3D Gaussian Splatting for Isolated Objects

MSc thesis in Computing Science for Utrecht University and the Royal Dutch Aerospace Centre

Lourens R. Verhage (6762115)	Author
Maxim van Oldenbeek	NLR (Daily) Supervisor
Dr. Deb Panja	UU Supervisor
Dr. Mihaela A. Mitici	UU Supervisor

July 10, 2024

Abstract

This thesis outlines a new scene object isolation method that is capable of isolating centre scene objects from a set of images that captures the scene. Doing this it shows some interesting quirks in the SMF data. The thesis experiments with the different configurations for the object isolation method, and presents these results. Beside this the thesis proposes some changes and extensions to Gaussian splatting to enable it to optimize the isolated objects, and re-implements depth-regularized Gaussian splatting. The modifications are explored by tweaking their hyper parameters, and in doing so showing their influence on the accuracy of the final trained object. The thesis is closed by speculating in possible improvements that could still be made to the object isolation method, and quickly proposes a method that could be used to allow for multiple objects, from the same scene, to be trained at the same time and be combined into one scene.

Cover figure: A Gaussian splatting optimized centre scene object overlapped over the original input image. Other objects found in the image are highlighted in different colors.

Layperson Summary

Lets say there are two or more pictures of the same scene, a segmentation model could be used on a single picture to give an outline of every visible object in that picture. This thesis describes a technique that can determine if two objects in different pictures are actually the same object. Secondly it outlines a few additions and modifications to a method known as Gaussian splatting. Gaussian splatting can be used to create a 3D representation of a scene captured by pictures from different positions and directions. By combining the technique of determining if objects in different images are actually the same object, and the modified Gaussian splatting, we are capable of creating a 3D representation of a single object in a scene captured by multiple pictures.

Unfortunately the new technique is not flawless, and is therefore currently only tuned to extract the object that is roughly at the centre of the scene. Therefore at the end this thesis also proposes some additional modifications that could be implemented to allow more objects to be extracted.

Preface

The thesis outlined in this document was made in collaboration between the Utrecht University and the Royal Netherlands Aerospace Center. Due to the collaboration between the Utrecht University and the Royal Netherlands Aerospace Centre the project has three supervisors, one supervisor at the Royal Netherlands Aerospace Centre and two supervisors at the Utrecht University.

The Ethics and Privacy Quick Scan of the Utrecht University Research Institute of Information and Computing Sciences was conducted. It classified this research as low-risk with no further ethics review or privacy assessment required.

Contents

Symbols and Notations

The thesis contains a lot of symbols and notations, especially in the methodology. Here is a short overview of the symbols and notations that are used.

Symbols

Symbol definition, unless stated otherwise.

Symbol	Definition
I	Image: 2D array
$\mathcal{I}$	Image set: $\{I\}$
H	Structure element: 2D array
F	Feature
$\mathcal{F}$	Feature set: $\{F\}$
p	A 2D pixel position: (x, y)
x	A 3D position: (x, y, z)
X	3D feature point: (F, x)
$\mathcal{X}$	3D feature point set/cloud: $\{X\}$
S	Segment mask: 2D array
$\mathcal{S}$	Segment mask set: $\{S\}$
M	Mask: $(S, \mathcal{X})$
$\mathcal{M}$	Mask set: $\{M\}$
O	Object: $(\mathcal{S}, \mathcal{X})$
$\mathcal{O}$	Object set: $\{O\}$

Set Notation

Here follows a short description about the set notation that is used in this thesis.

Conditions. When putting a condition on an element extracted from a set, the condition is written behind a vertical line ($|$). This vertical line $|$ can be read as where, multiple conditions are separated by a comma ($,$), for example the set with all even numbers is: $\{n \mid n \in \mathbb{N}, n/2 \in \mathbb{N}\}$.

Sizes. The size of a set is written as $\#\mathbb{N}$. For example the size of the set $\#(\mathcal{A} = \{1, 2, 3\}) = 3$.

For 2D array set types, like an image I, structure element H, or segment mask S, the size $\#S$ is the total size of the array, $width \times height$, regardless of the values of the elements inside the array.

Indexing. 2D array set types, like an image I, structure element H, or segment mask S, can be indexed using a 2D pixel position p. Doing this $I(p)$ will return the value of the element at position p in the 2D array.

Tuple Extraction. Some elements are a tuple of multiple elements, like the feature point X, mask M, and object O. To extract one of the elements from the tuple a dot (.) is used. For example $O.\mathcal{X}$ extracts the feature point set $\mathcal{X}$ from the object tuple O.

Abbreviations

Here is a table with abbreviations that are commonly used in this thesis.

D-GS	Depth-regularized Gaussian Splatting
GPU	Graphics Processing Unit
GS	Gaussians Splatting
IoU	Intersection over Union
IsO	Isolated Object splatting
LPIPS	Learned Perceptual Image Patch Similarity
NeRF	Neural Radiance Fields
PSNR	Peak Signal to Noise Ratio
SAM	Segment Anything Model
SFM	Structure-From-Motion
SH	Spherical Harmonics
SIFT	Scale Invariant Feature Transform
SQI	Squared Intersection
SSIM	Structural Similarity Index Measure

1 Introduction

The Royal Netherlands Aerospace Centre (NLR) is a research institute dedicated to innovation in aerospace. Research at the NLR is done for both civilian as military purposes. Aerospace Operations Training and Simulation (AOTS) focuses its research on modelling, simulation, and training. A few research projects within the NLR use techniques to reconstruct scenes based on images that capture the scene, like NeRF [1, 2] and Gaussian splatting [3], and is some case try to extract objects in these scenes.

The current way of extracting objects from scenes, for scene reconstruction, all require that the entire scene is trained in advance. This thesis describes a new method for isolating objects in a scene from images that capture the scene prior to reconstructing the scene. This thesis also proposes a few changes and additions to Gaussian splatting such that it is capable of training these isolated objects.

This section will give the problem description, some initial background that inspired this research, the research questions, and a short overview of what the new method will look like. The following section explains all the related work that is required to understand the new method. The third section gives an in depth explanation of how the new method works. This is followed by a section that gives a short explanation about how the new method was implemented and the tools that were used for the implementation. The fifth section presents the results, and the next section draws conclusions on the research questions based on the results. The final section gives some pointers to what future work could still attempted.

1.1 Problem Description

There are situations where intelligence gathering needs to be performed under time pressure, for instance a house search or a fast overflying object. To gather as much intelligence as possible images and video is often taken during these situations, these images and video can then later be analysed to gather even more data.

Images unfortunately do not always show the scale and position of objects clearly, and determining them manually by looking at multiple images or video is time consuming, and difficult. Therefore scene reconstruction (novel view synthesis) methods like NeRF [1, 2] and 3D Gaussian splatting [3] are useful to reconstruct the scene from the images and video. This allows someone to look through the entire scene again and look for objects that might have been missed.

If only a specific object is of interest, it would be useful to extract this single object such that it can be more thoroughly examined. Extracting a single object is also useful if this object can be analysed using other software, like extracting an aircraft from a scene such that it can be analysed using simulation software.

1.2 Initial Background

The current way to isolate objects is by first training the entire scene, and later extracting the relevant parts of the scene.

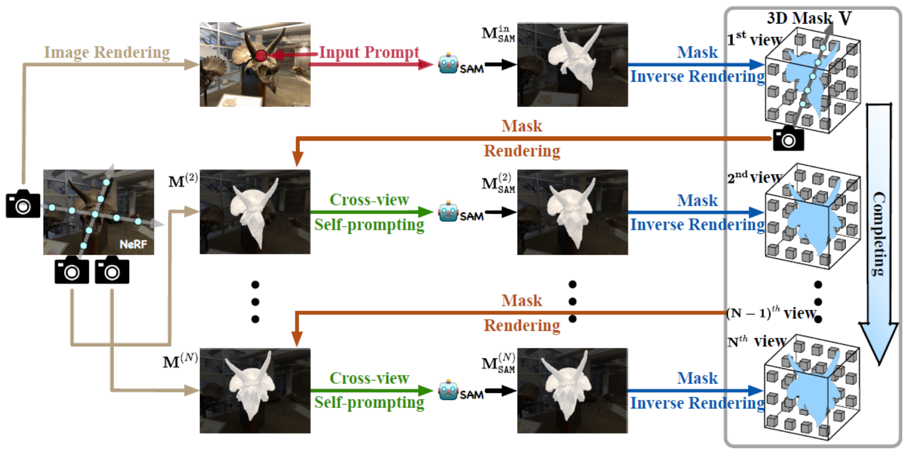

Figure 1: Figure from [4]. An initial image, and using a prompt a segment mask M_{SAM}^{in} is created. Using mask inverse rendering a part of the scene is isolated. This isolated scene is rendered and combined with a render from the full scene to create a new mask $M^{(2)}$, and using cross-view self-prompting a new segment mask $M_{SAM}^{(2)}$ is generated. This process is continuously repeated, till the entire object is isolated.

Segment Anything NeRF [4] does this by first rendering a single image and extracting a segment mask from this image. Using this mask, and inverse rendering, a part of the scene is isolated. Using a new camera view a new image is rendered from the entire scene and the isolated scene, and using cross-view self-prompting a new mask is generated that used to expand the isolated scene. This is repeatedly done till the entire object is isolated. See Fig. 1 for an overview.

The method for Gaussian splatting is Segment Any 3D Gaussians [5]. This method attaches a low-dimensional 3D feature to every Gaussian in the pre-trained scene. For every image in the image set segment masks are extracted and a neural network is used to extract features from the image. These features are pooled using the segment masks to create query masks. These query masks contain the features that belong to every segment mask. Now the features from the pre-trained scene are rasterized using the same camera configuration as the image. Using the query masks, these features are queried using the query masks, and Gaussians are selected based on the features. The selected Gaussians are rendered, and these renders are compared to the segment mask the query mask was created from. The loss in the rendering and the segment mask is used to update the 3D features in the Gaussians and the neural network that extracts features from the images. When the process is done learning the 3D features in the Gaussians and the neural network that extracts features from images are trained such that now the correct Gaussians can be filtered given an image of the scene and the segment mask of the object in that image. Figure 2 gives an overview of the method.

Both methods, for NeRF and Gaussian splatting, use fully trained scenes, and extract the object from the scene, or a combination of the scene and the

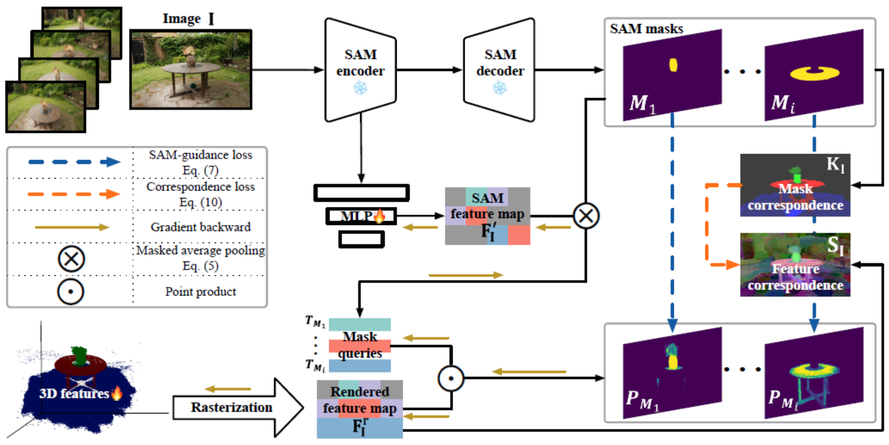

Figure 2: Figure from [5]. From image I segment masks $(M_1, ..., M_i)$ are extracted, and a neural network (MLP) is used to extract features F_I' from the image. The features F_I' are pooled by the segment masks $(M_1, ..., M_i)$ to create the mask queries $(T_{M_1}, ..., T_{M_i})$. The 3D features are rasterized into a rendered feature map F_I^r, these features are filtered using the query masks $(T_{M_1}, ..., T_{M_i})$ to select Gaussians and create rendered segment masks $(P_{M_1}, ..., P_{M_i})$. These rendered segment masks are compared with the original segment masks $(M_1, ..., M_i)$, and the loss is used to update the MLP and the 3D features.

input images $\mathcal{I}$. The Gaussian splatting method needs to undergo additional training to train the 3D features in the Gaussians and the feature extracting neural network. This thesis proposes a method that extracts the objects before the scene is trained, and is capable of training these individual isolated objects.

1.3 Research Questions

This all combines into the following research question:

Given a set of input images $\mathcal{I}$ that captures a single scene. Is it possible to extract all the visible objects in the scene captured in the image set $\mathcal{I}$, optimize these individual objects using Gaussian splatting, and combine the objects again to form a complete scene?

In this problem the following sub problems are identified:

1. Extracting objects from a single image $I \in \mathcal{I}$ can be done using a segmentation model, resulting in a segment mask set $\mathcal{S}$. To extract an object from the image set $\mathcal{I}$ the same object in different images need to be identified as the same object in the image set $\mathcal{I}$. *Given two segment masks $(S_a \in \mathcal{S}_i, S_b \in \mathcal{S}_j)|a \neq b$, is it possible to determine if they mask the same scene object?*
2. Gaussian splatting uses a feature point cloud $\mathcal{X}$ for its initial Gaussians. Only a subset of the points in the feature point cloud belong to any given

object in the scene. *Is it possible to filter the points belonging to a given object from the initial point cloud $\mathcal{X}$?*

3. Gaussian splatting compares the entire input image I to the rendered image I^* to determine the loss. When optimizing for individual objects, anything outside the object should be discarded. *Is it possible to modify the learning process of Gaussian splatting such that it is capable of only training a single object and removing all the parts that do not belong to the object?*

4. When multiple objects are trained and they are combined objects might intersect each other, due to parts of the scene being trained in multiple objects. Directly combining the objects into one scene might result in an incorrect representation due to the intersecting parts now containing more Gaussians, and therefore the Gaussians behind it are no longer rendered. *Given multiple trained objects, is it possible to combine the objects in such a way that the entire scene is correctly reconstructed?*

1.4 Overview

Here a short overview of the new object isolation method is given. First an image segment masks are generated. Next these segment masks are combined into one large segment mask. From this large segment mask parts that are not filled (missing regions) are extracted and used as additional segment masks. The point cloud gets projected to an image using the camera parameters of the original image I_i. The projected SFM points get filtered using both the original segment $\mathcal{S}_i$ masks, and the missing segment masks. This creates masks $\mathcal{M}_i$ for image I_i. This is done for every image in the image set $\mathcal{I}$. Next masks of different images are combined and created into objects $\mathcal{O}$. This is done by determining if the masks capture the same object, by checking if the masks contain the same SFM points. Finally an object $O_a \in \mathcal{O}$ is selected for optimization. See Fig. 3 for an overview.

2 Related Work

In this section the fundamental methods and techniques that are needed to understand the methodology of this thesis are explained. First structure-from-motion (SFM) is explained, which is a fundamental part of (depth-regularized) Gaussian splatting, and the new object isolation method. Next Gaussian splatting and its continuation depth-regularized Gaussian splatting are explained, these the models that optimize the Gaussians. After that a overview of segmentation and masks is given, and finally morphological filters are explained, these are all important for the new object isolation method.

2.1 Structure-From-Motion

Structure-From-Motion (SFM) [6, 7] is a method for acquiring 3D structures from an image set $\mathcal{I}$. The goal of SFM is to, given an image set $\mathcal{I}$ of a scene, extract the camera parameters for the cameras that were used to create the images, and reconstruct the geometry of the objects in the scene, represented

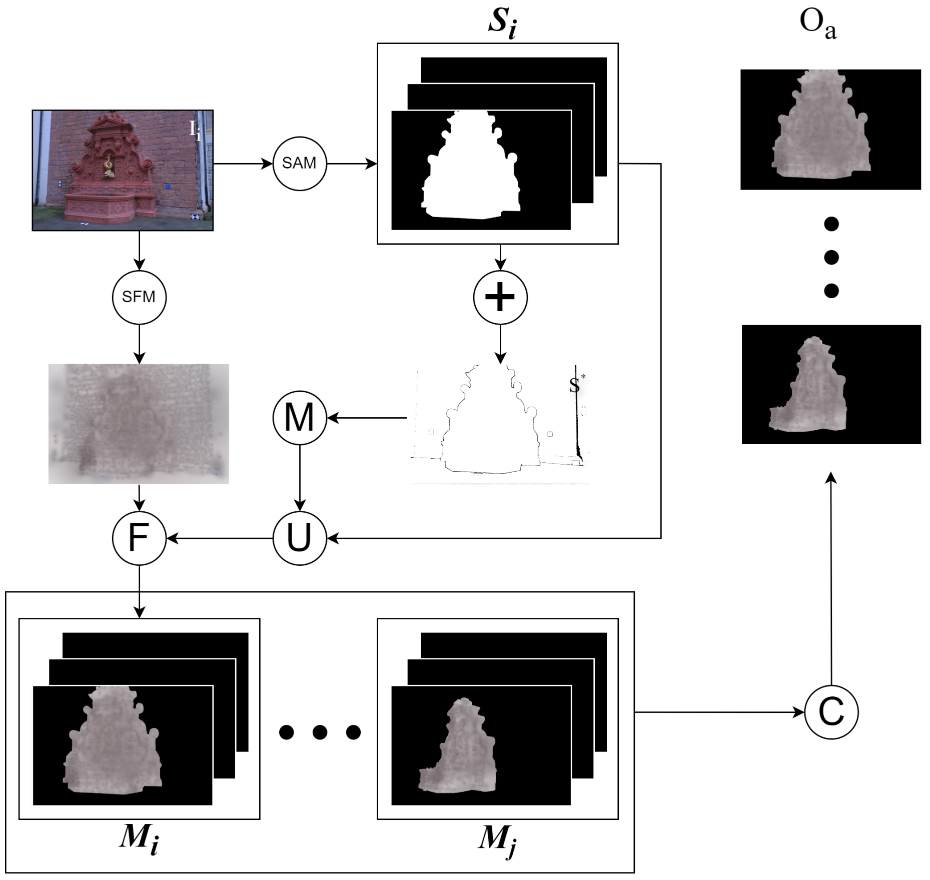

Figure 3: An overview of the object isolation method. The segment masks $\mathcal{S}_i$ of image I_i are generated and combined (+) to create the total segment mask S_i^*. The missing segments are extracted (M) from the total mask S_i^* and combined ($\cup$) with the original segment masks $\mathcal{S}_i$. The SFM points get projected to an image and get filtered using the segment masks, creating the masks M_i for the image I_i. Having done this for every image, the masks are then combined (C) into objects by checking matching SFM points, and an object O_a is chosen for optimization.

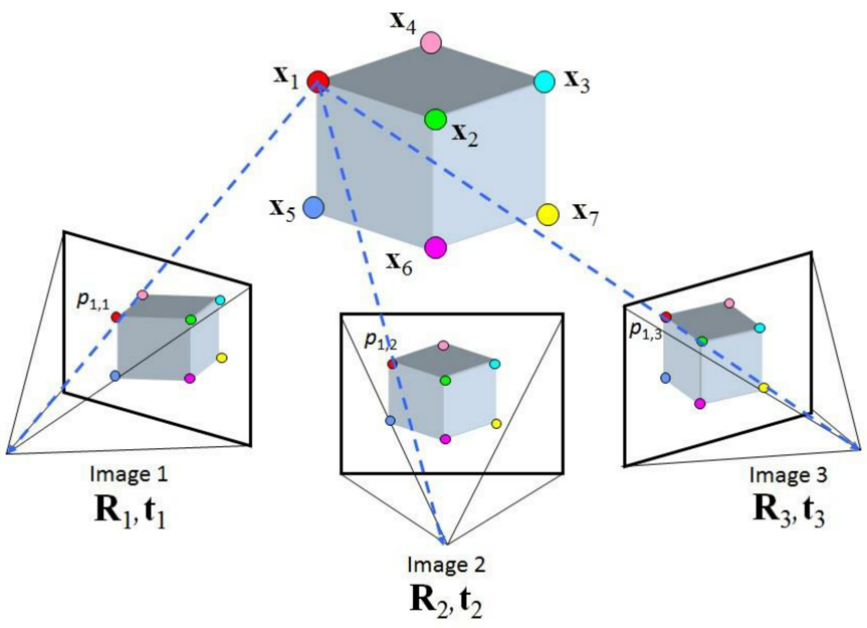

Figure 4: Figure from [8, 9, 10]. Feature p is observed in all images and is triangulated to point X_1. R_i, t_i are the rotation matrix and translation vector of image I_i.

as a point cloud $\mathcal{X}$. The camera parameters consist of the rotation matrix R, the translation vector t, the type of camera, and the parameters specific to the camera type such as focal length, principle point, and distortion parameters. The final point cloud $\mathcal{X}$ records for every point $X \in \mathcal{X}$ the position, color and reprojection error.

The most important parts of SFM are the feature extraction, feature matching, and triangulation.

For a visual guide for SFM, see Fig. 4.

Feature Extraction. SFM extracts a feature set $\mathcal{F}$ from the images I in the image set $\mathcal{I}$. Every feature F in the extracted feature set $\mathcal{F}$ consists of a 2D location x and a feature descriptor f. The features $\mathcal{F}$ outputted by a feature model need to be invariant to change in position, rotation, and scale, such that features can be accurately matched from different camera positions. Their are many different feature extraction methods both learned and fixed algorithms. Scale invariant feature transform (SIFT)[11, 12, 13] and variations on SIFT [14] are commonly used for fixed algorithms. Local invariant feature detectors (LIFD) [15] are common for learned feature extraction models.

Feature Matching. Now that for every image $I_i \in \mathcal{I}$ the features $\mathcal{F}_i$ have been extracted, SFM tries to match features from different images together. The result should be a set of possibly overlapping image pairs $\mathcal{C} = \{(I_a, I_b) \mid I_a, I_b \in \mathcal{I}, a < b\}$, and the feature matching that created this pair $\mathcal{M}_{ab} \in \mathcal{F}_a \times \mathcal{F}_b$.

Features are matched based on their similarity. How the similarity score between a feature pair $(F_a, F_b) \in (\mathcal{F}_a, \mathcal{F}_b)$ is determined depends on the method that is used to generate the features $\mathcal{F}$. For SIFT the feature descriptor is a 128-dimensional vector. The similarity between two features (F_a, F_b) is calculated as the distance between their vectors, where shorter distance means higher similarity [13]:

$$dist(F_a, F_b) = ||f_a - f_b||$$

Because the feature matching only looks at the similarity of the features, and not at their possible geometric positions, the image pair set $\mathcal{C}$ is later refined and verified.

Triangulation. To convert the matching feature pairs $\mathcal{M}_{ab}$ to a point cloud $\mathcal{X}$ the features need to be triangulated to a 3D position. The triangulation starts with two images $(I_a, I_b) \in \mathcal{C}$ and triangulates the matching features $\mathcal{M}_{ab}$ to a 3D position x. If a feature pair is triangulated, then both features $(F_a, F_b) \in \mathcal{M}_{ab}$ are registered as the same feature $F_a = F_b$. The 3D position x and the feature F that was used to triangulate the 3D position are then added as a point $X = (x, F)$ to the point cloud $\mathcal{X}$.

Now incrementally new images are added to the scene. The new image I_c must have a matching feature pairs with at least one image I_a that is already in the scene. The feature pair can be from an already existing point X, or it can extend the points in the point cloud $\mathcal{X}$ by triangulating a not yet triangulated feature F to a new point X. Triangulating a point X from more images increases the robustness of the point X.

Over the course of the triangulation, the camera pose, position, rotation, and scale, also get determined and refined. One of the ways to estimate the camera pose is using random sample consensus (RanSaC) [16]. RanSaC does this by taking a subset of triangulated feature points, and then estimating the camera pose of the newly added image that correctly projects these feature points back on the image. Next the remaining feature points get checked, and the feature points that do not correctly project back on the image get discarded. This way RanSaC also removes false feature matches.

The end result is a point set $\mathcal{X}$ of the features F that could be triangulated, and the camera poses from where these images where observed.

2.2 3D Gaussian Splatting

3D Gaussian splatting [**kerbl3DGaussians**] is a novel view synthesis method that uses 3D Gaussians to represent its scene. The goal of novel view synthesis is, give a image set $\mathcal{I}$, to create new views of the scene shown in the image set $\mathcal{I}$. A popular method for this is Neural Radiance Fields (NeRF), and its derivatives [1, 2, 17, 18, 19]. NeRF accomplishes this by using a neural network, the downside of this is the slow learning rate, and slow rendering. The goal of Gaussian splatting is to improve the training time and rendering. This is done by, instead of using a neural network, training (mean, scale, rotation, color, opacity) parameters of a collection of 3D Gaussians.

3D Gaussians. To capture the geometry of the objects as best as possible the Gaussians need to be anisotropic, not symmetric in every direction. To accomplish this the Gaussian is represented by a full 3D covariance matrix Σ [20]. The covariance matrix Σ only has a meaning if it is positive semi-definite. Because Gaussian splatting uses gradient descend it is not possible to optimize the covariance matrix Σ directly, it could result in an invalid covariance matrix. Therefore the covariance matrix is build from a scaling matrix S, and a rotation matrix R:

$$\Sigma = RSS^T R^T$$

Instead of optimizing the covariance matrix Σ the individual scaling S and rotation R matrices are optimized. The density of the Gaussian centered at the mean μ is then defined as:

$$G(x) = e^{-\frac{1}{2}(x)^T \Sigma^{-1}(x)} \tag{1}$$

To project the 3D Gaussians to 2D for rendering a new covariance matrix Σ' is needed. This new covariance matrix is created using the original covariance matrix Σ, the viewing transformation matrix W, and a Jacobian matrix J that approximates the projective transformation [20]:

$$\Sigma' = JW\Sigma W^T J^T$$

Removing the third row and column from Σ' a 2×2 variance matrix is obtained that can be used to directly get the density of the 2D projected Gaussian. Later during the blending process the densities are multiplied by a opacity value α.

To gain directional color c for the Gaussians spherical harmonics (SH) [21, 22] are used. They achieve directional values by plotting continuous functions on a sphere. All these functions are weighted and combined to create a single output, these weights are the values that are trained during the optimization. Every new SH band has 2 more functions than the previous band, with the first band only having a single, continous, function. Gaussian splatting uses up to 4 SH bands per color channel, red, green, and blue, resulting in 48 weights. Initially the Gaussians only use, and optimize, a single SH band, base color. Every 1000 iterations a new SH band gets introduced until all 4 SH bands are used.

Rasterization and Rendering. The fast rasterization is achieved by splitting the output render image I^* into 16×16 tiles. The size of the projected Gaussians is set to the 99% confidence interval. Then, for every tile a Gaussian overlaps, the Gaussian is instantiated with an id. This id is based on the Gaussians view space depth and the tile id it is instantiated on. The id is chosen in such a way that when sorted, using a GPU Radix sort [23], the Gaussians are sorted first based on tile, and then on depth. An example key function, given that the tile id t for every tile is an unique number, is:

$$K(z, t) = (16 \times 16) * N * t + z$$

Where z is the depth of the Gaussian, and N is the number of Gaussians, see Fig. 5. This sorted list is split into lists for every tile.

For every tile list a thread block is launched, with a thread for every pixel in the tile. In every thread the list is traversed front to back, and every Gaussian is checked if it overlaps with the threads pixel. If a Gaussian overlaps with the

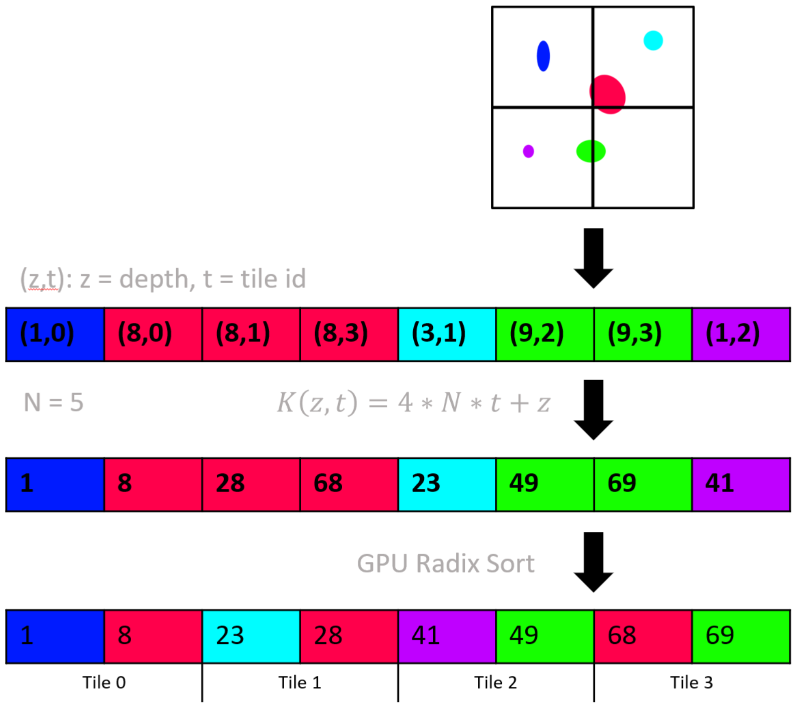

Figure 5: Gaussian instantiation and sorting of 5 Gaussians for 4 tiles.

pixel, its contribution to the pixel is calculated with the density (Eq. 1) and its opacity value α. This contribution is added to the opacity value α of the pixel. The final rendered color using the pixel opacity α is calculated as:

$$C = \sum_{i \in N} c_i \alpha_i T_i \tag{2}$$

Where

$$T_i = \prod_{i \in N} (1 - \alpha_i) \tag{3}$$

The thread is finished when the pixel opacity α is saturated $\alpha >= 1$. On the backward pass the same list is used again, only now from back to front, starting at the last Gaussian that contributed to the color of the pixel. Doing this, the Gaussians only have to be sorted once every iteration.

Initialization. Gaussian splatting uses the point cloud $\mathcal{X}$ created by SFM from the images $\mathcal{I}$ for its initialization. For every point $X \in \mathcal{X}$ a Gaussian is initialized. The initial covariance matrix Σ is estimated for an isotropic Gaussian with its size equal to the mean distance of the closest three points. The first band of the SH color is set to the color of the point X from the SFM point cloud $\mathcal{X}$, the other three bands are initialized with 0.

Culling and Densification. To control the number of Gaussians during the learning process, once every 100 iterations Gaussian culling and densification is performed, and every 3000 iterations the opacity of the Gaussians is reset.

For the culling unwanted or unneeded Gaussians are removed. Gaussians that are practically invisible, $\alpha < 0.005$, are removed. After the first opacity reset, the Gaussians that are large in world space $max(S) > 0.1 * scene\_extent$, or Gaussians that are large in view space $max(S2D) > 20$ are also removed during the culling. $S2D$ is the 2D projected size of the size S, the max extracts the largest dimension.

The purpose of the densification is to populate the areas where the geometry is not captured correctly with more Gaussians. There are two types of incorrectly captured geometry, under-reconstruction, where a small Gaussian cannot capture the entire geometry, and over-reconstruction, where a large Gaussian captures more than the intended geometry. It is observed that in both cases the Gaussian has a average high view-space position gradient $\tau_{pos} > 0.0002$. Small Gaussians, under-reconstruction, $max(S) <= 0.01 * scene\_extent$ are cloned. These Gaussians are copied and the new Gaussians are moved one step in the direction of the positional gradient. Gaussians in over-reconstruction, large Gaussians, $max(S) > 0.01 * scene\_extent$ cases are split. These Gaussians are replaced by two new Gaussians that are identical to the original Gaussian, except that they have their scale S devided by $\phi = 1.6$.

To remove Gaussians that are stuck close to the camera, once every 3000 iterations the opacity α is set close to zero $\alpha = 0.01$. During the optimization the Gaussians that are needed will have their opacity α raised again, while the culling process will remove all unnecessary Gaussians.

Optimization. The optimization is performed using a Stochastic Gradient Descent, and a sigmoid activation function is used for the opacity α, and the final calculated SH values, to clamp them between $[0, 1]$.

The loss function that is used for Gaussian splatting is a combination of two loss functions, the $\mathcal{L}_{color}$, which is the absolute loss in color difference, and the $\mathcal{L}_{DSSIM}$, which is the structural similarity loss. The structural similarity index measure (SSIM) calculated a score, when comparing two images, between $[-1, 1]$. Here -1 is perfect anti-correlation and 1 is perfect similarity. The structural dissimilarity (DSSIM) score simply inverts this score and bounds it between $[0, 1]$ ($DSSIM = \frac{1-SSIM}{2}$). These two loss functions are weighted with a hyperparameters λ_{SSIM} resulting in a final loss function then looks as follows:

$$\mathcal{L} = (1 - \lambda_{SSIM})\mathcal{L}_{color} + \lambda_{SSIM}\mathcal{L}_{DSSIM} \tag{4}$$

2.3 Depth-Regularized Optimization

One of the downside of Gaussian splatting is that it requires a large number of input images to correctly capture the geometry of the scene. Depth-regularized guassian splatting [24] tries to mitigate this problem by adding depth data, to the optimization process.

Depth Map. The depth map that is used for the depth regularization is composed of two separate depth maps. The first depth map D_{dense} is a depth map generated by a monocular depth estimation algorithm [25, 26]. Because this depth map D_{dense} depth is not scaled to the scene a second depth map D_{sparse} is used for scale. The second depth map D_{sparse} is generated by projecting the point cloud $\mathcal{X}$ generated by SFM back onto the image and storing the depth values. Using this second depth map D_{sparse} a scale s^* and offset t^* are calculated:

$$s^*, t^* = \arg\min_{s,t} \sum_{X \in D_{sparse}} \omega(X) * ||D_{sparse}(X) - (s * D_{dense}(X) + t)||^2$$

Where $\omega \in [0, 1]$ is used as a weight that represents the reliability of the feature points. The final depth map D^*_{dense} is then calculated by using the calculate scale s^* and offset t^*:

$$D^*_{dense} = s^* * D_{dense} + t^*$$

Optimization. The optimization adds two additional constraints to Gaussian splatting (Eq. 4). The depth loss $\mathcal{L}_{depth}$ is calculated as the absolute difference in the depth from the depth map D^*_{dense} and the rendered depth D. The depth is rendered in a similar way as the color (Eq. 2):

$$D = \sum_{i \in N} d_i \alpha_i T_i$$

Where d_i is the depth of Gaussian i.

Although the depth map D^*_{dense} is corrected using the SFM point cloud $\mathcal{X}$, there are often still conflicts. These are conflicts where the depth map D_{dense} estimates something to be in the foreground, while it should be in the background, or vice versa. To compensate for these conflicts another unsupervised constrained is added. This additional loss function $\mathcal{L}_{smooth}$ implies that neighbouring pixels that are part of the same plane should have similar depths. This is done by creating an edge map of the original image using a Canny edge detector [13, 27]. Pixels (p_i, p_j) that are next to each other, and neither are on an edge, are seen as part of the same plane. To ensure a larger loss for higher differences, and low loss for small differences an exponential loss is used:

$$\mathcal{L}_{smooth} = \sum_{p_j \in adj(p_i)} \mathbb{1}_{ne}(p_i, p_j) * ||p_i - p_j||^2$$

where $\mathbb{1}_{ne}$ is a function that indicates if both d_i and d_j are not in edge.

Both these new loss functions are weighted using a hyperparameters λ_{depth} and λ_{smooth}, resulting in a total loss function:

$$\mathcal{L} = (1 - \lambda_{SSIM})\mathcal{L}_{color} + \lambda_{SSIM}\mathcal{L}_{DSSIM} + \lambda_{depth}\mathcal{L}_{depth} + \lambda_{smooth}\mathcal{L}_{smooth} \quad (5)$$

2.4 Segmentation

Segmentation deals with the problem of creating a subdivision in an image corresponding to an object. Although similar and related to object detection there are differences. Object detection deals with identifying the type of object, while segmentation deals with extracting an object. Object detection models like Deformable Parts Models (DPM) [28], R-CNN [29], and You Only Look Once (YOLO) [30, 31] combine object detection and segmentation by giving a bounding box around the detected objects. Instead of a bounding box most image segmentation models return a mask [32] corresponding to the object.

Masks. While a simple axis aligned bounding box can be described using only 4 values, either two diagonal corner coordinates, or a corner with a width and a height, a segment mask takes a lot more data. A bounding box gives a general area the object is located, a segment mask is a lot more accurate with the location and shape of the object. A simple boolean mask will state for each pixel in the image if it belongs to the object the mask represents, while more advanced masks give for each pixel a certainty of the pixel belonging to the masked object.

Segment Anything. Segment Anything Model (SAM) [33] by Meta AI Research published in April 2023 is a segmentation model trained on a large dataset. The masks returned by same are all boolean masks. SAM allows the user to query the image by selecting a point, multiple points, or a bounding box within the image. Due to the ambiguity of the query, the point can select the billboard, text, or letter, see Fig. 6, SAM can outputs multiple valid masks. It was found that 3 masks was sufficient to address most common cases, and this is used in the final model.

In its fully automatic mode SAM samples a grid of 32x32 points to extract all the segments in the image. Because some points in the grid may select the same object and return similar masks non-maximum suppression (NMS) [34] is used to remove duplicate masks.

2.5 Morphological Filters

All information from the following section comes from the book Principles of Digital Image Processing, Fundamental Techniques [35]. For a more in depth explanation on the topic of morphological filters, look there.

"In their original form, morphological filters are aimed at binary images, images with only two possible pixel values, 0 and 1 or black and white, respectively." [35]. Morphological filters morph the structures in the binary images by shrinking and growing them.

A morphological filter uses a structuring element H. The structuring element is a binary structure with a hot spot at the origin of the structure, see Fig. 7. This structuring element is then moved across the image to change the structures in the image. There are two operations that can be performed using binary morphological filters.

Dilation. Growing or dilation is a operation that uses the structuring element H to grow the structures in the binary image I. This is done by overlapping the hot spot of the structuring element H with the every pixel p with a 1 value and pasting the 1 values from the structure element H to the output image I^*, see Fig. 8 for an example using the structuring element H from Fig. 7. Using the reflected structuring element H^*, see Fig. 7, dilation can be defined as:

$$I \oplus H = I^* = \forall p \mid p \in I \mid I^*(p) = (\exists q \mid q \in H^* \mid I(p+q) \land H^*(q)) \tag{6}$$

Where p and q are both 2D pixel coordinates, and $p + q = (p.x + q.x, p.y + q.y)$.

Erosion. Erosion does the opposite of from dilation, and shrinks the structures in the image I. Instead of overlapping the structure element H, and

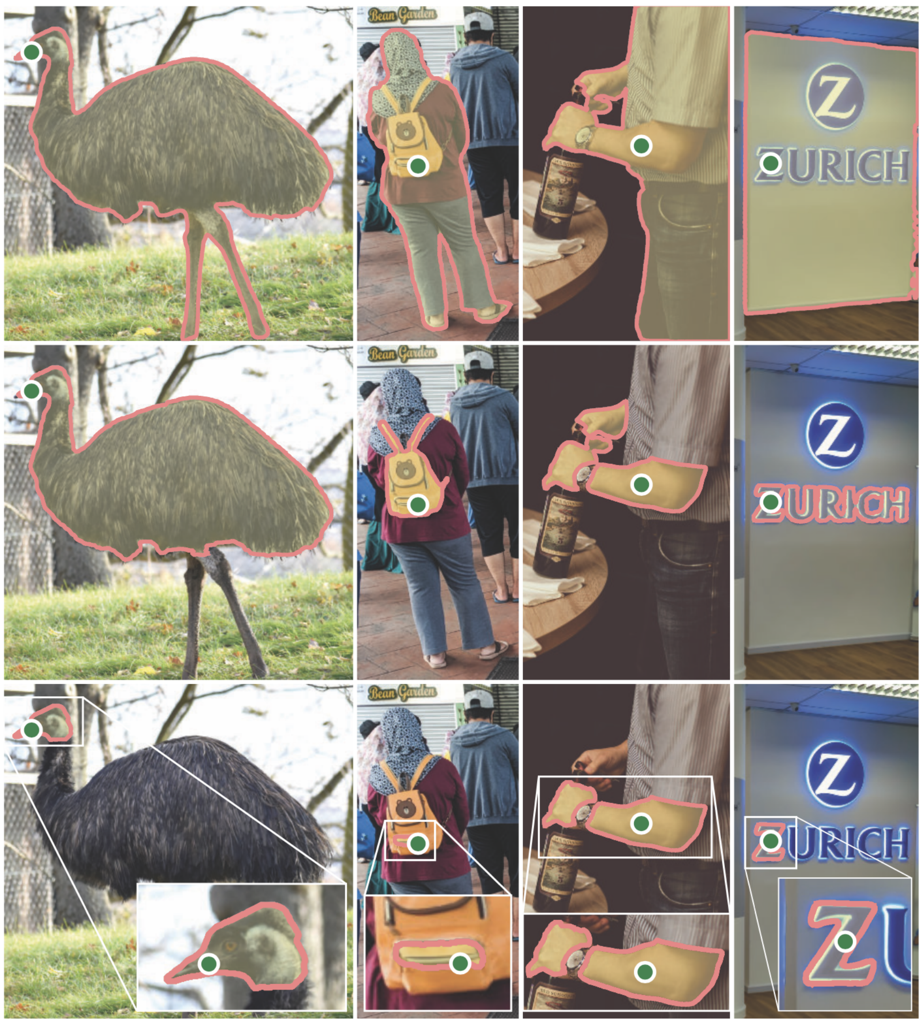

Figure 6: Each column shows 3 valid masks generated by SAM from a single ambiguous point prompt (green circle). (Directly from SAM paper [33])

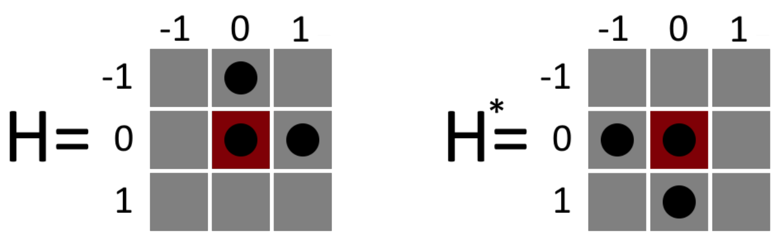

Figure 7: A binary structure element H and its reflect H^*. The red cell is the hot spot, cells with a 1 value are marked using a •, cells with a 0 value are empty.

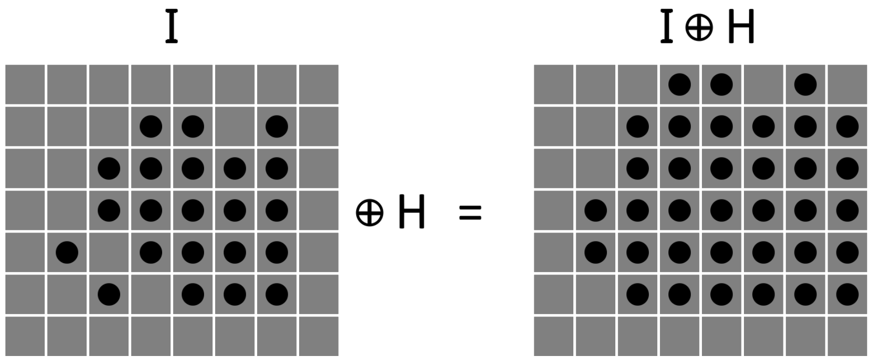

Figure 8: Dilation of image I using the structure element H from Fig. 7

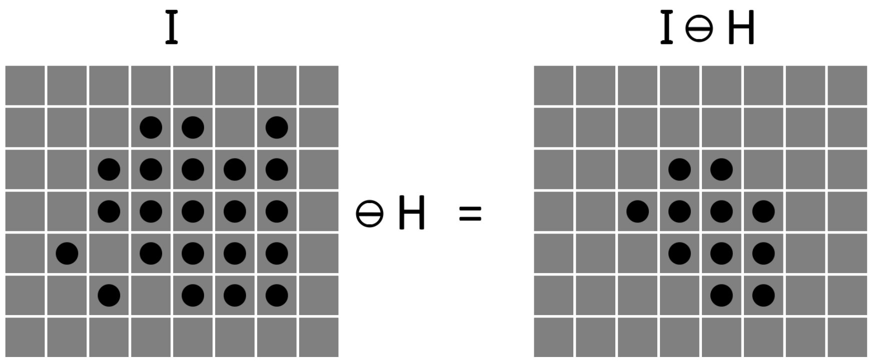

Figure 9: Erosion of image I using the structure element H from Fig. 7

pasting the values to the output image I^*, it only keeps the 1 values from the image I, if the 1 values in the structure element H all overlap with 1 values in the image I, see Fig. 9 for an example. Erosion can be defined as:

$$I \ominus H = I^* = \forall p \mid p \in I \mid I^*(p) = (\forall q \mid q \in H \mid \neg H(q) \vee (I(p+q) \wedge H(q))) \quad (7)$$

Where p and q are both 2D pixel coordinates, and $p+q = (p.x + q.x, p.y + q.y)$.

Erosion can be performed using dilation by using the inverted image $\bar{I}$, performing dilation using the reflected structuring element H^*, and inverting the end result:

$$I \ominus H = \overline{\bar{I} \oplus H^*}$$

Opening and Closing. A lot of operations can be performed by using dilation and erosion together, two of these operations are opening and closing.

Opening is used to open the structures in the image I by first eroding and then dilating the image I.

$$I \circ H = (I \ominus H) \oplus H$$

This results in small structures and smaller parts of bigger structures being removed, while larger and more solid structures remain unaffected, for an example see Fig. 10.

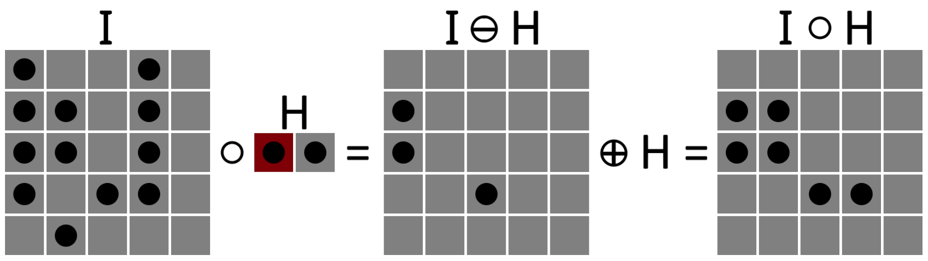

Figure 10: Opening operation with a small structuring element H. The small structures are removed from the image I.

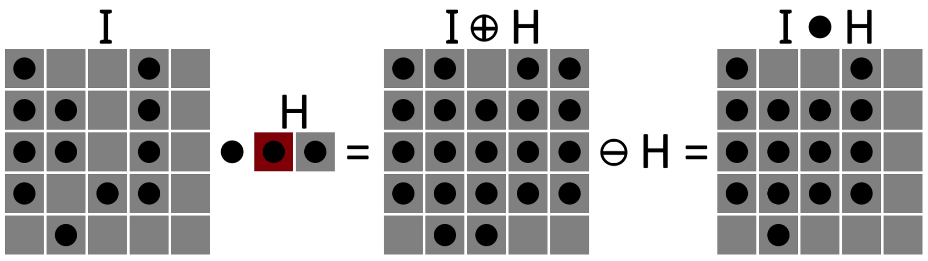

Figure 11: Closing operation with a small structuring element H. The small empty spaces inside or between structures in the image I are closed.

Closing does the opposite of opening. Closing first dilates the image I and then erodes the image I.

$$I \bullet H = (I \oplus H) \ominus H$$

Resulting in small openings inside or between structures being closed, for an example see Fig. 11.

3 Methodology

The next section describes how the objects in the images are isolated and selected, and how (depth-regularized) Gaussian splatting is modified to correctly optimize using these. It also describes problems in the SFM data that resulted in the focus shifting from optimizing multiple objects and later combining them, into only looking at centre scene objects.

3.1 Object Isolation

The goal of the object isolation stage is to create, for every object O, for every image I, a segment mask S that masks the object in that image. The segment masks $\mathcal{S}$ of an object O can then later be used to optimize that individual object.

To isolate objects from the input images the segments are used together with the generated SFM data of the scene.

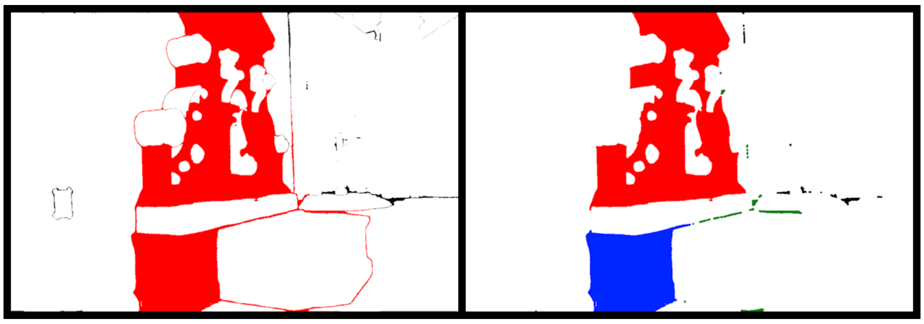
Figure 12: Left: combined segments S^*, region labeling would turn the red area into one large new segment. Right: closed combined segments $S^* \bullet H$, the region labeling now splits the original read area into multiple larger areas (red and blue), and multiple smaller areas (green).

Gathering Segments. To reduce the amount of memory needed during the object isolation stage, all segments are gathered and processed per image I_i from the input images $\mathcal{I}$. The index i denotes the current image that is being worked on.

The first stage of the isolation stage is to gather the segment masks $\mathcal{S}_i$, of the image I_i, that were generated using SAM.

Adding Missing Segments. Unfortunately the segment masks $\mathcal{S}_i$ created using SAM do not always cover the entire image, these missing region could still be part of objects and therefore should still be added to the total segments set $\mathcal{S}_i$.

To extract the missing segment masks all segment masks $S_a \in \mathcal{S}_i$ are combined into one large segment mask S_i^*. A closing filter with a 5×5 cross structure element H, see App. A, is used on this segment mask $S_i^* \bullet H$. This is done to remove the very small segments or the narrow spaces between segments. Very small segments removed by closing are to small to add anything useful, while the narrow spaces between segments are removed to completely separate two larger missing regions.

Next a region labeling algorithm [36] is used to extract the missing regions $\overline{S_i^* \bullet H}$ as new segment masks. Every region discovered with the region labeling is used as a new segment mask, these new segments masks are added to the image segment mask set $\mathcal{S}_i$. While SAM can create segment masks with multiple disjoint segments, the region labeling extracts every segment as a new segment mask, even when they might be part of the same object, just split by another object in the foreground.

Creating Masks. The next step is to gather the points, from the SFM generated point cloud $\mathcal{X}$, that were originally also visible on the image:

$$\mathcal{X}_i = \{X \mid X \in \mathcal{X}, X.F \in \mathcal{F}_i\}$$

Where $\mathcal{F}_i$ is the feature set of image I_i. Using these points $\mathcal{X}_i$ a mask is created by combining the subset of these points $\mathcal{X}_{ia} \in \mathcal{X}_i$ with the segment mask S_a

they are projected on:

$$M_a = (S_a, \mathcal{X}_{ia} = \{X \mid X \in \mathcal{X}_i, S_a(X)\})$$

Where $S_a(X)$ is true if point the projection of the 3D position x of point X is inside segment S_a. This is done for every segment mask $S_a \in \mathcal{S}_i$, creating the mask set $\mathcal{M}_i$. All masks $M_a \in \mathcal{M}_i$ that do not have any points $\#M_a.\mathcal{X} = 0$ inside its segment mask are, for now, discarded.

Add Discarded Segments. Some of the discarded segments masks might still be part of objects, but are sub parts that have no features triangulated and were therefore discarded. Because these segments could still be part of objects, they should still be added.

To try and add back a discarded mask S_a, it is first dilated using a $s \times 3$ cross structure element H, see App. A. This is done to enlarge the segment mask S_a by one pixel along the edges. If the discarded segment mask S_a did not originally have an overlap with another, not discarded, segment mask S, but was directly next to one, it will have an overlap when enlarged $S_a \oplus H$. The enlarged segment mask $S_a \oplus H$ is combined with the segment mask $S_b \in \mathcal{S}_i$ that has the most overlap $S_a \cap S_b$:

$$S_b^* = (M_b.S \mid M_b \in \mathcal{M}_i, b = \arg\max_c(\#(S_a \cap (M_c.S \mid M_c \in \mathcal{M}_i))))$$

This combined segment mask $S_{ab} = S_a \cup S_b^*$ is stored as the new segment mask of the mask M_b^* whose segment mask S_b^* the discarded segment S_a was combined with.

Combine Masks. The next step is to combine masks in the same image $(M_a, M_b) \in \mathcal{M}_i \times \mathcal{M}_i$ that have a lot of overlap. Segment masks created by SAM are not all disjoint from each other, and can have overlap. The overlap of masks is determined by the number of points they have in common $M_a.\mathcal{X} \cap M_b.\mathcal{X}$. There are two ways that are used for calculating an overlap score of two masks $Q(M_a, M_b)$, the first is intersection over union (IoU):

$$Q_{IoU}(M_a, M_b) = \frac{\#(M_a.\mathcal{X} \cap M_b.\mathcal{X})}{\#(M_a.\mathcal{X} \cup M_b.\mathcal{X})} \tag{8}$$

This is a standard score between $[0, 1]$ that calculates how similar the two masks (M_a, M_b) are. Here 0 means that there is no similarity, while 1 means perfect similarity.

This might not be a good score for the purpose of combining masks, because masks that are completely within another mask $M_a.\mathcal{X} \subset M_b.\mathcal{X}$ while still being significantly smaller will result in a small IoU score. This is counter intuitive, small masks fully within large masks should still get a decent score, therefore another new score is created. This new score, squared intersection score (SQI), tries to mitigate this problem:

$$Q_{SQI}(M_j, M_k) = (\frac{\#(M_a.\mathcal{X} \cap M_b.\mathcal{X})}{\#M_a.\mathcal{X}})^2 + (\frac{\#(M_a.\mathcal{X} \cap M_b.\mathcal{X})}{\#M_b.\mathcal{X}})^2 \tag{9}$$

SQI returns in a score between $[0, 2]$. Here 0 still means that there is no similarity, and the upper bound 2 still means perfect similarity, but all mask combinations where one mask is within another mask have a score > 1.

For the combination the size of a mask M_a is determined by the number of points $\#M_a.\mathcal{X}$ that are inside the mask. The largest mask M_a is selected first, and is combined with the mask M_b where the score $Q_{SQI}(M_a, M_b)$ is highest and higher than the set threshold t_{masks}. The threshold is set relatively low $t_{mask} = 0.1$, this is to ensure that most masks have no longer any overlap in points. This is done till no masks can be added to the mask M_a, then the next largest mask is selected. See App. B for the algorithm.

When combining two masks (M_a, M_b) both the segment masks, and the point sets get combined $M_{ab} = (M_a.S \cup M_b.S, M_a.\mathcal{X} \cup M_b.\mathcal{X})$. This new mask M_{ab} replaces the original chosen mask M_a, and the combined mask M_b is removed.

Create Objects. When all images have their masks created and combined the masks are stored in a global mask set $\mathcal{M}$. With the global mask set $\mathcal{M}$ the objects $\mathcal{O}$ can be created. Objects are a collection of segment masks from different images. The objects are created in a similar way that the masks are combined. Masks from different images are combined using the same scores (Eq. 8, 9).

There are two different algorithms for combining masks to create objects, growing and non-growing object creation. The growing algorithm directly transforms all masks into objects $\mathcal{O} = \{O = (\{M.S\}, M.\mathcal{X}) \mid M \in \mathcal{M}\}$, and similar to mask combination the object O_a with the most points $\#O_a.\mathcal{X}$ is selected first and combined with the object O_b with the highest score $Q(O_a, O_b)$ above the threshold t_{object}. This threshold can be set depending on the scene and the score function Q that is used. The new larger object is then again checked for objects to match with, see App. C for the algorithm.

The downside of the growing algorithm is that, because the object keeps growing, the score over time decreases. The non-growing algorithm mitigates this. Similar to the growing algorithm, the non-growing algorithm transforms all masks into objects $\mathcal{O} = \{O = (\{M.S\}, M.\mathcal{X}) \mid M \in \mathcal{M}\}$, and it also selects the object O_a with the most points $\#O_a.\mathcal{X}$. All objects O_b with a score $Q(O_a, O_b)$ higher then the threshold t_{object} are selected for combination, but are not combined yet. Once all objects have been checked the objects that were selected for combination are combined. This way the objects do not grow during the algorithm, see App. C for the algorithm.

During the combination of two objects (O_a, O_b), similar to the mask combination, the new object $O_{ab} = (\mathcal{S}_{ab}, \mathcal{X}_{ab})$ replaces the initial selected object O_a, and the combined object O_b is removed. The points get directly combined $\mathcal{X}_{ab} = O_a.\mathcal{X} \cup O_b.\mathcal{X}$. For the segment masks $(O_a.\mathcal{S}, O_b.\mathcal{S})$ of the objects, if two segment masks came from the same image, they are combined $S_{ab} = S_a \cup S_b \mid S_a \in (\mathcal{S}_i \cap O_a.\mathcal{S}), S_b \in (\mathcal{S}_j \cap O_b.\mathcal{S}), i = j$ and stored in the new segment mask set $\mathcal{S}_{ab}$. The other segment masks get added directly to the new segment mask set $\mathcal{S}_{ab}$. Therefore in the end, an object $O \in \mathcal{O}$ has at most the same number of segments masks as there are images.

Object Selection. Now that the objects have been created an object needs to be selected for optimization. Unfortunately one object O_i does not always completely encapsulate the entire object. This is due to features in SFM not matching over all images, and therefore creating separate objects for the same scene object, see Fig. 13 for example. Therefore it is very difficult to select

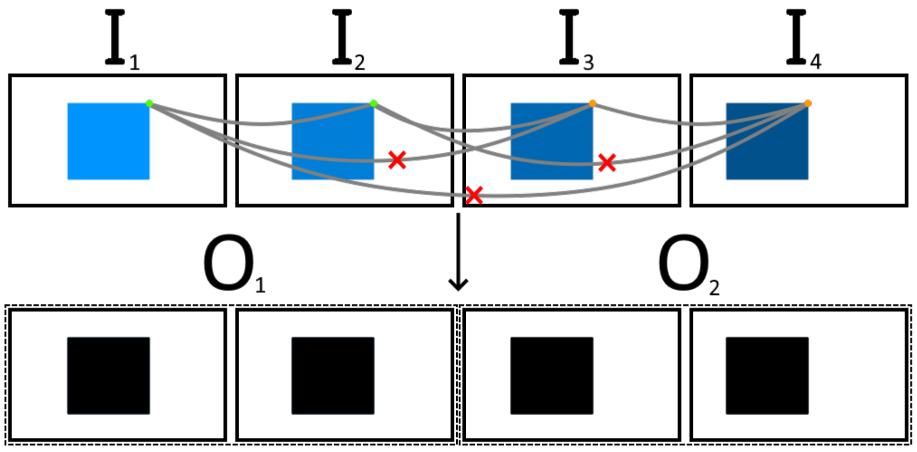

Figure 13: The green dots and the orange dots are both the same feature but could not be matched, resulting in two separate objects being created. Red crossed lines should be matched, but are not.

multiple objects, optimize them, and then later combine them. Thus the focus only lies on the object that is in the centre of most of the input images, a centre scene object.

To create the final object O^* that is going to be optimized, one or multiple objects need to be selected and combined. During the selection there is also a gathered images $\mathcal{I}^*$ set that keeps track of all the images $I_i \in \mathcal{I}$ that are no longer considered.

An object $O_a \in \mathcal{O}$ is selected to be part of the final object O^*, if it is the object with most segment masks $S_i \in O_a.\mathcal{S}$ at the center point c of the image I_i. Only segments S_i where the original image is not in the gathered images set $I_i \notin \mathcal{I}^*$ are counted:

$$O_a^* = O_a \mid O_a \in \mathcal{O}, a = \arg\max_b(\#\{S_i \mid S_i \in O_b.\mathcal{S}, I_i \notin \mathcal{I}^*, S_i(c) > 0\})$$

Once an object O_a^* has been selected, the mean size of all the segment masks S_i, where the original image is not in the gathered images set $I_i \notin \mathcal{I}^*$, is calculated. All the original images $I_i \notin \mathcal{I}^*$, where the size of the segment mask is larger than half the mean size $\#\{p \mid p \in S_i, S_i(p) > 0\} > 0.5 * mean$, are added to the gathered images set $\mathcal{I}^*$.

This is done until, either the gathered images set contains all images $\#\mathcal{I}^* = \#\mathcal{I}$, or no objects with segment masks in the center can be found. Because the algorithm only looks at segment masks S_i for which the original image is not in the gathered images set $I_i \notin \mathcal{I}^*$, every iteration adds at least one new image I_i to the gathered images set $\mathcal{I}^*$. Thus the algorithm runs at most the number of iterations as there are images. For the algorithm see App. D.

Diffuse Segments. As a final step, because the masks are not always perfect at the edges, the segment masks of the final object are diffused using a 7×7

Gaussian filter [13] with a $\sigma = 1$:

$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

This turns the boolean segment masks into floating point segment masks, these segment masks are used during the optimization as weights in the optimization functions.

In the end an object is returned with a floating point segment mask $S_i^{\sim}$ with values between $[0, 1]$ for every image $I_i \in \mathcal{I}$, and a point set of the points that was used to create this object $O = (\mathcal{S}^{\sim}, \mathcal{X})$.

3.2 Object Optimization

Now that the an object has been selected for optimization it needs to be initialized and then optimized. While the initialization, and optimization of the Gaussians differs from Gaussian splatting or depth-regularized guassian splatting, the densification of the Gaussians has not changed.

Object Initialization. The Gaussians for the objects are initialized similar to how Gaussian splatting initializes its Gaussians. The difference is that, instead of initializing all points from the SFM point cloud into Gaussians, only the points that are also part of the object are initialized.

Segmented Loss. Depth-regularize Gaussian splatting consists of four separate loss functions $(\mathcal{L}_{color}, \mathcal{L}_{DSSIM}, \mathcal{L}_{depth}, \mathcal{L}_{smooth})$, these loss functions compare the entire render I_i^* with the image I_i. Because only the Gaussians of the selected object are initialized most of the render I_i^* is empty. This empty space will have a fixed depth and background color, differing from the image I_i, resulting in a high loss value. This will also result in that Gaussians will start to move towards the empty space, to get the correct color and depth, to reduce the high loss. Therefore eventually the entire scene will be trained again, instead of only the selected object. Therefore the loss functions $(\mathcal{L}_{color}, \mathcal{L}_{DSSIM}, \mathcal{L}_{depth}, \mathcal{L}_{smooth})$ need to be changed.

The loss functions $(\mathcal{L}_{color}, \mathcal{L}_{DSSIM}, \mathcal{L}_{depth}, \mathcal{L}_{smooth})$ are changed by multiplying every pixel p with the same pixel in the segment mask $S_i^{\sim}$ of the image I_i. This will only give a loss to pixels p inside the segment mask $S_i^{\sim}$, and therefore only change the Gaussians that have pixels inside the segment mask $S_i^{\sim}$. The downside of this is that this will result in a 0 loss value for pixels that are not in the segment mask $S_i^{\sim}$ and are not of interest. This results in the loss functions being lower when the objects segment mask $S_i^{\sim}$ is smaller. As a result it is also difficult to directly compare the quality of objects in two different scenes, because the loss is now also dependant on the size of the segment mask $S_i^{\sim}$. Therefore only the pixels p in the segmentation mask $S_i^{\sim}$ where $S_i^{\sim}(p) > 0$ are counted. Making the final $\mathcal{L}_{color}$ loss function:

$$\mathcal{L}_{color} = \frac{\sum_{p \in S_i^*} ||S_i^{\sim}(p) * (I_i(p) - I_i^*(p))||}{\#\{p \mid p \in I_i, S_i^{\sim}(p) > 0\}}$$

This is done for all loss functions $(\mathcal{L}_{color}, \mathcal{L}_{DSSIM}, \mathcal{L}_{depth}, \mathcal{L}_{smooth})$.

Segment Bound Constraint. The modified loss functions no longer constrain any part of Gaussians that are completely outside the object segment mask $S_i^{\sim}$. This needs to changed, otherwise Gaussians at the edge of objects can grow infinitely into the empty space. To do this a new loss function is introduced, this new loss function $\mathcal{L}_{bounds}$ penalizes any Gaussians that grow outside the object segment mask $S_i^{\sim}$, while it does not do anything with Gaussians that are completely within the object segment mask $S_i^{\sim}$.

The new loss function is similar to the $\mathcal{L}_{color}$ loss function. Because the color does not matter, and only how visible a Gaussian is outside the object segment mask $S_i^{\sim}$, the loss function $\mathcal{L}_{bounds}$ does not look at the color but at the alpha α of the pixel. It calculates the absolute difference between the alpha α of the render image I_i^* and the image I_i. The alpha α value of a pixel is calculated as $1 - T_{final}$, see Eq. 3. Where T_{final} is the T value of the last Gaussian that contributed to the pixel.

To ensure that only a loss is applied to the Gaussians with a part outside the object segment mask $S_i^{\sim}$, the absolute alpha difference is weight using the inverse object segment mask $1 - S_i^{\sim}$. Finally to ensure that the size of the object does not influence the loss, only pixels p where $1 - S_i^{\sim}(p) > 0$ are considered. The final $\mathcal{L}_{bounds}$ loss function looks like:

$$\mathcal{L}_{bounds} = \frac{\sum_{p \in S_i^*} ||(1 - S_i^{\sim}(p))((I_i(p).\alpha * S_i^{\sim}(p)) - I_i^*(p).\alpha)||}{\#\{p \mid p \in I_i, 1 - S_i^{\sim}(p) > 0\}}$$

This new loss function $\mathcal{L}_{bounds}$ is now added to the total loss function (Eq. 5), and is weighted using a hyperparameter λ_{bounds}:

$$\mathcal{L} = (1 - \lambda_{SSIM})\mathcal{L}_{color} + \lambda_{SSIM}\mathcal{L}_{D-SSIM} \\ + \lambda_{depth}\mathcal{L}_{depth} + \lambda_{smooth}\mathcal{L}_{smooth} + \lambda_{bounds}\mathcal{L}_{bounds}$$

Scoring. Three scoring functions are used to compare a final rendered image I^* with the ground truth image I. These are the Peak Signal-to-Noise Ration (PSNR), the Structural Similarity Index Measure (SSIM), and the Learned Perceptual Image Patch Similarity (LPIPS).

The PSNR score uses the mean squared error (MSE) over the entire image and render $MSE = \frac{\sum_{p \in I}(I(p) - I^*(p))^2}{\#I}$. PSNR directly compares pixels in the images and does not look at the structures. For the PSNR score, higher scores are better.

The SSIM score is different from the PSNR score in that it does look at structures in the images. The SSIM score looks at structures and neighbouring pixels by diffusing the images. In SSIM higher scores represent more similar images.

The LPIPS score is a learned score that uses a pre-trained neural network that compares images. For this score, lower scores are better.

4 Tools and Implementation

There are a number of different tools that are used in the implementation of object isolation, and the Gaussian optimization model. While some of these tools can be changed for others, some, like the depth models, have direct influence on the result of the method.

4.1 Tools and Libraries

The main program is implemented in Python 3.11.5, this was the newest python version at the start of the project. although lower python versions could also be used.

The primary library used for the implementation is PyTorch. PyTorch is also used by (depth-regularized) Gaussian splatting, SAM, and Marigold. PyTorch is useful for optimization models, because it already has a back propagate already implemented in its tensors and functions, therefore mitigating the need to manually implement them.

All GPU code is implemented using CUDA kernels, and is compiled using NVIDIA CUDA Compiler (NVCC). Ninja 1.10.2 is used to create the build.

4.2 SFM Data

The SFM data is generated using PyColmap, which is a python implementation of Colmap [6]. The SFM features are extracted using SIFT, and matched using exhaustive feature matching.

4.3 Object Isolation

Object isolation uses a lot of different techniques. Some of these techniques are implemented on the GPU to speed up performance.

Segment Generation. Segments are generated using fully automatic SAM [33]. All settings are kept as default.

Filters. The morphological filters, and the Gaussian filter are all implemented on the GPU. The filters are implemented in such a way that for every pixel off the output a thread is initialized that only writes to that pixel in the output, seen in Eq. 6 and 7. This is done to remove racing condition, where multiple threads try to write to a single pixel.

Matching SFM Points to Segment. To speed up the process of matching SFM points to segments, this is also implemented on the GPU.

SFM already stores the projection of the original image I_i points $\mathcal{X}_i$, therefore mitigating the need to manually filter and project the points $\mathcal{X}_i$ from the entire point cloud $\mathcal{X}$.

All points $X \in \mathcal{X}_i$ are stored in an array. For every point X a thread is initialized that checks if the point is inside the currently checked segment mask S_a. The result of this check is written back into an array of the same size as the point array. This boolean array is later used as a mask to filter the correct points from the point set $\mathcal{X}_i$.

Region Labeling. The region labeling algorithm is an implementation of the sequential region labeling algorithm outlined in Principles of Digital Image Processing, Core Algorithms [36]. The regions are labeled using a 8-connected neighbourhood, therefore also connecting diagonal regions.

4.4 Gaussian Splatting

Due to the initial lack of access to the depth-regularized Gaussian splatting code [24, 37], there are two implementations of the optimization. The first is a modified version of the depth-regularized Gaussian splatting code. And the second is a complete new implementation based on GSplat [38, 39], named Isolated Object splatting (IsO).

Depth Maps. The original depth-regularized Gaussian splatting uses ZoeDepth [26] for the generation of its D_{dense} depth maps. Therefore the modified version also makes use of this for the generation of its D_{dense} depth maps. The new implementation uses Marigold [25] for its D_{dense} depth map generation.

Both methods work in a similar way in that they directly estimate the depth of a single image, zero shot, without looking at the other images. Both also return an estimated depth per pixel for the entire image. Marigold was chosen for the new method because it was a slightly newer method than ZoeDepth.

For determining the scale S^* and offset t^*, that are used to compensate the depth map D_{dense}, a separate optimization model is used. This model uses gradient descend to find the optimal scale S^* and offset t^*. This optimization model stops its search when the difference in loss between two iterations is smaller than $1e-5$.

Edge Map. The edge map for the $\mathcal{L}_{smooth}$ loss function is created the same way as in depth-regularized Gaussian splatting, using the Canny edge detector from OpenCV (openv-python 4.9.0.80). The Canny edge thresholds are set with a high threshold $t_{high} = 150/255$ and a low threshold $t_{low} = 50/255$.

Optimization. The modified depth-regularized Gaussian splatting has not only its loss functions changed such that they are capable of using the new object segment masks, but also has its backwards CUDA kernel updated such that it handles the delta opacity from the bounds loss $\mathcal{L}_{bounds}$.

The new implementation uses a clone of the GSplat CUDA kernels for its forward and backward pass. GSplat was chosen as a base, because it implements Gaussian splatting in separate projection and rasterization steps, this made for easier debugging.

The GSplat kernels do not initially return the depth value of the pixel, therefore they have been updated to also return the depth value, and back propagate the depth loss. The rest of the implementation, like the densification and culling, tries to mimic the original Gaussian splatting paper [3] as close as possible, by using the same criteria, and the same thresholds.

5 Results

The experiment results are subdivided into two sections, the first is focussed on the new object isolation section, while the second looks at the optimization of the objects.

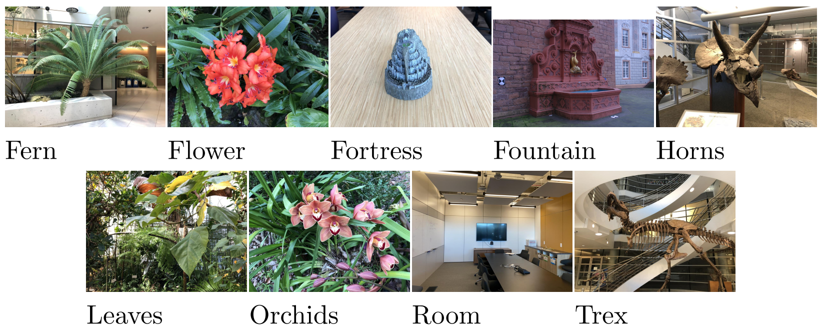

Figure 14: Dataset

5.1 Dataset

The dataset that is used is the same dataset as the depth-regularized Gaussian splatting paper [24]. This dataset consists of 8 scenes: Fern, Flower, Fortress, Horns, Leaves, Orchids, Room, and Trex, see Fig. 14. One additional scene is added to the test set, the Fountain scene is provided as a test scene for colmap, and is also used. The images of the fountain scene are scaled back to 1200×800 to be of similar size to the other scenes.

5.2 Mask Creation

The methodology of the object isolation, see Sec. 3, describes eight stages in the object isolation method: gathering segments, adding missing segments, create masks, add discarded segments, combine masks, create objects, object selection, and diffuse segments. This section covers the results of the first five stages of the object isolation method.

The thresholds for the scoring function, in the combine mask stage, is fixed to $t_{SQI} = 0.1$, see Eq. 9. Table 1 shows the progression of the number of segment masks and masks throughout first five stages of the object isolation method. The number of segment masks gathered in the gathering segments stage (initial), the number of missing segment masks found in the adding missing segments stage (missing), the number of masks created and discarded in the create masks stage (created, discarded), the number of discarded segment masks added back in the adding missing segments stage (added), the number of masks combined in the combine masks stage (combined), and the final number of masks at the end of the combine masks stage (combine).

The number of initial segment mask and missing segment masks combined should equal the number of created and discarded segment masks $initial + missing = created + discarded$. Similarly the number of final segments is equal to the number of combined segments subtracted from the number of created segments $final = created - combined$.

Ablations. The mask creation has a few ablations that can be explored. The first is the effect that the closing filter, in the missing adding segments stage, has on the number of missing segments that are extracted. The number of missing segments extracted for closed and not closed can be found in table 2.

	initial	missing	created	discarded	added	combined	final
Fern	1784	1278	1863	1199	1183	683	1180
Flower	6364	3094	7535	1923	1909	2285	5250
Fortress	1109	571	838	842	794	464	374
Fountain	1087	234	898	423	410	748	150
Horns	7349	2728	7469	2608	2597	3909	3560
Leaves	5450	795	2230	4015	3571	445	1785
Orchids	6586	3162	5764	3984	3739	1866	3898
Room	5469	1812	5247	2034	1995	2610	2637
Trex	6591	2116	6792	1915	1869	2987	3805

Table 1: Number of segment masks and mask during first five stages in the object isolation method, see Sec. 3. The number of initial segment masks, the number of missing segment masks, the number of masks created and discarded from the segment masks, the number of discarded masks that were added back, the number of masks that were combined, and the final number of masks at the end of the combine masks stage.

missing segments	closed	not closed
Fern	1278	2707
Flower	3094	14713
Fortress	571	2900
Fountain	234	1146
Horns	2728	20364
Leaves	795	4104
Orchids	3162	12103
Room	1812	9428
Trex	2116	13489

Table 2: The number of missing segment masks extracted in the missing segments stage of the object isolation method, with or without using the closing filter.

Here it is visible that the closing filter reduces the number of missing segments extracted. This is because, although the closing filter is capable of splitting one large missing segment into multiple smaller segments, see Fig. 12, the number of small missing segments that are completely removed by the closing filter is larger.

The next is effect the dilation filter, in the add discarded segments stage, has on the number of segment masks added back. The number of segments added back is shown in table 3. It is peculiar that without using the dilation filter on the discarded segment masks, no segment masks are added back. This shows that none of the discarded segment masks have any overlap with another segment mask. This is most likely because these were either small missing segment masks, who by the way that the missing segment masks are extracted do not have an overlap, or they are segment masks of completely disjoint objects where there are no features.

The last ablation study shows the effect the different scoring methods (Q_{IoU}, Q_{SQI}) have on the number of masks that are combined in the combine masks stage, visi-

added segments	dilated	not dilated
Fern	1183	0
Flower	1909	0
Fortress	794	0
Fountain	410	0
Horns	2597	0
Leaves	3571	0
Orchids	3739	0
Room	1995	0
Trex	1869	0

Table 3: The number of discarded segment masks that are added back in the add discarded segments stage, with or without using the dilation filter.

masks combined	SQI	IoU
Fern	683	536
Flower	2285	1783
Fortress	464	256
Fountain	748	228
Horns	3909	2310
Leaves	445	367
Orchids	1866	1604
Room	2610	2103
Trex	2987	2079

Table 4: The number of masks combined in the combine masks stage using either the SQI or IoU scoring function. Both scoring functions have a threshold of $t = 0.1$.

ble in table 4. Both scoring methods have the same threshold $t_{IoU} = t_{SQI} = 0.1$. Here it is visible that for every scene the Q_{IoU} (Eq. 8) scoring method combines less masks than the Q_{SQI} (Eq. 9) method. This is due to small masks with very few feature points inside larger masks with more feature points not reaching the threshold in the Q_{IoU} scoring method, while in the Q_{SQI} scoring method they have a score of at least 1.

5.3 Object Creation

This section covers the results of the last three stages, create objects, object selection, diffuse segments, of the object isolation method. The create objects stage uses the masks created by the previous stages, using all filters and combined using the SQI scoring method $t_{SQI} = 0.1$. These masks are combined into objects using the growing object creation algorithm with SQI scoring using a threshold of $t_{SQI} = 0.75$. This threshold was chosen as a good balance for combining objects, even when they do not completely overlap, and not combining objects when very little feature points overlap, like the feature points at the edges of the segment mask. The growing object creation algorithm was selected to prevent the objects from growing too much, and therefore no longer containing a single scene object. Finally object selection stage selects the centre

	final masks	created objects	selected objects
Fern	1180	434	20
Flower	5250	876	**1**
Fortress	347	106	**1**
Fountain	150	33	**1**
Horns	3560	1697	**1**
Leaves	1785	49	1
Orchids	3898	1566	**1**
Room	2637	979	30
Trex	3805	1258	50

Table 5: Number of created objects, and number of selected objects for every scene. Objects are created using the growing object creation algorithm with an SQI threshold $t_{SQI} = 0.75$. Selected objects boldface **1** are the centre scene objects that are correctly fully created by the object creation algorithm.

scene object.

Table 5 shows for every scene the number of objects created, in the create objects stage, from the final masks, see table 1, and the number of objects selected for the centre scene object by the object selection stage. Fig. 15 shows a selection of the segment masks of the selected centre scene object.

From the data it appears that, when there is a solid centre scene object, like Flower, Fortress, Fountain, Horns, or Orchids, the object creation algorithm will be able to fully construct the correct masks into a single object. It also shows that objects that are in the centre of the screen, but can not be correctly created into a single object due to the SFM data quirk, see Fig. 13, like Fern and Room, can still be correctly selected using the selection algorithm. Also is of note that sometimes additional parts get added, like Horns or Room, or parts get left behind, like Flower. Finally, scenes where there is no solid centre scene object, like Leaves and Trex, the algorithm is incapable of creating and selecting the correct object. In Leaves everything gets cluttered together into one large object, while in Trex there is a total mismatch.

Ablations. For the ablations both versions of the object creation algorithm, growing and non-growing, are used, with both scoring functions, Q_{IoU} Eq. 8 and Q_{SQI} Eq. 9, on different values of their scoring thresholds t_{IoU} and t_{SQI}. When using the IoU scoring in the object creation, the IoU scoring is also used during mask creation, similar with the SQI scoring. 81 different thresholds are used for every object creation algorithm and scoring method combination. The extracted data is the number of objects created and the number of objects selected.

The graph in figure 16 shows, for every combination of object creation algorithm and scoring method, the average number of masks per object. The object creation algorithm at most turns every mask into a single object. The average number of masks for a specific object creation algorithm and scoring combination at a threshold t is calculated by combining the average number of masks per object for every scene, and dividing that by the number of scenes.

The graph directly shows that, as expected, in almost all cases the SQI scoring combines more masks into a single object than the IoU scoring. It is also

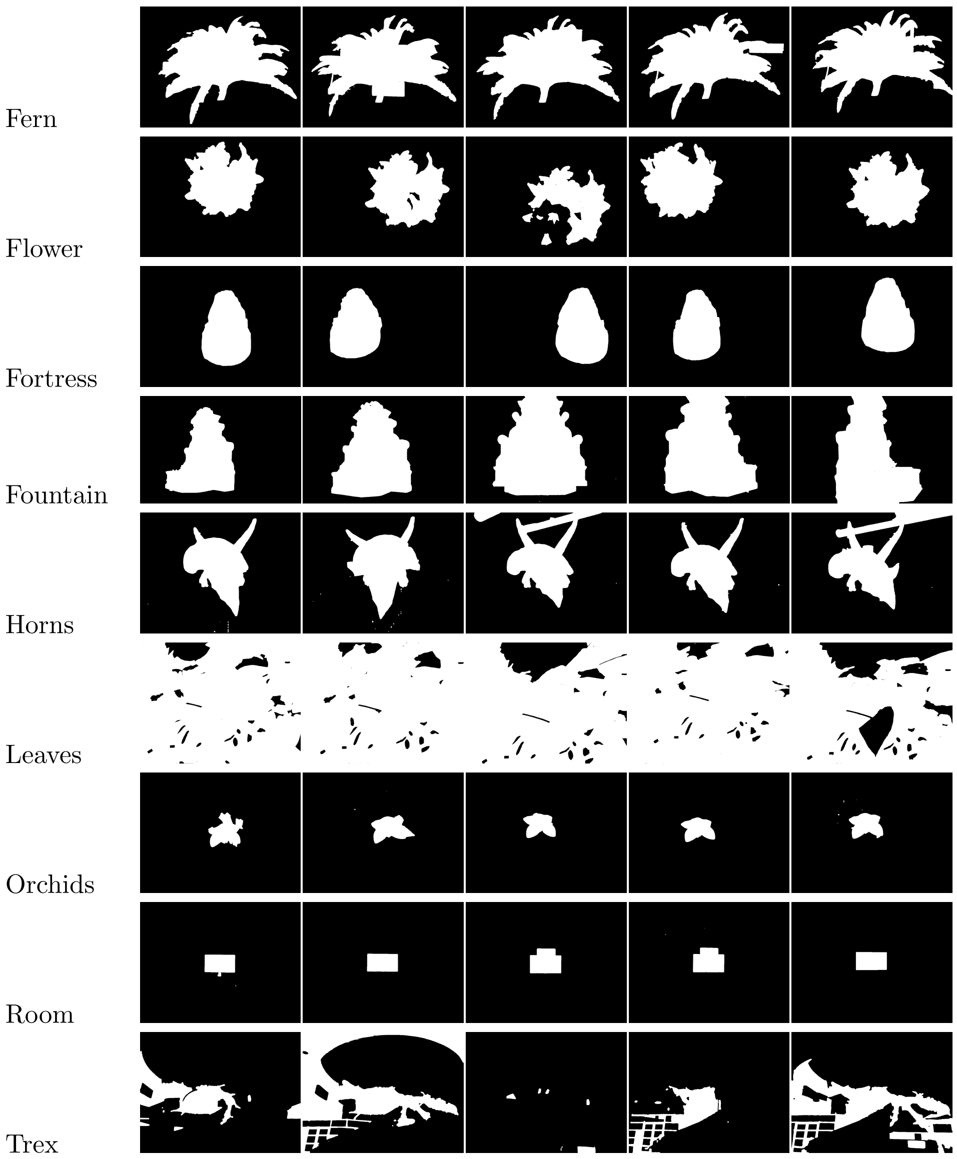

Figure 15: Final segment masks of the selected objects. The objects are created using the growing object creation algorithm with a SQI threshold $t_{SQI} = 0.75$.

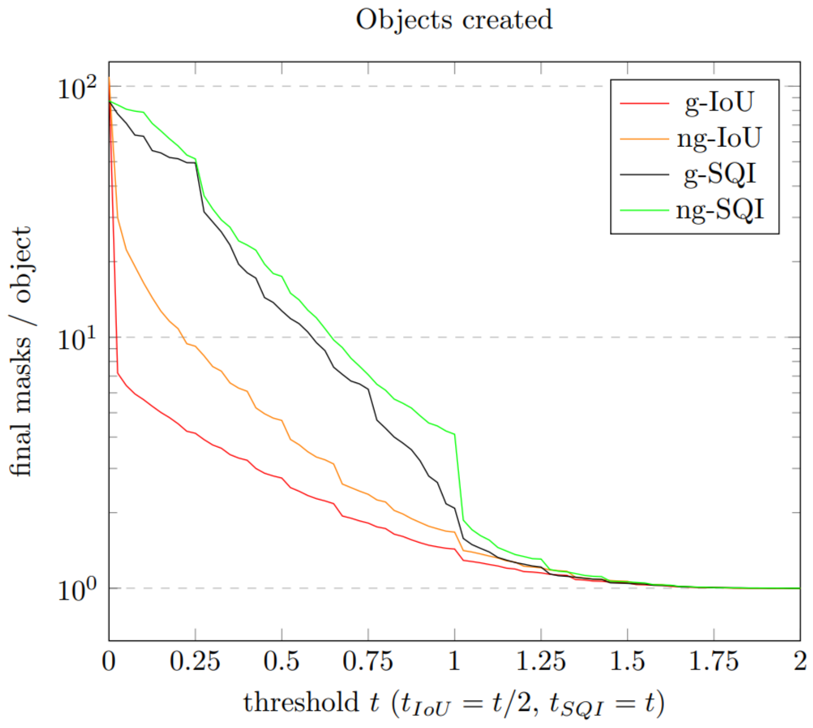

Figure 16: Graph of the average number of final masks per object for different object creation algorithms and scoring methods (g: growing algorithm, ng: non-growing algorithm).

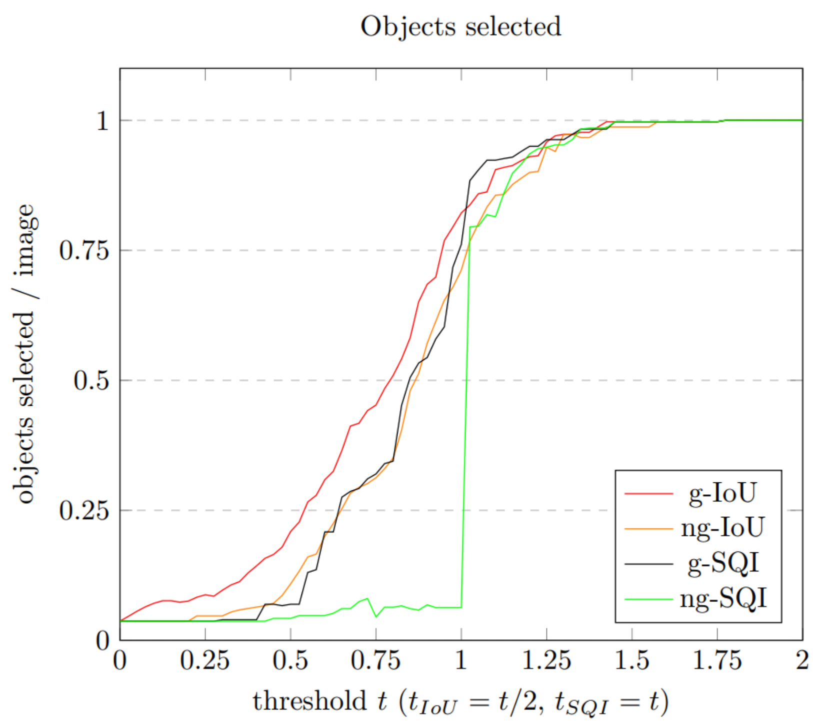

Figure 17: Graph of the number of objects selected for the centre scene object per image in the scene for different object creation algorithms and scoring methods (g: growing algorithm, ng: non-growing algorithm).

visible that the non growing algorithm combines more masks into a single object than the growing algorithm for both scoring functions. This can be explained by how the growing algorithm creates the objects. It grows the objects, adding more masks, therefore slowly no new masks will reach the threshold due to the size of the object. The non-growing algorithm does not do this, and therefore continues to add masks to the objects. Also of note is that at $t = 1$ the number of masks per object significantly decreases for all score and method combinations. For the SQI score function this can be explained by the objects that are completely inside other objects, here they will stop all being combined into one single object.

The graph in Fig. 17 shows the average number of objects selected, by the object selection algorithm, per image for every threshold t. The object selection algorithm selects at least 1 object, and at most 1 object per image. The average number of objects selected per image for a threshold t is calculated by combining the average number of objects selected per image for every scene, and dividing that by the number of scenes.

Here it is visible that the non-growing algorithm with SQI scoring, till a threshold $t = 1$, combines enough masks of different images into one object to create large objects such that only a few objects have to be selected for the centre scene object. Although this looks good, this might also show that it keeps combining objects too many objects, and therefore not only the wanted

Figure 18: Fern segments created with the non-growing object creation algorithm using SQI scoring with $t_{SQI} = 0.75$.

object is in the masks of the centre scene object, see Fig. 18. There is also a strange dip in the non-growing algorithm with SQI scoring around $t = 0.75$, this is a result of the Room scene all of a sudden selecting less objects around this point. It is unclear what causes this.

5.4 Object Optimization

For the optimization Gaussian splatting is run with objects created using a object creation scoring threshold $t_{SQI} = 0.75$. To test how solid the scene is, the scenes are trained on a black (0) background, and verified on a white (1) background. If a scene is completely solid, then the background color should not matter. Table 6 shows the scores for the trained scenes after 30.000 iterations of three different methods. The original Gaussian splatting (GS) method, the depth-regularized Gaussian splatting (D-GS) method, and the self implemented method, isolated object splatting (IsO). The base version is trained without any masking, and is verified over the entire scene. The mask version uses the centre scene object's segment masks, generated using the non-growing algorithm with SQI scoring ($t_{SQI} = 0.75$). These are verified by applying the segment masks over the ground truth images:

$$I^*_{gt} = I_{gt} * S^{\sim} + white * (1 - S^{\sim})$$

Where $S^{\sim}$ is the floating point segment mask from the diffuse segments stage, and *white* is an image with all values as 1. Figure 19 shows the resulting rendered images.

From the data it is visible that the original Gaussian splatting (GS) almost always outperforms the other two methods. While this is not surprising for the isolated object splatting (IsO) method, due to it being a complete new implementation, this is somewhat peculiar for the depth-regularized Gaussian splatting (D-GS) method. The loss in accuracy for the D-GS comes from the over compensation of the depth data, the method tries to get the correct depths and therefore removes Gaussians in the back to get the correct depth, and it will start to rely more on the background color. This is also visible when looking at the images.

Ablations. For the ablations the optimization is done with five different $\mathcal{L}_{bounds}$ values (0.00, 0.25, 0.50, 0.75, 1.00), and with a random color changing background. This is done for both the modified depth-regularized Gaussian splatting (D-GS), and isolated object splatting (IsO). All scenes are trained for 30.000 iterations. Table 7 shows the PSNR scores of the ablations, and figure

		PSNR ↑		SSIM ↑		LPIPS ↓	
		base	mask	base	mask	base	mask
Fern	GS	**20.89**	**17.48**	0.796	**0.840**	**0.121**	**0.124**
	D-GS	20.79	15.51	**0.809**	0.838	0.144	0.197
	IsO	18.78	17.46	0.609	0.710	0.476	0.333
Flower	GS	17.53	**20.57**	**0.701**	**0.934**	**0.161**	**0.057**
	D-GS	15.03	15.88	0.637	0.800	0.237	0.294
	IsO	**20.25**	18.40	0.663	0.885	0.343	0.168
Fortress	GS	**28.17**	**31.47**	**0.922**	**0.974**	**0.044**	**0.015**
	D-GS	19.36	23.60	0.874	0.952	0.088	0.053
	IsO	18.73	26.87	0.768	0.912	0.198	0.091
Fountain	GS	**30.48**	**29.85**	**0.965**	**0.978**	**0.022**	**0.016**
	D-GS	22.49	15.53	0.943	0.873	0.054	0.133
	IsO	26.74	26.30	0.855	0.932	0.196	0.099
Horns	GS	20.15	**24.22**	**0.808**	**0.968**	**0.113**	**0.033**
	D-GS	**21.16**	15.97	0.790	0.900	0.176	0.138
	IsO	18.42	17.31	0.617	0.860	0.502	0.189
Leaves	GS	**14.25**	7.99	**0.449**	0.349	**0.336**	**0.478**
	D-GS	12.615	**8.92**	0.384	**0.375**	0.416	0.484
	IsO	10.77	6.83	0.189	0.221	0.782	0.756
Orchids	GS	16.19	**31.12**	**0.694**	**0.992**	**0.148**	**0.016**
	D-GS	14.84	18.03	0.649	0.917	0.224	0.113
	IsO	**17.47**	20.66	0.506	0.961	0.475	0.061
Room	GS	**29.00**	23.89	**0.947**	0.981	**0.044**	**0.026**
	D-GS	20.95	**30.73**	0.860	**0.981**	0.147	0.040
	IsO	20.66	22.23	0.803	0.960	0.432	0.075
Trex	GS	17.05	**14.14**	**0.799**	**0.861**	**0.107**	**0.247**
	D-GS	**19.45**	13.61	0.790	0.826	0.193	0.269
	IsO	15.48	10.94	0.631	0.753	0.486	0.410

Table 6: Scores for final trained scenes after 30.000 iterations. Loss function weights, where applicable: $\mathcal{L}_{SSIM} = 0.8, \mathcal{L}_{depth} = 0.5, \mathcal{L}_{smooth} = 1.0, \mathcal{L}_{bounds} = 1.0$. **Boldface** are the highest scores.

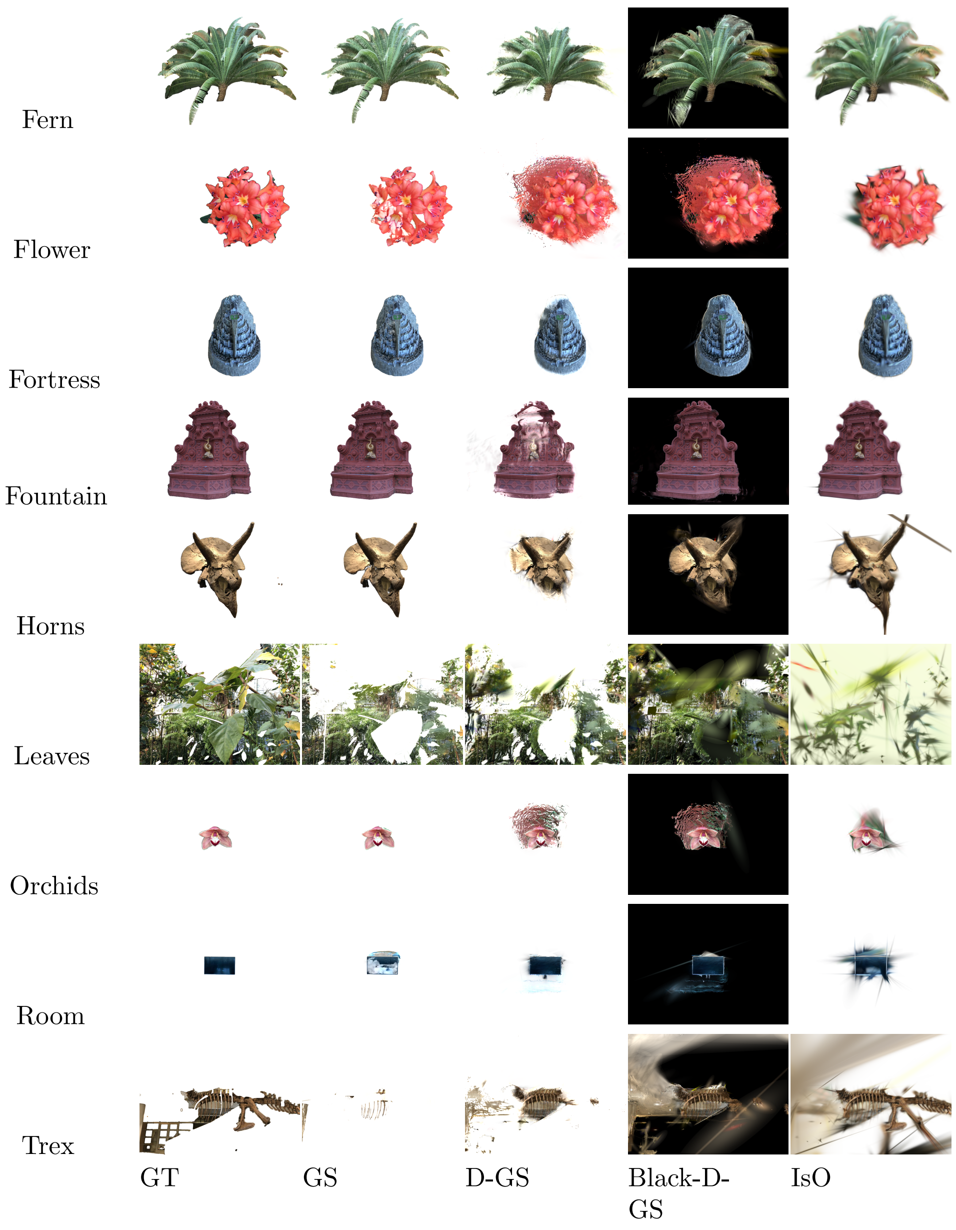

Figure 19: Renders from the different optimization methods. Ground truth (GT), Gaussian splatting (GS), depth-regularized Gaussian splatting (D-GS), result of depth-regularized Gaussian splatting rendered on a black background (Black-D-GS), isolated object splatting (IsO).

PSNR ↑		0.00	0.25	0.50	0.75	1.00	rb
Fern	D-GS	9.09	**16.36**	15.81	15.29	15.51	7.59
	IsO	10.91	16.12	16.74	17.05	**17.46**	9.84
Flower	D-GS	9.99	13.88	15.31	15.71	**15.88**	9.96
	IsO	8.11	17.20	17.70	18.17	**18.40**	8.43
Fortress	D-GS	17.10	23.86	**23.87**	23.81	23.60	16.36
	IsO	18.91	26.37	26.57	26.73	**26.87**	18.14
Fountain	D-GS	6.33	**17.12**	15.13	15.90	15.53	5.60
	IsO	6.09	22.62	24.36	25.51	**26.30**	5.78
Horns	D-GS	7.27	14.50	15.59	15.77	**15.97**	6.61
	IsO	4.26	15.22	16.72	17.02	**17.31**	5.67
Leaves	D-GS	11.67	11.56	10.90	9.96	8.92	**12.14**
	IsO	7.42	6.84	**11.15**	6.80	6.83	9.77*
Orchids	D-GS	10.49	14.68	16.07	16.78	**18.03**	9.08
	IsO	6.26	19.60	19.84	20.31	**20.66**	6.72
Room	D-GS	14.97	28.90	**31.14**	30.71	30.73	14.75
	IsO	8.10	19.08	20.65	21.62	**22.23**	5.85
Trex	D-GS	5.84	11.80	13.21	13.48	**13.61**	5.45
	IsO	6.57	9.38	10.26	10.63	**10.94**	6.15

Table 7: PSNR ↑ scores for the $\mathcal{L}_{bounds}$ ablations and the random background (rb). **Boldface** are the highest scores, red are the lowest scores, and orange are the second lowest scores. IsO Leaves rb (*) is evaluated at 7.000 iterations instead of 30.000 iterations. This method did not reach the 30.000 iterations, all Gaussians were culled.

20 shows the resulting renders for the Fountain scene. Similar to the other experiments, the Gaussians are trained on a black background, and verified on a white background. For the random background the Gaussians are trained on the random background color, this color is in the training also applied to the ground truth images, but still verified on a white background.

The data, see table 7 and 20, clearly shows that without the λ_{bounds} loss function ($\mathcal{L}_{bounds} = 0.00$) there is no control over over the Gaussians outside the centre scene object. Using a random color changing background for the optimization is also no compensation for the λ_{bounds} loss function. While the isolated object splatting (IsO) method prefers a high $\mathcal{L}_{bounds}$ value, the modified depth-regularized Gaussian splatting (D-GS) prefers a lower $\mathcal{L}_{bounds}$. This is most likely due to the Gaussians moving to the edge to get a correct edge at higher $\mathcal{L}_{bounds}$ values, and thus removing Gaussians from the centre. In the centre the Gaussians are then optimized in such a way that they still result in the correct color and depth, but now no longer for a completely solid object. When trained on a black background and verified on a white background this becomes visible.

6 Conclusion

Due to the quirks in the SFM data described in Sec. 3.1 Create Objects, it was not viable to accurately extract complete objects from the images. This

D-GS

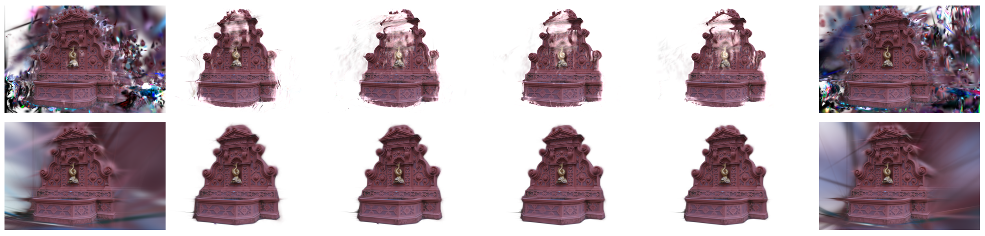

Figure 20: Fountain scene optimized under different $\mathcal{L}_{bounds}$ values, and random background (rb).

resulted in the focus shifting from extracting all objects to selecting only a single centre scene object. This also dropped the plan of combining multiple objects to reconstruct the entire scene. Although currently the conditions of the initial research question can not be satisfied, future improvements might still accomplish the initial goal.

When comparing the different Gaussian splatting methods it is visible that in most cases the original Gaussian splatting methods outperforms all other methods when optimizing a single object. Depth-regularized Gaussian splatting causes parts of the object to become opaque due to trying get the correct depth.

Looking at the individual sub problems:

1. *Given two segment masks $(S_a \in \mathcal{S}_i, S_b \in \mathcal{S}_j)|a \neq b$, is it possible to determine if they mask the same scene object?* This is limited by the SFM data. While it will never match segments masks that do not belong to the same object, it does miss segment masks that should be matched due to the quirks in the SFM data.

2. *Is it possible to filter the points belonging to a given object from the initial point cloud $\mathcal{X}$?* Yes, the point cloud $\mathcal{X}$ is generated by SFM and is also used in the creation of the objects, therefore selecting the points that belong to the object is already a part of the creation of the object itself.

3. *Is it possible to modify the learning process of Gaussian splatting such that it is capable of only training a single object and removing all the parts that do not belong to the object?* Yes, the modifications to the loss functions and the introduction of the new bounds loss λ_{bounds} allow Gaussian splatting to train individual objects. Although depending on the Gaussian splatting method used the trained objects are not always solid.

4. *Given multiple trained objects, is it possible to combine the objects in such a way that the entire scene is correctly reconstructed?* Due to the fact that only one object was selected for every scene, combination of multiple objects was never attempted. This leaves this question open for future research.

7 Future Work

7.1 Mask and Object Matching

Object isolation uses the SFM feature points X to both, combine masks, and create objects, by scoring them based on the number of matching features, see Eq. 8, 9. The scoring equation only counts features that are matched by SFM, due to the feature matching quirks, see Fig. 13, this can result in multiple seperate objects of the same scene object.

A possible way to improve this is by having a weighted scoring function that not only looks at the direct matched feature points, but also compares points (x_a, X_b) that are close to each other in the point cloud $\mathcal{X}$. Even though the features are not directly matched (Fig. 13 green and orange dots), because they are on the same position of the object, they should be triangulated close to each other in the point cloud $\mathcal{X}$. Doing this might improve the matching of masks and objects.

7.2 Object Combination

Currently due to the quirks in the SFM data and the limitations in the object isolation method, only a single, centre scene, object is selected for optimization. If the object isolation can be improved to extract multiple objects, it could be possible to train multiple individual objects, and then later combine them again into one full scene.

This would pose a series of new challenges, primarily maintaining the training and rendering speed advantage Gaussian splatting has. A possible way this might be done is by still sorting the Gaussians only once, and having a indexed boolean array for every object that, given the id of Gaussian g returns if this Gaussian is part of the currently trained object. This way the optimization of only sorting the Gaussians once is maintained.

7.3 Solid Objects

Currently there are no constraints to ensure that the object is optimized as a solid object, an additional loss function might mitigate this problem, and could therefore significantly improve the accuracy of the modified depth-regularized Gaussian splatting method.

A Structure Elements

0	1	1	1	0
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
0	1	1	1	0

The 5×5 structure element used for closing when extracting missing segments.

0	1	0
1	1	1
0	1	0

The 3×3 structure element used for the dilation when adding discarded segments.

B Segment Combination Algorithm

Algorithm 1 Mask combination algorithm
$masks : \mathcal{M} = \{(S, \mathcal{X})\}$
t_{mask}: threshold for feature matching

```
1:  function CombineMask($masks, t_{mask}$)
2:      $maskStack \leftarrow Sort(masks)$
3:      while $\#maskQueue > 0$ do
4:          $M_a \leftarrow maskStack.Pop()$
5:          $scores \leftarrow \{\ \}$
6:          for $M_b \in maskStack$ do
7:              $scores[M_b] \leftarrow Q(M_a, M_b)$
8:          end for
9:          $M_b \leftarrow Max(scores)$
10:         if $M_b > t_{mask}$ then
11:             $maskStack.Remove(M_b)$
12:             $masks.Remove(M_a)$
13:             $masks.Remove(M_b)$
14:             $M_a \leftarrow Combine(M_a, M_b)$
15:             $masks.Add(M_a)$
16:             $maskStack.Push(M_a)$
17:         end if
18:     end while
19: end function
```

C Object Creation Algorithms

Algorithm 2 Growing object creation algorithm
$masks : \mathcal{M} = \{(S, \mathcal{X})\}$
t_{object}: threshold for feature matching

```
function GrowingCreateObjects($masks, t_{object}$)
    $objectStack \leftarrow Sort(Object(masks))$
    $objects \leftarrow \{\ \}$
    while $\#objectStack > 0$ do
        $O_a \leftarrow objectStack.Pop()$
        $scores \leftarrow \{\ \}$
        for $O_b \in objectStack$ do
            $scores[O_b] \leftarrow Q(O_a, O_b)$
        end for
        $O_b \leftarrow Max(scores)$
        if $O_b > t_{object}$ then
            $objectStack.Remove(O_b)$
            $O_a \leftarrow Combine(O_a, O_b)$
            $objectStack.Push(O_a)$
        else
            $objects.Add(O_a)$
        end if
    end while
    return $objects$
end function
```

Algorithm 3 Non-growing object creation algorithm
$masks : \mathcal{M} = \{(S, \mathcal{X})\}$
t_{object}: threshold for feature matching

```
function NonGrowingCreateObjects($masks, t_{object}$)
    $objectStack \leftarrow Sort(Object(masks))$
    $combineObjects \leftarrow \{\ \}$
    $objects \leftarrow \{\ \}$
    while $\#objectStack > 0$ do
        $O_a \leftarrow objectStack.Pop()$
        for $O_b \in objectStack$ do
            if $O_b > t_{object}$ then
                $combineObjects.Add((O_a, O_b))$
            end if
        end for
    end while
    $objects \leftarrow CombineObjects(combineObjects)$
    return $objects$
end function
```

D Object Selection Algorithm

Algorithm 4 Object selection algorithm
objects : $\mathcal{O} = \{(\mathcal{S}, \mathcal{X})\}$
images : $\mathcal{I} = \{I\}$

```
1:  function ObjectSelection($objects, images$)
2:      $gatheredImages \leftarrow \{\ \}$
3:      $finalObject \leftarrow \{(\{\}, \{\})\}$
4:      while $\#gatheredImages < \#images$ do
5:          $centerSegments \leftarrow \{\ \}$
6:          for $O_a \in objects$ do
7:              $centers \leftarrow 0$
8:              for $S_i \in O_a.\mathcal{S}$ do
9:                                                  ▷ $I_i$ is the image $S_i$ came from
10:                 if $I_i \in gatheredImages$ then
11:                     continue
12:                 end if
13:                 if $CheckCenter(S_i, I_i)$ then
14:                     $centers \leftarrow centers + 1$
15:                 end if
16:             end for
17:             $centerSegments[O_a] \leftarrow centers$
18:         end for
19:         $O_a^*, maxCenters \leftarrow Max(centerSegments)$
20:         if $maxCenters = 0$ then
21:             break
22:         end if
23:         $meanSize \leftarrow MeanSize(O_a^*.\mathcal{S}, gatheredImages)$
24:         for $S_i \in O_a^*.\mathcal{S}$ do
25:             if $\#\{p \mid p \in S_i, S_i(p) > 0\} > meanSize/2$ then
26:                                                 ▷ $I_i$ is the image $S_i$ came from
27:                 $gatheredImages.Add(I_i)$
28:             end if
29:         end for
30:         $objects.Remove(O_a^*)$
31:         $finalObject \leftarrow Combine(finalObject, O_a^*)$
32:     end while
33:     return $finalObject$
34: end function
```

$MeanSize(O_a^*.\mathcal{S}, gatheredImages)$ returns the means size of the masks $S_i \in O_a^*.\mathcal{S}$ where the original image I_i is not in the *gatheredImages* set.

References

[1] Ben Mildenhall et al. "Nerf: Representing scenes as neural radiance fields for view synthesis". In: *Communications of the ACM* 65.1 (2021), pp. 99–106.

[2] Kyle Gao et al. "Nerf: Neural radiance field in 3d vision, a comprehensive review". In: *arXiv preprint arXiv:2210.00379* (2022).

[3] Bernhard Kerbl et al. "3D Gaussian Splatting for Real-Time Radiance Field Rendering". In: *ACM Transactions on Graphics* 42.4 (July 2023). URL: `https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/`.

[4] Jiazhong Cen et al. "Segment anything in 3d with nerfs". In: *arXiv prepring arXiv:2304.12308* (2023).

[5] Jiazhong Cen et al. "Segment any 3d gaussians". In: *arXiv preprint arXiv:2312.00860* (2023).

[6] Johannes L Schonberger and Jan-Michael Frahm. "Structure-from-motion revisited". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pp. 4104–4113.

[7] Shimon Ullman. "The interpretation of structure from motion". In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 203.1153 (1979), pp. 405–426.

[8] *Stereo and kinect fusion for continuous 3D reconstruction and visual odometry - Scientific Figure on ResearchGate.* accessed 1 Jun, 2024. URL: `https://www.researchgate.net/figure/Structure-from-Motion-SfM-process-is-illustrated-The-structure-in-the_fig2_269327935`.

[9] Ozgur Yilmaz and Fatih Karakus. "Stereo and kinect fusion for continuous 3D reconstruction and visual odometry". In: Nov. 2013, pp. 115–118. ISBN: 978-1-4799-3343-3. DOI: `10.1109/ICECCO.2013.6718242`.

[10] Sameer Agarwal et al. "Building rome in a day". In: *Communications of the ACM* 54.10 (2011), pp. 105–112.

[11] David G Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60 (2004), pp. 91–110.

[12] David G Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the seventh IEEE international conference on computer vision.* Vol. 2. Ieee. 1999, pp. 1150–1157.

[13] Wilhelm Burger and Mark J Burge. *Principles of Digital Imge Processing, Advanced Methods.* Springer, 2013. ISBN: 978-1-84882-918-3. DOI: `10.1007/978-1-84882-919-0`.

[14] Tinne Tuytelaars, Krystian Mikolajczyk, et al. "Local invariant feature detectors: a survey". In: *Foundations and trends® in computer graphics and vision* 3.3 (2008), pp. 177–280.

[15] Matthew Brown, Gang Hua, and Simon Winder. "Discriminative learning of local image descriptors". In: *IEEE transactions on pattern analysis and machine intelligence* 33.1 (2010), pp. 43–57.

[16] H Cantzler. "Random sample consensus (ransac)". In: *Institute for Perception, Action and Behaviour, Division of Informatics, University of Edinburgh* 3 (1981).

[17] Tao Hu et al. "Efficientnerf efficient neural radiance fields". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12902–12911.

[18] Stephan J Garbin et al. "Fastnerf: High-fidelity neural rendering at 200fps". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 14346–14355.

[19] Jonathan T Barron et al. "Mip-nerf 360: Unbounded anti-aliased neural radiance fields". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 5470–5479.

[20] Matthias Zwicker et al. "EWA volume splatting". In: *Proceedings Visualization, 2001. VIS'01.* IEEE. 2001, pp. 29–538.

[21] Volker Schönefeld. "Spherical harmonics". In: *Computer Graphics and Multimedia Group, Technical Note. RWTH Aachen University, Germany* (2005), p. 18.

[22] Sara Fridovich-Keil et al. "Plenoxels: Radiance fields without neural networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 5501–5510.

[23] Duane G Merrill and Andrew S Grimshaw. "Revisiting sorting for GPGPU stream architectures". In: *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*. 2010, pp. 545–546.

[24] Jaeyoung Chung, Jeongtaek Oh, and Kyoung Mu Lee. "Depth-Regularized Optimization for 3D Gaussian Splatting in Few-Shot Images". In: *arXiv preprint arXiv:2311.13398* (2024).

[25] Bingxin Ke et al. "Repurposing Diffusion-Based Image Generators for Monocular Depth Estimation". In: *arXiv preprint arXiv:2312.02145* (2023).

[26] Shariq Farooq Bhat et al. "Zoedepth: Zero-shot transfer by combining relative and metric depth". In: *arXiv preprint arXiv:2302.12288* (2023).

[27] John Canny. "A computational approach to edge detection". In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), pp. 679–698.

[28] In: *IEEE transactions on pattern analysis and machine intelligence* 32.9 (2009), pp. 1627–1645.

[29] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[30] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[31] Peiyuan Jiang et al. "A Review of Yolo algorithm developments". In: *Procedia Computer Science* 199 (2022), pp. 1066–1073.

[32] Shervin Minaee et al. "Image segmentation using deep learning: A survey". In: *IEEE transactions on pattern analysis and machine intelligence* 44.7 (2021), pp. 3523–3542.

[33] Alexander Kirillov et al. "Segment anything". In: *arXiv preprint arXiv:2304.02643* (2023).

[34] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. “Learning non-maximum suppression”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4507–4515.

[35] Wilhelm Burger and Mark J Burge. *Priniciples of Digital Image Processing, Fundamental Techniques*. Springer, 2009. ISBN: 978-1-84800-190-9. DOI: `10.1007/978-1-84800-191-6`.

[36] Wilhelm Burger and Mark J Burge. *Principles of Digital Imge Processing, Core Algorithms*. Springer, 2009. ISBN: 978-1-84800-194-7. DOI: `10.1007/978-1-84800-195-4`.

[37] Jaeyoung Chung, Jeongtaek Oh, and Kyoung Mu Lee. *DepthRegularizedGS*. URL: `https://github.com/robot0321/DepthRegularizedGS`.

[38] Vickie Ye and Angjoo Kanazawa. *Mathematical Supplement for the `gsplat` Library*. 2023. arXiv: `2312.02121 [cs.MS]`.

[39] Vickie Ye, Matias Turkulainen, and the Nerfstudio team. *gsplat*. URL: `https://github.com/nerfstudio-project/gsplat`.