Documentation for our 3D-Modelling Application

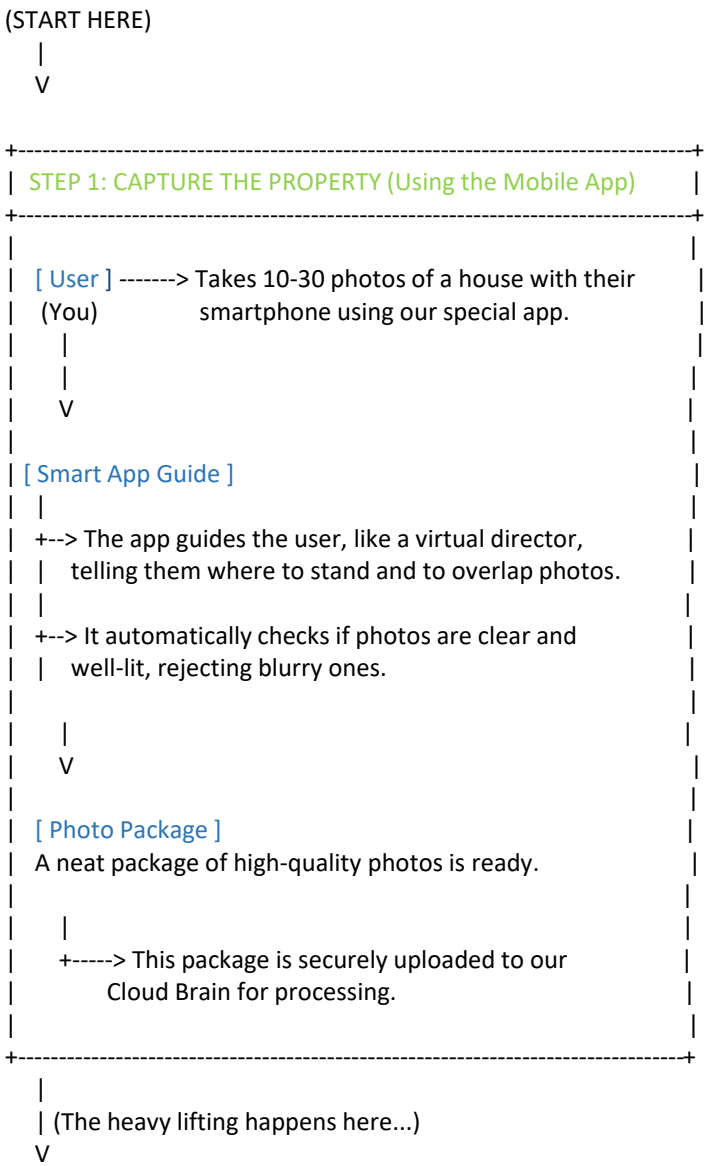
19 October 2025 19:27

==========

SYSTEM FLOW: From Smartphone Photos to Insurance-Ready Data

==========

This diagram shows how our system turns simple photos of a house
into accurate 3D models and measurements for insurance companies.



| or the edge of a roof).

|
V

It then "stitches" these photos together, not flat like a panorama, but in 3D, to build a digital skeleton of the house. This is called "Photogrammetry".

|
V

Finally, it drapes the original photo colors and textures over the 3D skeleton to create a realistic, life-like digital twin of the house.

|
V

[Accurate 3D Model]
The result is a complete and precise 3D model.

|
V

STEP 3: EXTRACT THE MEASUREMENTS (The Virtual Tape Measure)

[Automated Measurement Software]
Our software now analyzes the finished 3D model.

|
V

It automatically detects all the important surfaces:
- The roof planes
- The walls
- The windows and doors

|
V

It calculates all the key dimensions with high accuracy:
- Total Roof Area (in sq. ft.)
- Total Wall Area (in sq. ft.)
- Building Height and Perimeter

|
V

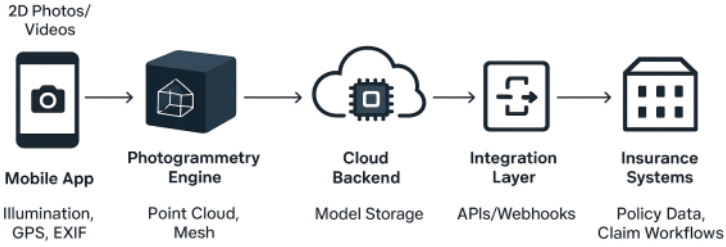
[Simple Measurement Report]
All these numbers are put into a simple, clean report.
(This is a "JSON" file for computers to read).

|
| (The final step is delivering the results...)
V

STEP 4: DELIVER TO THE INSURANCE COMPANY

```
+-------------------------------------------------------------------------------+
|                                                                               |
|  [ Automatic Notification ]                                                   |
|  Our system sends an automatic alert to the insurance                         |
|  company's software, saying "The report is ready!"                            |
|                                                                               |
|     |                                                                         |
|     V                                                                         |
|                                                                               |
|  [ Insurance Company Systems ]                                                |
|  Their existing software for pricing policies or handling                     |
|  claims can instantly access our measurement report.                          |
|                                                                               |
|     |                                                                         |
|     V                                                                         |
|                                                                               |
|  [ Final Result ]                                                             |
|  The insurance company can now create a price quote or                        |
|  process a damage claim much faster and more accurately,                      |
|  without needing to send someone to the property.                             |
|                                                                               |
+-------------------------------------------------------------------------------+

   |
   V
(END OF PROCESS)
```



Technical Components :-

1. Mobile App (React Native)

This is the "edge" of our system—the only part the end-user directly interacts with. Its primary job is to be a high-quality data collection tool.

a) React Native

Purpose:
The core framework for building the iOS and Android mobile applications.
Why it's used:
Cross-Platform Efficiency:
We write one JavaScript/TypeScript codebase that compiles to native UI on both iOS and Android. This drastically cuts development time and cost for an MVP, avoiding the need for two separate teams (Swift/Kotlin).
Rich Ecosystem:
It has a massive library of third-party modules, making it easy to access native device features like the camera, filesystem, and AR capabilities.
How it will be used:
We will build the entire UI—buttons, camera view, instructional overlays, and upload progress bars—as React components. We'll use state management (like Redux or Zustand) to handle the flow of capturing and queueing photos for upload.

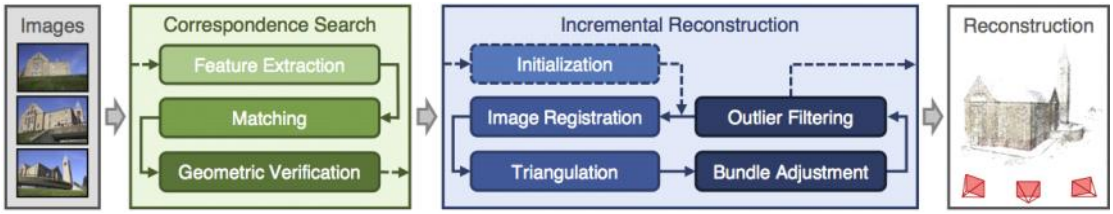
b) ARCore / ARKit (via a React Native module)

Purpose:
To provide real-time positional tracking for the camera. This is used for capture guidance, not for the final 3D model.
Why it's used:
Ensures Data Quality:
The single biggest reason photogrammetry fails is poor image sets (not enough overlap, bad angles). AR provides the 3D context needed to guide the user proactively.
Superior User Experience:
Instead of just telling the user to "take 30 photos," we can show them a "ghost image" of their last shot or a 3D grid around the house, encouraging them to get the 60-80% overlap required by COLMAP.
How it will be used:
We will integrate a library like react-native-arkit or ViroReact. As the user moves their phone, we will query the AR session for the camera's 6DoF (six degrees of freedom) pose. We can then use this positional data to render simple overlays on the screen, providing real-time feedback on their capture technique.

c) OpenCV
Purpose:
To perform on-device, pre-upload image quality analysis.
Why it's used:
Client-Side Validation:
It provides an instant feedback loop to the user and prevents wasting bandwidth and expensive cloud compute time on bad data. Rejecting a blurry photo on the device is cheap and fast; rejecting it after a 30-minute failed cloud process is a terrible experience.
How it will be used:
We will use a library like react-native-opencv-tutorial or a WebAssembly (WASM) version of OpenCV. After a picture is taken, we will pass the image buffer to an OpenCV function to:
Calculate a blur score using a Laplacian variance check. If the score is below a set threshold, we prompt the user to retake the photo.
Check brightness using a histogram analysis to detect under or over-exposed images.

2. Photogrammetry Engine
This is the core computational engine that performs the magic of turning 2D images into a 3D model.

a) COLMAP
Purpose:
COLMAP is an open-source software pipeline. It's not one single program, but a set of tools that run in a specific order. Its one and only job is to take a collection of overlapping 2D images and produce a geometrically accurate 3D model.
How it works:
- Feature Extraction (Finding unique clues)
- Feature Matching (Connecting the clues)
- Structure from Motion or SfM (Building the 3D skeleton)
- Dense Reconstruction (Fleshing out the model)
- Meshing & Texturing (Giving it a solid skin)



COMAP generates output of 3D cloud scene in ".ply" format.
Why it's used:
Accuracy & Control:
It is the academic and open-source gold standard for geometric accuracy. It provides granular control over every step of the reconstruction process, which is essential for achieving survey-grade results.
No Licensing Fees:
As a free and open-source tool, it allows us to build a scalable backend without the per-use costs of a

commercial SDK like RealityCapture or Metashape.
How it will be used:
COLMAP is a set of command-line executables. Our backend processing script will orchestrate calls to these executables in a precise sequence, using the output of one step as the input for the next. A typical workflow script would execute:
colmap feature_extractor: To find SIFT features in each image.
colmap exhaustive_matcher: To match those features across images.
colmap mapper: To perform the SfM reconstruction, creating a sparse point cloud and calculating camera positions.
colmap image_undistorter: To prepare images for dense reconstruction.
colmap patch_match_stereo: To create the dense point cloud.
colmap poisson_mesher: To convert the dense cloud into a solid mesh.

b) Three.js
Purpose:-
Three.js is a **JavaScript library** that helps you interact with **3D graphics on the web** (generated by photogrammetry) using your browser. It simplifies WebGL (Web Graphic Library)
Why it's used?
Web based accessibilty
Enables interaction and measurement
Performance and great ecosystem
How it will be used?
Scene --> Mesh --> Lights --> Camera --> Render

3. Cloud Backend
This is where the heavy lifting, storage, and automation happens. It's designed to be scalable and cost-effective.

a) GCP / AWS
Purpose:
The foundational cloud platform providing all the necessary infrastructure services.
Why it's used:
On-Demand Power: Provides access to high-end GPU compute instances, which are essential for photogrammetry.
Managed Services: Offers services for storage, databases, container orchestration, and more, which saves significant DevOps time compared to managing our own physical servers.

b) GPU Compute Instances (e.g., Google Compute Engine with NVIDIA T4/V100)
Purpose:
The virtual servers that will actually run the COLMAP processing jobs.
Why it's used:
Massive Parallelism: The dense reconstruction and feature matching steps are highly parallelizable tasks. A GPU can perform these calculations orders of magnitude faster than a CPU, reducing processing time from hours to minutes.
How it will be used:
These will be provisioned ephemerally. A new job will request a GPU instance, the processing will run, and the instance will be terminated upon completion to minimize costs.

c) Docker
Purpose:
To create a portable, self-contained environment for our entire processing pipeline.
Why it's used:
Dependency Management & Reproducibility:
COLMAP has many specific dependencies (CUDA, C++, various libraries). A Docker container bundles the OS, all dependencies, COLMAP itself, and our Python scripts into a single, immutable image. This guarantees that the process will run identically every single time, regardless of the underlying server.
How it will be used:
We will write a Dockerfile that defines the environment. This image will be built and stored in a container registry (e.g., GCR or ECR). Our orchestration service will simply pull and run this pre-configured container for each job.

d)GKE (Google Kubernetes Engine) or Cloud Run
Purpose: The "brain" that automates and scales our processing jobs.

Why it's used:
Automation & Scalability:
We need a system to automatically trigger a new job when photos are uploaded. Cloud Run is excellent for a simple, event-driven approach (e.g., "on new file in bucket, run one container"). GKE is more powerful for managing a persistent cluster that can handle hundreds of concurrent jobs, auto-scaling the number of GPU nodes up and down based on demand. For the MVP, Cloud Run is likely sufficient and simpler.
How it will be used:
We will configure a Cloud Run service to be triggered by a new folder being created in a Google Cloud Storage bucket. The service will be configured to run our Docker container, passing the path to the new images as an environment variable.

4. Integration Layer

This layer makes our system's output useful to other software.

a) REST API (e.g., using FastAPI in Python)

Purpose:
To provide a secure, structured "front door" for external systems to request data from our platform.
Why it's used:
Standardization:
REST is the universal standard for web-based communication. Every modern programming language and platform can easily interact with a REST API.
How it will be used:
We will build a small web service with endpoints like GET /api/v1/projects/{id}/measurements. When an authenticated insurance system makes a request to this endpoint, our API will fetch the corresponding JSON file from our Cloud Storage and return its contents.

b) Webhooks

Purpose:
To proactively notify external systems when a job is complete.
Why it's used:
Efficiency & Real-Time Updates:
The alternative to webhooks is "polling," where the external system has to repeatedly ask our API, "Is it done yet?". This is inefficient. Webhooks reverse this; our system tells them the moment the data is ready.
How it will be used:
The very last step of our cloud processing script (after saving the 3D model and JSON file) will be to make a simple HTTP POST request to a URL provided by the client's system. This request will carry a payload like {"projectId": "xyz-123", "status": "SUCCESS", "dataUrl": "/api/v1/projects/xyz-123/measurements"}.

Open Sources Repos (For MVPs) :-

1. https://github.com/colmap/colmap :- COLMAP (For 3D modelling)
2. https://github.com/mrdoob/three.js/ :- Three.js (Contains many inbuild use-cases of three.js library including hittest ie measuring distance using webXR)