A Brief Overview of Neuronavigation AR Application

Parsa Mojarad and Ahmadreza Tavana

Student, Department, Sharif University of Technology

Introduction

This paper conducts a novel method for Neuronavigation. Neuronavigation is the set of computer-assisted technologies used by neurosurgeons to guide or "navigate" within the confines of the skull or vertebral column during surgery, and used by psychiatrists to accurately target rTMS.This article introduce the methods which are implemented for this application.
For this application, augmented reality methods are used for having sense of reality by using the application. Augmented reality is an interactive experience that combines the real world and computer-generated content. The content can span multiple sensory modalities, including visual, auditory, haptic, somatosensory and olfactory. Now we introduce the AR application and used methods in detailed. This paper contains explanation of used methods, actions for making the application better after primary tests, problems and possible solutions and future works. Actually our group could not develop the application more than a specific options because of limitation of AR packages in this time because Google packages don't have the options that we need for improving the application which are discussed in problems and possible solutions and future works in continue of the paper.

Methods

Implementing 3D reconstruction and Augmented Reality in Unity:

MR imaging output files come in different formats such as DICOM and NIFTI. In order to fully reconstruct MR images that were, at first, only available in DICOM format, a volume rendering approach "Volume rendering by object space raymarching" was used. This method was easily implemented in the 3D workspace given to us by the Unity game engine. After configuring shaders to mimic the color and texture of brain tissues, a series of DICOM images were used to test the algorithm.

Fig. 1. A 3D reconstructed monkey brain MRI

As it can be seen in Fig1. the reconstructed image, although it had some minor shape problems, was fully reconstructed and shown in a virtual 3D space. The next step was to program the application to use an Android mobile phone to run an Augmented Reality (also referred to as AR) space and load the reconstructed image. In this stage, the problems began to occur. The output 3D object was extremely large in file size, and caused the whole application to lag and become unusable. The other problem was, even though Unity provided for an extremely rich environment to develop an AR app, loading DICOM file series from storage was not viable, and for this reason alone, we opted to migrate our app to other platforms.

Migrating to Java/Android

After encountering problems said in the previous section, we decided to use basic libraries provided for AR applications from Google. The first step in this application is to reconstruct the MR images since we don't have access to the tools in Unity. In this case we use NIFTI image format and then build 3D reconstructed images in another program other than ours. The mentioned program, Slicer(1), has the feature to load NIFTI image series and after selecting the tissue, generate an output with multiple formats such as ".obj", ".ftx" and ".stl", all which are 3D objects formats.

Next step is to implement Google's AR library in the Java/Android environment (Android Studio). This library, Sceneform, which allowed the application to render 3D objects in an AR environment was a somewhat good platform to develop an application similar to what we did in Unity. Unfortunately, at the time, Google declared that this library was deprecated and no longer received updates. A group of developers however, maintain the library, to this date, not to add more features but only to fix the bugs. Yet there were no alternatives at this point, so we decided to go through with the library since it had some of the features that we needed. In order to use this library, the 3D model exported in Slicer, had to be in ".gltf" format and also uploaded in either a local server or a remote one. These were the three tasks that needed to be done before building the app.

Exporting ".obj" model

As it said in the previous section, in order to use AR libraries, the format of 3D object files must be .gltf. For having this format, at first we must convert .nii files to .obj files. This conversion can be done in Slicer with using segmentation toolbox. As you know we worked in monkey brain MRI files. The files are skull and brain segmented atlas of monkey. First step of a 3D output of the monkey skull and the brain is shown in figure 2.

Converting ".obj" to ".gltf" and reducing size

In the previous section, we came to this conclusion that for using AR libraries, file formats must be .gltf. In the Slicer

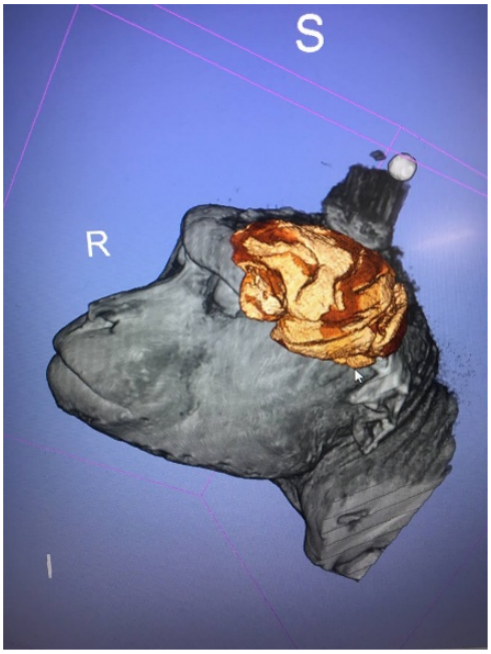

Fig. 2. As it can be seen the brain is fitted to monkey skull and first step of segmentation is done properly.

application we changed the format of .nii files which are for afnii files to .obj. Now these .obj files which are for skull and atlas, must be converted to gltf and attached to each other to have a single 3D file to use in the application and also size of the file must be reduced for having faster loading in the application. We must be careful to not reduce size in a way to loose quality of the 3D images. These approaches can be done in "Blender"(2) application. We attached atlas and skull and sync them and centralize them and then we reduce the size by sampling from the whole file. For this project we just kept 0.2 of pixels to reduce the size properly and also not losing the quality of the objects.

Reducing Alpha of 3D object

An innovation approach for having better sense of brain in the skull of the monkey, we tried to reduce alpha of both skull and atlas in order to fit atlas inside of the monkey skull and the observer has a better sense of location of the brain. Figure 3 shows a sample of atlas with lower amount of alpha.

Local Server

As explained the previous part, our Neuronavigation AR Application is written for android phones. For accessing and loading 3D files in the application we must have accessibility to files of the phone and new regulations of android don't allow to access to local files of the phone because of security and the user have to download their needed files from an online server. For solving this problem, we had to run a server which has to be synced with the application to download wanted files. So for this application we run a local server which is written in node.js platform which gives us the ability of downloading 3D files from the server. This server works as a data base that we can storage 3D MRI files on the server

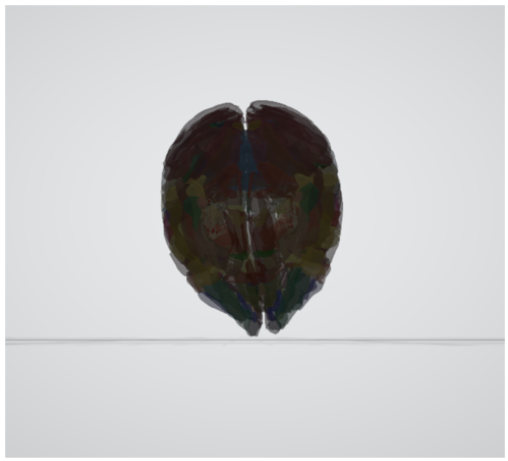

Fig. 3. Atlas with reduced alpha

and load each of them while using the application. Files can be both downloaded and uploaded on this local server for using the application.

Actions taken to fix some problems

After completing the mentioned tasks, in Android development environment "Android Studio", using "SceneForm maintained"(3) , the models were loaded and tested in AR space. This library used "node", "ray hit" and "anchor" concepts. Node refers to a fixed location either relative to the "world" which was in our case the mobile phone itself or "local" which was another node previously defined. Ray hit is a concept that is the basis of AR. Simply put, after scanning the area and detecting the surfaces, and touching anywhere on the screen, it simulates a virtual ray that starts from the mobile phone camera and hits the detected surface, the location is calculated and a node is placed there. Anchor is a type of node which can attach models and other node types to. To briefly introduce the changes that were done after testing the app multiple times, these actions were taken to fix some problems encountered:

- Implemented Z-axis change slider to move the 3D object along Z-axis.
- Implemented hidden nodes to chain both head and atlas models in order for to rotate, move and scale them both at the same time
- Added a feature to stop scanning and detecting surfaces to improve accuracy
- Added the feature to show and hide the models instead of loading and removing them constantly
- Removed the artificial shadow component that SceneForm automatically added to the camera scene.
- Removed the translation, rotation and scale controllers attached to the anchor node.
- Added sliders to rotate and scale the models.

Also some algorithmic steps were taken to improve the accuracy as much as possible, such as working with multiple nodes and node types (world and local), also changing between world locations and local locations and rotations whenever needed.

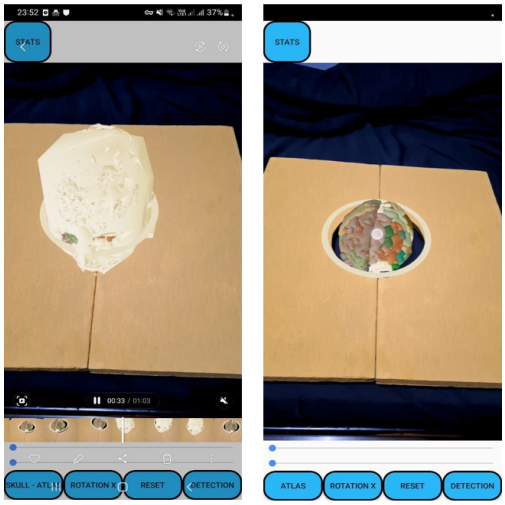

Fig. 4. Images from inside the application

Some of the problems that were encountered in Unity were resolved here such as there was no lag and the app ran smoothly and images could be loaded from an external source rather than being prebuilt into application.

Problems and Possible Solutions

After running multiple tests, adjusting and refining the application, it was not approved as it had some major flaws that will be discussed in this section.
1- The first major problem is AR based apps need the nodes to anchor themselves to a surface (as discussed before). This surface detection forces some physical constraints as we had to constantly test the application with some flat surfaces around the head (target location) and wait for AR core libraries to register these surfaces and allow the application to use them as anchor points. So as it can be seen in the images from inside the application a flat surface is used.
2- The accuracy needed for this application to actually work like a conventional NeuroNavigation device is extremely high, and no matter all the refining and adjusting AR is not designed to reach these standards. These standards include being able to register the 3D model of head and brain atlas to real world at least in two planes (sagittal, coronal or transverse). This brings us to another physical constraint of AR which has a dislocation every time the camera moves because it renders and registers node locations relative to the phone camera location. This dislocation, though very small, ruins the registration when we switch between planes and we cannot accurately place the model on the subject's head.
3- Another physical constraint in AR based applications is solid modeling. All models placed in AR environments are solid no matter how we changed the alpha value (transparency). This caused a disruption in fixing the model on target, as it was impossible to place the model inside the target and it was always in front of the target. This means the application cannot meet the standards before.
After dealing with many problems and fixing them, the 3 problems discussed before remained and there was no solution to them. Some new features that were developed later by the developers of SceneForm such as depth occlusion was tested to see if it can at least solve one of the problems but it did not.

Future Works

There may be a way to solve these problems in future works, here are some possible routes to take:
- Switching to the new libraries the may be introduced by google instead of SceneForm
- Manipulating the basic features of ARcore(4) such as depth detection to place the model inside the target instead of front or back of the target. Google is also adding some new features to the library and it may have a possible solution for this particular problem.
- Completely changing the approach and using another platform instead of AR such as XR. This course of action however, is not suggested. Because re-implementing the data that AR provides and using the data that mobile phone sensors record is like re-inventing the wheel.

Referenced Links

1. https://slicer.org/. .
2. https://www.blender.org/. .
3. https://github.com/sceneview/sceneform-android. .
4. https://developers.google.com/ar. .