Eyemantics - Eye Tracking Semantics

Wenqian Yang
ETH Zürich
yangwen@ethz.ch

Marino Müller
ETH Zürich
muemarin@ethz.ch

Joram Eickhoff
ETH Zürich
jeickhof@ethz.ch

Elisa Hoskovec
ETH Zürich
ehoskovec@ethz.ch

Abstract

This work is driven by the goal of extending the usability of advanced image segmentation methods through Mixed Reality (MR) technology for immersive and interactive user experience. We explore the integration of advanced 2D image segmentation techniques with the Magic Leap 2 (ML2) headset to achieve real-time 3D instance segmentation in mixed reality (MR) environments. The pipeline comprises image capture, gaze point analysis, image conversion, and projection, facilitated by a robust TCP communication channel between the ML2 and a processing PC. We employ the Segment Anything Model (SAM) to efficiently process segmentation tasks. Despite computational challenges, our system maintains a balance between speed, performance, and usability, as evidenced by user feedback and the mean System Usability Scale score of 82.20*. The study concludes with a demonstration of the practicality and user-friendliness of our MR segmentation system, while also identifying potential areas for enhancement in gaze accuracy, viewpoint offset, and mesh detail. This work not only showcases the feasibility of seamless 3D segmentation in MR but also opens up new avenues for immersive user experiences across various applications. The code is available on our GitHub repository* [1].

1. Introduction

Semantic Segmentation is a widely researched task in computer vision. The goal is to categorize each pixel in an image into a class or object to produce a dense pixel-wise segmentation map of an image. Recent developments, e.g. the *Segment Anything Model* by Kirillov *et al*. [3] improved the performance on zero-shot instance segmentation which allows obtaining the mask of an object in an image. However, these methods only generate segmentation masks of the 2D image without the possibility to display them in the real world. While methods for 3D image segmentation exist, they rely on the image data being fully available in 3D. Additionally, 3D information of most environments is not available, and obtaining it through Simultaneous Localization and Mapping (SLAM) or Dense 3D reconstruction is a computationally demanding task.

In the realm of mixed reality (MR) technologies, the evolution of head-mounted displays has been pivotal in shaping user experiences and interaction modalities. The introduction of the Magic Leap 2 headset marks a significant milestone in this continuum. A salient feature of the Magic Leap 2 is its expanded field of view, addressing a critical limitation that has historically impeded the immersive potential of MR devices. This enhancement not only enriches user immersion but also broadens the applicability of the headset in various professional and recreational contexts.

The pipeline proposed in this work aims at overcoming the domain gap and enabling 3D instance segmentation by combining advances in 2D image segmentation methods with capabilities offered by modern mixed reality headsets. The user's gaze point is used to prompt the segmentation model, thus the user can select, which object should be segmented by simply looking at it.

2. Related Work

A very important part of our project is the domain of image segmentation, particularly models that are promptable and can generate segmentations according to an input image and gaze point prompt.

Segment Anything Model (SAM) by Kirillov *et al*. [3] offers a promptable model pretrained on a broad segmentation dataset. Input prompts can be point vectors, boxes, or text, and they can be used to generate masks for all possible objects within an image. The paper's goal is to build a "foundation model for image segmentation". The SAM model consists of three components: an image encoder, a prompt encoder, and a mask decoder. The heavyweight image encoder outputs the image embeddings from a MAE pretrained Vision Transformer model. The prompt encoder encodes sparse (points, box, text) and dense (mask) inputs. The mask decoder maps the image and prompt embedding to a segmentation mask. For prompts that correspond to more than one object, SAM can output multiple masks with

[1] https://github.com/dbt-ethz/Eyemantics

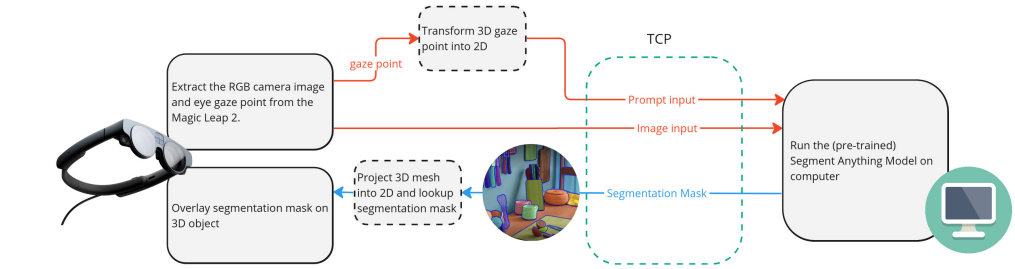


Figure 1. **Pipeline:** (1) Image and gaze point capture and projection (2) TCP transmission between the ML2 and the PC (3) SAM mask generation with the captured image and 2D gaze point (4) The mask is generated by processing the image and the gaze point with a lightweight SAM model (5) Mesh vertex coloring to get an overlay over the 3D object

associated confidence scores. A dataset of over 1 billion masks on 11 million images was built by using the SAM model in a data collection loop. The authors claim the model shows competitive zero-shot performance on several tasks compared to fully supervised results.

Fast Segment Anything by Zhao *et al*. [5] proposes a method that accelerates the same promptable segmentation task while maintaining comparable performance. This is done by simplifying the task to a conventional instance segmentation task. This method involves training a regular CNN detector with an instance segmentation branch on only 1/50 of the SA-1B dataset published by SAM [3]. In the paper, the claim is made that this approach has a 50× higher run-time speed while achieving comparable performance to the SAM model.

3. Method

The pipeline consists of five components split onto the ML2 device and a PC. The application on the ML2 device tracks the gaze point and captures the image before sending the data to the PC. A Python script on the PC uses SAM to segment the object the user is focusing on. The segmentation mask is then sent back to the ML2 and the mask is overlaid onto the real object. Figure 1 displays an overview of the whole pipeline. The following subsections outline the different components in detail.

3.1. Capturing Images

The primary phase in our methodology involves the on-demand capture of both image and gaze point using the Magic Leap 2 (ML2). This process is methodically segmented into several distinct steps:

1. Camera Preparation: This initial step involves activating the camera and configuring its settings for image capture. Key parameters such as output format, resolution, exposure, and white balance are meticulously adjusted. Notably, we utilize the computer vision camera (CV camera) for this task, ensuring that it operates independently on a separate thread from the RGB camera. The latter is primarily used for recording live demonstrations.

2. Image Capturing: The image capture function is mapped to the ML2 controller's press button. Upon activation, the pre-configured CV camera captures an image in the YUV format.

3. Image Conversion: Post-capture, the YUV image undergoes a multi-stage conversion process. Initially, it is transformed into a single RGB image. Subsequently, this image is converted into a byte array and finally serialized into JSON strings. This serialization facilitates the communication process.

4. Storage of Camera Parameters: The final step involves recording the current intrinsic and extrinsic parameters of the camera. This data is crucial for the subsequent step.

3.2. 3D to 2D projection

The gaze point is used as input for the image segmentation. The Magic Leap 2's built-in functionality to estimate the user's gaze point is used. The gaze point is obtained in world 3D coordinates and must therefore be projected into the image frame. For that, the camera's intrinsic and extrinsic parameters are used.

Given a 3D coordinate in the world frame $P^w = \begin{bmatrix} X^w & Y^w & Z^w & 1 \end{bmatrix}^T$ (homogeneous coordinates), it can be transformed into the local camera frame by using the camera's position and orientation

$$P^c = \begin{bmatrix} R & t \end{bmatrix} P^w \tag{1}$$

with R and t being the camera's rotation matrix and translation vector. The 3D point in camera frame can be projected in the 2D image frame via the intrinsic parameter matrix K:

$$p = KP^c \tag{2}$$

with $p = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T$ which is then transformed into Cartesian coordinates

$$(u, v) = (\frac{p_x}{p_z}, \frac{p_y}{p_z}) \tag{3}$$

to obtain the pixel coordinates u and v in the image frame. The captured image from section 3.1 and the projected gaze point is processed to extract the segmentation mask of the object at the user's gaze point.

3.3. TCP Connection

The Unity application on the ML2 and the Python script on the PC need to be connected to enable communication with each other. On the one hand, the Unity app needs to send the captured image with the corresponding eye-gazing coordinates. On the other hand, the Python script has to communicate the resulting mask back to the ML2. A Transmission Control Protocol (TCP) [4] is implemented to that end.

TCP is a communication standard and part of the transport layer of the TCP/IP suite. It ensures an end-to-end reliable protocol between a client and a server. The server creates a socket with its IP address and a given port number, where it listens for new clients. The client can now connect to this socket if the IP address and the given port number from the server are known.

In our implementation, the Unity app on the ML2 poses as the server, while the Python script on the PC tries to connect to it as a client. As mentioned above, the client needs to know the IP address and the port number from the server to establish a connection. Therefore, the user must provide this information. Since the average user is more familiar with typing on a PC than the ML2, we chose these roles to improve the user experience.

The whole communication pipeline between the ML2 and the PC works as follows: As soon as the image is captured on the ML2 the communication pipeline is triggered unless there is already one currently in progress. The image and its corresponding eye gaze coordinates are packaged as byte arrays and sent to the Python script on the PC. The bytes are then decoded and taken as input for SAM. After the resulting mask is generated, the true/false array is encoded into a byte array and sent back to the Unity script, where it gets decoded into a Boolean array again. With this step, the communication process is completed and ready for a new iteration. The communication between the two instances is stopped as soon as the Unity app on the ML2 is terminated.

3.4. Segment Anything Model

The goal is to process the captured image and gaze point from the Magic Leap device through the *Segment Anything Model (SAM)* [3], in order to return a generated segmentation mask to the device. Although we managed to get the model running both locally and on Google Colab, the process was constrained by timing challenges.

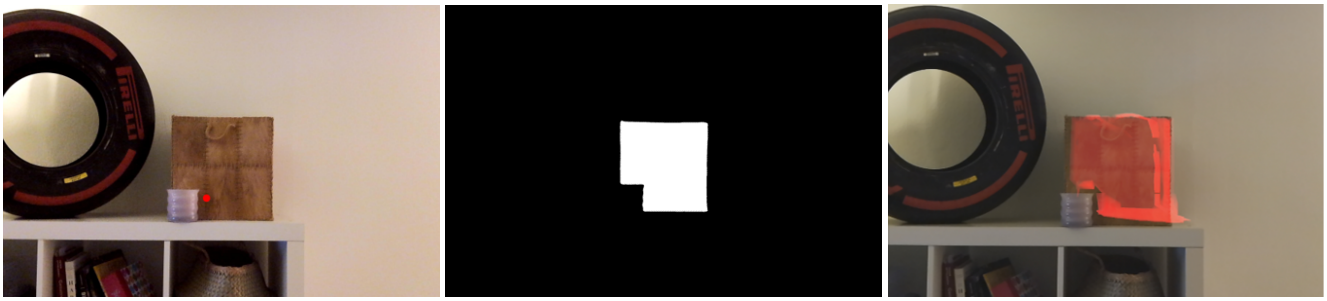
Depending on the model backbone sizes, a local segmentation run on a laptop with a newer CPU takes between 10 and 30 seconds. The duration on Colab is very inconsistent as it varies between 0.5 and 2.5 seconds depending on the GPU available. We get the best runtime results with a GPU, on which the smallest SAM model run takes between 1 and 1.5 seconds with only minor deviations. More about the runtime of the whole pipeline can be seen in Section 4.

Attempts to speed up the process with *Fast SAM* [5] do not result in a significant enough speed increase for both local CPUs/GPUs and the GPUs available on Colab. Other methods, such as resizing the image, introducing blur, and varying color information were tried, but with little success. Based on all attempts and our available hardware, a delay of over 1 second is unavoidable.

For the segmentation process itself, we run a custom Python script that receives the captured image and the gaze point from the ML2 device as a bytes array via a TCP connection, as seen in Section 3.3. After receiving the bytes array we transform it into a suitable RGB image *numpy* array with the help of the *OpenCV* library and we transform the gaze point coordinates into a *numpy* array as well. With these as inputs, we query the most lightweight *SAM* model available, whereby the output of subsequent multiple masks with confidence scores was also disabled. After segmenting is completed we prepare the resulting segmentation mask to be sent back to the ML2 device via the TCP connection (Section 3.3).

3.5. Display the Segmentation Mask

After receiving the segmentation mask via the TCP connection, it is displayed in the 3D environment by overlaying it on the segmented object. For that purpose, the meshing functionality of the Magic Leap 2 device is leveraged via the Meshing Subsystem. The device is capable of generating a 3D mesh representing the environment composed of vertex 3D coordinates. The segmentation mask is visually represented by altering the color of the mesh vertices that



(a) Captured image on the ML. (b) Mask generated by SAM. (c) Overlayed segmentation mask on the ML2.

Figure 2. From the captured image with the red gaze point to the 2D mask generated by SAM to an overlay on the real object.

fall within the segmentation mask after projecting the 3D vertex locations into the 2D image plane.

The device employs a Meshing Subsystem which utilizes a Time Of Flight (TOF) camera to map the environment. Due to the distance measurements of the TOF being noisy, the mesh is planarized and the number of vertices on planar surfaces is reduced to reduce computation overhead through unnecessary vertices. The Meshing Subsystem generates multiple meshes with unique mesh IDs. Interaction with these meshes, including their creation, updating, and deletion, is facilitated through function handles provided to the Meshing Subsystem. Accordingly, any changes to a mesh trigger an update in both the vertex colors and the comprehensive list of all meshes available. Thus, the mesh coloring is triggered if one of two events occur: creation or update of a mesh triggers the update of its vertex colors, and receiving a new mask triggers the update of the vertex colors for all available meshes.

The coloring of mesh vertices is determined by their corresponding positions in the 2D camera image. Vertices located within the segmentation mask are colored in a bright red, whereas those outside retain the mesh's base color, which is fully transparent (α = 0 in RGBA color space). The determination of a vertex's position in the 2D image is conducted through 3D to 2D projection, similar to the projection method used for the gaze point in section 3.2. However, due to the computational delay inherent in evaluating the SAM and the latency in image data transmission via the TCP protocol, discrepancies in the viewpoint can occur between the moments of image capture and mask reception due to movements by the user. To mitigate this, the intrinsic and extrinsic camera parameters at the time of image capture are stored and subsequently utilized during the projection of mesh vertices into the 2D camera image for coloring the mesh vertices.

For rendering purposes, the "URP Particles Lit" shader is employed for the mesh material, owing to its capability for α pass-through and vertex-based coloring. The application of mesh color occurs during the creation or updating of a mesh, or for all meshes when a new mask is received. For this, a constantly updated list of all meshes is maintained.

This methodology ensures that the segmentation mask is accurately overlaid on the segmented 3D object, allowing for considerable viewpoint variations during the examination of the segmentation mask.

4. Results

In the presented application, we seamlessly incorporate the components described in section 3 into a fully functional pipeline. Figure 2 visualizes the different stages, from capturing the image on the ML2 to the red segmentation mask overlaying the object.

The pipeline can be triggered arbitrarily many times as long as the ML2 and the PC stay connected. As mentioned in Subsection 3.4, SAM can require a considerable amount of time. However, with the use of a GPU, the overhead can be reduced significantly to make the segmentation as instant as possible. Figure 3 displays the runtime of one iteration of the whole pipeline, from the image capture to the overlaid mask. It is shown, that a run on a laptop with a modern CPU (Intel Core i7) takes between 26 and 30 seconds, while the use of a GPU (NVIDIA GeForce RTX 2060) results in a much lower runtime of around 1.7 seconds with only minor fluctuations.

5. Ablation

To evaluate the user experience of our app, we use a quantitative user study. To that end, we asked 17 users to fill out a questionnaire provided by Brooke [2]. The questionnaire provides ten statements about the usability of the system, which the user can rate on a 5-point scale. Choosing one is equivalent to strongly disagreeing and five to strongly agreeing.

The results of the individual questions suggest that most users were able to navigate through the app without too many problems – on average, they found the system easy and not cumbersome to use, not too complex, learned how to use it quickly, did not require prior knowledge and felt confident using the system. However, six out of the 17

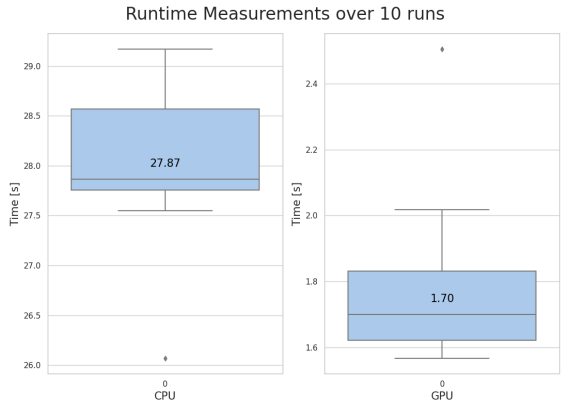


Figure 3. Runtime comparison of the whole pipeline using a CPU (Intel(R) Core(TM) i7-7600U CPU) and a GPU(NVIDIA GeForce RTX 2060).

users, were not certain if they might need help from a technical person and one user agreed with needing help to navigate in the app. Furthermore, the average user did not experience too many inconsistencies in the system and thinks the features are well integrated. Finally, seven users would use the system frequently, three users disclosed that they would not, while the seven remaining users are not certain.

However, it is important to note, that the evaluation of the single questions is not meaningful as Brooke [2] states, but one should rather consider the resulting System Usability Scale score. The SUS score, which can range from 0 to 100, is calculated as mentioned by Brooke [2] and combines the evaluation of all ten questions into a single score. The evaluation of the questionnaires shows a SUS mean score of approximately 82.20. The corresponding boxplot of the SUS scores can be found in Fig 4 for a more detailed evaluation of the SUS scores. Based on the evaluation provided by Bangor, Kortum, and Miller [1], the score suggests an *excellent* user-friendliness.

6. Discussion

A pipeline capable of gaze point-based image segmentation and mask overlay with *excellent* user-friendliness has been presented. While the pipeline is fully functional and creating accurate results, multiple approaches for improvement are conceivable.

First, points for improvement of the current application were identified by the fact that six out of 17 users were uncertain about needing help from a technician. This could be due to the design decision that the mesh rendering can be toggled by the press of a button. Therefore, the users had to press two buttons instead of one, posing a challenge for inexperienced users. This feature was expected to be helpful but the user study indicated otherwise. Additionally, a detailed analysis of the user study reveals that the users are

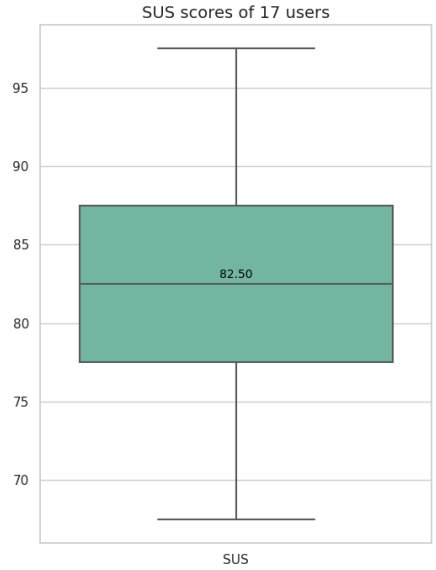


Figure 4. Evaluation of the SUS score provided by 17 users. The median, as shown in the figure, is 82.5. The minimal SUS score is 67.5 and the maximal SUS score is 97.5.

split on whether the system is overall useful with multiple users reporting that they would not use the app. This could be due to the fact that the questioned users are not part of the target audience. Potential use cases are conceivable for people with low vision where the system helps visualize the exact boundaries of an object through overlaying the bright red mask. However, the app in its current form offers only limited assistance for people with normal vision. Extending the pipeline could enable further use cases, e.g. by prompting the SAM with object classes and thus visualizing all objects of that class in the 3D environment.

Second, the pipeline relies on being triggered by the user due to long computation times and unreliable gaze point information. Continuously running the pipeline by iteratively processing a new image once the previous one has been segmented would be desirable. However, it was found that due to unreliable gaze point measurements, especially when contact lenses or glasses are worn, there is a high probability that the projected gaze point position is not inside the object the user was actually looking at. In combination with long calculation times, it was found to be an uncomfortable user experience to have the mask overlaid on the wrong object. Therefore, it was decided to trigger the pipeline manually. Filtering the gaze point was evaluated by applying median, average, and low-pass filters which did not improve the gaze point measurement significantly. Fine-tuning the filters could potentially improve the gaze point measure-

ment.

Third, it was observed due to the camera being at a slightly different position than the user's eyes, a change in viewpoint between the camera and the user exists. This is translated into the overlaid masks not being perceived to be exactly on top of the segmented object. This was fixed by adding a rotation offset of 2° around the device's x-axis. The amount of offset was found heuristically and alleviated the problem. However, a more rigorous approach to compensate for the viewpoint offset would be desirable.

Fourth, since the mask is visualized by coloring the vertices of the meshes, the amount of detail is limited by the detail and quality of the mesh. The mesh is created by a TOF camera which limits the detail to a maximum of about 4cm. Additionally, black and specular surfaces pose a significant challenge for the Magic Leap 2, resulting in incomplete meshes in these situations. Ideas to improve the level of detail of the overlaid mask include using 2D-3D projection of the segmentation mask given the camera extrinsic at the time of capture. Using the inverse camera transform, a 3D position of every mask pixel could be calculated. However, this requires knowledge of the depth relative to the camera which is not available in case of specular or black surfaces. Additionally, projecting every pixel of the segmentation mask into 3D poses a significant computational challenge due to the high pixel count of the image of 2880px by 2160px. Two approaches to still use the 2D-3D projection for mesh refinements are feasible: only project the edges of the mask to obtain a sharper boundary of the mesh being displayed and only project the full mask into 3D if a significantly sized mask was received but (almost) no edge vertices are colored, indicating an incomplete mesh. In the latter approach, the distance to the camera of the closest vertex could serve as a rough estimate for the projection.

Finally, only the last received mask is used to color the mesh. Thus, only one segmentation mask is visualized at a time. This was done so that the user's view is not unnecessarily occluded. However, the pipeline could be easily adapted to allow the simultaneous visualization of multiple segmentation masks.

7. Conclusion

This work demonstrates a novel approach in 3D instance segmentation using the Magic Leap 2 (ML2) headset, combining advanced 2D image segmentation techniques with the MR capabilities of ML2. An accurate display of the segmentation mask was achieved even for large viewpoint changes. By focusing on the user's gaze as a prompt for real-time segmentation, we have developed a seamless integration of image capturing, processing, and projection in an MR environment.

The study identifies areas for improvement, including gaze point accuracy, viewpoint offset compensation, and mesh detail enhancement. Addressing these challenges will be crucial for the further advancement of MR technology.

In summary, our research represents a novel contribution by combining advanced 2D image segmentation method with Mixed Reality, closing the domain gap and elevating 2D segmentation masks into the 3D world.

References

[1] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009. 5

[2] John Brooke. Sus: a "quick and dirty'usability. *Usability evaluation in industry*, 189(3):189–194, 1996. 4, 5

[3] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023. 1, 2, 3

[4] Jon Postel. Transmission control protocol. Technical report, 1981. 3

[5] Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast segment anything. *arXiv preprint arXiv:2306.12156*, 2023. 2, 3