March 15, 2019

Exploring *Augmented Reality* for mobile

Srikar Mutnuri
XR Developer, TCS Interactive

Vision

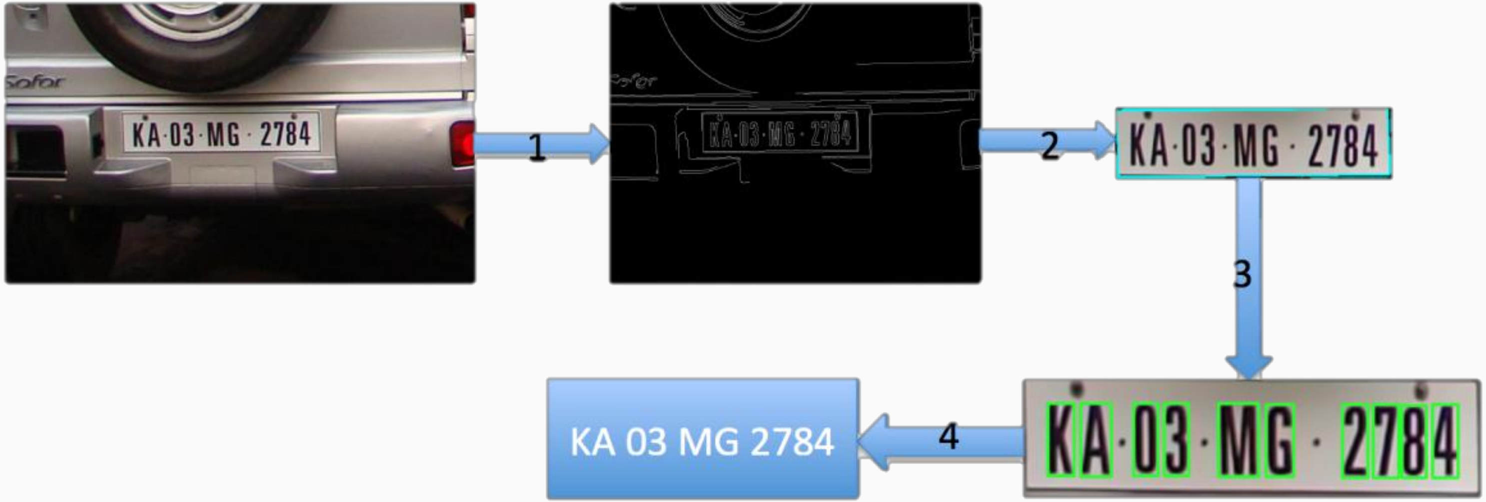
Vision

At a high level, given an image, do the following to gain insights about it:

- **Acquire** – sensors, camera
- **Process** – image processing
- **Analyze** – image analysis

Image Source: PyImageSearch

(Computer) Vision



1. Histogram + Canny edge detection (**Process**).
2. Crop the detected number plate (**Process**)
3. Detect character locations (**Analyze**)
4. Perform OCR (**Analyze**)

Milgram’s Mixed Reality Spectrum

Combination of AR, VR and MR is generally called Extended Reality (XR)

(Or the *X* could just be a filler…)

Source: Microsoft Docs.

Augmented Reality

Augmented Reality (AR) n.

“A technology that superimposes a computer-generated image on a user's view of the real world, thus providing a composite view.”

AR: Defining Characteristics

- Blending of the real with the imaginary
- Interacting real-time with virtual content
- Virtual objects are either fixed or have predictable movements

Pokémon Go (Source: Niantic)

Source: Azuma, R.T (1997), ***A Survey of Augmented Reality*** Presence, 6(4), 355 - 385

Doing proper AR in real world is chaotic…

A lot can change while you’re using an AR app:

- Camera angle/perspective
- Rotation
- Scale
- Lightning
- Blur from motion or focusing
- General image noise

Source: Blogpost by Andreas Jakl

Vision + Augmented Reality

So what makes AR possible?

```
while(condition):
        update tracking data,
        update environment data,
        poll input for object placement/interactions
        update placed virtual objects
```

So what makes AR possible?

These two steps can often be simultaneous (SLAM):

- Update tracking data
- Update environment data

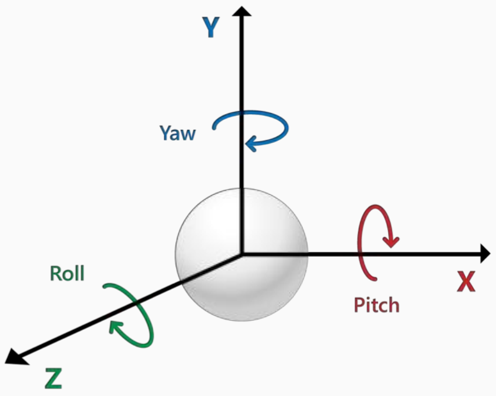
We’ll see them together here.

Update understanding

AR requires tracking to really work well.

Continually locating/updating the user's viewpoint when in motion involves:

- ***Positional tracking***: (x, y, z) coordinates
- ***Rotational tracking***: roll, pitch & yaw - (r, p, y)

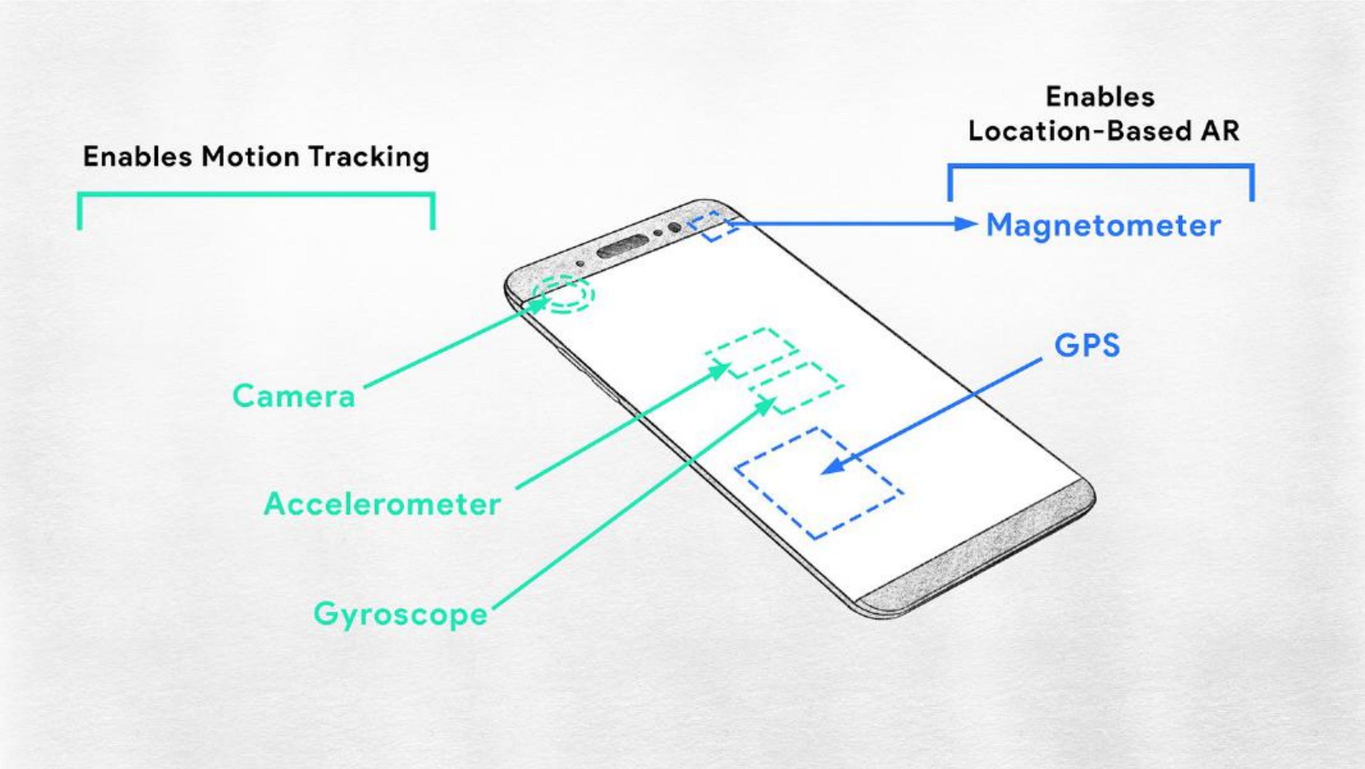


Known as *Pose Estimation* in XR speak.

A common co-ordinate system is important!

Update understanding

Sensors help!



Hardware that enables inference

Source: Coding Blocks, Medium article

Update understanding

Shouldn't there be some reference points to do all this?

YES!

Keypoints are the key here:
distinctive locations in images - corners, blobs or T-junctions.

Together, they describe ***features*** of the environment.

Properties:

- ***Reliability*** – a feature should be reliable always.
- ***Invariance*** –same point in different views.

Update understanding

Some SLAM algorithms used to identify the keypoints to track a feature reliably:

- SIFT – Scale Invariant Feature Transform
- SURF – Speeded up robust features
- BRISK – Binary Robust Invariant Scalable Keypoints

Update understanding

Two parts to any algorithm:

- **Keypoint detection** : detect sufficient keypoints to understand environment well
- **Keypoint description** : give unique fingerprint to each keypoint detected.

Keypoints == Spatial Anchors

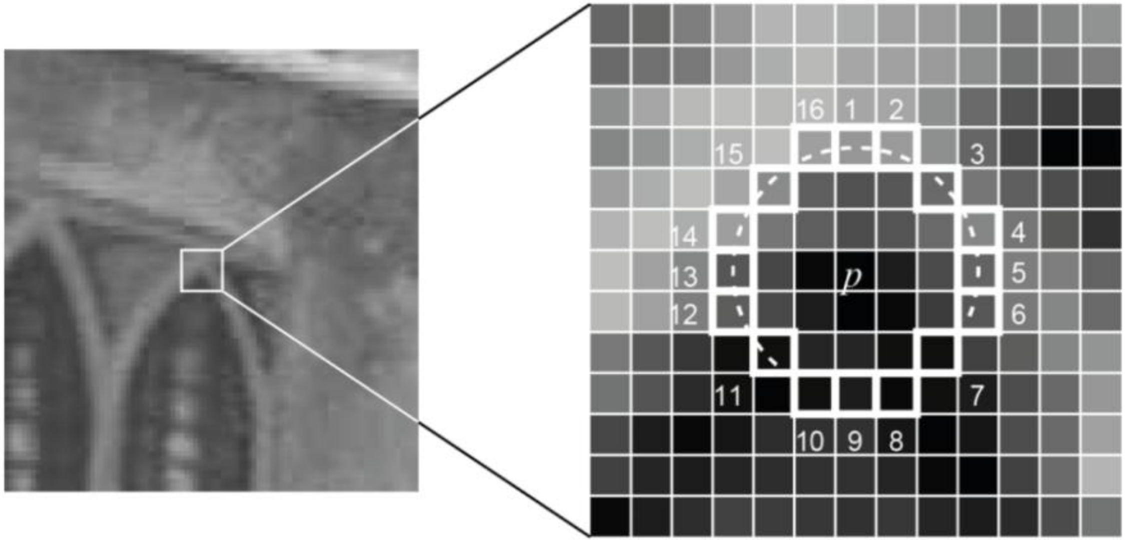
Should happen each frame!

Update understanding

For *BRISK*:

Keypoint detection:
At least 9 pixels should be brighter or darker than ***p***

Keypoint description:
Create binary string with 512 bits, with comparison results



FAST algorithm by Rosten and Drummond. *Image credits: Rosten E., Drummond T. (2006) Machine Learning for High-Speed Corner Detection. In: Leonardis A., Bischof H., Pinz A. (eds) Computer Vision – ECCV 2006. ECCV 2006. Lecture Notes in Computer Science, vol 3951. Springer, Berlin, Heidelberg*

FAST Algorithm, the basis of **BRISK**

Update understanding

Reference Image

Update understanding

Some processing later…

Update understanding

Tracking Points, aka ***Trackables***

Update understanding

For tracking, create database of **all** keypoints

Look at corners at multiple scales:

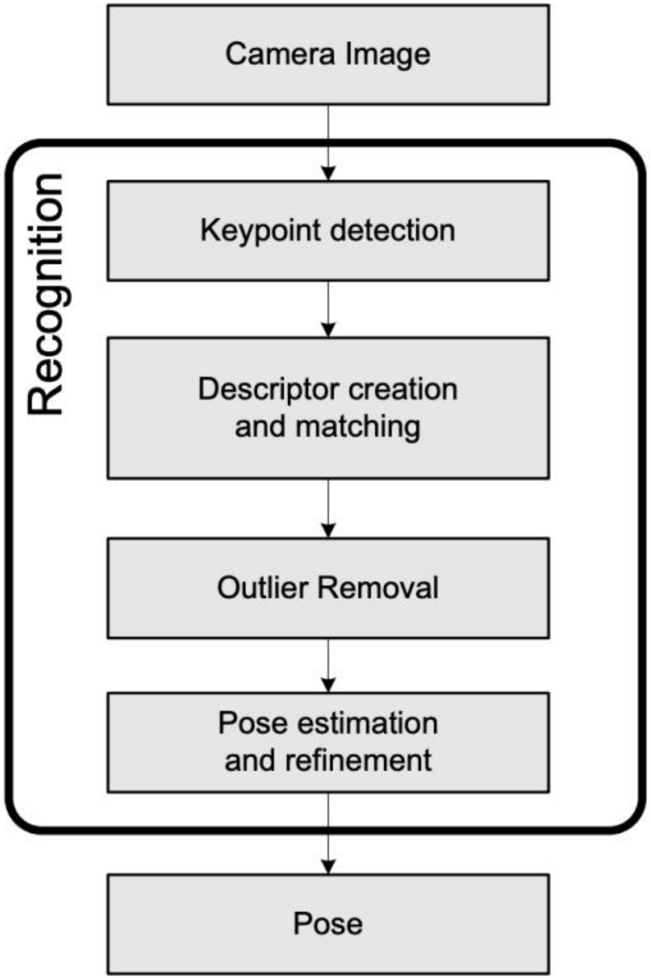
varied scale => more descriptive!

Keypoints : more the better… Expensive to track too!

Update understanding

- Error correction is an important step in maintaining the pose.
- Removal of outliers helps in pose refinement: Simple geometry-based or maybe homography-based (2D-3D relationships)
- Use remaining keypoints to calculate the pose

Source: COMP4010, Bruce Thomas, University of South Australia



So what makes AR possible?

```
while(condition):
        update tracking data,
        update environment data,
        poll input for object placement/interactions
        perform error correction
        update placed virtual objects
```

Update understanding – other factors

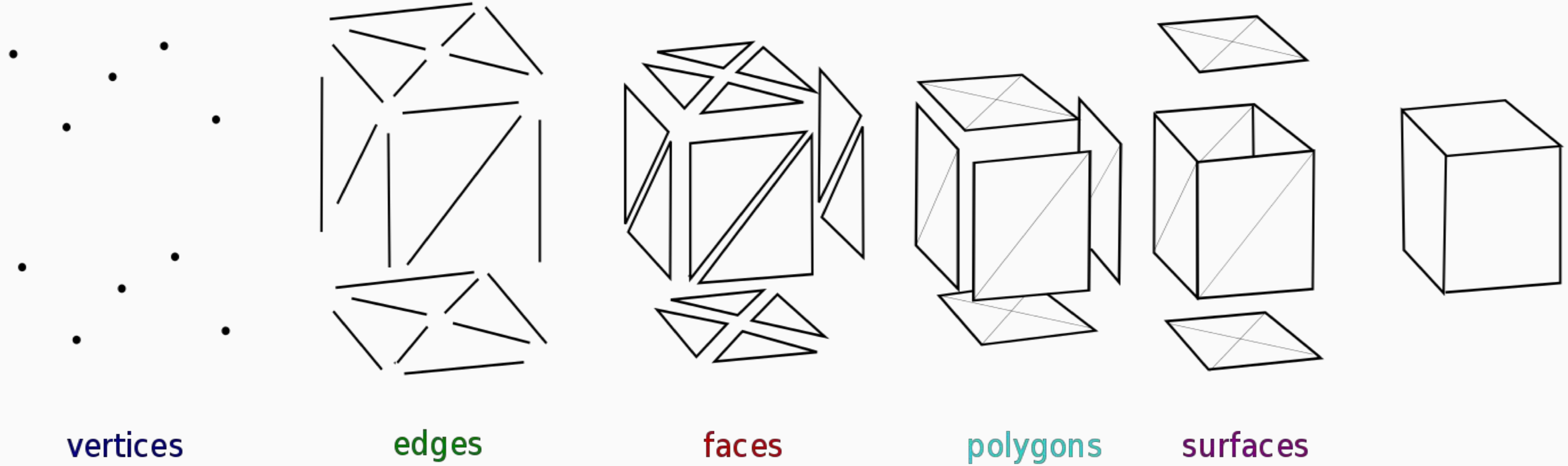
There are some other important parts to enabling a good AR experience:

- Lighting of the surrounding environment
- User interactions (hit testing, raycasts etc.) with the virtual objects
- Oriented points - to place objects on angled surfaces
- Occlusion – hide objects when behind something else

Source: COMP4010, Bruce Thomas, University of South Australia

Virtual objects

All objects are made up of meshes



From points to meshes

Image source: Wikipedia

Virtual objects

Environmental understanding and tracked pose might effect the following:

- Shadows casted by/upon
- Lighting intensity
- Orientation/size in space
- Amount of object visible

The last part, aka Occlusion, is especially difficult. We'll see this in detail soon.

Image source: ARCore documentation

How do AR libraries handle these?

Some libraries...

Many libraries exist for Augmented Reality:

- ARCore (Google)
- ARKit (Apple)
- ARFoundation (Unity)
- Vuforia (PTC/Qualcomm)
- EasyAR
- Selerio
- 6D.ai

Exclusive behind the scenes from your AR app…

On launching an AR app:

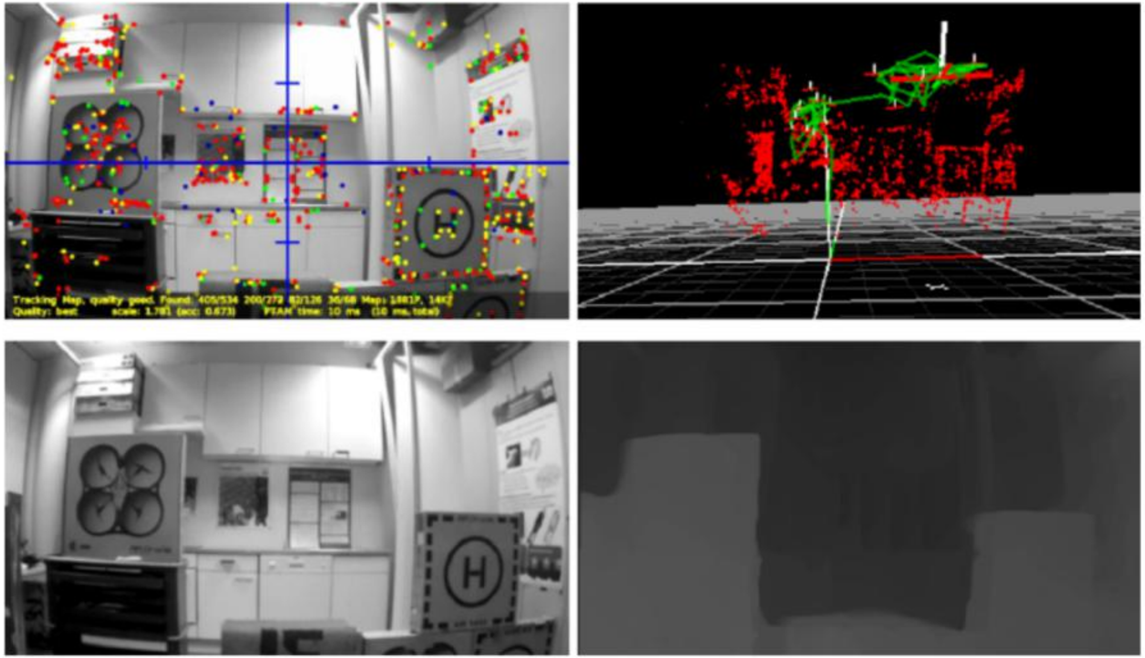
- Scan the surroundings
- Check for matches with pre-downloaded keypoint maps and initialize a new map if needed
- Update map with movement in scene
- Use this data to create experience

Bigger maps → more computations to manage!

Bit of a problem child.

Source: 6D.ai blog on Medium

Current SOTA



A sparse map might look something like the top right image. The top left shows how the points match the real world (colors are used to indicate how reliable that point is). Bottom left is the source image. Bottom right is an intensity map, which can be used for a different type of SLAM system (semi-direct—which are very good by the way, but aren't yet in production SLAM systems like ARCore or ARKit yet)

Image source: 6D.ai blog on Medium

Now to the tricky part – problems!

Some major issues in current AR technologies:

- **Improper Occlusion** – objects are not hidden when they are supposed to
- Depth distortions – unreliable depth sensor data
- Inaccurate tracking data - drifting objects

- **Performance drops** – too data much to process

Could have potential workarounds, rarely work in real scenarios.

An interesting paper reference:
“**Perceptual issues in augmented reality revisited**” by E. Kruijff, J. E. Swan II, and S. Feiner, IEEE International Symposium on Mixed and Augmented Reality, Oct 2010.

An example of improper occlusion

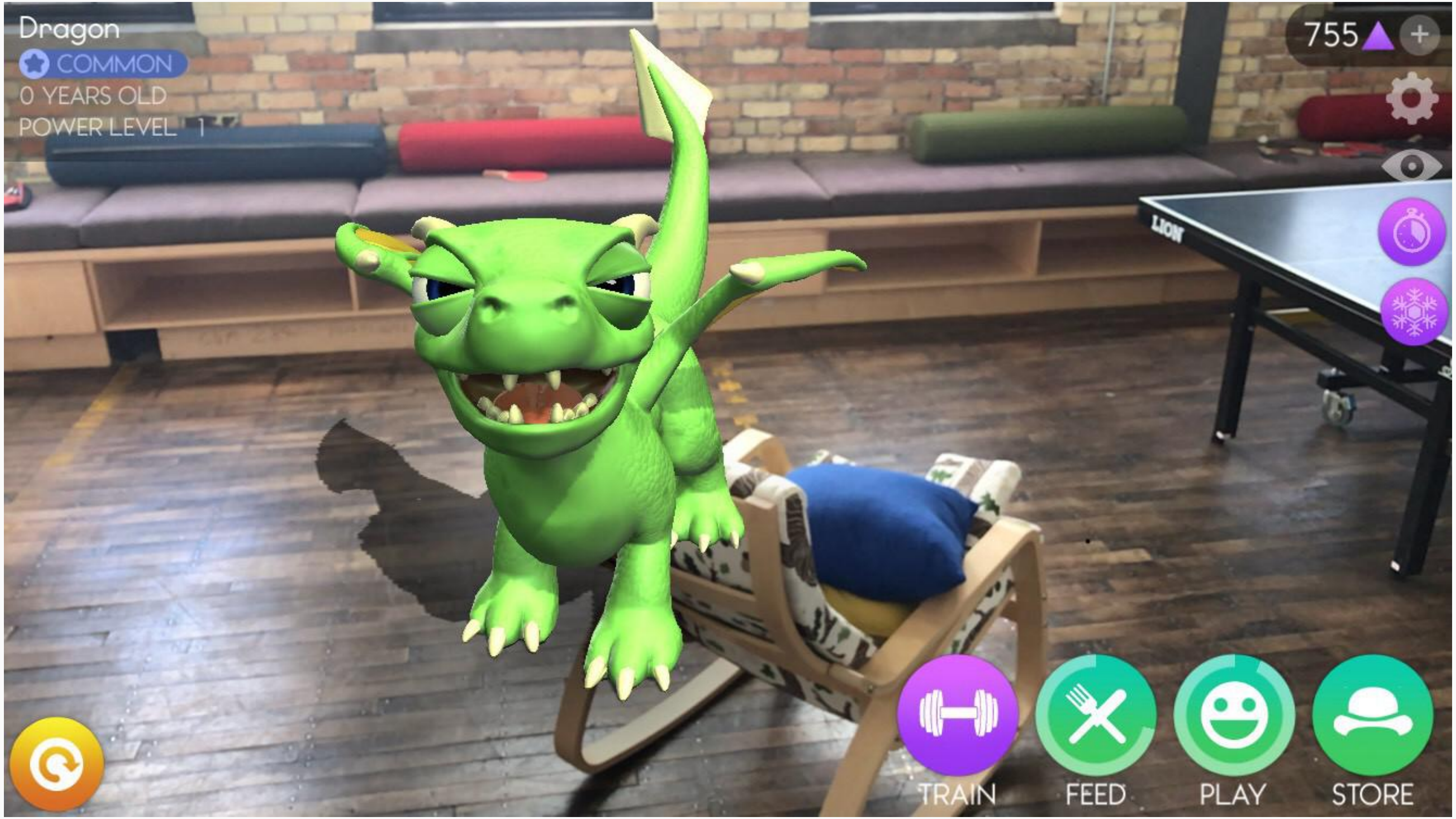


Image source: Hacker noon blog on Medium

(Attempting to) Solve Occlusion

To handle Occlusion, do these:

- Use depth cameras to get a 3D location of each pixel (trivial?)
- Aggregate the depth info across frames to generate a 3D geometry of the scene

The catch: most phones don't have a depth sensor to begin with.

Source Selerio blog on Medium

(Attempting to) Solve Occlusion

3D reconstruction can be split into two sub-problems:

- Generate 3D depth information from 2D images (dense point clouds/depth images)
- Integrate this over several frames to generate meshes in real time

Point cloud → 3D mesh → Occlude!

Still a hard task with more performance drops!

Source Selerio blog on Medium

A good solution?

??

AI in AR ☺

The task at hand...

Do the following in real time, on a mobile device:

- Detect depth in a given image
- Given the depth data generate a mesh out of it.

Way(s) to do this...

Some SDKs already handle this using neural networks along with other tools.

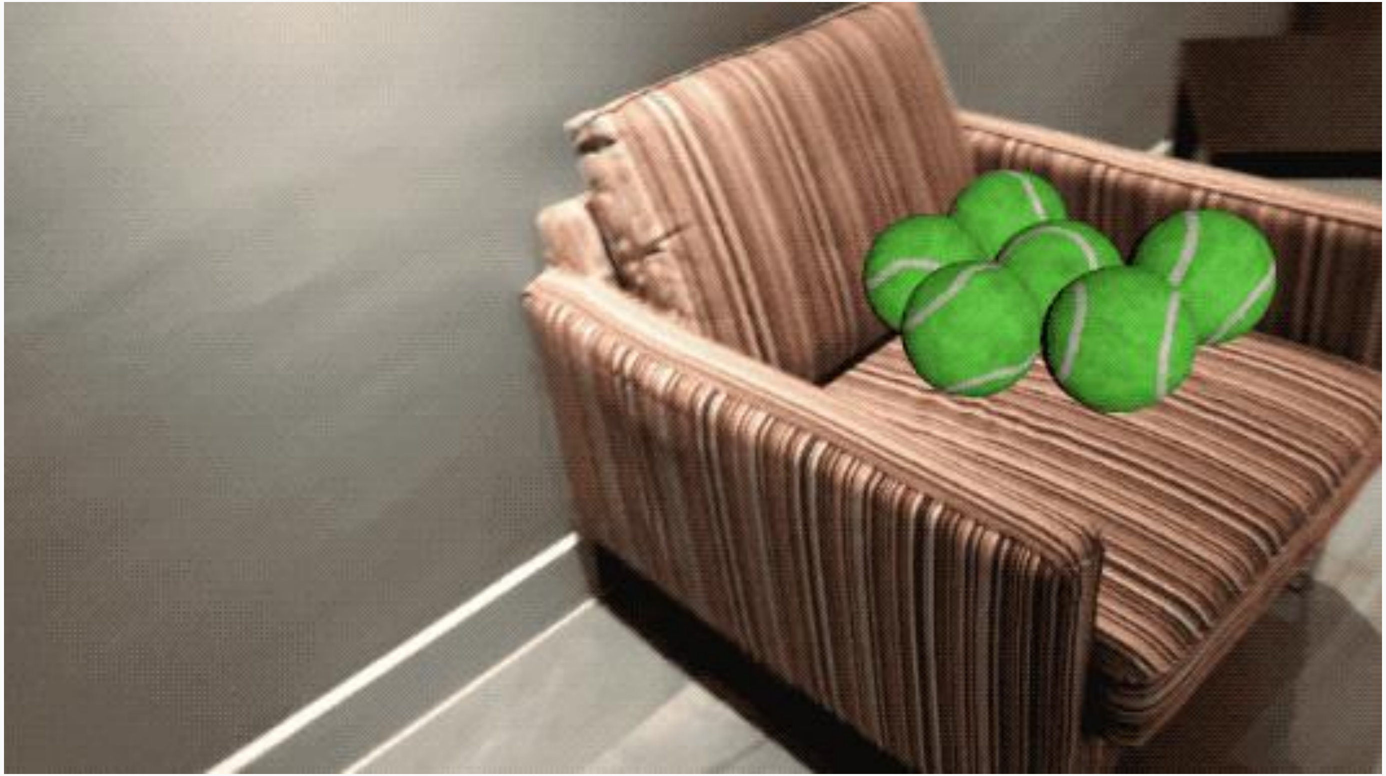
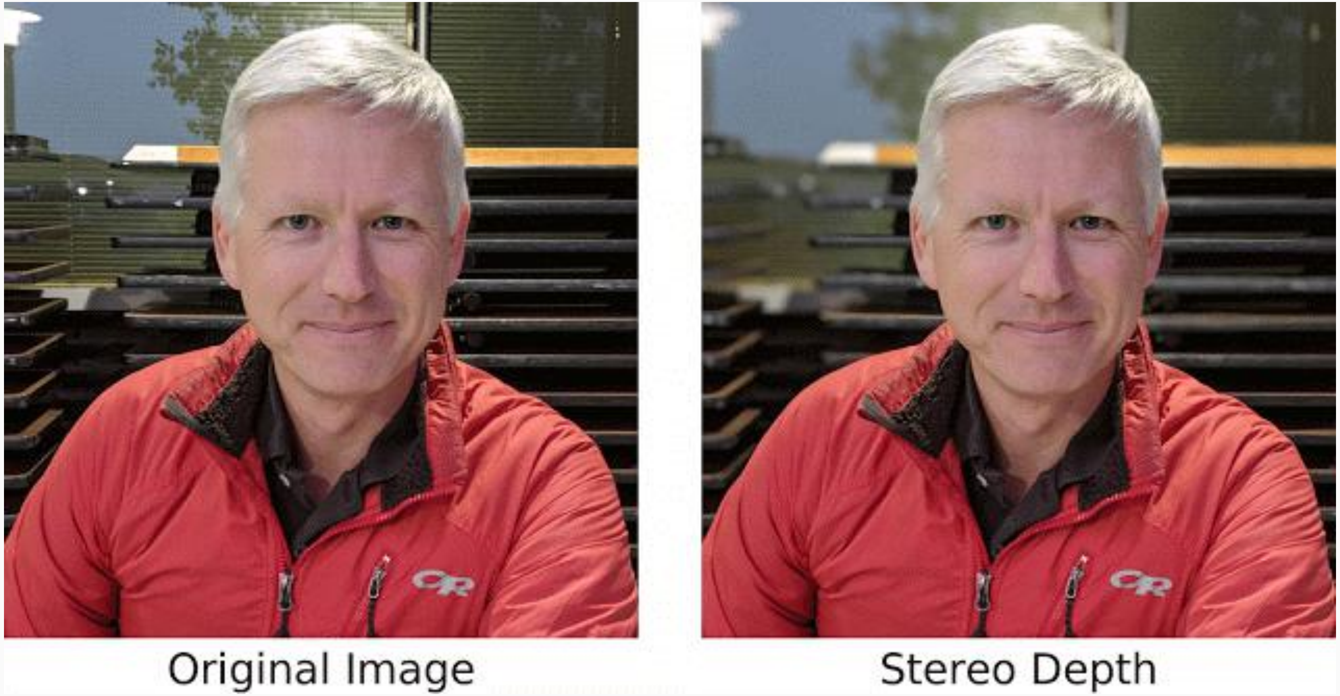


Image source: Selerio blog on Medium

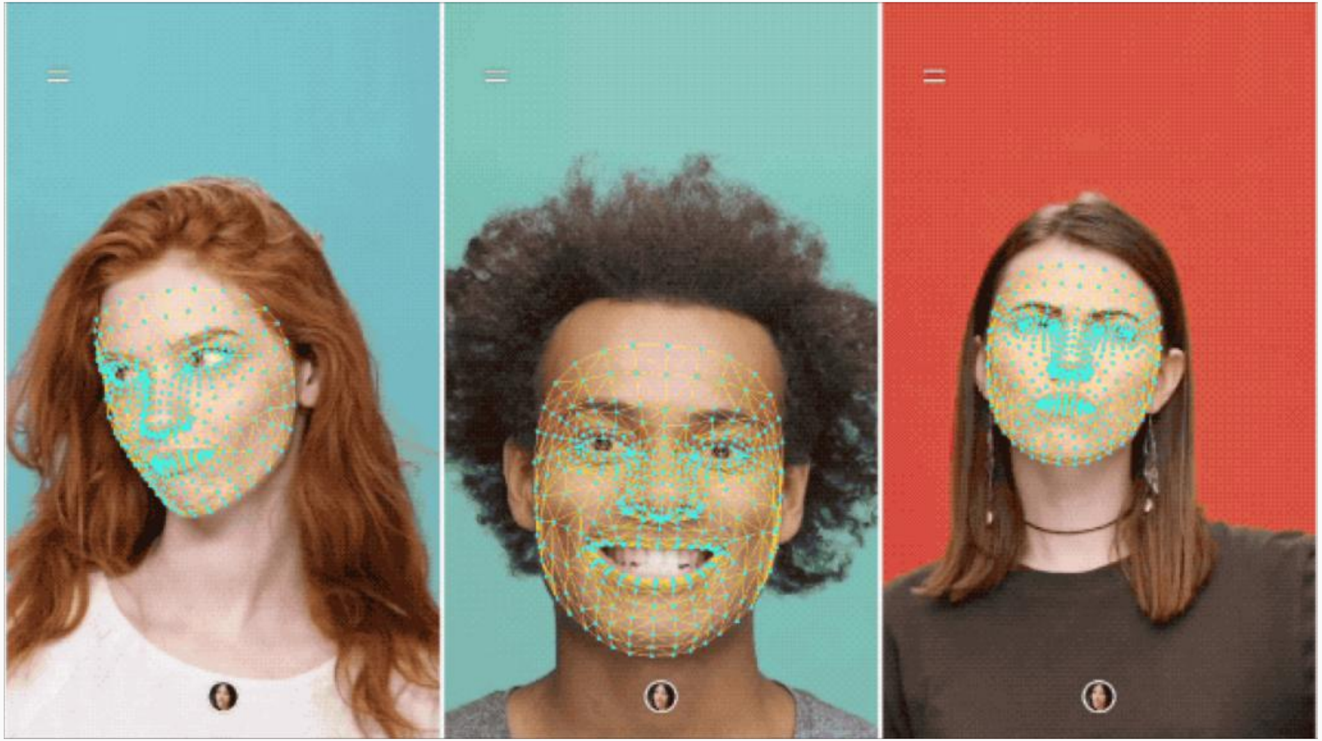
A slight detour - part one:



Portrait mode in Pixel phones

Source: Google AI Blog

A slight detour - part two:



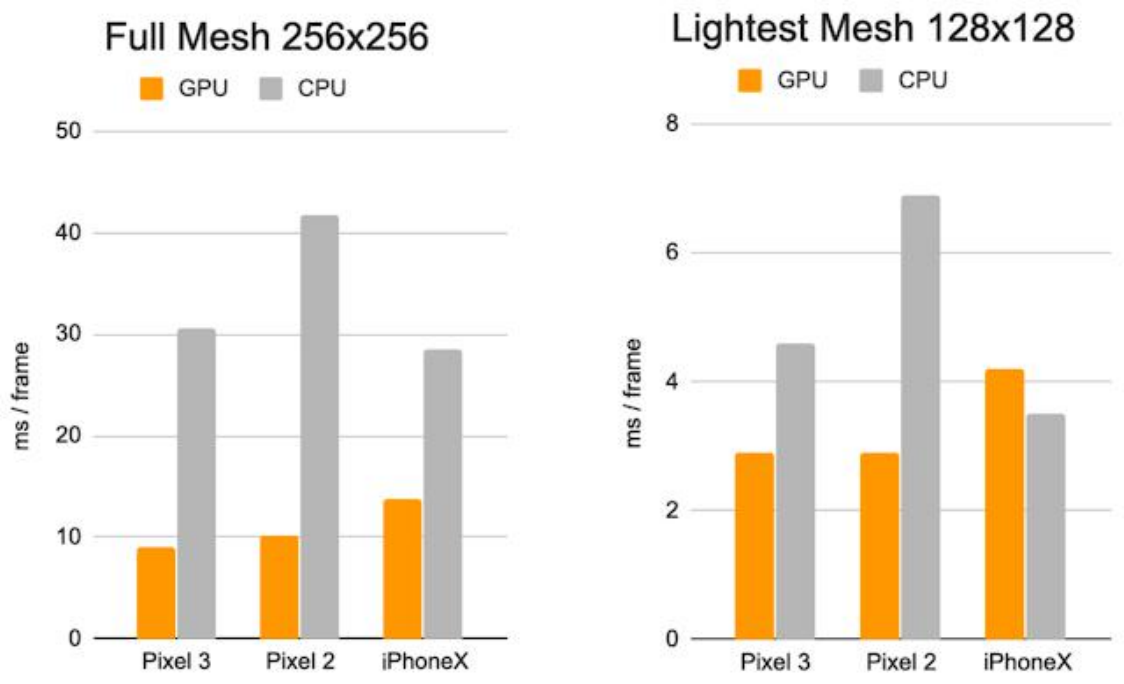
Face tracking for ARCore Augmented Faces

Source: Google AI Blog

The common denominator

TensorFlow Lite

Comparison chart...



Inference times per frame: CPU vs GPU for Augmented Faces

Source: Google AI Blog

Conclusion

Conclusion – the hype cycle

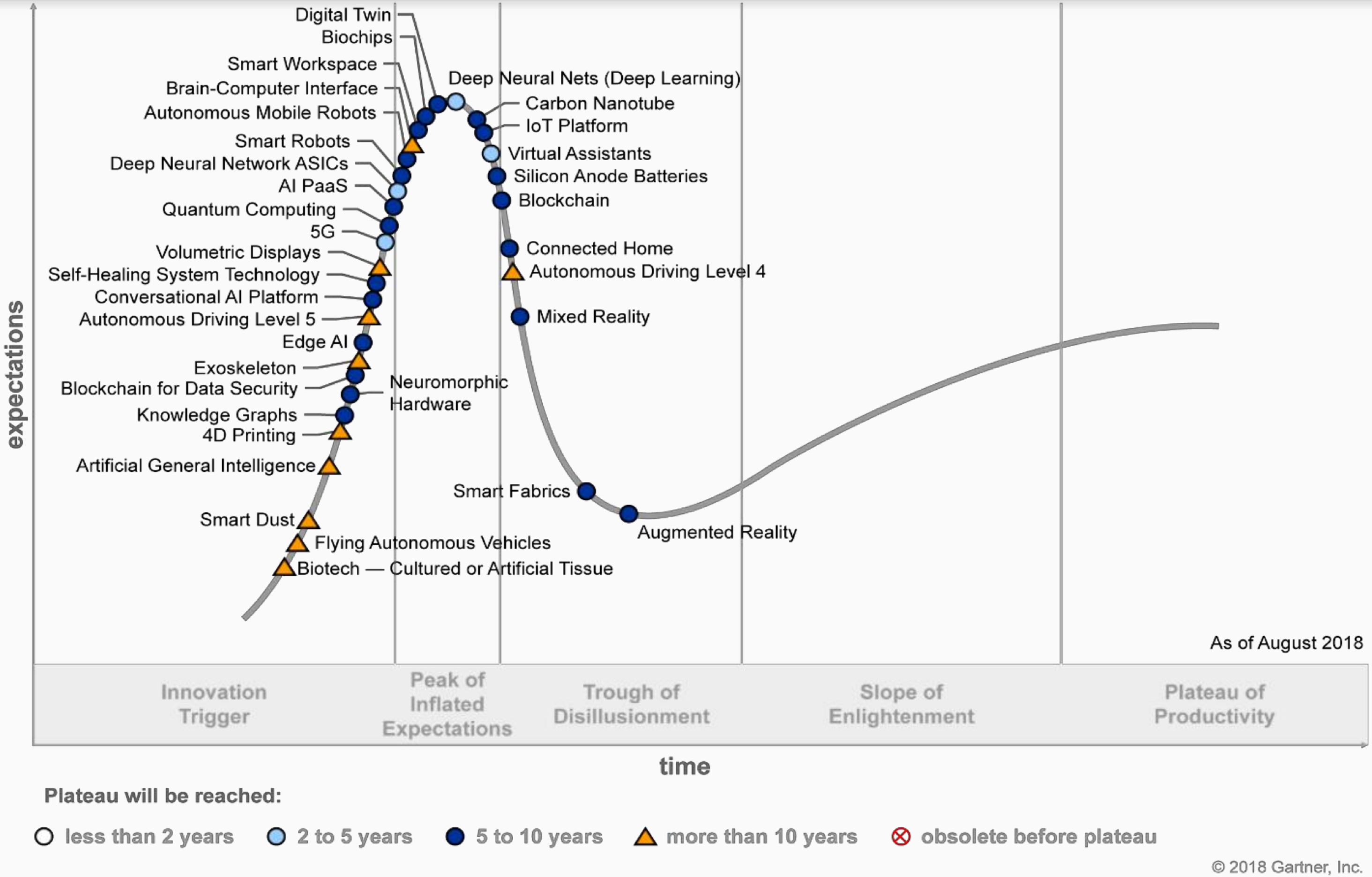


Image Source: Gartner

Some (great) references

References:

(in no particular order)

- https://medium.com/6d-ai/how-is-arcore-better-than-arkit-5223e6b3e79d
- https://medium.com/6d-ai/why-is-arkit-better-than-the-alternatives-af8871889d6a
- https://ai.googleblog.com/2019/03/real-time-ar-self-expression-with.html
- https://ai.googleblog.com/2018/11/learning-to-predict-depth-on-pixel-3.html
- https://medium.com/coding-blocks/introduction-to-motion-tracking-in-arcore-f3e584ce0ba0
- http://stanford.edu/~rqi/pointnet/docs/cvpr17_pointnet_slides.pdf
- https://arxiv.org/pdf/1801.07829.pdf
- https://medium.com/@Umbra3D/a-scalable-pipeline-for-processing-massive-point-clouds-8b96a433d8d5
- http://www.cs.cornell.edu/projects/megadepth/
- https://medium.com/selerio/occlusion-is-important-in-ar-9a672ff9ca2d

References:

(in no particular order)

- http://blog.guidigo.com/blog/how-to-work-around-occlusion-issues-in-augmented-reality/
- https://hackernoon.com/a-machine-and-humans-perception-of-the-world-in-augmented-reality-2342f4a6750
- http://visual.cs.ucl.ac.uk/pubs/cofusion/index.html
- http://visual.cs.ucl.ac.uk/pubs/learningOcclusion/
- http://vision.gel.ulaval.ca/~jflalonde/pubs/papers/lalonde_wio_18.pdf
- https://arxiv.org/pdf/1703.09771.pdf

Thank you!

Feel free to reach out on Twitter: @itsmutnuri