com/princeton-vl/OcMesher for the code.

2. Related Work

2.1. Direct SDF Renderers.

A potential alternative to mesh extraction is to directly render an implicit function using ray marching algorithms such as sphere tracing [19], segment tracing [13], quasi-analytic error-bounded (QAEB) ray tracing [39], others [7, 16]. Raymarching methods also generally do not accept arbitrary occupancy functions, instead requiring an SDF [19], lipschitz continuity [13], or that the scene is composed of certain primitives [1]. Our method supports any occupancy function, including those used in Infinigen, and produces a mesh, which is necessary to integrate with existing synthetic dataset generators [17, 42] which are overwhelmingly mesh-based.

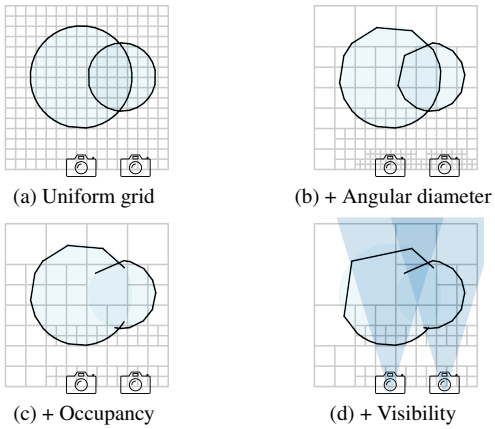
2.2. Multi-Resolution Mesh Extraction.

Uniform iso-surface extraction algorithms like Marching Cubes [34], Marching Tetrahedra [9] and others [12, 48, 49] are the standard approach to mesh extraction for occupancy functions. However, they are less well suited when scene content is unbounded or not all equally important. Therefore, many existing works divide space into multiresolution tetrahedra [14, 62], or even polyhedra [40, 56]. Some works also divide space into an octree and use dual contouring algorithms [22, 23, 41, 46]. Please refer to [57] for a more complete discussion. However, these works do not answer the question of how to determine the level of detail (LOD) when we have multiple cameras.

2.3. View Dependent Mesh Extraction.

To the best of our knowledge, all existing works focus on a single camera at a time and reuse the mesh for very few neighboring cameras. Many construct volume grids in world space and subdivide them recursively [18, 32, 33, 47]. These works usually do not optimize the mesh size based on criteria such as occlusion and do not scale well for high-resolution images. For example, Scholz et al. [47] focuses on real-time approaches and their generated mesh contains at most 700k triangles for a 1920×1080 resolution image of more than 2 million pixels. Considering the deep depth dimension and the fact that they do not coarsen occluded triangles, their meshes are far from achieving pixel-level details in unbounded scenes.

Infinigen [42] uses Marching Cubes in camera-space spherical coordinates, which is more scalable to high-resolution images. However, as mentioned previously, it is impractical for real-time rendering and suffers flickering and low poly artifacts due to switching between view-dependent meshes as the camera moves.



(a) Uniform grid (b) + Angular diameter (c) + Occupancy (d) + Visibility

Figure 3. The mesh representation (2D analogy) gets increasingly efficient as we apply the 3 LOD criteria one by one.

2.4. Flickering Removal.

Besides generating a static mesh, alternative flickering removal approaches attempt to hide or smooth out LOD transitions by continuously deforming the mesh [20, 30]. However, these techniques usually require the geometry to be a height map [30] or given as an initial high-poly mesh [20], which is not applicable in our case with an occupancy function as input. Other blending algorithms only consider changes in image space [15], ignoring 3D geometry. These solutions also cannot export a static mesh for use in real-time rendering engines.

3. Method

Given an occupancy function and a set of camera poses, we aim to extract an iso-surface mesh that is high-detail when viewed from any camera but has minimum overall polygon count and rendering cost. Our strategy is to construct an octree by recursively subdividing nodes until several level-of-detail criteria are met (Sec. 3.1), using a coarse-to-fine strategy to ensure computing the criterion is feasible (Sec. 3.2). We provide an overview in Fig. 4.

3.1. Octree Level of Detail Criteria

We use an octree to achieve different levels of detail for different parts of the scene. Octrees represent an irregular, multi-resolution grid using a tree structure, where each node represents a cube in the space, and nodes can be recursively subdivided into eight octants. To construct an octree, we recursively subdivide based on the 3 criteria described below and in Fig. 3, using a coarse to fine strategy as described in Sec. 3.2.

3.1.1 LOD based on Projected Angular Diameter.

The most important LOD criterion is the projected angular diameter of the octree node in the camera views, which is

inversely proportional to the distance to the cameras. We compute the maximum angular diameter A_{node} considering all cameras:

$$A_{\text{node}} = L_{\text{node}} / \min_{\forall \text{cam}} \text{dist}(\text{node}, \text{cam}) \quad (1)$$

Where L_{node} is the side length of the cube represented by the node, `cam` iterates through all the cameras, and `dist` computes the distance of the center of the node to each camera. We compare A_{node} to our target angular diameter $\hat{A}$, which is computed based on the field of view (FOV) of the cameras and the image resolution.

If $A_{\text{node}} > \hat{A}$, we either subdivide the node into 8 children and compute the angular diameter of each child recursively, or subdivide the node into a denser grid. In practice, there will always be some node containing one of the cameras, and A_{node} from Eq. 1 can never be less than $\hat{A}$. To avoid this issue, we clamp all distances to be at least D_{min} (a hyperparameter set by the user), and assert that the camera is never actually within distance D_{min} of the surface.

3.1.2 LOD based on Occupancy.

To save computation, we avoid subdividing any node determined to be wholly unoccupied, as these nodes should not contain zero crossings or affect the final mesh.

Exactly determining whether a node is occupied is expensive. For example, we could completely subdivide every node to meet the angular diameter criterion, and then merge empty nodes, but this is prohibitively expensive. Instead, we determine node occupancy using heuristics based on the occupancy function value of its 8 bounding vertices, without performing a full subdivision. This misses some nodes that are actually occupied if we further subdivide, and unless handled carefully, will cause dual contouring to create spiky meshes near any sharp LOD transitions. Therefore, we need to propagate the occupancy from the existing surface to its neighbors, as discussed later in Sec. 3.2.

3.1.3 LOD based on Visibility.

Finally, we avoid subdividing nodes that are either occluded by a surface or outside the camera frustum. In either case, we say that the octree node is not *visible* since it can only affect the image through shadows, reflections, or bounce lighting. For these not-directly-visible areas, we stop subdivision earlier, i.e. we use $\hat{A}_{\text{inv}} > \hat{A}$ as the diameter criterion. This reduces overall computation and final mesh size since the sum of angular sizes of nodes visible to the camera is bounded and should not grow with depth. However, similarly to occupancy checking, computing precise visibility would require completely subdividing all nodes, which is too expensive. We instead decide which node is visible in the middle of the subdivision.

Table 1 evaluates the effect of the 3 criteria for the scene in Fig. 5 in a low-resolution setting, and for the uniform grid baseline, we could only give the theoretical prediction. We can see all the criteria, especially the angular diameter criterion and the visibility criterion, can significantly reduce redundancy in the octree nodes and the resulting mesh.

Table 1. Ablations on LOD Criteria

	# Octree Leaf Nodes	# Mesh Vertices	# Mesh Faces
Ang. ✗ Occup. ✗ Vis. ✗	3.5×10^{13}	$> 1 \times 10^{9}$	$> 1 \times 10^{9}$
Ang. ✓ Occup. ✗ Vis. ✗	8,686,602	143,726	287,560
Ang. ✓ Occup. ✓ Vis. ✗	1,035,232	121,802	243,616
Ang. ✓ Occup. ✓ Vis. ✓	736	1,535	3,079

3.2. Coarse-to-Fine Octree Construction

We use a coarse-to-fine strategy to compute occupancy and visibility criteria efficiently. As illustrated in Fig. 4, we construct the octree in 3 steps: the coarse full octree, the occupancy and visibility test, and the fine visible octree. We explain them step by step.

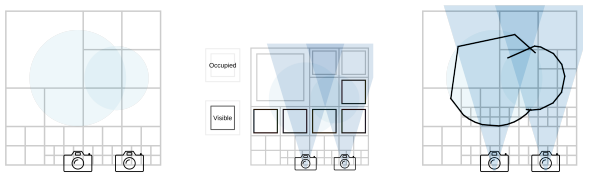
3.2.1 Coarse Full Octree

First, we construct a coarse full octree based solely on the projected angular diameter criterion. We initialize the octree with a single root node with size L_{root}. We use $L_{\text{root}} = 1000\text{m}$ for our Infinigen experiments, but this value is unbounded - one could equally use 5000km (horizon at sea-level) or 50,000km (approx maximum horizon distance on earth) with minimal overhead. We initialize a max-priority queue with the root node, which is sorted by key A_{node}, then repeatedly subdivide the top node while $A_{\text{node}} > \hat{A}_{\text{inv}}$.

We limit the size of the coarse octree to $S_{\text{max}} = 500,000$. Once this value is reached, we no longer subdivide and instead mark unprocessed nodes as having a dense grid of N^3 virtual children. N is the smallest power of two such that each virtual child has $A_{\text{node}} \leq \hat{A}_{\text{inv}}$. Crucially, we avoid storing virtual child nodes in memory, since we can cheaply compute them on the fly given just A_{node}.

3.2.2 Occupancy and Visibility Test

We use a flood-fill algorithm to avoid querying the occupancy of nodes that are unlikely to be occupied. We start with a subset $\mathcal{N}$ of nodes in the coarse octree (specifically, only the nodes on the boundaries of the N^3 virtual grids). For each node in $\mathcal{N}$, we compute the occupancy function value on the 8 vertices of the node. If both positive (≥ 0)



(a) Coarse full octree (b) Occup. and vis. test (c) Fine visible octree

Figure 4. We construct the octree from coarse to fine.

and negative (< 0) vertices exist, we mark this node as occupied and put its neighboring nodes into $\mathcal{N}$. We iterate until we completely test $\mathcal{N}$.

We may miss thin iso-surfaces due to insufficient sampling. But thin objects like plants are usually generated separately as standalone objects. In any case, this does not hurt the goal of data generation as we still have accurate and view-consistent ground truth, even though the mesh is not perfectly faithful to the procedural occupancy function.

Next, we compute whether each occupied node is also visible or not. For each camera, we project all nodes onto a depth buffer and mark nodes as visible in this camera if the projected depth is smaller than any of the values within a neighborhood of its projected pixel. A node is then visible overall if it is visible in at least one camera camera. To further avoid missing potentially visible nodes in the final result, we dilate the set of visible nodes by including all $< k$-th degree neighbors, where $k = 2$ is a user-specified hyperparameter.

3.2.3 Fine Visible Octree

Finally, we divide all visible nodes into high-resolution octrees until all of these octrees's leaf nodes (referred to as subnodes) have $A_{\text{subnode}} \leq \hat{A}$. This process can cause sharp LOD transitions if a neighboring node that was not considered occupied turns out to be occupied. Therefore, if we find a sign change while dividing a node on the border of a node marked unoccupied, we mark it as occupied such that it is meshed at high resolution too. This propagation without restriction can undo the work done by the visibility test. Therefore if a neighbor node is already marked as invisible, we don't convert it to be visible.

3.3. Mesh Extraction

We use the dual contouring algorithm [22, 23, 41] to extract a mesh from the octree. But, instead of quadratic error functions (QEF), we use bisection to locate the center vertex of each node, which is more robust to discontinuous occupancy functions.

3.4. Computational Complexity

The majority of the time and the memory are spent in the fine octree step, where we need to densely query the occupancy functions and extract dense meshes. Such functions are usually optimized for parallel computation. Therefore we do these operations in batches on parallel devices such as GPU to save time. The total complexity depends on many factors:

$$\text{Time and Memory} \propto K_{\text{scene}} K_{\text{cam}} N_{\text{cam}} \frac{S_{\text{fov}}}{\hat{A}^2} \qquad (2)$$

Where K_{scene} is the complexity of the scene, K_{cam} describes how fast the camera moves, i.e., how much new content each camera has (for completely non-overlapping cameras, $K_{\text{cam}} = 1$, but in practice, $K_{\text{cam}} \ll 1$), N_{cam} is the number of cameras, and S_{fov} is the solid angle of the FOV. The last factor $\frac{S_{\text{fov}}}{\hat{A}^2}$ comes from the fact that we only construct the fine octree for the visible part.

4. Experiments

To generate synthetic scenes and videos, we integrated our method with Infinigen [42]'s scene generator but replaced their mesh extracting solution with ours. Because Infinigen's assets except terrains do not use occupancy functions, we turn them off in the experiments. We randomly generated 42 scenes and for each scene, we rendered a video consisting of $N = 192$ frames of resolution $H \times W = 720 \times 1280$ on a cluster of GPUs including NVIDIA GeForce RTX 3090, RTX A6000, and A40. Each scene is generated under several settings:

- With Infinigen's spherical marching cubes, at 3 target resolutions. Higher mesh resolution reduces flickering but incurs more computational cost. We show the 3 target resolutions to demonstrate this trade-off. We regenerate the mesh every 8 frames, which is the default setting from Infinigen and is a generous comparison due to the low-poly artifacts at image edges during camera motion, as shown in Fig. 1.
- With our method using a moderate resolution. We create a single mesh from all N frames except for when the resulting mesh exceeds Blender's (our rendering Engine) capacity, which happens for 5 scenes out of the 42 scenes in total. In those cases, we split the entire video clip into two clips. This makes the video of these 5 scenes flicker in the middle frame, but it has negligible effects on the experiment results compared with the theoretical results.

We render RGB images with Blender's Cycles Engine, with 8192 samples per pixel, and save ground-truth depth maps and camera parameters for each frame. Because we only consider static terrain, we can compute the ground truth optical flow $\mathbf{F}_{i\to j}$ ($1 \leq i, j \leq N$) for any pair of frames # i and # j. Each $\mathbf{F}_{i\to j}[x, y]$ means pixel (x, y) in frame # i goes to pixel $(x, y) + \mathbf{F}_{i\to j}[x, y]$ in frame # j.

4.1. View Consistency

First, we show that our method has less flickering in the rendered videos, i.e., ours has better view consistency. Qualitatively, we show a zoomed-in region of a muddy mountain scene for a pair of mesh-switching frames (we call such frames transition frames) rendered with both methods in Fig. 5. Because the Infinigen solution switches between meshes, we can see the differences between the two images, focusing on those reflective regions.

In addition, we can quantitatively measure the view consistency score $S_{i\to j}$ between two frames i and j given the ground truth optical flow $\mathbf{F}_{i\to j}$:

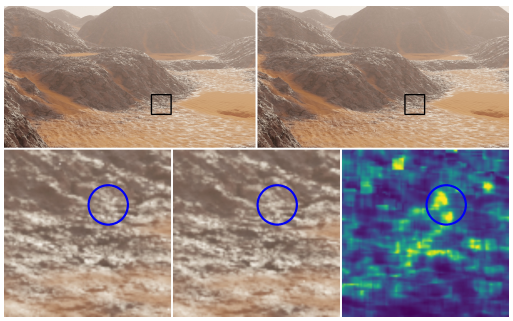
$$S_{i\to j} = \text{SSIM}(\mathbf{I}_i, \text{warp}(\mathbf{I}_j, \mathbf{F}_{i\to j})) \tag{3}$$

Where $\mathbf{I}_i$ and $\mathbf{I}_j$ are the RGB images of frame i and frame j respectively, warp is a function that warps back the input frame according to the forward flow, and SSIM computes the structural similarity score (SSIM) [55] of two images. This produces a 2D map the same size as the image, with values proportional to how consistent the two frames are. Fig. 5 shows the SSIM map for the zoomed-in regions.

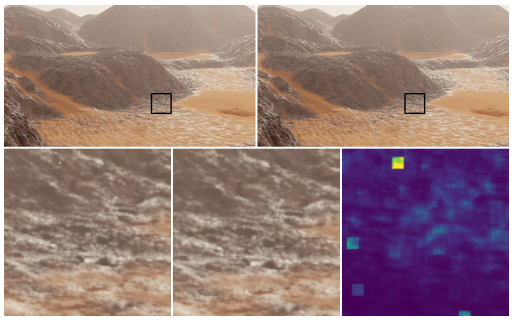
We analyze the complete video sequence for 3 scenes. In Fig. 6, column (a) shows an overview of the scene for reference; column (b)(c) shows the SSIM score map $S_{8\to 9}$ between the first pair of mesh-switching frames, i.e., frame #8 and frame #9. Column (b) shows 3 different resolution settings with the Infinigen solution; (c) shows ours. The brighter the color, the worse the SSIM score. From left to right in (b), as the resolution increases, the score gets better. Yet, ours has much better view consistency even compared with the best in Infinigen. The only significant inconsistency (yellow area) lies in occlusion boundaries and where volume scattering makes the rendering noisy.

Fig. 7 (a) shows the SSIM score for adjacent frames, which details how much flickering is experience when we watch the video continuously. It shows the frame-wise average against time on the left. The sharp comb-like peaks indicate periodical flickering in Infinigen during transition frames. On the right, we show scene-wise average endpoint error (EPE) for transition frames versus the mesh extraction time. We do not include rendering time because it is very dependent on the rendering engine. Infinigen's flickering is reduced by additional mesh resolution, but this incurs increased runtime. However, even in its highest resolution, it is still much worse than our method.

Fig. 7 (b) shows the SSIM score along many individual pixel trajectories, comparing frame 1 to frame i at every timestep. It also shows the frame-wise average against time and the scene-wise average against the generation time. We

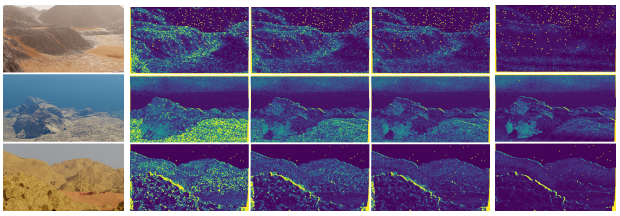


(a) Infinigen



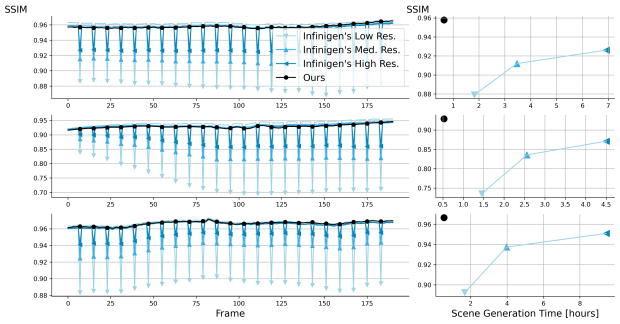
(b) Ours

Figure 5. Two adjacent frames of meshes extracted with (a) Infinigen (b) OcMesher (Ours). In the zoomed-in images, we can see more visual inconsistency in (a). The heatmap shows the quantitative measurement (via flow ground truth, explained in Sec. 4.1).

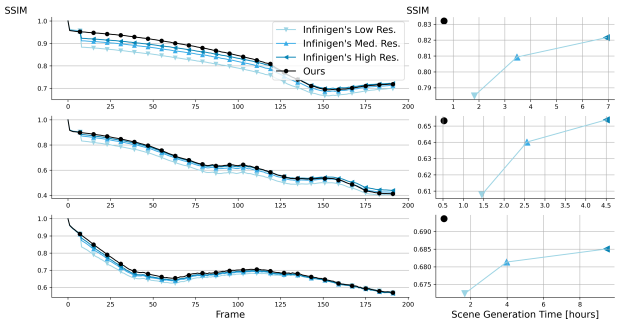


(a) Images (b) Infinigen with increasing resolution (c) Ours

Figure 6. Quantitative measurement of view consistency for the first pair of transition frames. The brighter, the worse.



(a) Comparison of frame-by-frame SSIM



(b) Comparison of first-to-nth frame SSIM

Figure 7. Quantitative measurement of view consistency (a) for adjacent frames and (b) along a pixel trajectory. On the left, Infinigen's transition frames introduce sharp spikes in frame-by-frame inconsistency. On the right, Infinigen suffers worse $0 \to i$ consistency except for the first 8 before any transition frames occur. In both cases, our method provides a superior SSIM-vs.-runtime tradeoff.

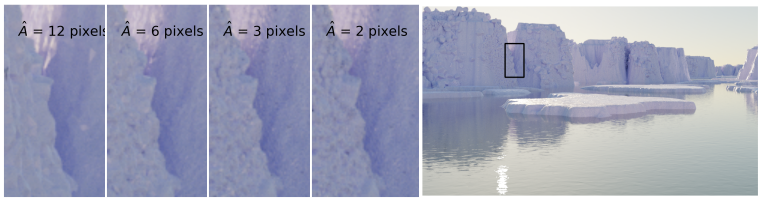


Figure 8. Rendering quality of different $\hat{A}$ values

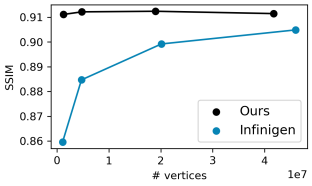
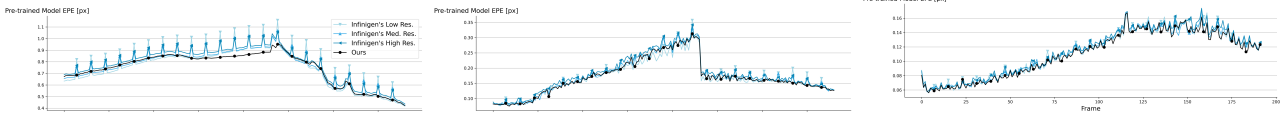


Figure 9. SSIM against number of vertices for different $\hat{A}$ values



(a) Gunnar Farneback's algorithm (b) RAFT (c) VideoFlow

Figure 10. End-point-error (EPE) for 3 pre-trained optical flow methods evaluated on OcMesher vs. Infinigen at various resolutions.

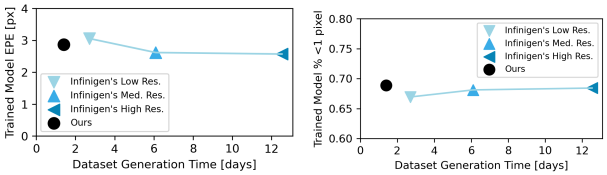
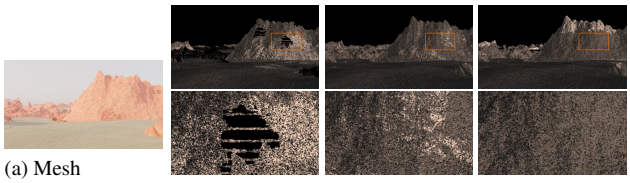


Figure 11. RAFT models trained on our dataset achieve better or comparable results with a lower cost.



(a) Mesh rendering (b) Sphere tracing w. $1\times, 0.1\times, 0.01\times$ compensation

Figure 12. For an occupancy function derived from a non-distance function f, we compensate for this by using $0.1 \times f$ or even $0.01 \times f$ as the SDF. Sphere tracing still produces noisy results.

can see as soon as the frame goes beyond the first group of frames where the mesh is reused, the score drops significantly in Infinigen, so it is much harder to match corresponding points there. The trade-off curve is similar.

4.2. Effect of Angular Diameter

We set the angular diameter threshold $\hat{A}$ such that the individual faces of the mesh are not visible in the given image resolution (720 $\times$ 1280 here). And empirically, we found that setting $\hat{A} = 3$ pixels is good enough. The resulting angular diameters A of the nodes are between 1.5 and 3 pixels. Fig. 8 shows qualitative results for a range of $\hat{A}$ values and we can see that rendering quality saturates as $\hat{A}$ decreases.

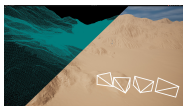
Fig. 9 shows that the number of vertices of the mesh increases proportional to $1/\hat{A}^2$ in our method. Compared with meshes of the same size in Infinigen, our view consistency is high all the time. So we can choose the $\hat{A}$ value that makes the rendering quality saturate.

4.3. Synthetic Data for Model Evaluation

We compare these datasets as benchmarks for the optical flow estimation task. With the same method and the same scene, if a dataset setting produces stable and reasonable errors as the camera moves smoothly, this means the dataset has fewer distracting factors and can serve as a better benchmark. We evaluate optical flow for each pair of adjacent frames using 3 existing methods: Gunnar Farneback's algorithm [11] from OpenCV, RAFT [51], and VideoFlow [50]. Fig. 10 shows the results of each method for one scene. On the left, we show frame-wise mean endpoint error (EPE). Again, we see spikes during transition frames in Infinigen, especially in the first two methods which estimate flow independently without neighbor information. This demonstrates that Infinigen has significant measurement noise when used as a benchmark dataset. In contrast, evaluating the models on our data produces consistent and smooth errors, except for the middle frame where the camera motion suddenly changes direction, which is an expected phenomenon.

4.4. Synthetic Data for Model Training

We also use the generated datasets as training sets for the optical flow task. Particularly, we focus on wide baseline scenarios, which is a more challenging task and more useful for relevant applications like stereo matching. We use RAFT [51], and train the model from scratch for 50k iterations with batch size 2 on image pairs from 32 out of the 42 scenes, with the rest as a validation set. We train on images sampled 10 frames apart and exclude images with median optical flows greater than 50px. We compare the training results versus the mesh generation cost of the dataset in Fig 11. We can see that models trained on our dataset attain better EPE and % $<$1 px metrics at a lower cost. Infinigen is prohibitively expensive to achieve similar performance.



(a) The scene and wireframe visualization

(b) Screenshots of a real-time video. FPS: 53.46

Figure 13. Our method enables real-time (>50 FPS) scene exploration in Unreal

4.5. Comparison with Ray-marching Alternatives

Ray-marching with compensated SDF does not work well in practice for rendering. For an occupancy function derived from a non-distance function f where positive values mean occupied, and negative unoccupied, we can compensate for this by using $0.1 \times f$ or even $0.01 \times f$ as the SDF. We used an open-sourced sphere tracing implementation with Phong shading. As shown in Fig. 12, compared with mesh-based rendering on the left, the result is noisy and has holes in the images rendered by sphere tracing. In addition, once the mesh is generated, it takes only 3 minutes to render the mesh; for sphere tracing, the smaller the multiplier is, the longer it takes to render the mesh, and it takes an hour to do sphere tracing in the $0.01 \times f$ setting.

4.6. Creating Embodied AI Environments

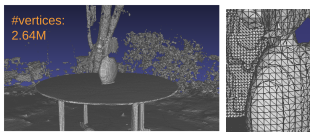
Most importantly, our method can create simulated virtual environments in real-time rendering engines like Unity or Unreal Engine, where an embodied AI agent can explore and learn. To demonstrate this, we took a scene from the previous dataset and extracted a mesh that is valid for a large region of camera rotations and translations along a path of interest. This results in an unbounded mesh that can be explored interactively within a certain range of camera poses. We export the resulting mesh to Unreal Engine. We experimented on a 4K (3840×2160) display with one NVIDIA GeForce RTX 3090 GPU and achieved >50 FPS rendering frame rate. Fig. 13(a) shows the scene and the wireframe visualization, and we can see the mesh is denser on those surfaces closer to the camera view and sparser far away. Fig. 13(b) shows the screenshots of the real-time video. In contrast, Infinigen needs to create a new mesh for each new view, which takes about 0.5 seconds even at low resolution, as shown in Fig. 14. Even ignoring mesh loading overhead, this yields a theoretical frame rate of 2.41 FPS. This could be increased by refreshing the mesh only every few frames, but this would produce visible seams and low-poly artifacts.

4.7. Mesh Extraction from Neural Functions

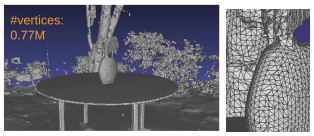
We used OcMesher to extract a mesh from a BakedSDF [60] trained on a standard demonstration scene, with results shown in Fig. 15. Compared to the mesh produced by SDFStudio's[61] uniform marching cubes, ours has similar visual detail, but less than 30% the total cost (measured in vertex-count).

Figure 14. Re-extracting new view-dependent meshes with Infinigen's solution cannot achieve interactive frame rates, even at extremely low detail ignoring any mesh load-time.

(a) Scene



(b) Uniform marching cubes



(c) OcMesher

Figure 15. OcMesher creates camera-aware meshes for neural occupancy functions

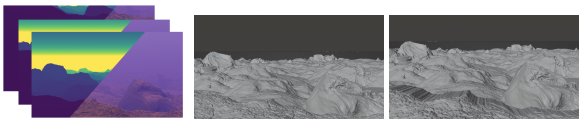


Figure 16. Application of OcMesher to depth maps

4.8. Mesh Extraction from Depth Maps

OcMesher can also extract meshes for arbitrary views from scenes represented as a set of depth maps, shown in Fig. 16. We define an occupancy function for the scene by projecting each 3D point into every depthmap and checking whether the projected depth is smaller than the interpolated depth in any frame, in which case we classify it as unoccupied. We use rendered depth-maps from Infinigen as an example, but the same technique applies to any set of depthmaps with known poses. Note that the occluded geometry is underconstrained and has artifacts.

5. Conclusion

We propose OcMesher, a method to extract meshes for unbounded scenes based on a given occupancy function and a given set of cameras. Unlike previous methods, we generate a high-resolution mesh that can be reused for all predefined cameras. We expect OcMesher to be a useful multipurpose tool for computer vision, and have demonstrated its usefulness for synthetic training & benchmark datasets, generating real-time environments for embodied AI, mesh extraction from novel view synthesis methods, and mesh extraction from depth maps.

6. Acknowledgements

This work was partially supported by the National Science Foundation and Amazon.

References

[1] Vectron volume for cinema 4d: https://www.machina-infinitum.com/vectron-volume. 3

[2] Shaojie Bai, Zhengyang Geng, Yash Savani, and J Zico Kolter. Deep equilibrium optical flow estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 620–630, 2022. 1

[3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022. 1

[4] Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. A naturalistic open source movie for optical flow evaluation. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part VI 12*, pages 611–625. Springer, 2012. 1

[5] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 1

[6] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 1

[7] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 15–22, 2009. 3

[8] Jeevan Devaranjan, Amlan Kar, and Sanja Fidler. Meta-sim2: Unsupervised learning of scene structure for synthetic data generation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVII 16*, pages 715–733. Springer, 2020. 1

[9] Akio Doi and Akio Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE Transactions on Information and Systems*, 74:214–224, 1991. 3

[10] Epic Games. Unreal engine. 1, 2

[11] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Image Analysis: 13th Scandinavian Conference, SCIA 2003 Halmstad, Sweden, June 29–July 2, 2003 Proceedings 13*, pages 363–370. Springer, 2003. 7

[12] Sarah F Frisken. Surfacenets for multi-label segmentations with preservation of sharp boundaries. *The Journal of computer graphics techniques*, 11(1):34, 2022. 3

[13] Eric Galin, Eric Guérin, Axel Paris, and Adrien Peytavie. Segment tracing using local lipschitz bounds. In *Computer Graphics Forum*, pages 545–554. Wiley Online Library, 2020. 3

[14] Thomas Gerstner and Renato Pajarola. *Topology preserving and controlled topology simplifying multiresolution isosurface extraction*. IEEE, 2000. 3

[15] Markus Giegl and Michael Wimmer. Unpopping: Solving the image-space blend problem for smooth discrete lod transitions. In *Computer Graphics Forum*, pages 46–49. Wiley Online Library, 2007. 3

[16] Enrico Gobbetti, Fabio Marton, and José Antonio Iglesias Guitián. A single-pass gpu ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer*, 24:797–806, 2008. 3

[17] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, et al. Kubric: A scalable dataset generator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3749–3761, 2022. 1, 3

[18] Benjamin Gregorski, Mark Duchaineau, Peter Lindstrom, Valerio Pascucci, and Kenneth I Joy. *Interactive view-dependent rendering of large isosurfaces*. IEEE, 2002. 3

[19] John C Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996. 1, 3

[20] Hugues Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 189–198, 1997. 3

[21] Chenfanfu Jiang, Siyuan Qi, Yixin Zhu, Siyuan Huang, Jenny Lin, Lap-Fai Yu, Demetri Terzopoulos, and Song-Chun Zhu. Configurable 3d scene synthesis and 2d image rendering with per-pixel ground truth using stochastic grammars. *International Journal of Computer Vision*, 126:920–941, 2018. 1

[22] Tao Ju and Tushar Udeshi. Intersection-free contouring on an octree grid. In *Proceedings of Pacific graphics*. Citeseer, 2006. 3, 5

[23] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 339–346, 2002. 2, 3, 5

[24] Samin Khan, Buu Phan, Rick Salay, and Krzysztof Czarnecki. Procsy: Procedural synthetic dataset generation towards influence factor studies of semantic segmentation networks. In *CVPR workshops*, page 4, 2019. 1

[25] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An interactive 3D environment for visual AI. *arXiv preprint arXiv:1712.05474*, 2017. 1

[26] Hei Law and Jia Deng. Label-free synthetic pretraining of object detectors. *arXiv preprint arXiv:2208.04268*, 2022. 1

[27] Jiankun Li, Peisen Wang, Pengfei Xiong, Tao Cai, Ziwei Yan, Lei Yang, Jiangyu Liu, Haoqiang Fan, and Shuaicheng Liu. Practical stereo matching via cascaded recurrent network with adaptive correlation. In *CVPR*, 2022.

[28] Wenbin Li, Sajad Saeedi, John McCormac, Ronald Clark, Dimos Tzoumanikas, Qing Ye, Yuzhong Huang, Rui Tang, and Stefan Leutenegger. Interiornet: Mega-scale multi-sensor photo-realistic indoor scenes dataset. *arXiv preprint arXiv:1809.00716*, 2018. 1

[29] Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin.

Neuralangelo: High-fidelity neural surface reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 1

[30] Peter Lindstrom, David Koller, William Ribarsky, Larry F Hodges, Nick Faust, and Gregory A Turner. Real-time, continuous level of detail rendering of height fields. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 109–118, 1996. 3

[31] Lahav Lipson, Zachary Teed, Ankit Goyal, and Jia Deng. Coupled iterative refinement for 6d multi-object pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6728–6737, 2022. 1

[32] Zhiyan Liu, Adam Finkelstein, and Kai Li. Progressive view-dependent isosurface propagation. In *Data Visualization 2001: Proceedings of the Joint Eurographics—IEEE TCVG Symposium on Visualization in Ascona, Switzerland, May 28–30, 2001*, pages 223–232. Springer, 2001. 3

[33] Yarden Livnat and Charles Hansen. View dependent isosurface extraction. In *Proceedings Visualization'98 (Cat. No. 98CB36276)*, pages 175–180. IEEE, 1998. 3

[34] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Seminal graphics: pioneering efforts that shaped the field*, pages 347–353. 1998. 1, 3

[35] Zeyu Ma, Zachary Teed, and Jia Deng. Multiview stereo with cascaded epipolar raft. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXI*, pages 734–750. Springer, 2022. 1

[36] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning. *CoRR*, abs/2108.10470, 2021. 1

[37] Donald Meagher. Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer, 1980. 2

[38] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022. 1

[39] F Kenton Musgrave. Qaeb rendering for procedural models. pages 509–528. Morgan Kaufmann, 2003. 3

[40] Valerio Pascucci and Kree Cole-McLaughlin. Efficient computation of the topology of level sets. In *IEEE Visualization, 2002. VIS 2002.*, pages 187–194. IEEE, 2002. 3

[41] Ronald N Perry and Sarah F Frisken. Kizamu: A system for sculpting digital characters. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 47–56, 2001. 3, 5

[42] Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, et al. Infinite photorealistic worlds using procedural generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12630–12641, 2023. 1, 2, 3, 5

[43] Christian Reiser, Stephan Garbin, Pratul P. Srinivasan, Dor Verbin, Richard Szeliski, Ben Mildenhall, Jonathan T. Barron, Peter Hedman, and Andreas Geiger. Binary opacity grids: Capturing fine geometric detail for mesh-based view synthesis, 2024. 1

[44] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *ECCV*, pages 102–118. Springer, 2016. 1

[45] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. 1

[46] Scott Schaefer and Joe Warren. Dual marching cubes: Primal contouring of dual grids. In *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.*, pages 70–76. IEEE, 2004. 3

[47] Manuel Scholz, Jan Bender, and Carsten Dachsbacher. Real-time isosurface extraction with view-dependent level of detail and applications. In *Computer Graphics Forum*, pages 103–115. Wiley Online Library, 2015. 3

[48] William Schroeder, Rob Maynard, and Berk Geveci. Flying edges: A high-performance scalable isocontouring algorithm. In *2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV)*, pages 33–40. IEEE, 2015. 3

[49] Will Schroeder, Spiros Tsalikis, Michael Halle, and Sarah Frisken. A high-performance surfacenets discrete isocontouring algorithm. *arXiv preprint arXiv:2401.14906*, 2024. 3

[50] Xiaoyu Shi, Zhaoyang Huang, Weikang Bian, Dasong Li, Manyuan Zhang, Ka Chun Cheung, Simon See, Hongwei Qin, Jifeng Dai, and Hongsheng Li. Videoflow: Exploiting temporal cues for multi-frame optical flow estimation. *arXiv preprint arXiv:2303.08340*, 2023. 7

[51] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 402–419. Springer, 2020. 7

[52] Apostolia Tsirikoglou, Joel Kronander, Magnus Wrenninge, and Jonas Unger. Procedural modeling and physically based rendering for synthetic data generation in automotive applications. *arXiv preprint arXiv:1710.06270*, 2017. 1

[53] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *CoRR*, abs/2106.10689, 2021. 1

[54] Wenshan Wang, Delong Zhu, Xiangwei Wang, Yaoyu Hu, Yuheng Qiu, Chen Wang, Yafei Hu, Ashish Kapoor, and Sebastian Scherer. Tartanair: A dataset to push the limits of visual slam. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4909–4916. IEEE, 2020. 2

[55] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. 6

[56] Gunther H Weber, Oliver Kreylos, Terry J Ligocki, John M Shalf, Hans Hagen, Bernd Hamann, and Kenneth I Joy. *Extraction of crack-free isosurfaces from adaptive mesh refinement data*. Springer, 2003. 3

[57] Rephael Wenger. *Isosurfaces: geometry, topology, and algorithms*. CRC Press, 2013. 3

[58] Magnus Wrenninge and Jonas Unger. Synscapes: A photorealistic synthetic dataset for street scene parsing. *arXiv preprint arXiv:1810.08705*, 2018. 1

[59] Yao Yao, Zixin Luo, Shiwei Li, Jingyang Zhang, Yufan Ren, Lei Zhou, Tian Fang, and Long Quan. Blendedmvs: A large-scale dataset for generalized multi-view stereo networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1790–1799, 2020. 1

[60] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P Srinivasan, Richard Szeliski, Jonathan T Barron, and Ben Mildenhall. Bakedsdf: Meshing neural sdfs for real-time view synthesis. *arXiv preprint arXiv:2302.14859*, 2023. 3, 8

[61] Zehao Yu, Anpei Chen, Bozidar Antic, Songyou Peng, Apratim Bhattacharyya, Michael Niemeyer, Siyu Tang, Torsten Sattler, and Andreas Geiger. Sdfstudio: A unified framework for surface reconstruction, 2022. 8

[62] Yong Zhou, Baoquan Chen, and Arie Kaufman. Multiresolution tetrahedral framework for visualizing regular volume data. In *Proceedings. Visualization'97 (Cat. No. 97CB36155)*, pages 135–142. IEEE, 1997. 3