CS7GV4 Augmented Reality

Project Report: 3D Copy&Paste

Jan Jiménez Serra - ID: 20303235

Introduction

The goal of the project is to create a real-time 3D Copy&Paste solution, first by obtaining a 3D mesh using a mobile device, and then seamlessly send the mesh to a 3D modeling program by pointing the camera to it. In this case, an iPad Pro with LiDAR sensor is used to make a 3D scanner with Apple's ARKit.

That is done by creating a mobile application accompanied by a desktop server in a similar way as done in 2D by ClipDrop [1]. While ClipDrop pastes an image onto Photoshop, this project takes that concept into 3D by pasting a 3D object onto Blender.

The motivation is to improve current cumbersome methods of obtaining 3D models. Also, it would be a practical and easy way to introduce beginner creators into AR, as you would only need a mobile device. It would also be useful for 3D modeling artists.

Theoretical background

3D scanning for mobile devices has been made with a variety of different technologies. Most of those technologies come with limitations.

Solutions for 3D scanning work with sensors, including optical, acoustic, laser scanning, radar, thermal, and seismic [2]. Those sensors are not widely available on mobile devices, therefore external devices are needed, which lowers the amount of people that will be able to use them.

Mobile 3D scanning apps that do not need external devices generally acquire 3D data using stereo image pairs. Photogrammetry is the approach based on a block of overlapped images used for 3D mapping and object reconstruction [3]. While that can output high quality 3D reconstructions, it comes with higher computational power needed, and most importantly, time needed to generate it. Because of this, mobile apps nowadays require payment to compensate for the costs of generating the 3D scan.

Therefore, current photogrammetry does not satisfy real-time solutions, therefore another sensor would be needed. Here is where LiDAR comes in. The fourth-generation iPad Pro from 2020 comes with a LiDAR sensor already integrated on the device, which solves the problem of needing an external device to generate real-time reconstructions.

LiDAR stands for light detection and arranging. It is a method for determining ranges by targeting an object with a laser and measuring how long it takes for the reflected light to return to the receiver [4].

A digital surface model (DSM) is generated with lidar data and then the objects higher than the ground are automatically detected from DSM, which allows for general knowledge of the objects such as size, height, shape, etc.

LiDAR technology has been extensively used in a wide range of applications, in surveying, geodesy, geomatics, archaeology, geography, geology, geomorphology, seismology, forestry, atmospheric physics, laser guidance, and more. Most importantly for this solution, it is commonly used for 3D laser scanning.

Apple's reasons to add a front-facing LiDAR sensor on its new devices are mainly optical camera focusing and the ability for augmented reality apps to perform much more accurately. Regardless of the reasons why Apple added it, the LiDAR system can also be used for 3D scanning. While LiDAR produces a single depth map, It is possible to move the camera around to generate a 3D model by combining different LiDAR images.

Implementation

The technical implementation comprises three different parts. First, the iOS mobile application written in Swift that uses ARKit to extract LiDAR's information into an OBJ file. Then a server running on the computer that correlates the center position of the phone's camera to the center of the computer's screen using a python module called screenpoint[ref]. Finally, a Blender script imports the OBJ file and puts it at the (x, y) coordinates generated by screenpoint. The technical architecture can be seen at figure 1.

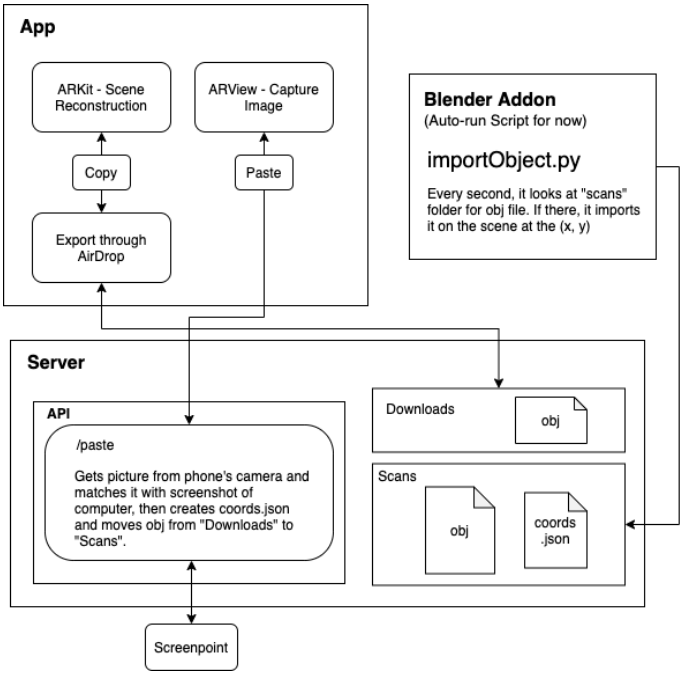

Figure 1. Technical structure of the project, comprises an iOS application, a local server coded with Python and Flask, and a Blender script.

Mobile iOS App

The iOS app is based on “Visualizing and Interacting with a Reconstructed Scene” [5] from Apple’s official documentation. There, they showcase the capabilities of LiDAR, mostly for more accurate physics on objects in the scene. The important part that allows for 3D reconstruction is the instance of *sceneReconstruction*, which when enabled, provides a polygonal mesh that estimates the shape of the physical environment. The following code shows how to enable it; it only works with the iPad Pro 4th Gen or iPhone 12 Pro with a LiDAR scanner.

```
arView.automaticallyConfigureSession = false
let configuration = ARWorldTrackingConfiguration()
configuration.sceneReconstruction = .meshWithClassification
```

Apple’s ARKit framework is further divided into what they call “Content technologies", which are RealityKit, SceneKit, SpriteKit and Metal. SceneReconstruction with LiDAR only works with RealityKit’s arView.

Getting the scene reconstruction into an OBJ file is not straightforward. Firstly, all ARMeshAnchor are fetched, which are anchors for physical objects that ARKit detects and recreates using a polygonal mesh. Then, each ARMeshAnchor is converted into a MDLMesh by converting each vertex of the geometry from the local space of their ARMeshAnchor to the world space. Each MDLMesh is added into a MDLAsset, which is the iOS container for 3D objects. Finally, the MDLAsset can be easily exported as an OBJ file using UIActivityViewController. The code for that can be seen at the *saveButtonPressed* function in *ViewController*. This method was borrowed from “iPadLIDARScanExport” by Stefan Pfeifer [6].

This all happens once the user presses the *copy* button, which prompts an export interface. The app trusts the user to export the OBJ through AirDrop into the computer. The initial idea was to send the OBJ through a Rest API to the server, but sending 3D objects through HTTP is not standardised yet, therefore this solution was applied instead.

By default, the app starts the scene reconstruction and shows a polygon mesh of the scene. The 3D scan can be reconstructed anytime by pressing the reset button as seen at figure 2.

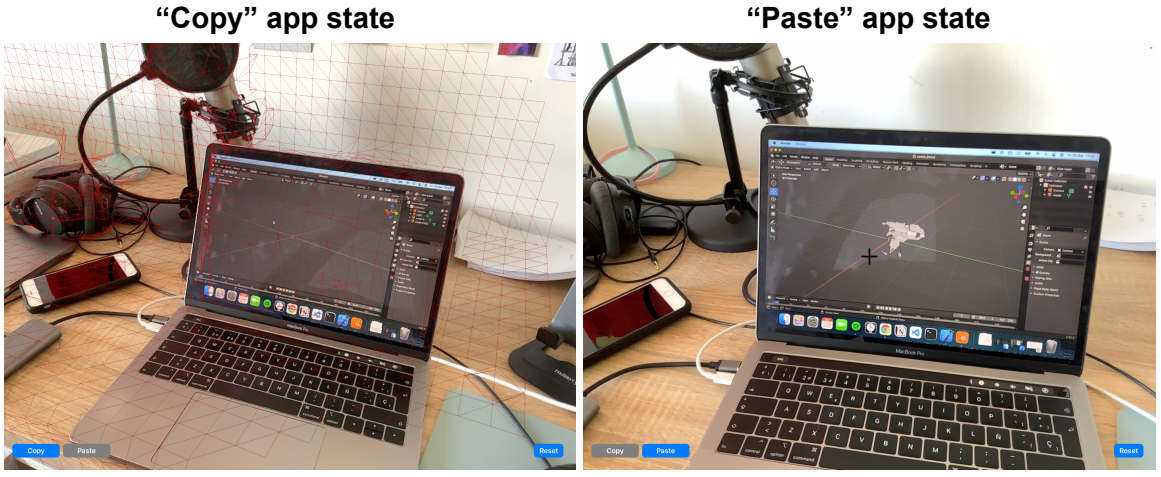

Figure 2. User Interface of the iOS application showing both main states.

Once the OBJ is exported, the *paste* button becomes available, as well as a cross icon in the center of the screen. The scene reconstruction also stops and the app stops showing the polygon mesh.

Finally, the user points to the computer's screen where it wants the 3D object to be pasted, once the *paste* button is pressed, the app takes a picture from the phone's camera and sends it to the server's API with a POST function.

Local server

The server is coded in Python and uses Flask to create an API. Using Python allows for a wide range of handy modules like OpenCV, Pyscreenshot, Pillow, etc.

For now, the API is just a single endpoint */paste* which is a POST HTTP function that receives an image. The image received should be a picture captured from the iOS app that looks at the computer with a Blender project opened.

The function will then take a screenshot of the computer using the Pyscreenshot module, then it finds the centroid coordinates in screen space using Screenpoint.

Screepoint is a python module created by Cyril Diagne, who also did the AR 2D Cut&Paste [7]. Screenpoint uses OpenCV SIFT. SIFT is a class for extracting keypoints and computing descriptors using the Scale Invariant Feature Transform (SIFT) algorithm by D. Lowe [8]. It does the feature matching between two images, and then finds the centroid of those feature points. An example illustrating the feature matching done from the demo is shown at figure 3.

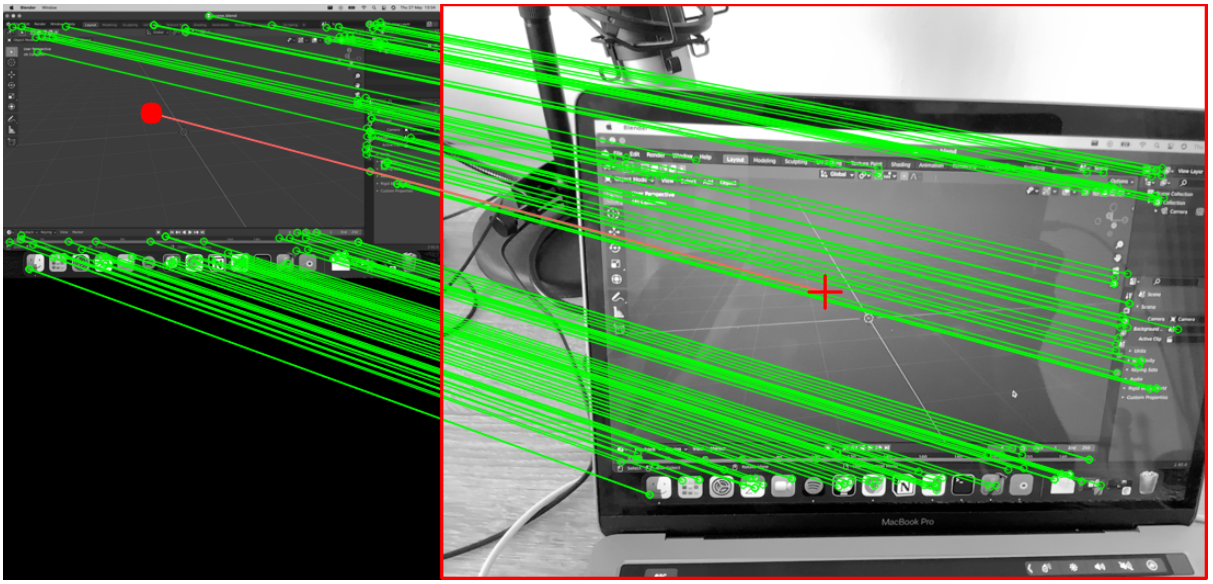

Figure 3. Illustration of feature matching made with Screenpoint and OpenCV SIFT.

If found, the centroid coordinates are saved into "scans/coordinates.json" as a JSON object with x and y keys.

Lastly, the python script moves the OBJ file from the local "downloads" folder, to the "scans" folder of the project as seen in the following code snippet.

```
x, y, img_debug = screenpoint.project(view_arr, screen_arr, True)
# Write debug image.
cv2.imwrite('feature_matching.png', img_debug)

found = x != -1 and y != -1

if found:
    # Bring back to screen space
    x = int(x / screen.size[0] * screen_width)
    y = int(y / screen.size[1] * screen_height)
    logging.info(f'{x}, {y}')

    # Paste the current model in blender at these coordinates.
    logging.info(' > sending to blender...')

    # creating coordinates
    data = {
        'x': x,
        'y': y
    }

    # write coordinates
    with open('./scans/coordinates.json', 'w') as outfile:
        json.dump(data, outfile)
    # move model from local downloads to scans folder
    os.replace(downloads_loc+"/model.obj",
               "./scans/model.obj")
```

Blender Script

Blender has a scripting system where you can write commands to the scene. That scripting system cannot be used as an external python module, which makes it difficult for external tools to modify Blender files.

The workaround was having a Blender script run automatically on start-up in the background. The function of that script is to look every second at the “scans” folder of the project for an OBJ file; if found, it imports it on the (x, y) coordinates extracted from “coordinates.json” file on the same folder. Knowing that the paste function from the local server will add the OBJ file on that folder once the user presses the paste button on the app, it creates this illusion of pasting the 3D model into Blender.

Any delay on the pasting functionality comes from the time that Blender takes to import the OBJ file. A future improvement will be creating a Blender addon instead of a simple script, which would improve performance.

Results

Experiments were focused on measuring the iPad's LiDAR sensor for 3D reconstruction on different scenarios. User experience was also measured, with the goal to improve current 3D scanning solutions.

The experiments did not have the purpose of finding pros and cons of using LiDAR as a technology for 3D scanning, as that has been extensively researched already [ref]. Instead, the experiments were conducted to find out if current on-device LiDAR sensors were accurate enough for 3D scanning.

Using Meshroom [9], the same scene was reconstructed using photogrammetry to be compared with the LiDAR scan, from an input of 11 images rotating on a middle-sized object. The comparison can be seen at figure 4.

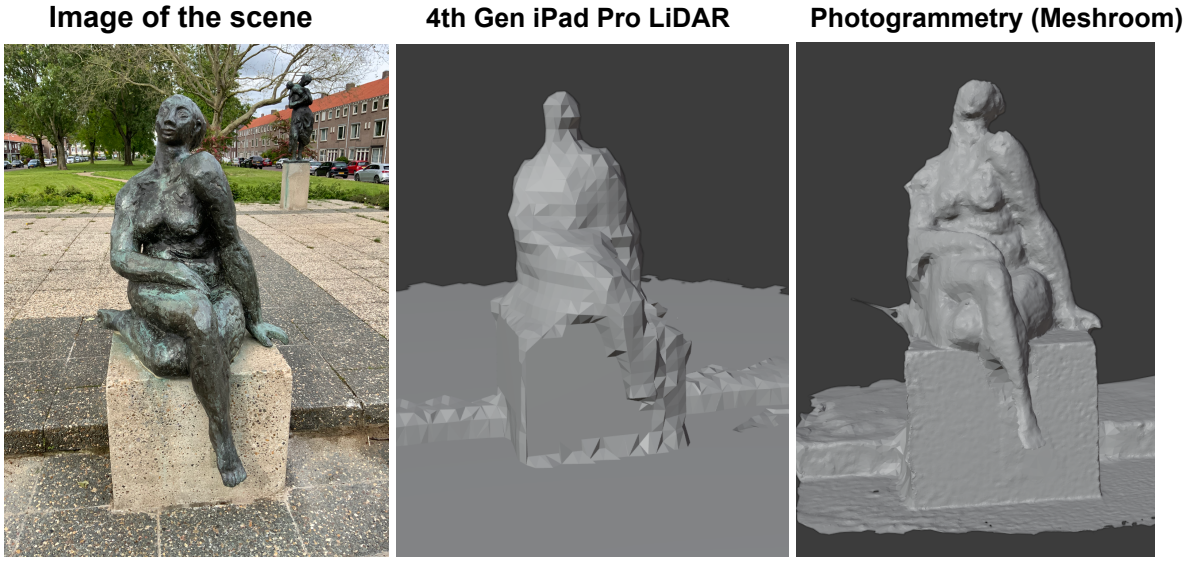

Figure 4. From left to right: picture of the scene to be reconstructed, reconstruction using the LiDAR scan from a 4th Gen iPad Pro, reconstruction using photogrammetry with Meshroom.

As clearly seen, LiDAR quality is way lower than using a photogrammetry solution. The iPad LiDAR sensor cannot properly scan any small items, which is due to the LiDAR sensor having a far less dense beam array. At figure 5, you can see the difference between the iPhone 12 Pro front camera used for face recognition, versus the view of the LiDAR dot array, taken from iFixIt [10].

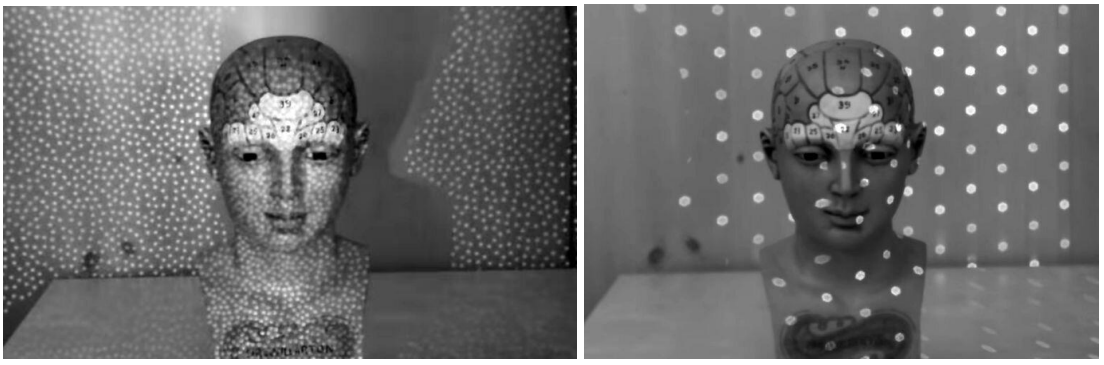

Figure 5. On the left, beam array of front-facing face recognition sensor of iPhone 12 Pro, on the right beam array of LiDAR on the same device.

That being said, LiDAR's solution works in real-time, compared to the 20 minutes that it took for a GeForce GTX 1050 equipped Windows computer.

Conclusions

A 3D scanning solution for new iOS mobile devices was made with a unique pasting feature that sends the scan to Blender in real-time.

It is unlikely that the iPad Pro LiDAR can be used in a professional setting, where more accurate representations are needed. That being said, this solution might be helpful for a more hobbyist user base that wants to quickly try out 3D scanning without the need of buying external devices.

3D models obtained by this solution could be improved sufficiently once in Blender, or used as starters for 3D models based on real-life objects.

The goal of improving on current cumbersome 3D scanning user experiences was met. Moreover, modifications on part of the pipeline could be easily incorporated. For example, instead of using a LiDAR sensor, a photogrammetry solution could be implemented, which would be more time consuming and computationally expensive on the "copy" stage, but the "paste" stage would be just as fast.

Further improvements on the user interface could be made. I tried adding an overlay of the 3D model when pasting, but adding 3D models into RealityKit is not straightforward. Instead, SceneKit would be needed in a second screen when pasting, which made the process less smooth.

Another clear improvement would be turning the Blender script into a Blender addon, which is in the process of being implemented. A similar idea could be implemented in other 3D modeling softwares.

An issue when pasting might be that the coordinates found by Screenpoint might not show up on Blender where we want, because the Blender scene could be not centered or zoomed. A solution to that would be centering the scene to default at the start of the script.

The unique pasting system could also be incorporated into any other current 3D scanning solution as a feature.

References

1. Diagne, Cyril. “ClipDrop (AR Copy Paste).” ClipDrop, clipdrop.co. Accessed 28 May 2021.
2. Brian Curless (November 2000). "From Range Scans to 3D Models". ACM SIGGRAPH Computer Graphics. 33 (4): 38–41. doi:10.1145/345370.345399. S2CID 442358.
3. Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz and Richard Szeliski, “Building Rome in a Day”. International Conference on Computer Vision, 2009, Kyoto, Japan.
4. National Oceanic and Atmospheric Administration (26 February 2021). "What is LIDAR". oceanservice.noaa.gov. US Department of Commerce. Retrieved 15 March 2021.
5. “Apple Developer Documentation.” Visualizing and Interacting with a Reconstructed Scene, developer.apple.com/documentation/arkit/content_anchors/visualizing_and_interacting_with_a_reconstructed_scene. Accessed 28 May 2021.
6. Zeitraumdev. “Zeitraumdev/IPadLIDARScanExport.” GitHub, github.com/zeitraumdev/iPadLIDARScanExport. Accessed 28 May 2021.
7. Cyril, Diagne. “Cyrildiagne/Screenpoint.” GitHub, github.com/cyrildiagne/screenpoint. Accessed 28 May 2021.
8. Lowe, D. G., “Distinctive Image Features from Scale-Invariant Keypoints”, International Journal of Computer Vision, 60, 2, pp. 91-110, 2004.
9. “AliceVision | Photogrammetric Computer Vision Framework.” Meshroom, alicevision.org. Accessed 28 May 2021.
10. Purdy, Kevin. “How LiDAR Works, and Why It’s in the IPhone 12 Pro.” IFixit, 28 May 2021, www.ifixit.com/News/45482/how-lidar-works-and-why-its-in-the-iphone-12-pro.