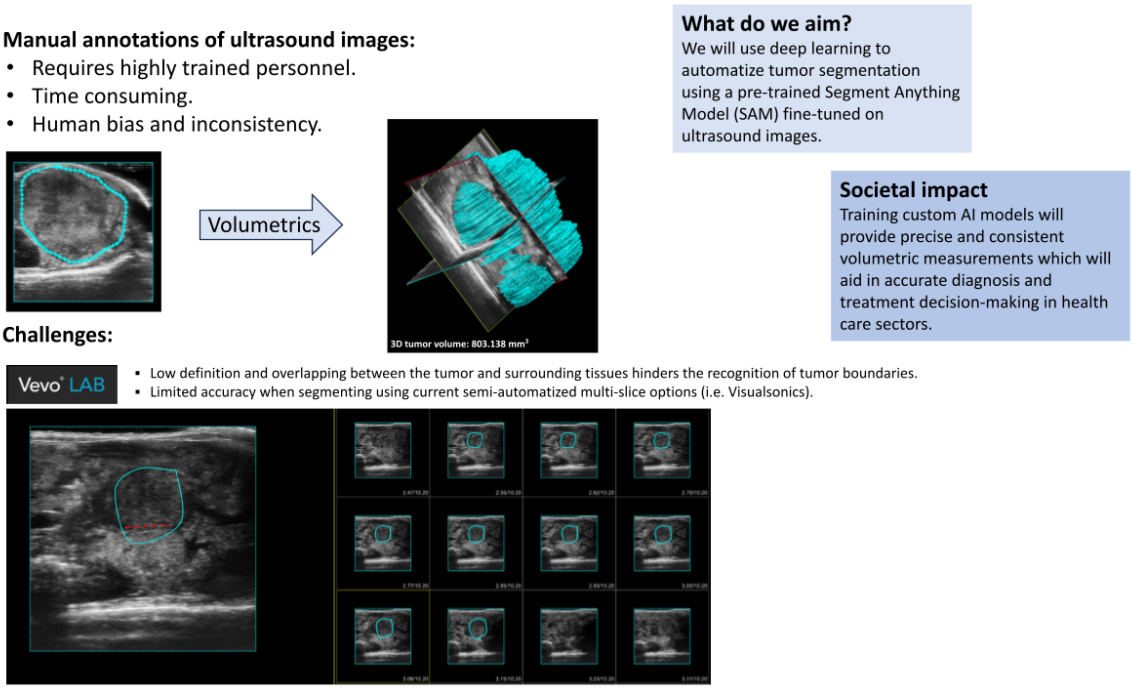
ParaSAM

Deep learning in Medical images for tumor segmentation and volumetrics: From human bias to AI precision



Abstract

Manual annotation in medical imaging faces significant challenges including susceptibility to human bias, time consumption, and inaccurate image interpretation either in research and healthcare settings.

In the OpenCV AI Contest 2023, our project "ParaSAM" introduces a groundbreaking approach for tumor segmentation and volumetrics in medical images, leveraging deep learning to transcend human biases inherent in manual annotation. This collaborative effort with Umeå University and Daniel Öhlund Lab focuses on pancreatic cancer research, utilizing an extensive dataset of 3D ultrasound images from a KPC mouse model. ParaSAM builds upon the MedSAM model, refining it for enhanced accuracy in automatic detection and segmentation of pancreatic tumors. We developed an interactive 3D application using Unity and OpenCV for importing ultrasound images, performing automatic segmentation, and visualizing tumors in 3D. Our preliminary results demonstrate significant improvements in tumor detection and volumetric analysis over conventional methods, marking a crucial step towards precision oncology diagnostics.

1. Summary

ParaSAM *(from the Greek prefix Para- meaning 'Beyond' and SAM from 'Segment Anything Model')* is an advanced tool designed for precise segmentation and volumetric analysis of tumors in medical images, developed using OpenCV and Unity. As a significant evolution and refinement of the "SAM" model, ParaSAM specifically addresses the challenges in detecting and analyzing pancreatic tumors.

In collaboration with Daniel Öhlund's Lab (Umeå University), this project aims to:

- Enhance the accuracy of tumor segmentation in ultrasound images, moving beyond the capabilities of the traditional SAM and its medical adaptation, MedSAM.
- Automate the tumor annotation process, reducing the time and effort involved in manual annotations.
- Develop a comprehensive workflow using OpenCV and Unity, encompassing data preprocessing, automated capture and extraction of annotations, dataset generation, and advanced 3D visualization techniques.
- Throughout the three-month contest period, we have focused on annotating a vast number of tumors and developing a Proof of Concept application. This application demonstrated the entire workflow, including the training of a MedSam refinement, post-processing of inference results, and advanced visualization in 3D and mixed reality environments.

Preliminary results (detailed in the later section) confirm that ParaSAM significantly improves upon the segmentation and volumetric analysis capabilities of the original SAM and MedSAM models, marking a substantial advancement in medical imaging technology.

Immediate goals following the success of this Proof of Concept (PoC) developed for the contest, is to create a tool for laboratory use which will be available for the scientific community. In the long-term, this tool holds the potential to set the ground for developing optimized AI models for clinical and diagnostic use in healthcare sectors. The source code repository will be publicly opened in line with MedSAM's philosophy and results will be published in a high-impact scientific journal.

Deliverables:
Video: https://youtu.be/qNulJCA25TQ

Public repository: https://github.com/gespona/ParaSAM

Team Members:
Margarita Espona-Fiedler
Gerard Espona Fiedler
Daniel Öhlund (in an advisory capacity)

2. Introduction

Research at Umeå University and Öhlund Lab

Annotated ultrasound images are essential in research and healthcare for monitoring tumor growth and aiding diagnostics. However, manual annotations, which require specialized training and detailed pixel-by-pixel identification of regions of interest, are labor-intensive, time-consuming, and susceptible to human bias. This is particularly challenging in pancreatic cancer where tumor areas are often poorly defined.

At Öhlund lab, ultrasound images have been acquired from a KPC mouse model expressing Pdx1-Cre, a pancreas-specific promoter, to drive conditional mutations in KRAS and TP53 (Kras+/G12D Tp53+/R172H) involved in the development of pancreatic cancer. Tumor growth from 57 tumors was monitored over time and images from a total of 217 3D ultrasound scans were manually annotated using VEVOLab software (Visualsonics) obtaining a total of 32,550 annotated images with their respective volume estimations. After necropsy, tumor dimensions were measured using a caliper and the volume was calculated using the formula V=(W^2 x L)/2 evidencing a second major challenge: the underestimation of real tumor volumes in ultrasound images.

Commercially available softwares like VEVOLab offers multi-slice views of the 3D scans and semi-automated options which facilitates the segmentation; nonetheless, this is often imprecise and manual re-adjustments of the selected areas are required which is laborious and time-consuming taking into account that each 3D scan might contain around 100 images per tumor. To address these limitations, we propose using deep learning to fully automate tumor segmentation. Our approach involves refining the pre-trained Segment Anything Model (SAM), MedSAM, tailored for ultrasound images. We are developing a custom AI model capable of processing large numbers of images efficiently and accurately predicting volumetric measurements as part of the proof-of-concept (PoC) stage. To this aim, we have used OpenCV and Unity (for 3D visualization) since these allow both automated detection and adjusted annotations, reconstruction 3D of the tumor, volumetric calculations, and further comparison with imported/extracted manual annotations from VEVO labs software.

Introduction to SAM and MedSAM

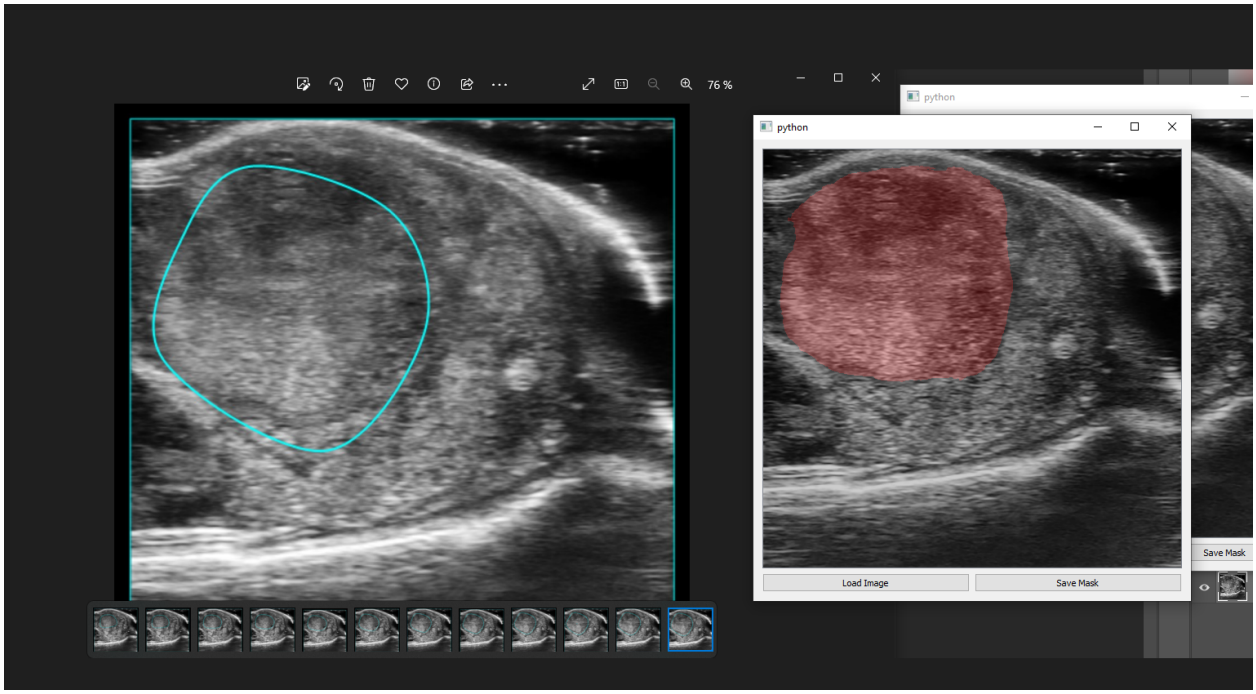
MedSAM Overview

- MedSAM, the first to adapt SAM for medical imaging, aims to create a universal segmentation tool (1, 2). It involves a large dataset with over 200,000 masks across 11 modalities.
- The technical approach uses a transformer-based architecture with a vision transformer-based image encoder and focuses on fine-tuning the mask decoder.
- Dataset and Training: MedSAM uses 33 segmentation tasks across various modalities. The images were normalized and resized for stable training, with an 80/20 training/testing split.
- Performance: MedSAM significantly improves upon SAM, especially in identifying small objects and handling weak boundaries.
- Limitations and Future Directions: While MedSAM surpasses SAM in several aspects, it still trails specialized models in areas such as boundary consensus. Future enhancements could involve larger models and expanded datasets.

Conclusion: MedSAM's advancements demonstrate the potential for building comprehensive segmentation foundation models in medical imaging, paving the way for more nuanced and effective segmentation methods in this domain.

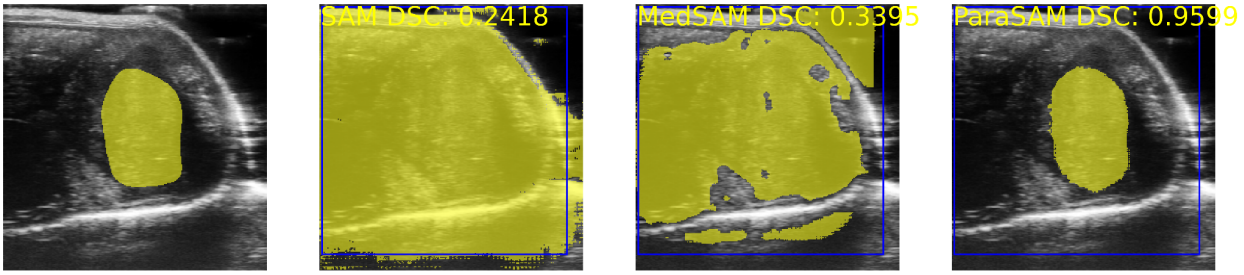
ParaSAM Model

- Following MedSAM's philosophy, ParaSAM builds on MedSAM's work by freezing the embedding and prompt portions and refining the mask decoder with our images and annotations.
- Prompt: We continue using a 2D bounding box as the prompt. We are exploring semi-automatic annotation, where users select the tumor region with a bounding box, and automatic annotation, where the bounding box covers the entire image, and the model is trained to isolate the tumor.



Semi-automatic annotation using vainilla MedSAM with tight user bounding box

The subsequent image demonstrates that while MedSAM improves over SAM in medical image segmentation, the results on our ultrasound images are not optimal, especially on full bounding box.

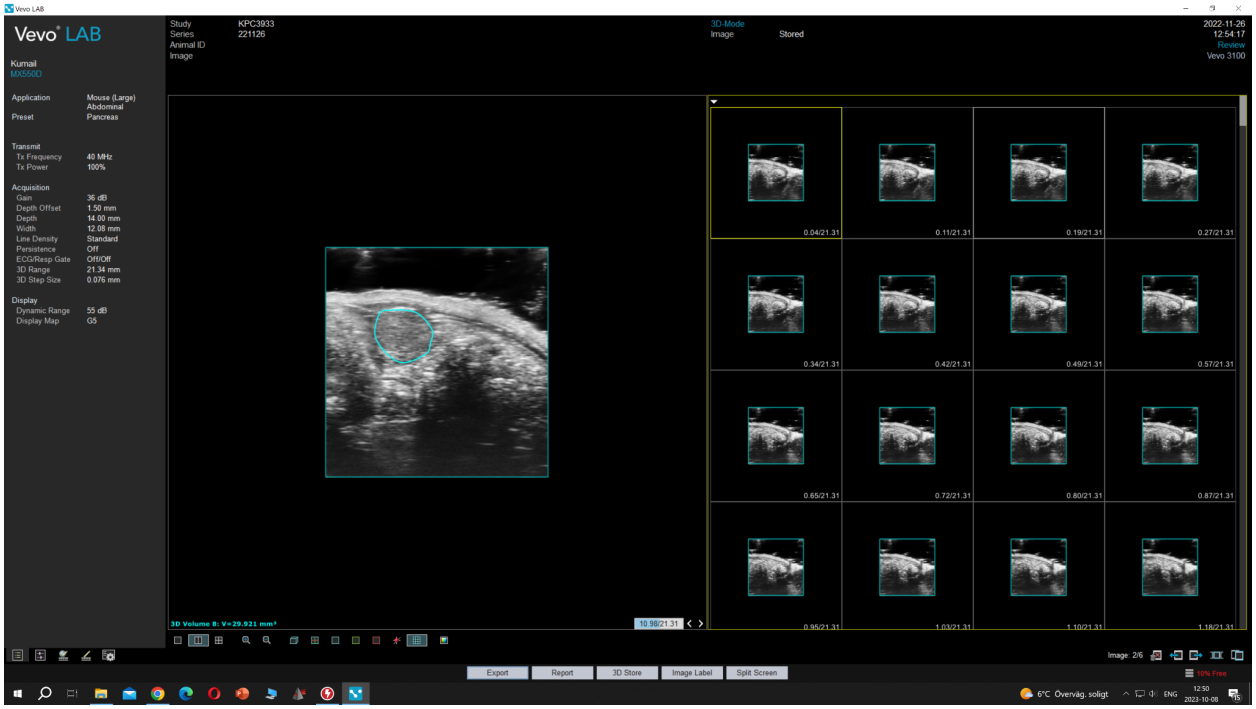


Manual annotation (GT), original SAM inference, MedSAM inference, ParaSAM inference

3. Project Development

Data Collection:

At Öhlund lab, we have acquired and manually annotated 3D ultrasound images from pancreatic tumors developed in KPC mouse models using VEVO labs software. Present report includes an initial dataset of 8 sequences/tumors, comprising 517 annotated images. We designated 80% of these images for training and 20% for validation. To ensure optimal model generalization, we are currently training with the full set of over 32,000+ images.



Example of 3D Ultrasound sequence manually annotated with Vevo LAB

Notice: We provide a .zip file under Releases (https://github.com/gespona/ParaSAM/releases) with Windows binaries for preprocess and 3D reconstruction with Data folder generated. So don't really need to save any preprocess or do any training to evaluate the work here.
That said, following instructions includes detailed information about OpenCV/Unity and python projects that allows any user to go through the whole process, using binaries or Unity project itself (paid third-party assets needed). For training you need to follow instructions to set up python env.

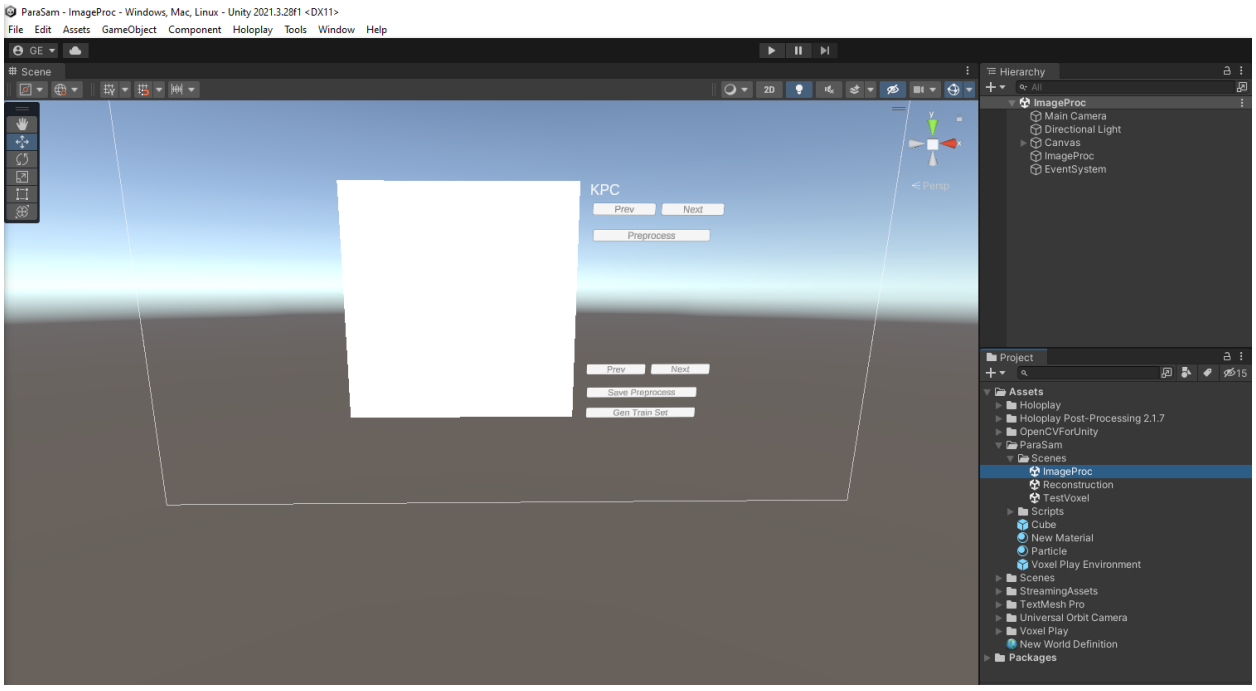
For installation instructions, please follow repository README:
https://github.com/gespona/ParaSAM/tree/master

Pre-processing:

Using OpenCV/Unity, we load original images and annotation captures. Through OpenCV, we process annotation images to detect the image frame of interest and then the VEVO Labs annotation, which is a polyline. This polyline is converted into a binary mask, and images and masks are resized according to MedSAM's input requirements. The final output is a directory structure ready for MedSAM's pre-processing.

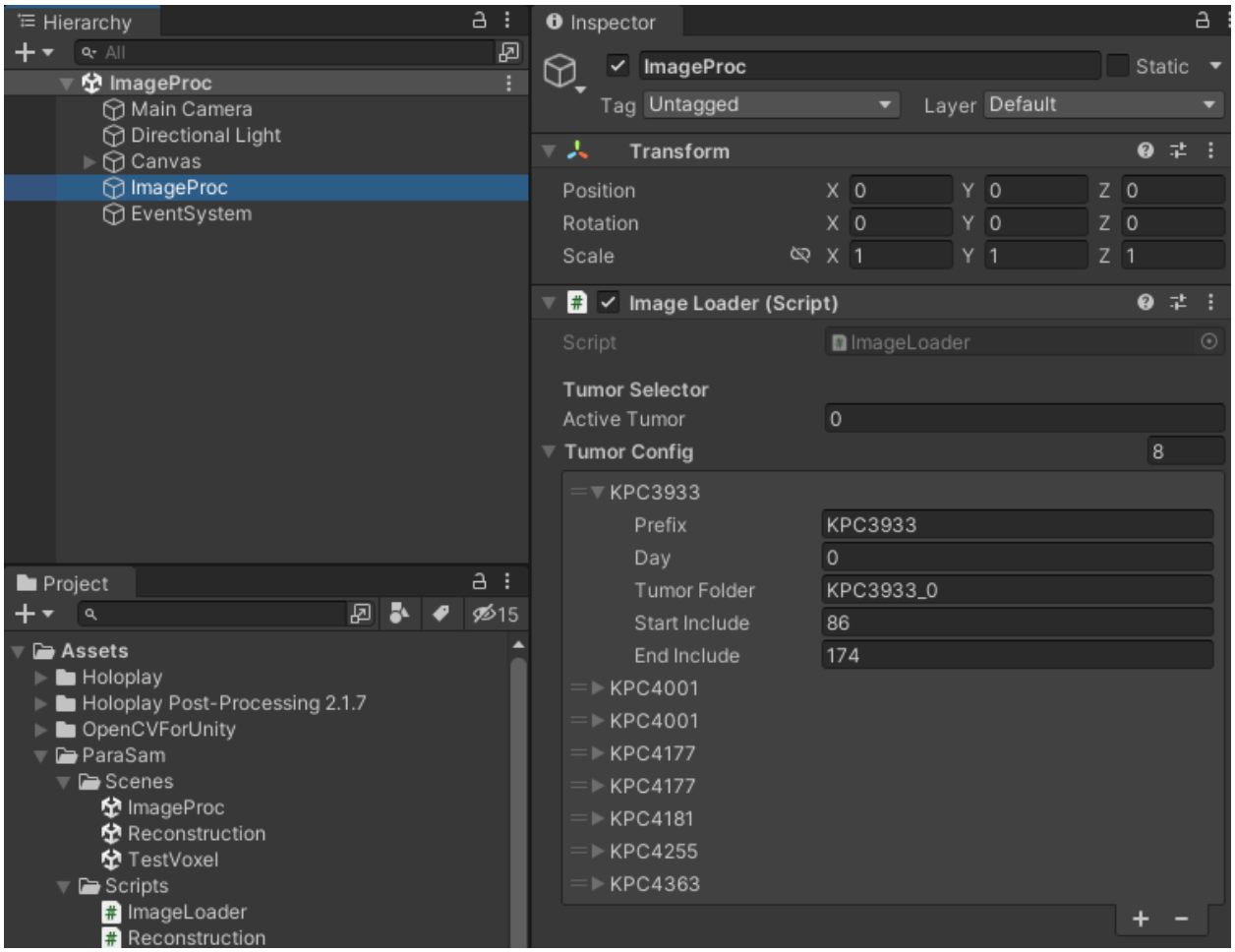
More in detail, the OpenCV/Unity application consists of two parts (scenes). The scene/binary named 'ImagePreproc' is responsible for:

- Loading the original images and annotation captures.
- Using the OpenCV library to process the annotation images, first detecting the frame of the image of interest, and then detecting the annotation made in VEVO Labs, which is a polyline.
- Again using OpenCV, we detect the contours and convert the polyline into a binary mask.
- Finally, we standardize the resolution of the images and masks according to the input size required by MedSAM.
- Once the images are generated, we can also ask the application to divide each sequence into a training and validation dataset (in this case, we parameterize the split at 80%-20%).



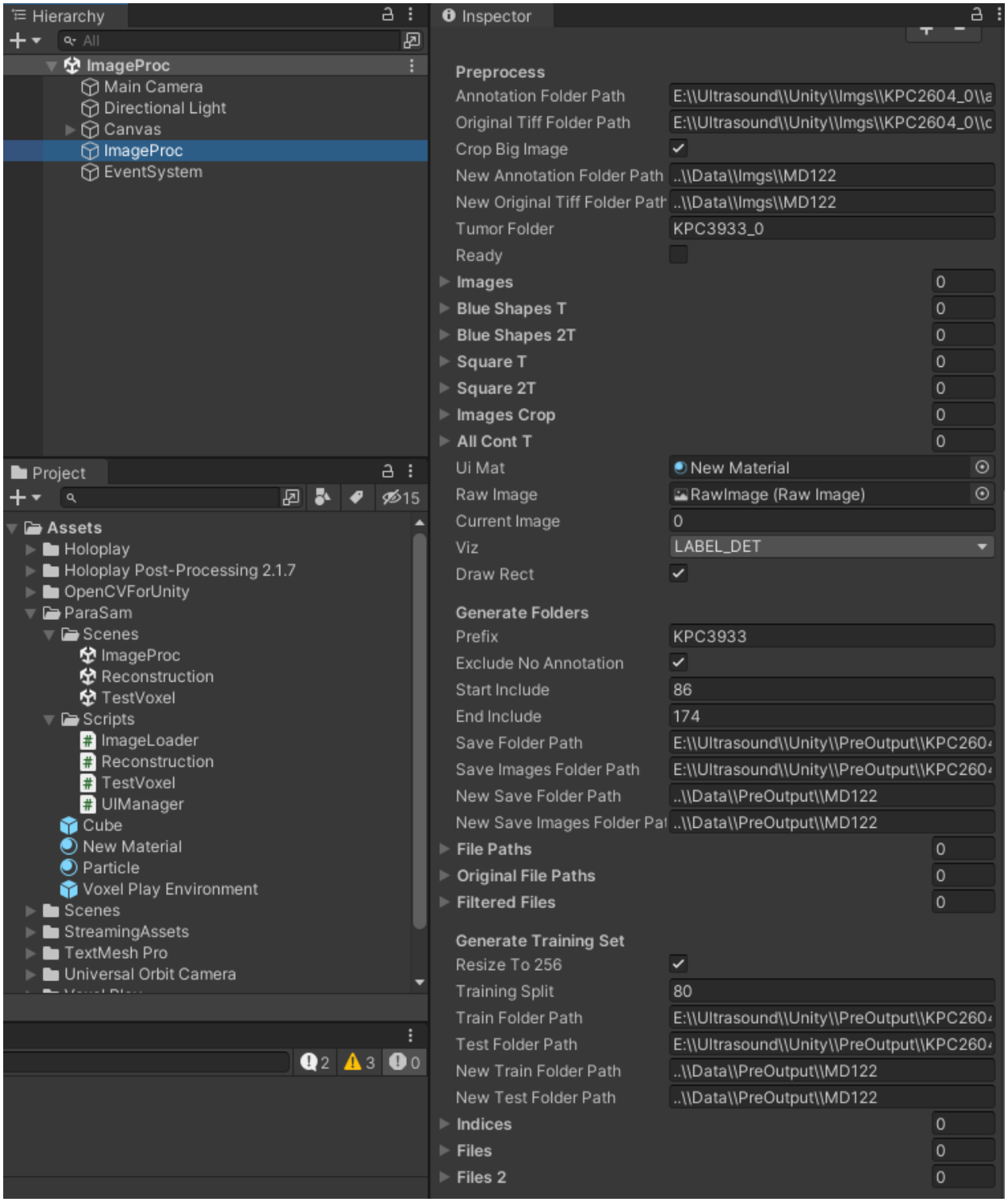
OpenCV / Unity application for preprocessing VEVO Lab captures

The main work is done by C# script called ImageLoader.cs
This script you can find attached in the scene to ImageProc gameObject, you can find tumor configuration section:



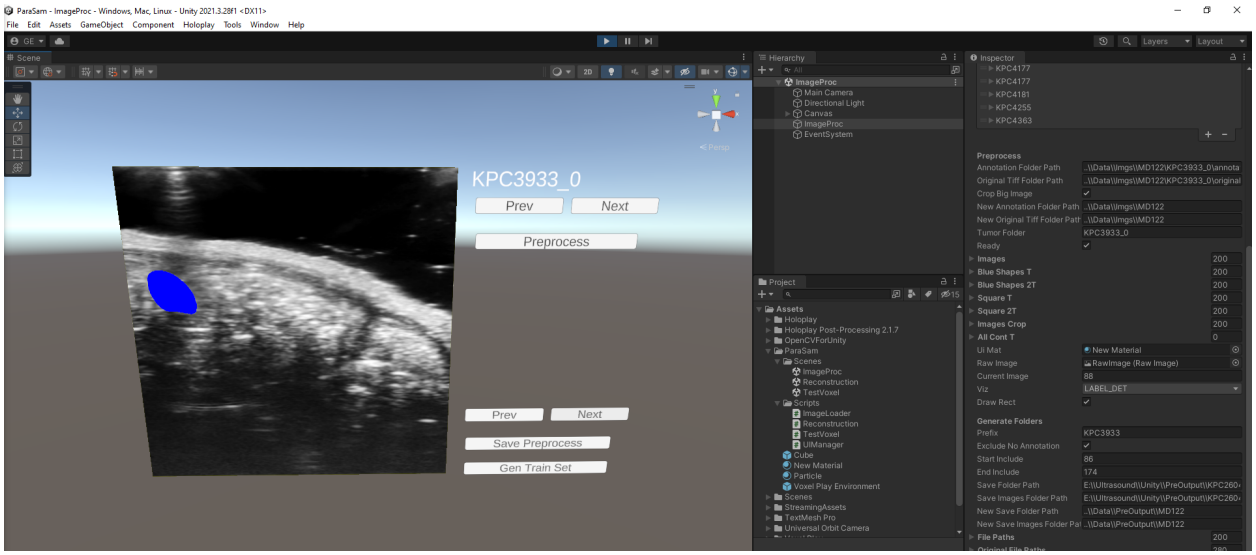
Tumor configuration

Preprocess and dataset generation:

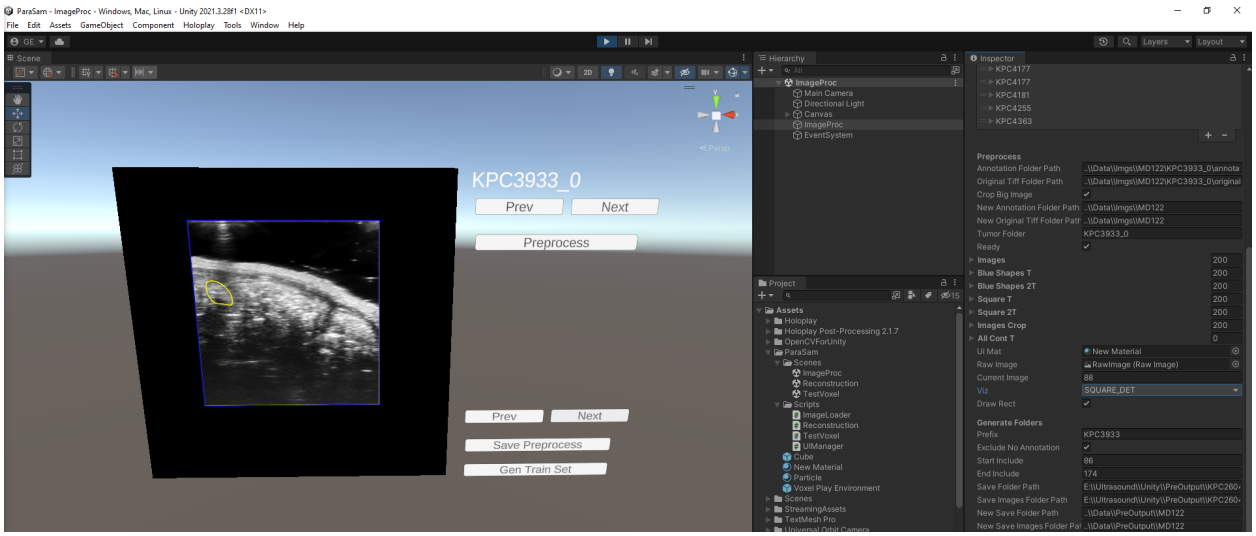


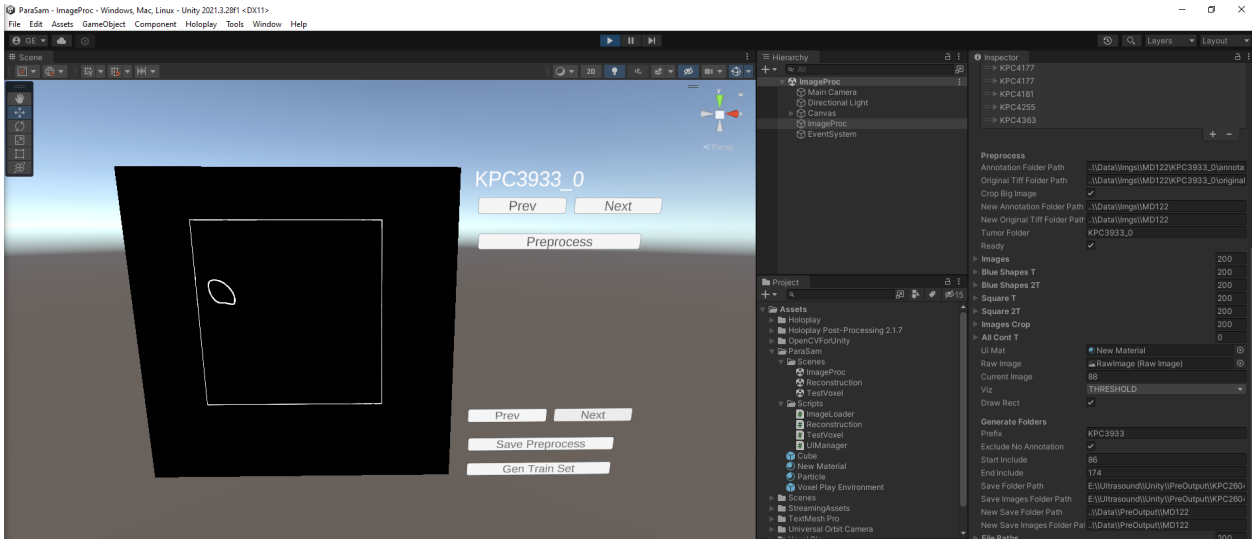
With the UI you can move through the tumor sequences, each image inside the sequence, save the preprocessing and split in train/validation dataset.

Load and preprocess could take several minutes.



You can visualize/debug the different stages of the preprocessing using the dropdown Viz in the inspector





The main code is OpenCV image processing:

For each image in the sequence, we perform color threshold to detect manual annotation:

```
Mat hsv = new Mat();
Imgproc.cvtColor(img, hsv, Imgproc.COLOR_BGR2HSV);
Scalar lowerBlue = new Scalar(0, 100, 100);
Scalar upperBlue = new Scalar(100, 255, 255);
Mat blueMask = new Mat();
Core.inRange(hsv, lowerBlue, upperBlue, blueMask);
blueShapes.Add(blueMask);
```

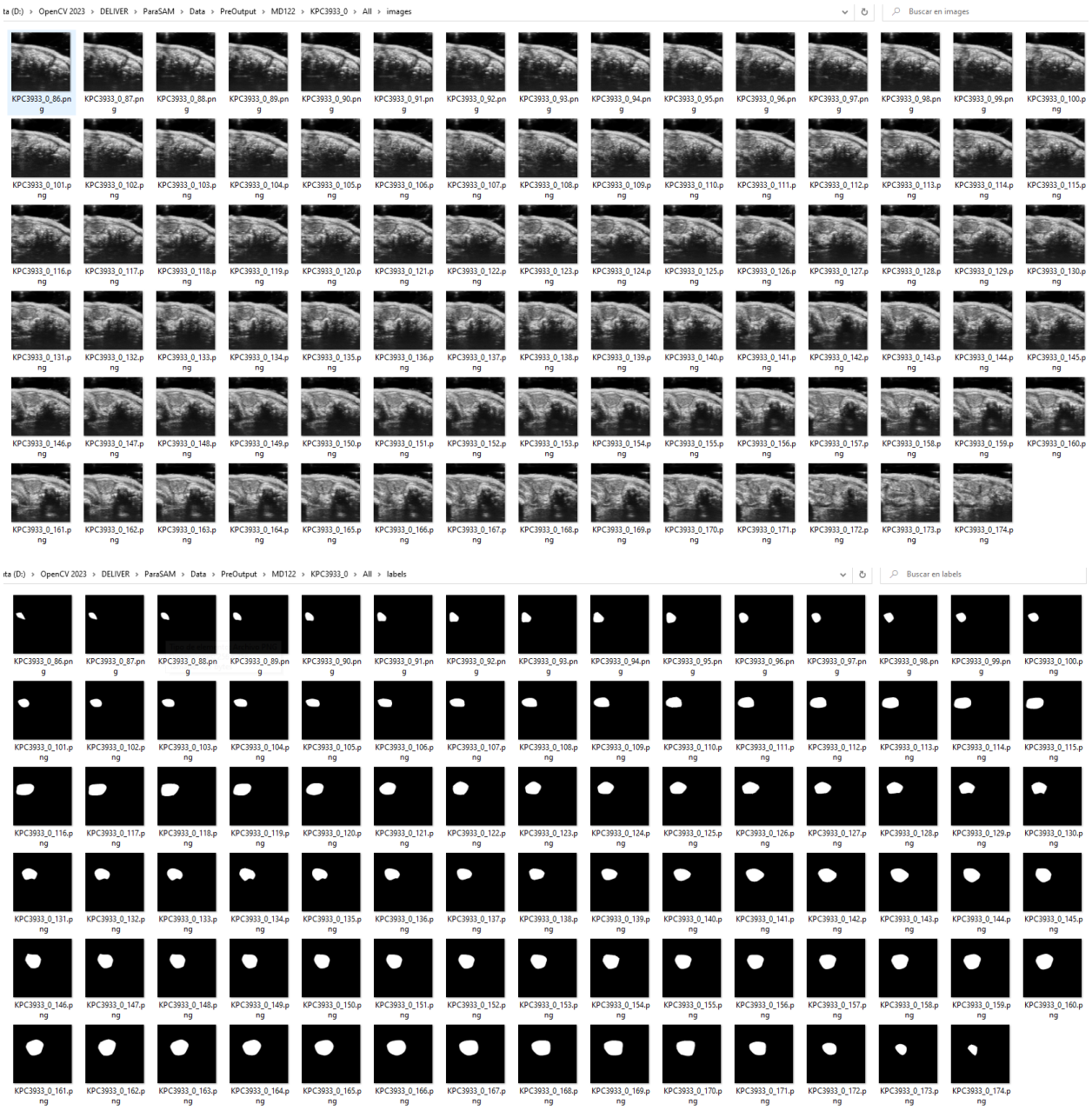
We extract the contours:

```
List<Rect> blueSquareRects = new List<Rect>();
List<MatOfPoint> contours = new List<MatOfPoint>();
Mat hierarchy = new Mat();
Imgproc.findContours(blueMask, contours, hierarchy, Imgproc.RETR_EXTERNAL,
Imgproc.CHAIN_APPROX_SIMPLE);
```

And we generate the mask:

```
List<MatOfPoint> contourList = new List<MatOfPoint> { contour };
Scalar color = new Scalar(255, 0, 0); // Blue color
int thickness = 2;
Imgproc.drawContours(imgCopy, contourList, -1, color, thickness);
```

Results can be found on any Data\PreOutput folder (either on release .zip file or Data folder in repo). For each sequence you can find All, Train and Test folders with images and masks.



ParaSAM Model Training:

To train the ParaSAM model, we'll fine-tune MedSAM with our ultrasound images and the manual tumor annotations from the Öhlund lab. In other words, we'll freeze image encoder embeddings, prompt and we'll focus on train mask decoder.

To this aim, we followed the instructions from the MedSAM repository, where we found materials for refining models.

To facilitate both training and inference, as well as mask extraction, we created a Python script, which we currently run in a conda environment. In the future, our goal is to integrate Python calls into the Unity application and, if possible, execute especially the inference part within Unity in real-time using either the OpenCV library (DNN) or the new Unity Sentis (an evolution of Barracuda), a framework that allows running ONNX models within Unity. In both cases, it requires model transformation from pytorch but should be doable.

The first step is to prepare the dataset, performing preprocessing to normalize the images, calculate the image embeddings (encoder) using the MedSAM model, and create a .npz file with all the information for subsequent training:

```
python .\pre_grey_rgb2D.py -i .\data\KPCMIX1\Train\images\ -gt
.\data\KPCMIX1\Train\labels\ -o .\data\KPCMIX1 --data_name kpcmix1
--checkpoint .\work_dir\MedSAM\medsam_vit_b.pth
```

Then we use our Python script for training.

```
python .\parasam.py --mode train --compare_models --checkpoint
.\work_dir\MedSAM\medsam_vit_b.pth --epochs 200 --data_folder
.\data\KPCMIX1\ --npz_folder .\data\KPCMIX1_vit_b\ --task_name kpcmix1
```

It's important to note that training, even just the mask decoding part, while freezing the encoder and prompts, is a GPU-intensive process. With this in mind, the work presented here was carried out using a home computer powered by RTX 3060 Ti.

To avoid memory issues, we conducted the training in batches of 200 epochs.

To continue training in subsequent batches, we take the last best checkpoint generated in the previous process:

```
python .\parasam.py --mode train --compare_models --checkpoint
.\work_dir\kpcmix1_bbb_medsam\sam_model_best.pth --epochs 200
--data_folder .\data\KPCMIX1\ --npz_folder .\data\KPCMIX1_vit_b\
--task_name kpcmix1
```

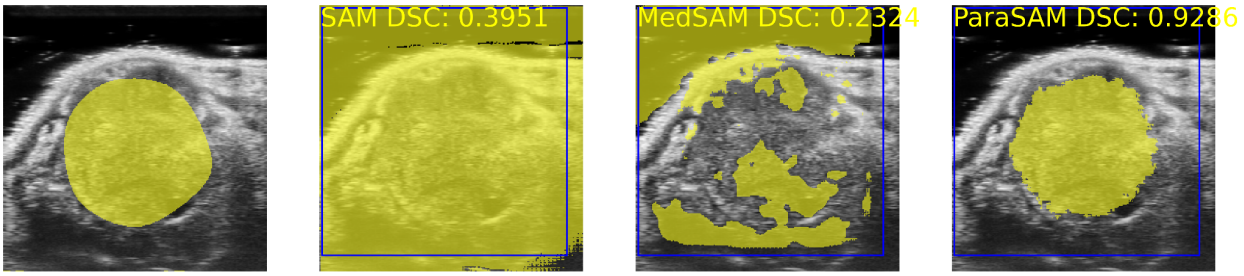
Each batch takes about 5 hours to complete, and the first model was trained for about 800 epochs.

Once satisfied with the training, we can perform inference and compare results using the same bounding box between SAM, MedSAM, and ParaSAM.

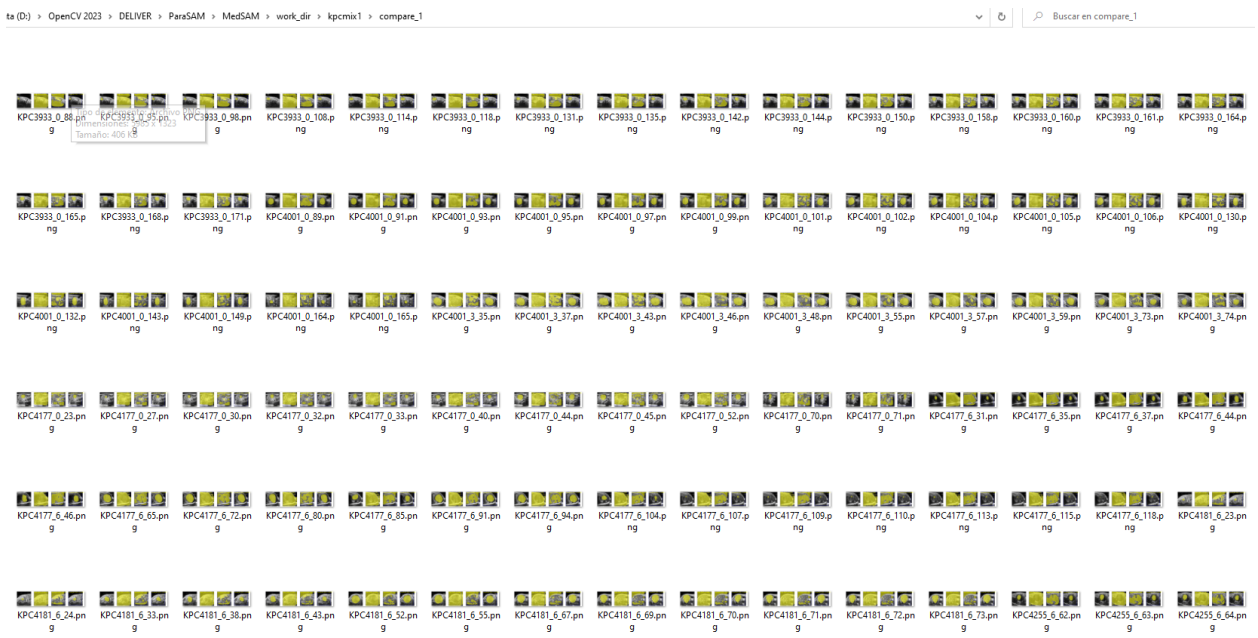
```
python .\parasam.py --mode inference --checkpoint
.\work_dir\kpcmix1_bbb_medsam\sam_model_best.pth --npz_folder
.\data\KPCMIX1_vit_b\ --task_name kpcmix1 --data_folder .\data\KPCMIX1\
--compare_inference

Or

python .\parasam.py --mode inference --checkpoint
.\work_dir\kpcmix1_bbb_medsam\sam_model_best.pth --npz_folder
.\data\KPCMIX1_vit_b\ --task_name kpcmix1 --data_folder .\data\KPCMIX1\
--inference_all
```



You can find these results under the folder where we store the model checkpoints, for example: ParaSAM\MedSAM\work_dir\kpcmix1\compare_1



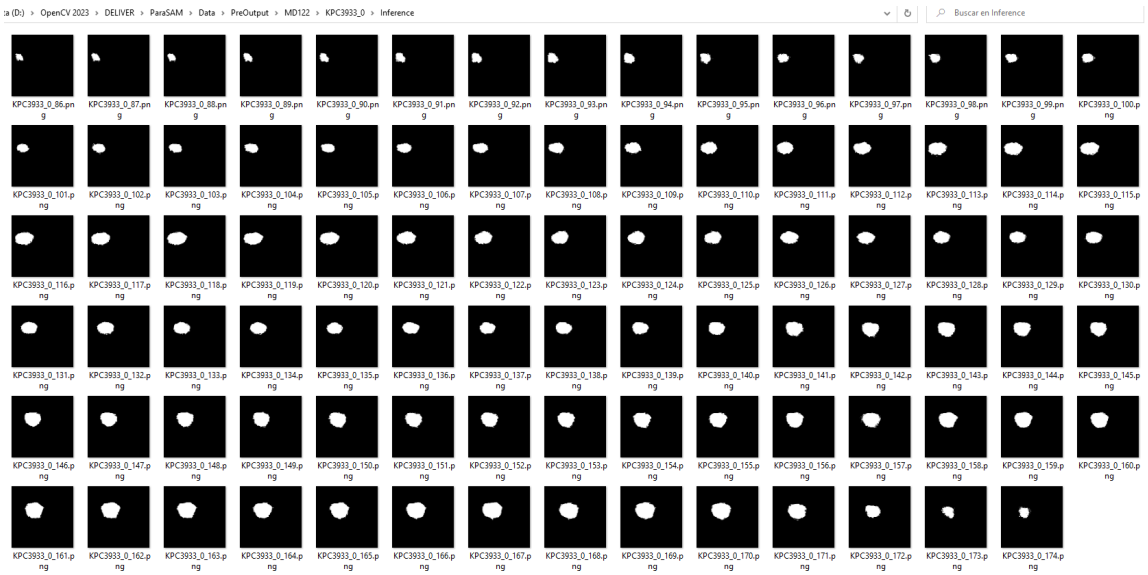
As we can see, ParaSAM significantly improves the results obtained with SAM and MedSAM.

Post-processing: 3D Tumor Reconstruction and Volumetric Calculation

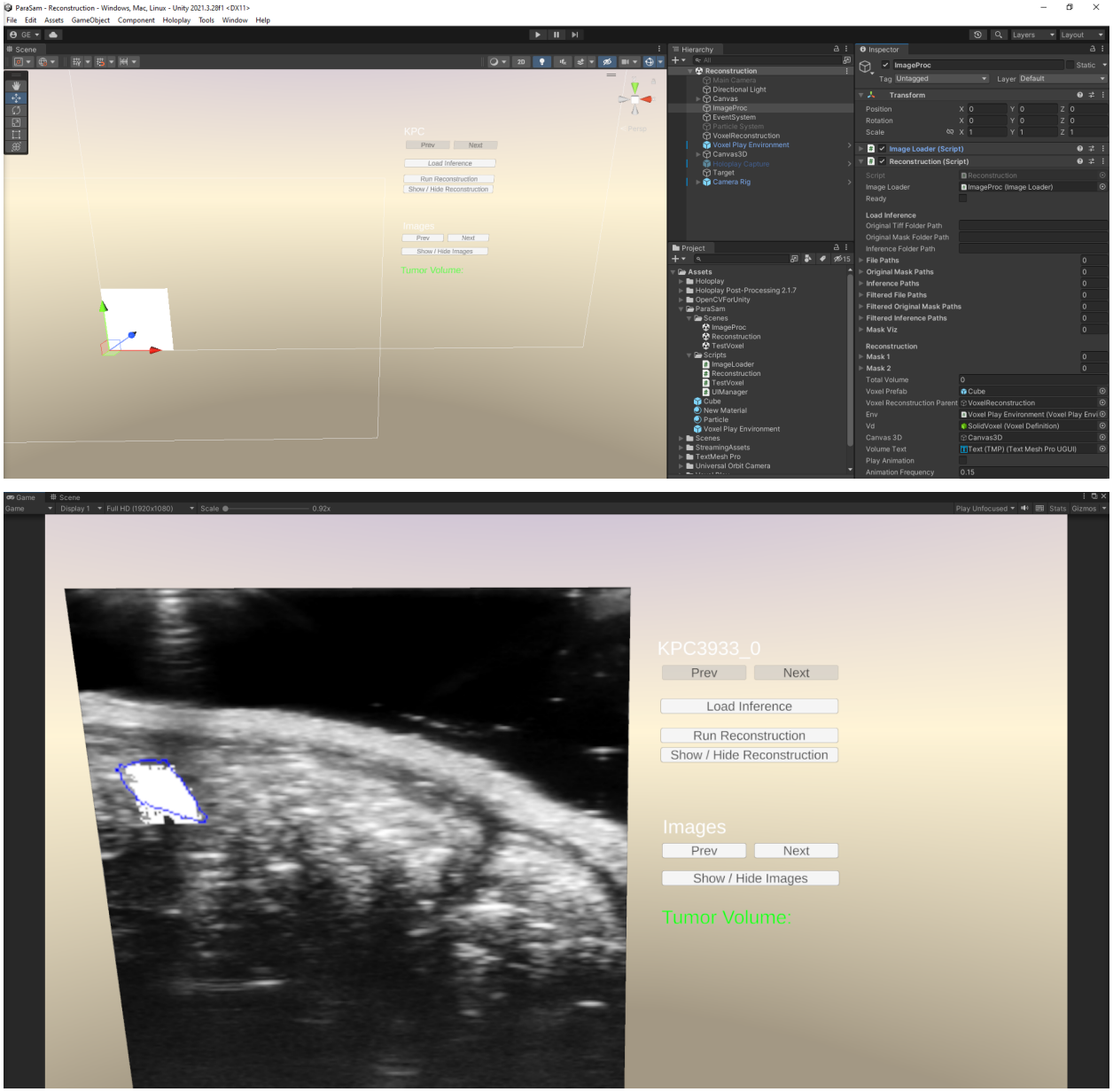
The Unity application/binary contains a second scene named 'Reconstruction' with an interface similar to the first scene.

First, we'll load a new folder with inference results. In this case example for tumor KPC3933 can be found here:

ParaSAM\Data\PreOutput\MD122\KPC3933_0\Inference



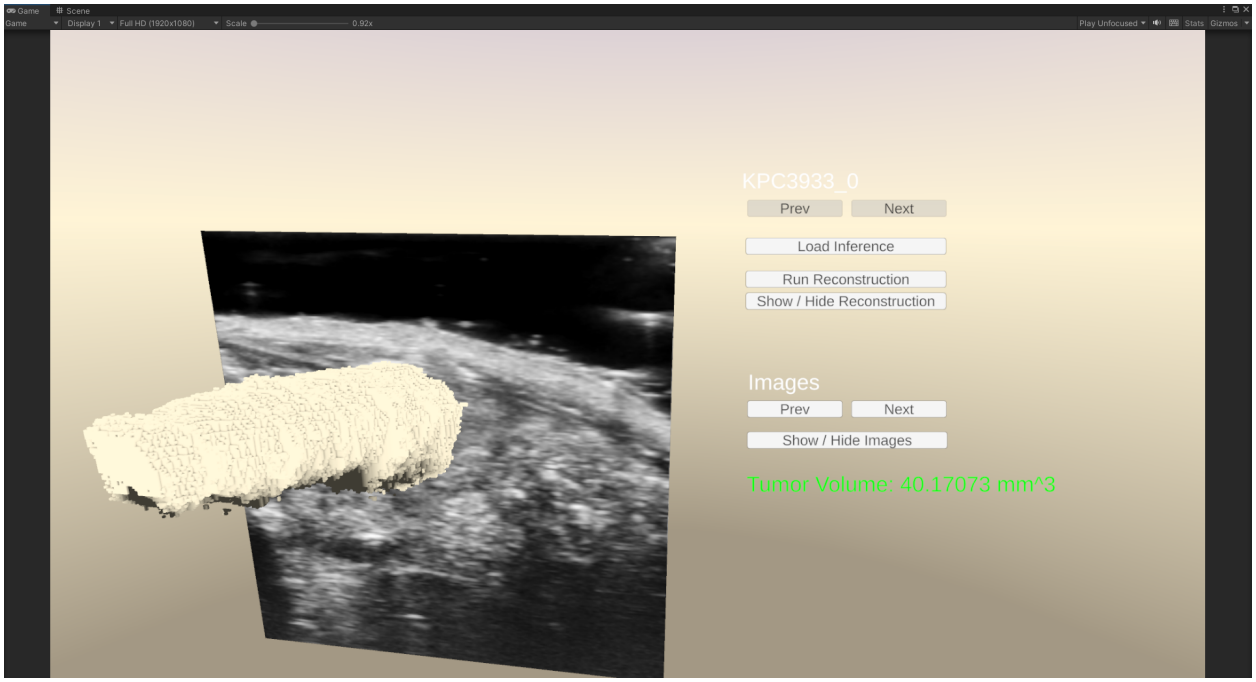
From a basic UI, we can navigate through the tumor image sequence and compare the manual annotation, which we consider as ground truth, with the inference made by ParaSAM.

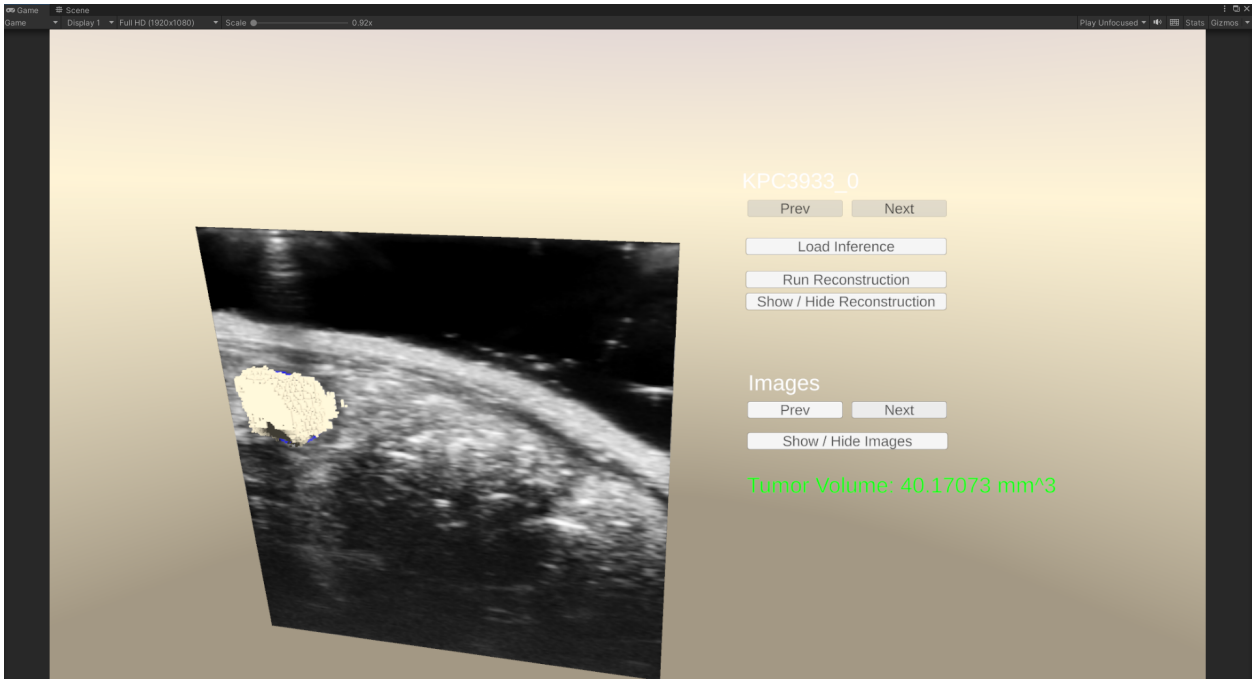


Comparison between GT (blue line) and ParaSAM inference (white mask)

In the future, we will be able to adjust the inference if we are not satisfied with it.

We also have the option to start the 3D reconstruction of the tumor based on the masks inferred by ParaSAM. This process is done with OpenCV and essentially involves processing each mask to generate a 3D occupancy grid. For visualization, we use a third-party Unity asset to display this grid as a set of voxels. We use a specialized asset, based on a special shader, to achieve good performance in Unity due to the high number of voxels generated, without having time to delve into finer generation by subdividing the space between image pairs.





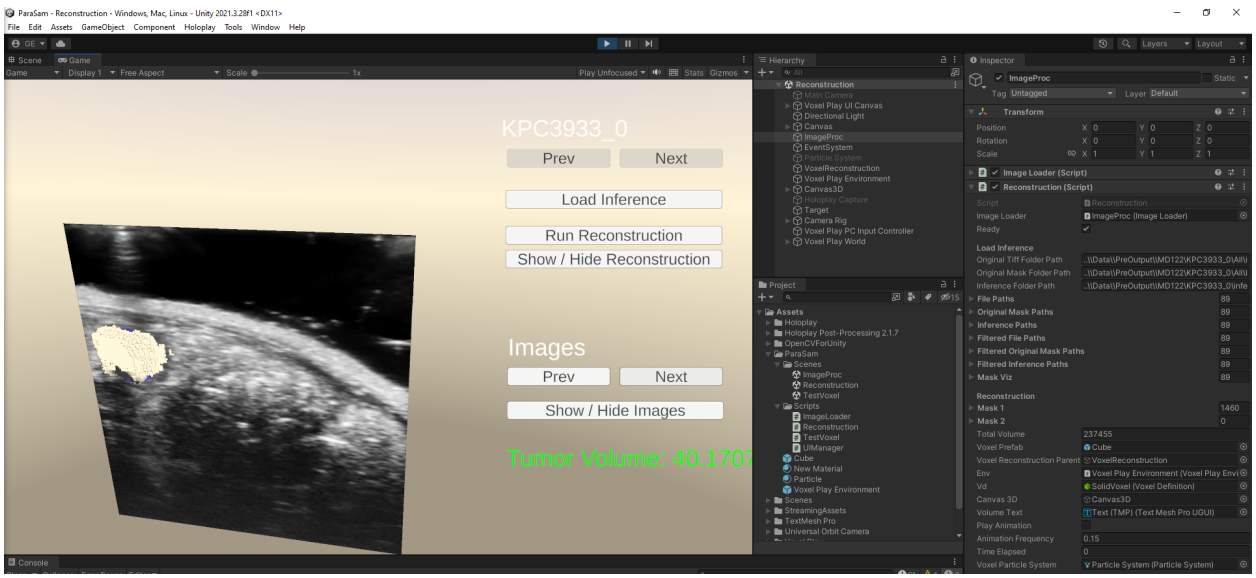
You can inspect tumor reconstruction with each cross-section (image of the sequence)

As a reference, the following 3D reconstruction is composed of 237455 voxels.

Once the tumor is reconstructed and the 3D occupancy matrix obtained, it's easy to calculate the volume knowing the dimensions of each voxel.

In this case, each voxel is 0.04718 mm x 0.04718 mm x 0.076 mm = 0.0001691723824 mm^3

Multiplying by the number of voxels gives us the total volume: 40.17 mm^3



In this case the main code is OpenCV code you can find on C# script reconstruction.cs
For each image and mask inference, we prepare visualization:

```
// Overlay the mask onto the RGB image with some transparency
Mat maskOverlay = new Mat();
Core.addWeighted(images[i], 1.0, masks[i], 1.0, 0.0, maskOverlay);
```

We transform mask in binary mask:

```
Mat binaryMat = new Mat();
Imgproc.threshold(grayMat, binaryMat, 127, 255, Imgproc.THRESH_BINARY); //
Apply threshold
// Find GT mask contours for overlay
List<MatOfPoint> gtContours = new List<MatOfPoint>();
Imgproc.findContours(binaryMat, gtContours, new Mat(), Imgproc.RETR_EXTERNAL,
    Imgproc.CHAIN_APPROX_SIMPLE);

// Draw contours onto the RGB image
foreach (MatOfPoint gtContour in gtContours)
{
    Imgproc.drawContours(maskOverlay, gtContours, -1, new Scalar(0, 0, 255), 1);
}
```

For occupancy grid computation, we've been exploring several ways to interpolate between each pair of images in the 3D sequence. For now we leave the simplest way, just transforming directly each point of the mask in a voxel, knowing the distance between each capture in the sequence is 0.076mm

```
public void RunReconstruction2()
{
    for (int i = 0; i < masks.Count; i++)
    {
        Debug.Log("LAYER " + i);
        Mat maskMat1 = masks[i];
        Core.flip(maskMat1, maskMat1, 0); // 0 means flipping around x-axis
(vertical flip)
        mask1 = ExtractMaskPoints(maskMat1);

        FillVoxels(mask1, i);

    }

    /*totalVolume = CalculateVolume(voxelGrid);
    volumeText.SetText("Tumor Volume: " + totalVolume * 0.000169172f + "
mm^3");*/

    imageLoader.currentImage = 0;
    playAnimation = true;

}
```

To extract the mask points:

```
List<Vector2> ExtractMaskPoints(Mat mask)
{
    List<Vector2> points = new List<Vector2>();
    for (int y = 0; y < mask.rows(); y++)
    {
        for (int x = 0; x < mask.cols(); x++)
        {
            // Check if the pixel is part of the mask (assuming white means
part of the mask)
            // You might need to adjust this condition based on your specific
mask format
            if (mask.get(y, x)[0] == 255) // Assuming mask is a grayscale image
            {
                points.Add(new Vector2(x, y));
            }
        }
    }
    return points;
}
```

And FillVoxels method is filling 3D occupancy grid matrix

```
public bool[,,] voxelGrid = new bool[256, 256, Mathf.CeilToInt(_sliceDistance)
];
```

And voxel visualization at same time:

```
public void FillVoxels(List<Vector2> sourceMask, int layer)
{
    foreach (Vector2 sourcePoint in sourceMask)
    {
        env.VoxelPlace(new Vector3(sourcePoint.x,sourcePoint.y,layer), vd);
        voxelGrid[(int)sourcePoint.x,(int)sourcePoint.y,layer] = true;
    }
}
```

Advanced 3D Visualization. LookingGlass Holographic Display. Mixed Reality with Quest 3.

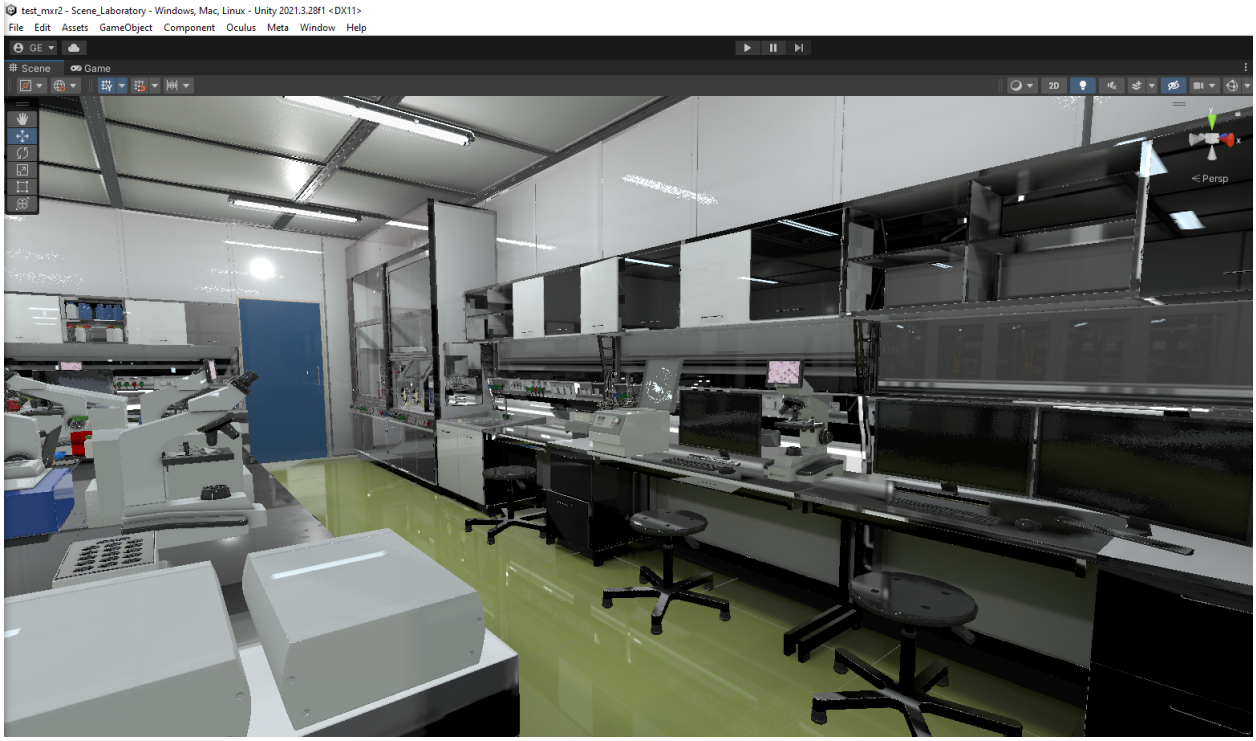
An aspect we have barely been able to work on in this project is the development of an appropriate UI that facilitates the exploration of the 3D sequence, as well as manual annotations (which we consider as ground truth), ParaSAM model inferences, and the 3D reconstruction of the tumor.

Within this last section, it makes sense to visualize the 3D reconstruction using advanced devices designed for this purpose.

As an experiment, and taking advantage of the integration of LookingGlass and Meta Quest with Unity, we have used:

- Visualization on LookingGlass holographic screen.

- Visualization in mixed reality using Meta Quest 3: Currently work in progress, we're building our own virtual lab, where we can inspect 3D reconstruction using VR and MXR



Beyond complementing 3D generation with "3D displays", the ability to jointly explore each layer of the 3D reconstruction opens new options for overlaying additional information about tumor treatments, thus understanding how they evolve and respond to treatments.

4. Preliminary Results

Given the short timeframe (3 months), most of our time was dedicated to data extraction and curation.

Nevertheless, we have been able to validate the end-to-end workflow, starting from manually annotated images using the VEVO labs program.

Using the application developed with Unity and OpenCV, we pre-processed the annotated images and converted the manual polyline annotations into masks - input for SAM/MedSAM/ParaSAM. With the same application, we generated the dataset prepared for the training process, splitting it into training (80%) and validation (20%).

Thanks to the work done in MedSAM, we were able to prepare a Python script to refine the MedSAM checkpoint.

The results presented in this work substantially improve the outcomes of SAM and MedSAM for our tumor images, especially using a fixed bounding box as a prompt (automatic segmentation approach).

Epochs: Approximately 800 (around 20h training on RTX 3060 Ti)

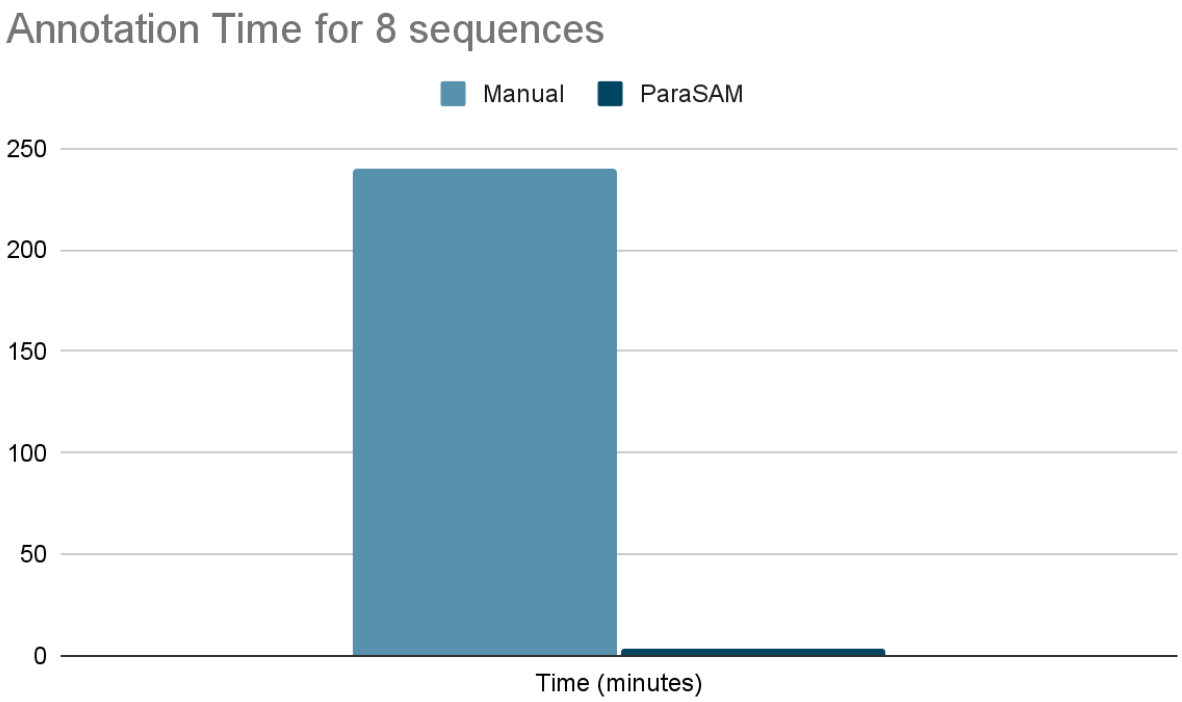
Later, using the application developed with Unity and OpenCV, we processed the masks inferred with ParaSAM and used voxelization techniques to calculate the volume of the tumor reconstructed in 3D.

For the selected tumor KPC3233, the preliminary volume data is 40.17 mm^3, compared to volumes obtained with the VEVO lab program (in two different annotations) ranging between 50 mm^3 and 28 mm^3 (acceptable difference range)

This end-to-end process confirms some important points:

- Consistent results are always obtained for a sequence without any bias.
- Manual annotations made in VEVO labs use a polyline tool, which in practice is imprecise. In contrast, the masks inferred by ParaSAM are at the pixel level, being much more accurate to the shape of the tumor (grayscale, shapes, etc.)

The annotation process of a tumor/sequence by a specialist in the lab takes between 30-60 minutes depending on the complexity of the tumor. Annotating the 8 tumors would take about 4 hours in the best case. ParaSAM performs the inference of the 8 tumors in 4 minutes.



5. Conclusions and Future Steps

As previously mentioned, these first three months, encompassing the contest period, have allowed us to develop a PoC and validate the basic functionality end-to-end, including validating the refinement of MedSAM for automatic (and semi-automatic) annotation.

The next major milestone is to optimize the model and validate the results using the full set of over 32,000+ images; include annotated images of different observers which will provide the Kappa coefficient (interpersonal variation); and compare these results with both manual annotations and measurements of the real tumor taken after necropsy. This data will be publicly available in the form of a scientific publication by early 2024.

Additionally, we are not limited to SAM/MedSAM as reference models, and we plan to compare results with models.

In a more research-oriented approach to generative AI, the fact that tumor sequences are 3D but each image is typically taken at 0.75mm intervals, means that the 3D space has been quantized or discretized. This implies the need to approximate the space between images, either by known geometry or by interpolating information like contours between image pairs. We believe that recent advancements in generative AI could be used to fill in missing images in that 0.75mm space to obtain more precise volumetrics.

For more immediate and specific improvements to the application, we are focusing on:

- Editing masks inferred by ParaSAM.
- Integrating inference (and/or training) within the Unity application via OpenCV (ML) library and/or Unity Sentis.
- Improving the voxelization process for more accurate volumetric calculations.

6. Acknowledgements and References

We thank MedSAM authors for opening the code (https://github.com/bowang-lab/MedSAM), model checkpoint and instructions on further fine-tuning and for extension we thank Meta AI for making the source of SAM publicly available