MACARONS: Mapping And Coverage Anticipation with RGB Online Self-Supervision

Supplementary Material

Antoine Guédon Tom Monnier Pascal Monasse Vincent Lepetit
LIGM, Ecole des Ponts, Univ Gustave Eiffel, CNRS, France
https://imagine.enpc.fr/~guedona/MACARONS/

In this supplementary material, we provide the following elements:

1. Further details about the architecture of our model MACARONS and its different modules.
2. Further details about the training process, as well as the implementation of MACARONS.
3. Further quantitative and qualitative results.

We also provide on this webpage a video that illustrates how MACARONS explores and reconstructs large 3D structures.

1. Architecture

In the following subsections, we provide details about the architecture of MACARONS and its different modules.

1.1. Depth prediction module

The figure 1 illustrates the architecture of the depth prediction module of MACARONS, which takes inspiration from Watson *et al.* [9].

In our experiments, we follow [9] and use a set of $n_{depth} = 96$ ordered planes perpendicular to the optical axis at I_t. The depths are linearly spaced between extremal values d_{min} and d_{max}. We adapt d_{min} and d_{max} depending on the size of the bounding boxes of the scenes seen during training. We use images with size 456×256 pixels, which corresponds to a widescreen aspect ratio of 16:9.

This architecture is essential for MACARONS to learn how to compute a volume occupancy field and predict NBV in a self-supervised fashion. Indeed, we use a dense depth map prediction module rather than SfM or keypoints matching approaches because we need dense depth maps for space carving operations to generate a pseudo-GT volume occupancy and train the corresponding module. Moreover, the depth prediction module is also a fast and precise model that allows for reconstructing in real-time the surface points seen by the camera.

1.2. Volume occupancy module

The figure 2 presents the architecture of the volume occupancy module. We implement this module using a Transformer [8]: The module takes as input the point p, the surface point cloud S_t and previous poses c_i, and outputs a scalar value in $[0, 1]$.

This volumetric representation is a deep implicit function inspired by [4], and is convenient to build a NBV prediction model that scales to large environments. Indeed, as we explained in the main paper, it has a virtually infinite resolution and can handle arbitrarily large point clouds without failing to encode fine details since it uses mostly local features at different scales to compute the probability of a 3D point to be occupied.

In particular, for any 3D point $p \in \mathbb{R}^3$, we compute the k-nearest neighbors $(q_1, ..., q_k)$ of p in the dense point cloud S_t and transform the sequence $(q_1 - p, ..., q_k - p)$ using self-attention units followed by pooling operations. The resulting feature encodes information about the local state of the geometry. Then, we iterate the process at different scales: we down-sample the point cloud, compute the k-nearest-neighbors, encode the sequence, and reiterate. The spatial extension of the neighbors grows as we down-sample S_t, which helps to encode information about geometry at larger scales.

Because this architecture relies on local, neighborhood features, it can process arbitrarily large point clouds without failing to encode fine details or producing memory issues: Indeed, for a 3D point p, adding distant surface points to S_t does not change the local state of the geometry, nor the neighbors of p.

In practice, we use $k = 16$ and compute features at 3 different scales.

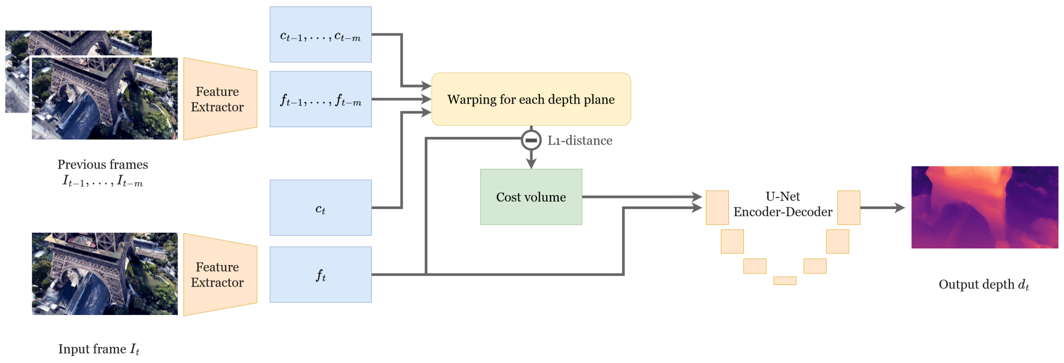


Figure 1. **Architecture of the depth prediction module.** The depth prediction module relies on a cost volume to predict depth from multiple RGB inputs: Features extracted from previous images $I_{t-1}, ..., I_{t-m}$ are warped into the view space of image I_t for multiple depth planes, and compared to the features extracted from I_t using L1-distance.

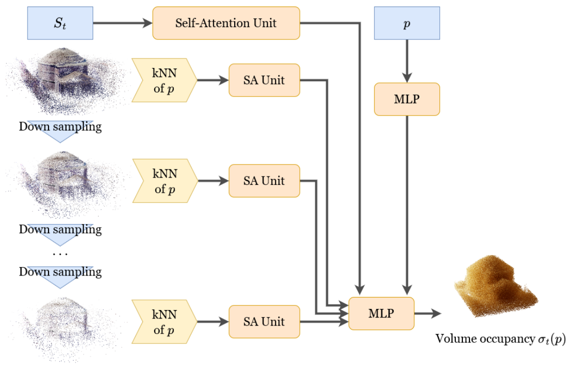


Figure 2. **Architecture of the volume occupancy module.** At time step t, the volume occupancy module relies mostly on neighborhood features to compute its output $\sigma_t(p)$. To compute the neighborhood features of an input 3D point p, we apply self-attention units on the k-nearest neighbors of p at different scales.

1.3. Surface coverage gain module

The figure 3 presents the architecture of the surface coverage gain module. This final module computes the surface coverage gain of a given camera pose c based on the predicted occupancy field, as proposed by [4]. However, as we explain in the main paper, we brought key modifications to the surface coverage gain estimation formula to adapt the model to NBV computation in large environments. In the following paragraphs, we detail some technical improvements that we did not mention in the main paper.

Occlusion-aware camera history features. In particular, we introduce a critical change to camera history features h_t. In [4], the authors compute camera history features by projecting all previous camera positions c_i^{pos} on a sphere around p. On the contrary, we compute $h_t(p)$ by projecting on a sphere around p only the positions c_i^{pos} of the cameras for which p was in the field of view delimited by c_i^{rot}, and for which p was not too far behind the surface reconstructed in the depth map d_i.

Therefore, we encode only the previous camera poses for which p, whether it is empty or occupied, is likely to be visible. This results in camera history features that reflect previous occlusion effects. This technical detail is actually of great importance to improve performance in large environments, since there is great variability in camera fields of view.

Indeed, to compute the visibility gain of a 3D point p, the surface coverage gain module exploits information about the volume occupancy, the camera history and the occlusions. For the specific case of a centered object with cameras sampled on a sphere around it, the volume χ is entirely contained in the field of view of all cameras. Therefore, by encoding occlusion effects with its transformer architecture, the module is able to identify, for any camera c_i in the camera history, if the point p was visible from c_i. Consequently, projecting the positions of all previous cameras c_i on a sphere around p does not decrease performance since the model is able to identify which camera p was visible from by encoding all occlusion effects in χ.

However, in a large environment, a subset of points in χ that occludes p in the direction of a previous camera c' could be located outside the field of view of another new camera c. Thus, the visibility module could lack information about previous occlusion effects when processing the field of view of the new camera c while still be provided with the information that p was observed in the direction of c' because of the camera history feature. This could trick the model into thinking p is empty even if it is not. Consequently, modifying the camera history feature to reflect previous occlusion effects leads to better performance in large environments.

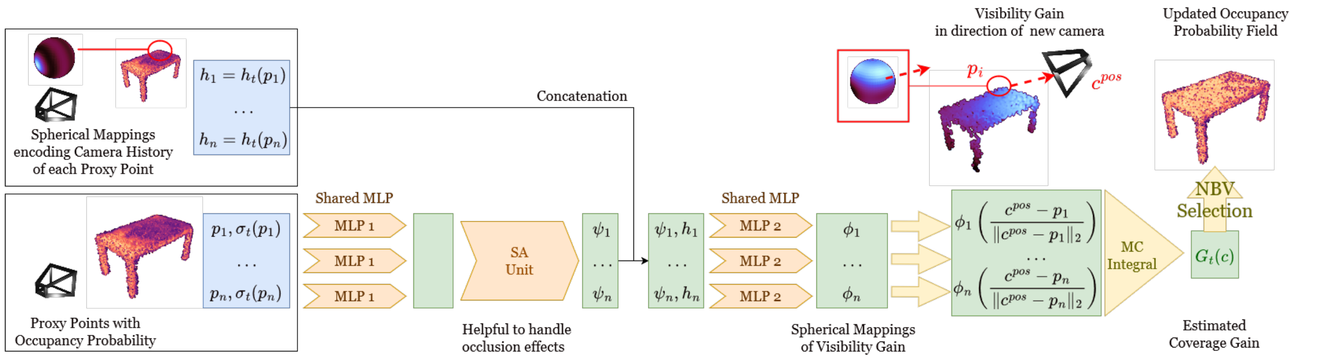

Spherical Mappings encoding Camera History of each Proxy Point
$h_1 = h_t(p_1)$
$\dots$
$h_n = h_t(p_n)$
Concatenation
Proxy Points with Occupancy Probability
$p_1, \sigma_t(p_1)$
$\dots$
$p_n, \sigma_t(p_n)$
Shared MLP
MLP 1
SA Unit
ψ_1
ψ_n
Helpful to handle occlusion effects
ψ_1, h_1
ψ_n, h_n
Shared MLP
MLP 2
ϕ_1
ϕ_n
Spherical Mappings of Visibility Gain
Visibility Gain in direction of new camera
p_i
c^{pos}
$\phi_1\left(\frac{c^{pos} - p_1}{\|c^{pos} - p_1\|_2}\right)$
$\phi_n\left(\frac{c^{pos} - p_n}{\|c^{pos} - p_n\|_2}\right)$
MC Integral
Updated Occupancy Probability Field
NBV Selection
$G_t(c)$
Estimated Coverage Gain

Figure 3. **Architecture of the surface coverage gain module, inspired by [4].** which predicts a visibility gain for a sequence of 3D points p_i in the field of view of camera c. To make this prediction, the model encodes the points p_i concatenated with their occupancy probability $\sigma_t(p_i)$. We use an attention mechanism to take into account occlusion effects in the volume between the 3D points and their consequences on the visibility gains. We finally use a Monte Carlo integration to compute the coverage gain $G_t(c)$ of camera c.

Additional details. At time step t, all surface coverage gain predictions are made in the view space of the current camera pose c_t. This increases performance since it slightly simplifies the problem (on the contrary, estimating surface coverage gains from any random coordinate space, for example, would make the problem more complex).

Finally, Even if its coverage gain is known to be equal to zero, we do compute a prediction for the surface coverage gain of the current camera c_t and use it to compute the loss during the online, self-supervised training. This information is actually useful to help the module set the lowest visibility gain value at zero.

2. Implementation details

2.1. Camera management

We follow [4] and first discretize the set of all camera poses $\mathcal{C}$ in the scene on a 5D grid, that correspond to coordinates $c^{pos} = (x_c, y_c, z_c)$ of the camera as well as the elevation and azimuth to encode rotation c^{rot}. The number of poses depends on the dimensions of the bounding box of the scene: As we explain in the main paper, this box is an input to the algorithm, as a way for the user to tell which part of the scene should be reconstructed. We follow [4] in our experiments, and discretize the 5D grid to obtain approximately 10,000 different poses in each scene.

At each time step t, we define the set of possible camera poses, denoted by $\mathcal{C}_t \subset \mathcal{C}$, as the immediate neighbors of the current camera pose c_t within the 5D grid. Specifically, these poses lie within a 6-neighborhood on the 3D position grid and a 4-neighborhood on the 2D rotation grid. We exclude any neighboring pose that shares the same position c_t^{pos} as c_t, as the depth module requires camera movement to generate depth maps using warping operations.

2.2. Initializing the neural modules

We have observed that during the initial training phase of MACARONS, the volume occupancy module and surface coverage gain module sometimes exhibit instability when trained from scratch with a naive initialization. This instability arises from the great variability and noise in the input data, which make training challenging for self-attention units and transformer architectures. Indeed, in contrast to SCONE [4], which has similar modules trained under ideal conditions with perfectly known 3D objects and 3D supervision, MACARONS reconstructs 3D from partial observations of unknown scenes, which results in a wide variation in batch size as well as noise in the 3D reconstructions.

However, we have found that this instability occurs only during the first few minutes of training, while both SCONE [4] and MACARONS require several dozen hours to converge. To stabilize the modules of MACARONS, we have developed a simple initialization process that involves a few iterations under ideal conditions: We first initialize the modules using standard techniques, such as Kaiming [5] for all layers except the self-attention units, for which we use Xavier initialization [2]. We then perform a few iterations with perfectly known objects, similar to [4], which takes less than 5 minutes (1 minute is sufficient).

We can use either a few ShapeNet [1] meshes, or simple virtual cube meshes generated online for this initialization process, eliminating the need for an additional 3D dataset.

2.3. Training the neural modules

To train our model in an online, self-supervised fashion for our experiments, we start by loading a 3D scene from a subset of the dataset introduced in [4]. We sample a random camera pose in the scene, and let our model explore. The camera captures images with size 456×256 pixels, which corresponds to a widescreen aspect ratio of 16:9. The model performs NBV training iterations as described in the main paper: In particular, it builds a Memory in real-time and simultaneously learns to reconstruct surfaces and optimize its path in the volume to increase its coverage of the surface. When starting a trajectory, we select a certain number N of 3D points within the bounding box of the scene, which we refer to as *proxy points*. During the online training process, we solely use these points to represent the volume: We calculate the volume occupancy exclusively for these proxy points and sample from them to predict the surface coverage gains. We save both the proxy points and their pseudo ground-truth volume occupancy values in memory. In our implementation, we typically sample 100,000 proxy points and evaluate surface coverage gains for 4 camera poses at each iteration.

After 100 NBV iterations, we load another scene and start a new trajectory. We perform data augmentation during training: we apply color jitter on RGB images, and perform rotations and mirroring operations on 3D inputs. Multi-GPU programming can be used to let the model explore several scenes at the same time and speed up convergence; In practice, we use 4 GPUs Nvidia Tesla V100 SXM2 32 Go to let the model explore 4 different scenes in parallel.

We perform up to 360 trajectories to make the model converge. However, such numbers are prone to variations: they depend not only on the complexity of the scenes but also on the size of the Memory and the number of Memory Replay iterations. Increasing the number of Memory Replay iterations slows down the exploration process during online training but considerably accelerates the convergence.

2.4. Memory Replay

The use of Memory Replay iterations allows for training the model with more complex camera configurations, for example by evaluating the surface coverage gains of distant camera poses stored into the memory. On the other hand, decreasing the number of memory replay iterations results in the model relying mostly on the current images for training, thus comparing surface coverage gains between nearby camera poses.

In our experiments, for each GPU, we store the data from the last 10 trajectories into the memory. We use 5 memory replay iterations for the depth module and only 1 for both volume occupancy and surface coverage gain modules, by using the 4 latest images captured from nearby camera poses. The self-supervision signal is built by comparing the current state of the scene to the state of the same scene before capturing these images. The use of a single memory replay iteration for the volume occupancy and surface coverage gain modules results in the best performance since we select the next best view from nearby camera poses in our naive path planning strategy.

However, more complex strategies for selecting the Next Best View (NBV), which include distant camera poses, could be devised. In such cases, increasing the number of memory replay iterations to create a self-supervision signal that aligns with the expected camera configuration should improve the performance.

2.5. Computational cost

Generally, computing a whole trajectory only takes a few minutes at inference, even on a single GPU Nvidia GeForce GTX 1080 Ti. During online, self-supervised training, computing a whole trajectory can take up to 10 minutes using GPU Nvidia Tesla V100 in our main experimental setup (which consists in 5 memory replay iterations for depth module and only 1 for volume occupancy and surface coverage gain modules), and up to 25 minutes for a different setup (5 memory replay iterations for depth module and 3 for the other modules).

However, we train MACARONS in synthetic scenes, which requires the GPU to perform numerous rendering operations to produce RGB inputs. As a result, the model's processing speed should be much faster in real-world scenarios. Specifically, with online learning activated, MACARONS can process 1.33 frames per second. After sufficient training, online learning can be disabled even in new, unfamiliar scenes, increasing the processing rate to 2.35 frames per second. These processing rates make MACARONS well-suited for real-time exploration, as it is not necessary to process every frame captured by the camera, but only a small subset of them.

3. Memory Building

3.1. Partial surface point cloud

As we explained in the main paper, to compute the reconstructed surface point cloud S_t at time step t we backproject the depth map d_t in 3D, filter the point cloud and concatenate it to the previous points obtained from $d_0, .., d_{t-1}$. We filter points associated to strong gradients in the depth map, which we observed are likely to yield wrong 3D points: We remove points based on their value for the edge-aware smoothness loss appearing in [3, 6, 9] that we also use for training. We hypothesize such outliers are linked to the module incapacity to output sudden changes in depth, thus resulting in over-smooth depth maps.

To avoid processing an excessively large point cloud, we choose to backproject only a randomly selected subset of the 456×256 pixels contained in the depth map. In the experiments we conducted, we sampled 5% of the pixels, resulting in the backprojection of 5836 pixels for each new depth map produced by the model.

3.2. Pseudo-GT volume occupancy

We rely on Space Carving [7] using the predicted depth maps to create a supervision signal to train the prediction of the volume occupancy field. As explained in the main paper, our key idea is as follows: When the whole surface of the scene is covered with depth maps, a 3D point $p \in \mathbb{R}^3$ is occupied iff for any depth map d containing p in its field of view, p is located behind the surface visible in d.

In practice, at time step t, we only have access to the depth maps $d'_{t,1}, ..., d'_{t,n}$ predicted for the images captured so far. We can still compute an intermediate occupancy field, which is an approximation but can be used as supervision signal. Since it is not reliable far away from the depth maps when the whole surface has not been covered, we only sample points around the newly reconstructed surface within a margin that increases with the total number of depth maps. In our experiments, this margin increases during the trajectory from 0 to approximately half the length of the bounding box, depending on the scene. We use the $\arctan$ function to compute the margin, rather than using a linear growth.

Finally, the depth maps generated by our model are not perfect, and some of them could contain errors. Therefore, eliminating all proxy points that are not located behind depth maps can be too aggressive and may produce inaccurate volume occupancy fields. To alleviate this harsh space carving approach, we introduce two simple ideas: a *score-based carving operation* and a *carving tolerance*.

Score-based carving. We assume that only a small portion of the depth maps produced by our model are inaccurate. Thus, we suggest to keep proxy points iff they are located behind a sufficient number of depth maps. Specifically, for a proxy point p, we denote by $n_d(p)$ the number of depth maps for which p is the field of view of the depth map, and we denote by $n_b(p)$ the number of depth maps for which p is not only in the field of view of the depth map, but also located behind the surface shown in the depth map. We finally define the score of p as $s(p) = \frac{n_b(p)}{n_d(p)}$. To perform space carving, we preserve proxy points with a score exceeding a certain threshold. We set this threshold to 0.95 in our experiments.

Carving tolerance. To further alleviate space carving, we allow 3D points to be located *in front of* a depth map but only up to a certain distance that depends on the size of the bounding box of scene. We refer to this range as the *carving tolerance*, and we set it to roughly 5% of the spatial extent of the bounding box in our experiments.

3.3. Pseudo-GT surface coverage gain

When computing pseudo-GT surface coverage gains, we count the number of new visible surface points in newly acquired depth maps. However, surface points have to be uniformly sampled on the whole surface to allow for accurate supervision coverage gain computation. Indeed, if surface points in S_t are not uniformly sampled on the surface, then the pseudo-GT coverage gain will be higher than expected in areas where the surface points are the most concentrated.

To address this issue, we apply a filtering process to the surface point cloud S_t during online training: we regularly recompute a filtered version S'_t of S_t by redistributing the surface points in small cells across the volume, which contain approximately the same number of surface points. In our experiments, we typically use 50 to 150 cells depending on the spatial extent of the scene, and set the maximum capacity of a cell to 1,000 points. This simple approach asymptotically promotes the uniform distribution of points on the surface. We recompute S'_t every 20 training iterations and fill the cells incrementally by adding 1,000 points at a time. Indeed, we encountered some issues when filling cells with a large number of points at the same time; Therefore, filling incrementally the cells gives better performance.

4. Experiments

In this section, we first provide extensive details concerning the experiment presented in subsection 5.2. of the main paper. Then, we present additional analysis about the benefits brought by our approach.

4.1. Exploration of large 3D scenes

We first provide further analysis concerning the main experiment presented in the paper, which compares our approach MACARONS to different NBV baselines for automated exploration and reconstruction of large 3D scenes.

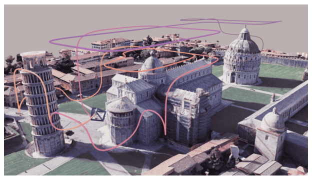
In particular, to complement the quantitative results presented in Table 1 of the main paper, we provide in Figure 6 details about the convergence speed of the surface coverage in large 3D scenes for MACARONS and the baselines from [4].

Next, we give additional qualitative results computed with MACARONS. Figure 4 pictures trajectories computed in real-time by MACARONS after 100 NBV iterations.

Figures 7 give examples of surface point clouds reconstructed with our model from RGB images during exploration.

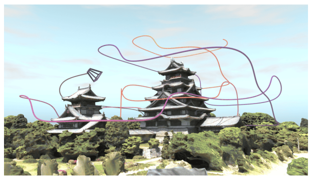
Figure 8 shows examples of volume occupancy fields that MACARONS predicted from the reconstructed surface point clouds.

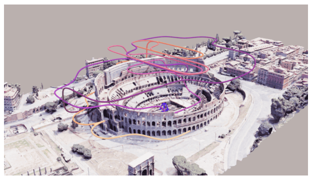
Finally, we provide on this website a video that illustrates how MACARONS explores and reconstructs efficiently a

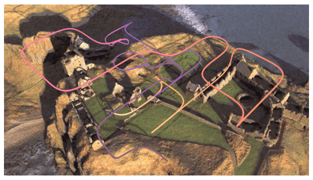

(a) Pisa Cathedral

(b) Statue of Liberty

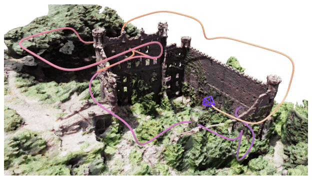
(c) Pantheon


(d) Fushimi Castle


(e) Colosseum

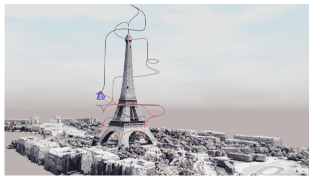

(f) Dunnottar Castle

(g) Christ the Redeemer


(h) Bannerman Castle

(i) Alhambra Palace

(j) Neuschwanstein Castle


(k) Eiffel Tower

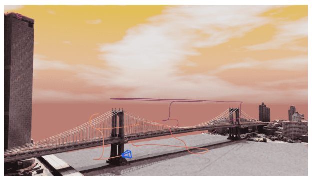

(l) Manhattan Bridge

Figure 4. **Trajectories computed in real-time by MACARONS in large 3D structures.** At each time step t, MACARONS predicts a NBV and builds trajectories that consistently cover most of the surface of the scene. MACARONS performed 100 NBV iterations in these images.

subset of three large 3D scenes. In particular, the video shows several key-elements of our approach:

1. The trajectory of the camera, evolving in real-time with each NBV iteration performed by the surface coverage gain module (left).
2. The RGB input captured by the camera (top right).
3. The surface point cloud reconstructed using the depth prediction module of MACARONS (right).
4. The volume occupancy field computed and updated in real-time using the volume occupancy module (bottom right). In the video, we removed the points with an occupancy lower or equal to 0.5 for clarity.

4.2. Ablation study: Loss function

We illustrate how the novel loss we designed for surface coverage gain estimation is grounded in the theoretical framework introduced in [4].

Improving surface coverage gain estimation. As we discussed in the main paper, the formalism introduced in [4] aims to estimate the surface coverage gain by integrating over the volume occupancy. However, this estimation can only be performed to a scale factor that cannot be computed in closed form.

Consequently, the predicted surface coverage gains are supposed to be proportional to the real values. We build

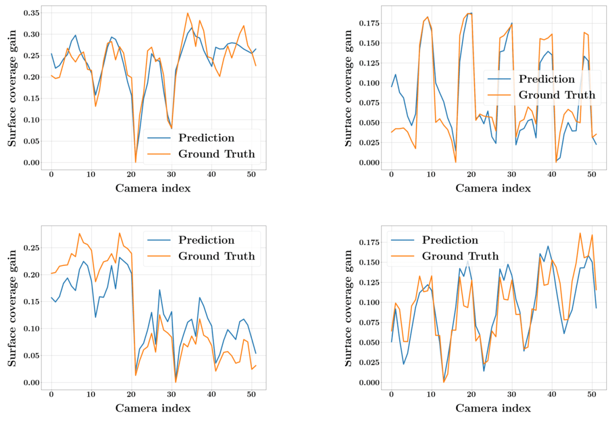
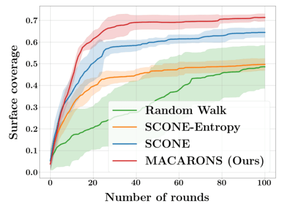
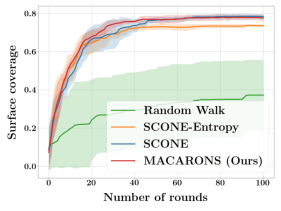


Figure 5. **Comparison between true surface coverage gains and predicted surface coverage gains on ShapeNet models, using our novel loss.** As expected, the normalized true and predicted coverages have great similarity, which verifies the hypothesis about the proportionality of true surface coverage gains and the predicted volumetric integrals.

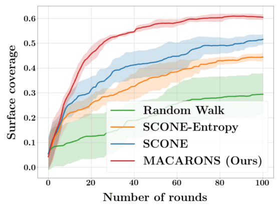
our novel loss on this single hypothesis: If the values are proportional, then dividing both predicted and pseudo-GT coverage gains by their respective means should result in similar values, that we directly compare with a L1-norm during training. As shown in Figure 5, we verify this theoretical proportionality on different samples of the ShapeNet dataset [1]. To this end, we use a version of MACARONS,



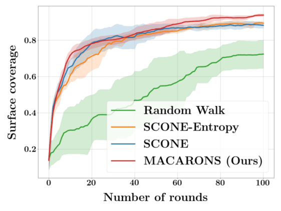
(a) Pisa Cathedral



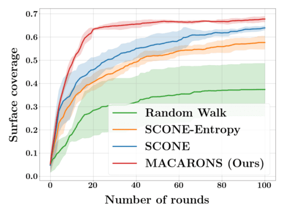
(b) Statue of Liberty



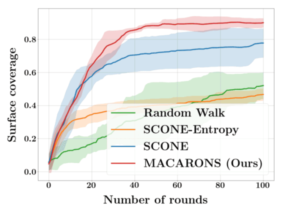
(c) Pantheon



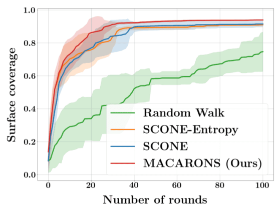
(d) Fushimi Castle



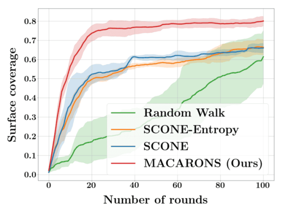
(e) Colosseum



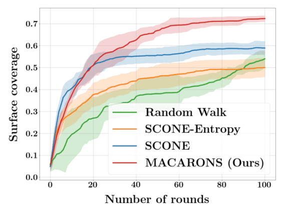
(f) Dunnottar Castle



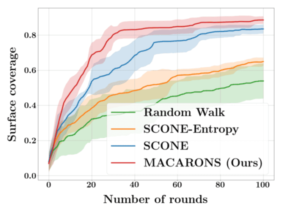
(g) Christ the Redeemer



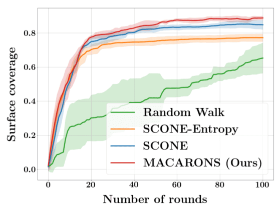
(h) Bannerman Castle



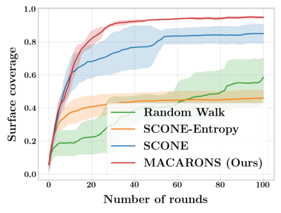
(i) Alhambra Palace



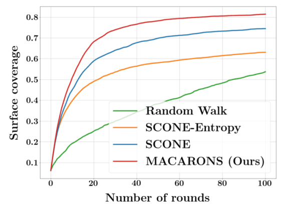
(j) Neuschwanstein Castle



(k) Eiffel Tower



(l) Manhattan Bridge



(m) Average on all scenes

Figure 6. **Convergence speed of the surface coverage in large 3D scenes by MACARONS and several baselines from [4].** All methods use perfect depth maps as input except for MACARONS, which takes RGB images as input. We follow [4] and plot the evolution of the total surface coverage during exploration, after 100 NBV iterations. We average surface coverage on several trajectories for each scene, starting from random camera poses. Standard deviations are shown on the figures. Our model MACARONS has been trained on a set of previous scenes; all weights are frozen and we only perform inference computation. Other methods are trained on ShapeNet [1] with 3D supervision. The first two rows depict scenes that were already seen by MACARONS during its online, self-supervised training. The third row depicts scenes the model has never seen before. Even if it only uses RGB images, our model MACARONS is able to outperform the baselines in large environments since, contrary to other methods, its self-supervised online training strategy allows it to scale its learning process to any kind of environment.

called MACARONS-NBV, trained on ShapeNet only with perfect depth maps. We sample random meshes in the dataset as well as random initial camera poses; Then, we apply the volume occupancy and the surface coverage gain modules to predict the surface coverage gains of cameras sampled on a sphere around the object. We divide predicted coverage gains and ground truth coverage gains by their respective means and compare the two distributions. As expected, the two are highly similar for many objects in the dataset.

Additional details. In the main paper, we compare in Table 1 our self-supervised method MACARONS trained with our novel loss to the original pipeline SCONE [4] trained on ShapeNet with the loss from [4]. We did not evaluate our new online, self-supervised pipeline with the loss from [4] because this loss needs a dense set of cameras, which is not available during online exploration. Indeed, this loss gives

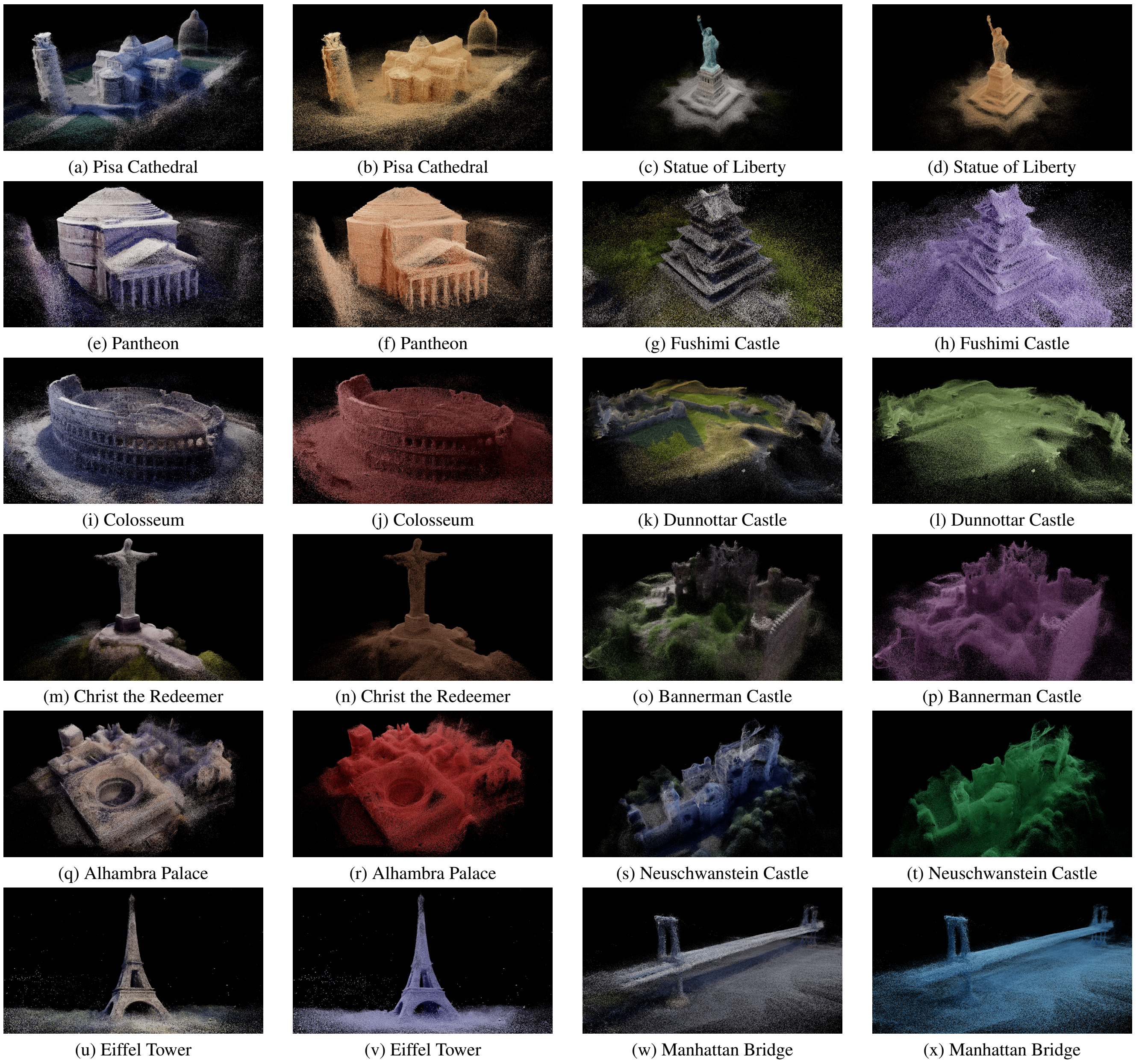
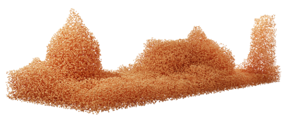
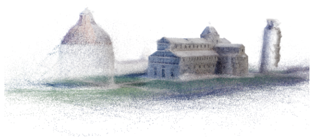
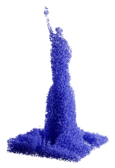


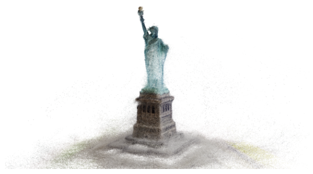
Figure 7. **Automated reconstruction of large 3D structures from RGB images with our approach MACARONS.** Our model reconstructs the surface in real time during exploration: We show here the reconstruction after 100 NBV iterations. The model has been trained on a set of previous scenes; all weights are frozen and we only perform inference computation. The first four rows depict scenes that were already seen by the model during its online, self-supervised training. The two last row depicts scenes the model has never seen before. For each scene, we provide images with colors computed from the input RGB images as well as images for which all points share the same color. The latter help to better estimate the quality of the reconstructed geometry. MACARONS is not only able to reconstruct surfaces thanks to its depth prediction module (even for scenes it has never seen before), but is also able to optimize its path around the structure and consistently cover most of the surface of the scene thanks to its NBV prediction.

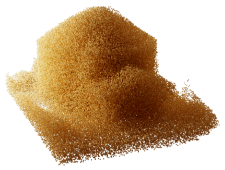
chaotic results when trained with a sparse set of cameras in our pipeline. However, we proposed in Table 3 of the main paper an ablation in large environments, that uses the same MACARONS-NBV, trained on ShapeNet only with both our loss and the loss from [4]. MACARONS-NBV performs slightly worse than SCONE [4] when trained on ShapeNet with the loss from [4], because authors from [4] use additional, hand-crafted operations to help their model

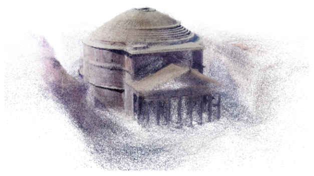

(a) Pisa Cathedral

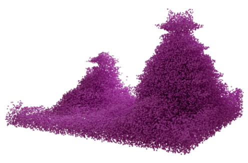

(b) Pisa Cathedral

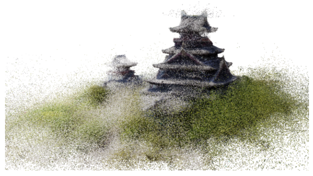

(c) Statue of Liberty

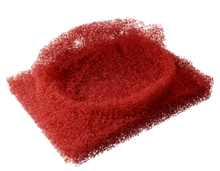

(d) Statue of Liberty

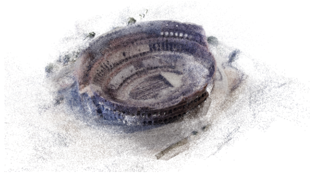

(e) Pantheon

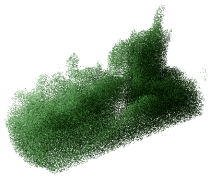

(f) Pantheon


(g) Fushimi Castle


(h) Fushimi Castle


(i) Colosseum


(j) Colosseum


(k) Neuschwanstein Castle

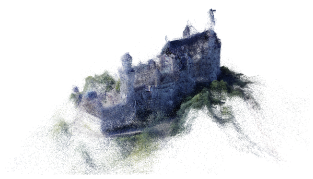

(l) Neuschwanstein Castle

Figure 8. **Volume occupancy fields computed with our approach MACARONS.** Our model updates the volume occupancy field in real time during exploration: For several scene, we show here the volume occupancy after 100 NBV iterations (left), as well as the final reconstructed surface for comparison (right).

process large and unknown scenes. On the contrary, we do not use such post-processing tricks but let our model learn to process large scenes by itself with our online self-supervised training, which explains the superiority of the full model MACARONS.

4.3. Ablation study: Pretraining, Memory Replay

We conducted an additional ablation study in Table 1 to assess the influence of memory replay iterations and pretraining with explicit 3D supervision on [1]. We trained MACARONS using different setups: we initialize the model using our previously described initialization process (*Initialized*), or with a complete pretraining on ShapeNet with explicit 3D supervision, following [4] (*Pretrained*). Additionally, we perform either one memory replay iteration (*1 MRI*) or three memory replay iterations (*3 MRI*) for both the volume occupancy and surface coverage gain modules. For the depth module, we computed five memory replay iterations in each setup.

As previously mentioned, the model does not gain from additional memory replay iterations when trained from scratch because it prioritizes learning to predict surface coverage gains for nearby camera poses due to our unsophisticated path planning strategy. However, if the model has already acquired knowledge about NBV prediction through a full pretraining program involving explicit 3D supervision on ShapeNet, it appears to benefit from more memory replay iterations as they allow for further specializing in NBV prediction in large, unknown and complex scenes.

Finally, as stated in the main paper, a basic pretraining approach on ShapeNet with explicit 3D supervision, without incorporating a self-supervision strategy in unfamiliar environments, fails in achieving performance comparable to the full model MACARONS. Indeed, even with additional hand-crafted tricks to adapt NBV-prediction to larger scenes, SCONE's performance is still inferior to MACARONS when trained from scratch with self-supervision only.

3D scene	SCONE [4]	MACARONS Initialized 1 MRI	MACARONS Initialized 3 MRI	MACARONS Pretrained 1 MRI	MACARONS Pretrained 3 MRI
Dunnottar Castle	0.650 ± 0.093	0.774 ± 0.049	0.793 ± 0.055	0.784 ± 0.025	**0.805** ± 0.026
Colosseum	0.532 ± 0.032	0.622 ± 0.005	0.618 ± 0.011	**0.629** ± 0.011	0.627 ± 0.011
Bannerman Castle	0.552 ± 0.020	0.764 ± 0.026	**0.778** ± 0.018	0.716 ± 0.029	0.752 ± 0.016
Pantheon	0.401 ± 0.030	0.488 ± 0.012	0.487 ± 0.010	0.499 ± 0.020	**0.509** ± 0.021
Christ the Redeemer	0.833 ± 0.037	0.848 ± 0.024	0.831 ± 0.040	0.865 ± 0.037	**0.868** ± 0.028
Statue of Liberty	0.695 ± 0.020	0.696 ± 0.010	0.670 ± 0.021	**0.708** ± 0.014	0.700 ± 0.013
Pisa Cathedral	0.555 ± 0.033	0.633 ± 0.023	0.629 ± 0.005	**0.661** ± 0.008	0.645 ± 0.012
Fushimi Castle	0.806 ± 0.029	0.807 ± 0.014	0.827 ± 0.016	0.842 ± 0.019	**0.860** ± 0.011
Alhambra Palace	0.528 ± 0.030	0.632 ± 0.017	0.636 ± 0.012	0.634 ± 0.019	**0.647** ± 0.011
Neuschwanstein Castle	0.662 ± 0.041	0.738 ± 0.043	0.731 ± 0.013	0.760 ± 0.020	**0.773** ± 0.014
Eiffel Tower	0.753 ± 0.013	0.766 ± 0.033	0.754 ± 0.025	**0.789** ± 0.010	0.784 ± 0.027
Manhattan Bridge	0.745 ± 0.069	0.820 ± 0.021	0.830 ± 0.023	0.825 ± 0.034	**0.847** ± 0.011
Average on all scenes	0.643	0.716	0.715	0.726	**0.735**

Table 1. **AUCs of surface coverage on large 3D scenes for SCONE [4] and different training configurations of MACARONS.** SCONE [4] uses perfect depth maps as input, and MACARONS takes RGB images as input. We follow [4] and compute the area under the curve representing the evolution of the total surface coverage during exploration. The 8 scenes above the bar were seen by MACARONS during self-supervised training (but with different, random starting camera poses and trajectories), and the 4 scenes below the bar were not. SCONE [4] is trained on ShapeNet [1] with 3D supervision. We trained MACARONS using different setups: we initialize the model using our previously described initialization process (*Initialized*), or with a complete pretraining on ShapeNet with explicit 3D supervision, following [4] (*Pretrained*). Additionally, we perform either one memory replay iteration (*1 MRI*) or three memory replay iterations (*3 MRI*) for both the volume occupancy and surface coverage gain modules. For the depth module, we computed five memory replay iterations in each setup.

References

[1] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An Information-Rich 3D Model Repository. Technical report, Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 3, 6, 7, 9, 10

[2] Xavier Glorot and Yoshua Bengio. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *International Conference on Artificial Intelligence and Statistics*, 2010. 3

[3] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised Monocular Depth Estimation with Left-Right Consistency. In *Conference on Computer Vision and Pattern Recognition*, 2017. 4

[4] Antoine Guédon, Pascal Monasse, and Vincent Lepetit. SCONE: Surface Coverage Optimization in Unknown Environments by Volumetric Integration. In *Advances in Neural Information Processing Systems*, 2022. 1, 2, 3, 5, 6, 7, 8, 9, 10

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *International Conference on Computer Vision*, 2015. 3

[6] Philipp Heise, Sebastian Klose, Brian Jensen, and Alois Knoll. PM-Huber: Patchmatch with Huber Regularization for Stereo Matching. In *International Conference on Computer Vision*, 2013. 4

[7] Kiriakos N. Kutulakos and Steven M. Seitz. A Theory of Shape by Space Carving. *International Journal of Computer Vision*, 2000. 5

[8] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, 2017. 1

[9] Jamie Watson, Oisin Mac Aodha, Victor Prisacariu, Gabriel J. Brostow, and Michael Firman. The Temporal Opportunist: Self-Supervised Multi-Frame Monocular Depth. In *Conference on Computer Vision and Pattern Recognition*, 2021. 1, 4