# Augmented Reality using Computer Vision

Instructor - Simon Lucey

**16-623 - Designing Computer Vision Apps**

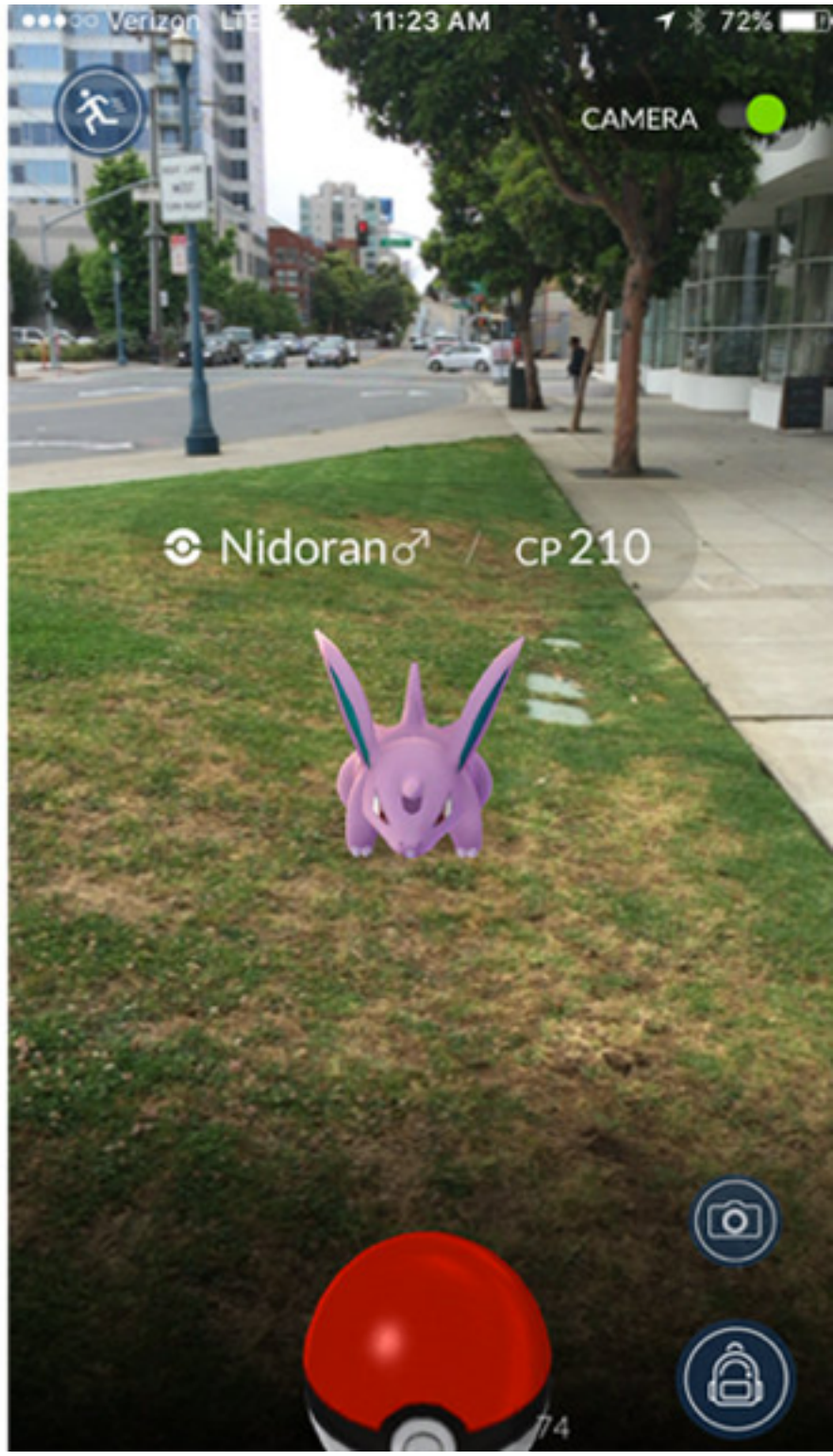
Carnegie Mellon
THE ROBOTICS INSTITUTE

# Today

- Augmented Reality
- Review: Homographies & Pinhole Cameras
- ARToolkit in iOS

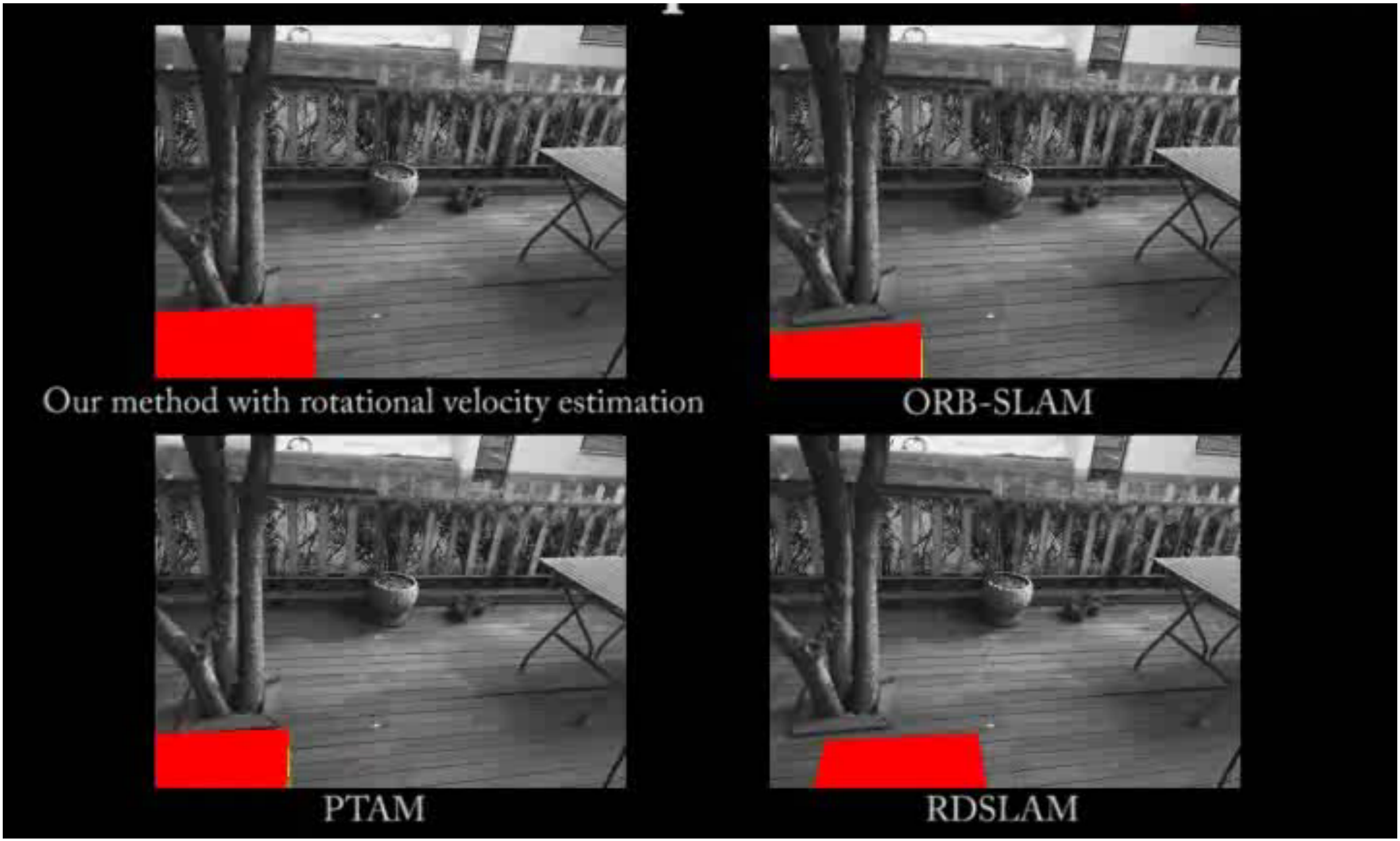

Voltorb
CP 282
CAMERA


Verizon LTE
11:21 AM
74%
Growlithe
CP 472
CAMERA


11:23 AM
72%
Nidoran♂
CP 210
CAMERA

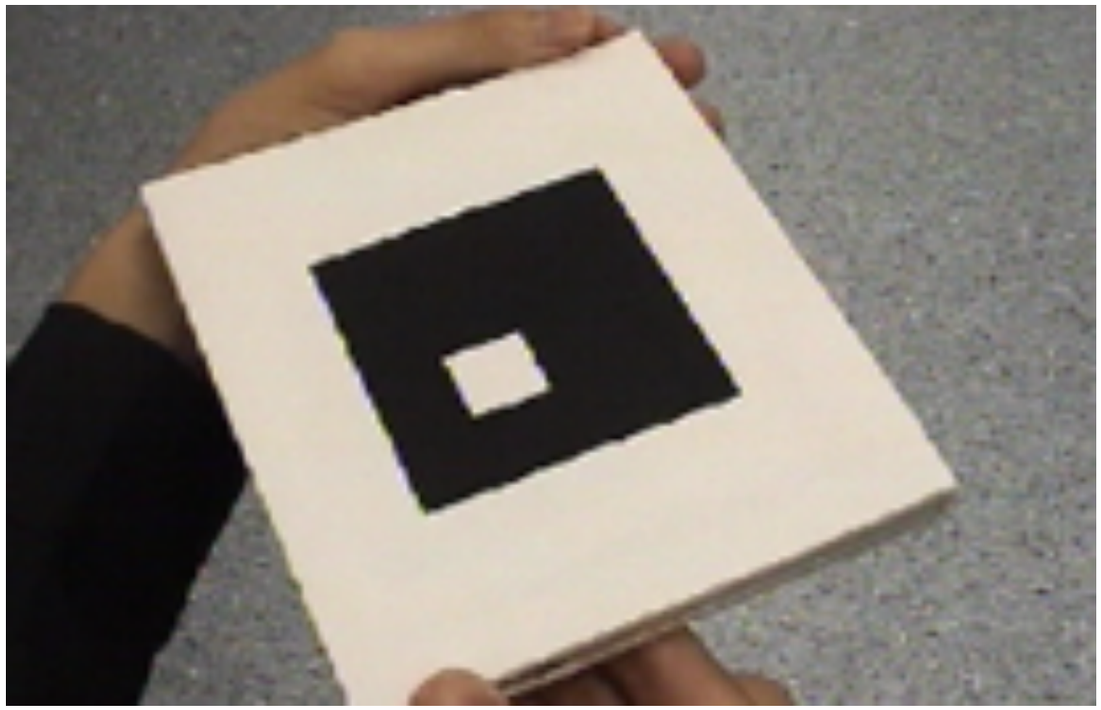

Pokémon
GO

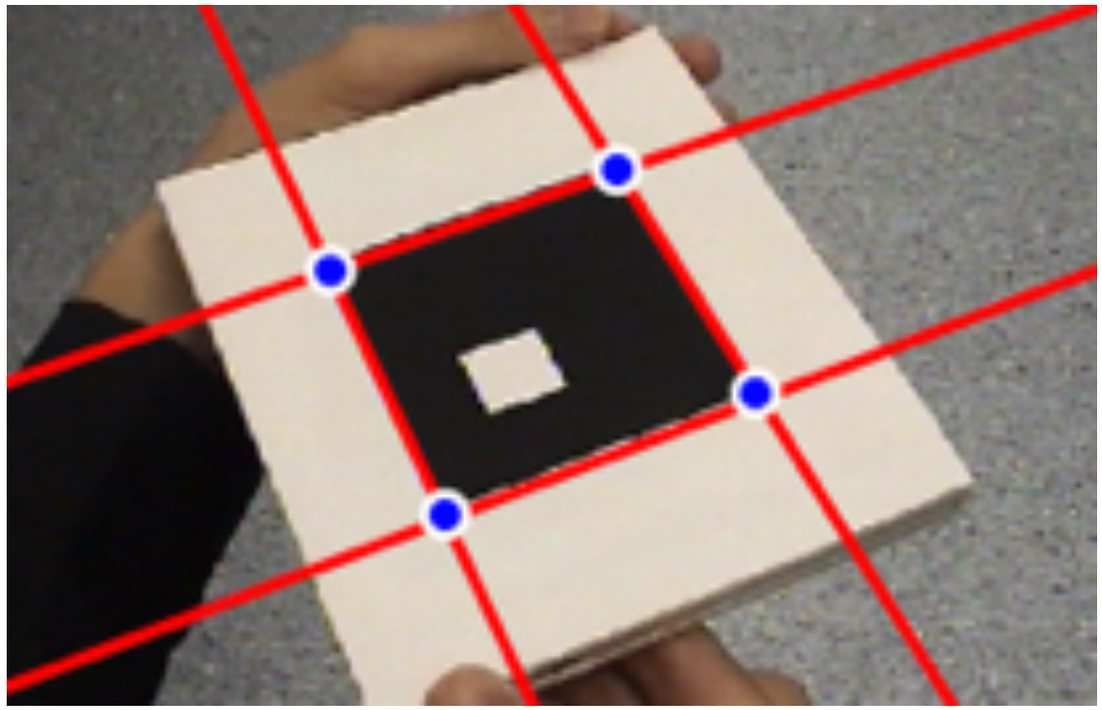
# Example of SLAM for AR

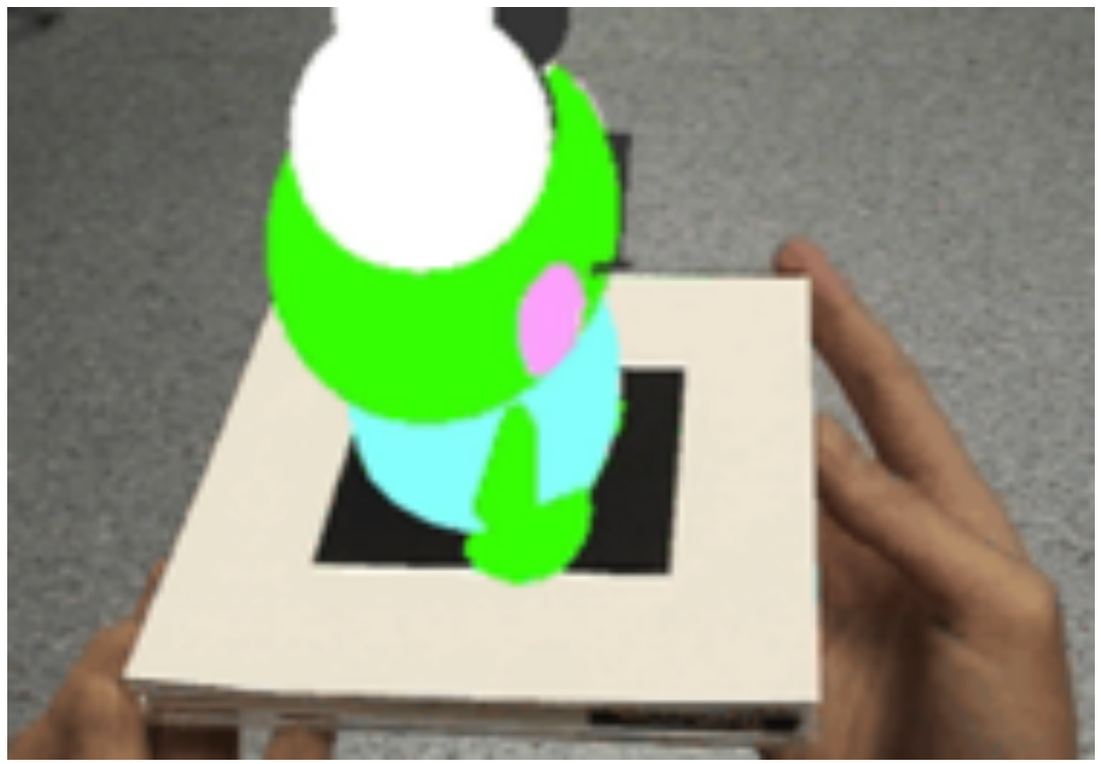



Taken from: H. Liu et al. “Robust Keyframe-based Monocular SLAM for Augmented Reality”, ISMAR 2016.

# Motivation

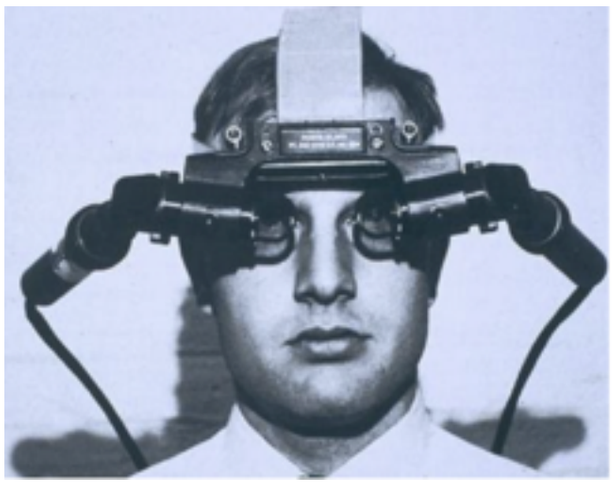






Adapted from: Computer vision: models, learning and inference.  Simon J.D. Prince

# Short AR History



"Sword of Damocles"
Ivan Sutherland

- Head mounted displays were first developed in the 1960s, and is considered the first step in making AR possible.

Thomas P. Caudell

- Tom Caudell first coined the term while working on a project for Boeing in the early 1990s.
- Used the term to describe a digital display used by aircraft technicians that blended virtual graphics onto a physical reality.

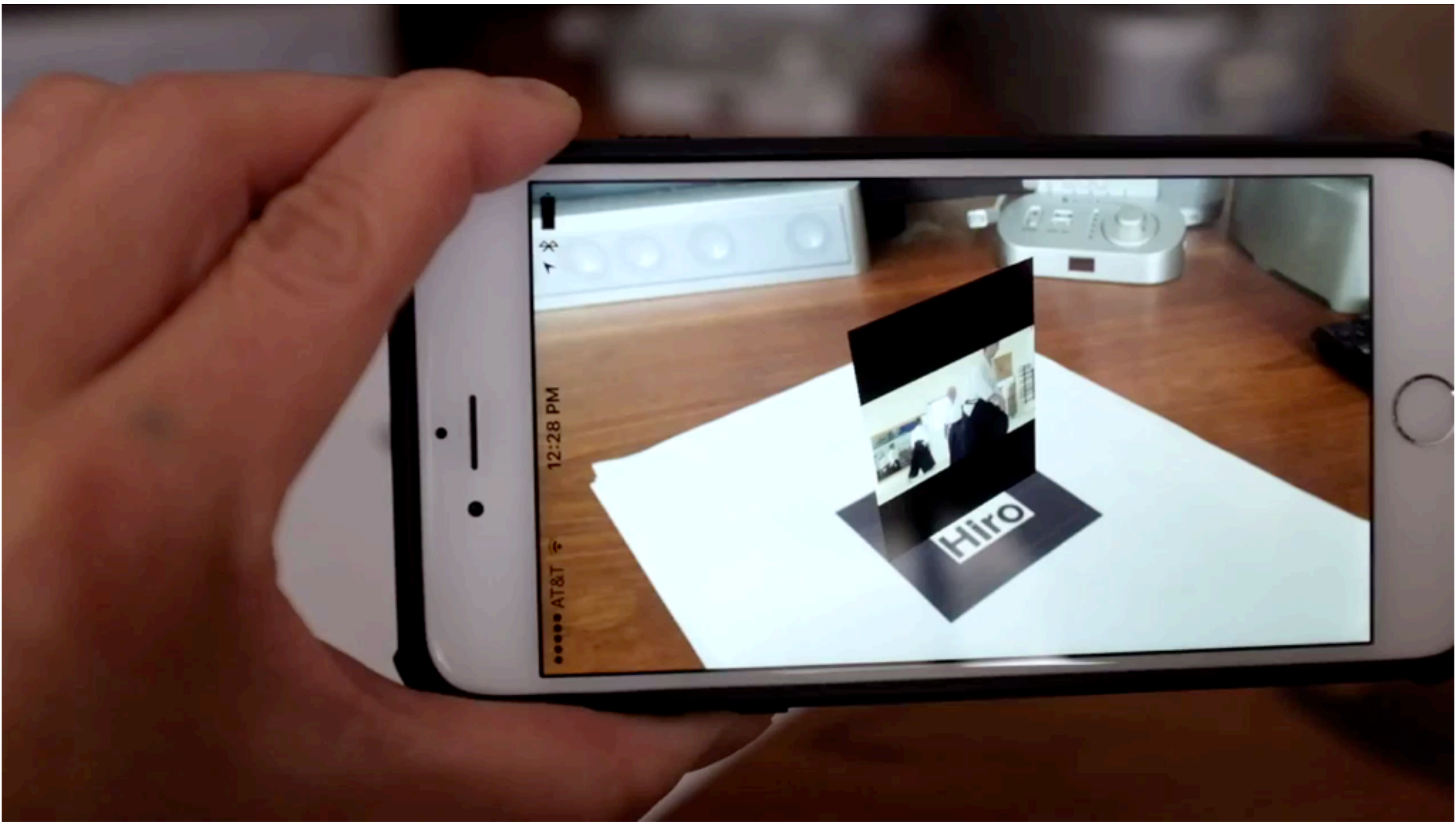
# Short AR History


AR Toolkit

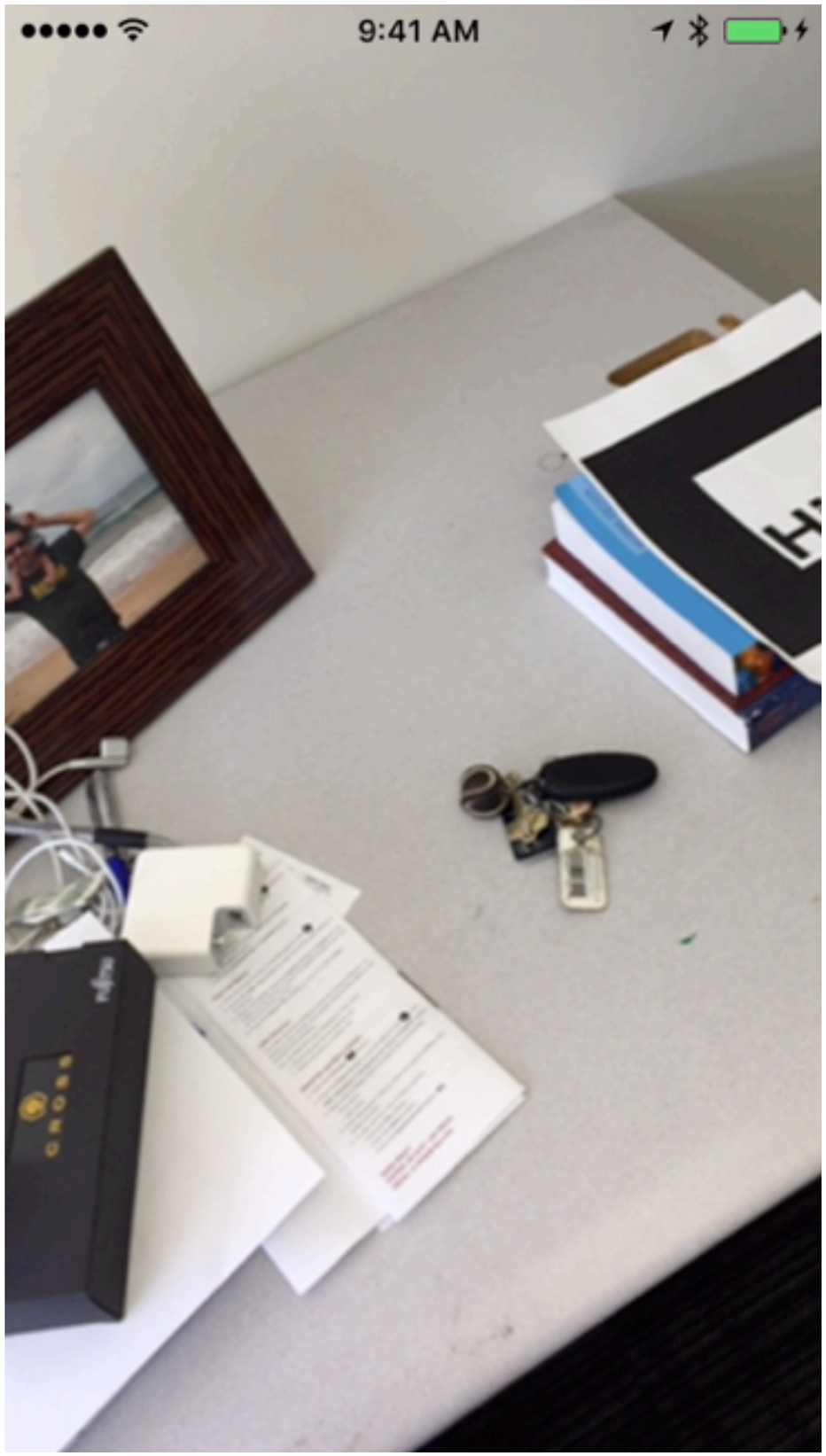
Hirokazu Kato

- In the late 1990s, Hirokazu Kato of the Nara Institute of Science and Technology developed ARToolkit.
- Originally released by the University of Washington's HIT Lab to the open source community.
- ARToolkit probably the most well known and commonly used package for AR.
- ARToolkit now owned and maintained by West Coast startup DAQRI.

# ARToolkit Example



# ARToolkit My Example



# Other SDKs for AR

- Vuforia is another popular SDK for AR development.
- Like ARToolkit is portable for Android and iOS.
- Lots of nice examples on their developer portal.
- Check out more at - `https://developer.vuforia.com/`

# Google Glass

# The Future?? - DAQRI Smart Helmet

# AR versus VR

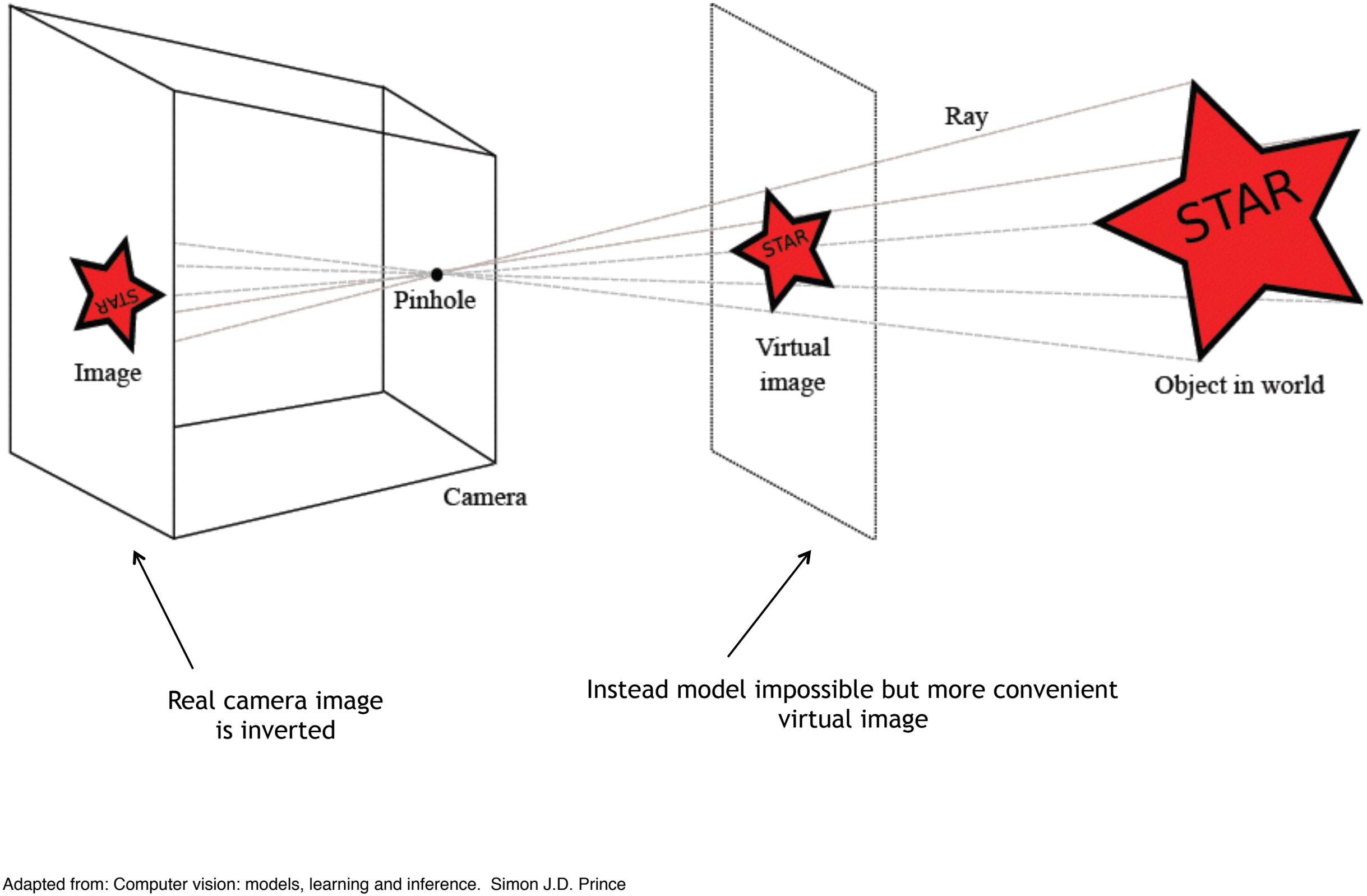
**Augmented reality** will further blur the line between what's real and what's computer-generated


Taken from http://www.slideshare.net/brainberryglobal/augmented-reality-meetup-in-kiev-hakan-mutlu-sonmez.

# Today

- Augmented Reality
- Review: Homographies & Pinhole Cameras
- ARToolkit in iOS

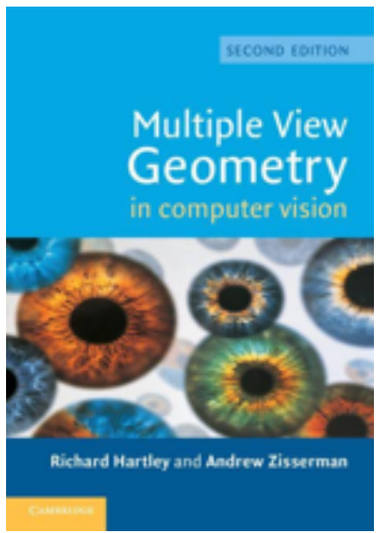
# Review: Pinhole Camera




Adapted from: Computer vision: models, learning and inference. Simon J.D. Prince

# Notation - Cheat Sheet

| Description | Hartley & Zisserman | Prince |
|---|---|---|
| 3D Point | $\mathbf{X}$ | $\mathbf{w}$ |
| 2D Point | $\mathbf{x}$ | $\mathbf{x}$ |
| Rotation matrix | $\mathbf{R}$ | $\mathbf{\Omega}$ |
| Intrinsics matrix | $\mathbf{K}$ | $\mathbf{\Lambda}$ |
| Homography matrix | $\mathbf{H}$ | $\mathbf{\Phi}$ |
| translation vector | $\mathbf{t}$ | $\boldsymbol{\tau}$ |




# Pinhole camera

- Camera model:

$$x = \frac{\phi_x u + \gamma v}{w} + \delta_x$$

$$y = \frac{\phi_y v}{w} + \delta_y,$$

- In homogeneous coordinates:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x & 0 \\ 0 & \phi_y & \delta_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} \quad \text{(linear!)}$$

Adapted from: Computer vision: models, learning and inference. Simon J.D. Prince

# Pinhole camera

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x & 0 \\ 0 & \phi_y & \delta_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}$$

- Writing out these three equations

$$\begin{aligned} \lambda x &= \phi_x u + \gamma v + \delta_x w \\ \lambda y &= \phi_y v + \delta_y w \\ \lambda &= w. \end{aligned}$$

- Eliminate $\lambda$ to retrieve original equations

Adapted from: Computer vision: models, learning and inference. Simon J.D. Prince

# Adding in extrinsics

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x & 0 \\ 0 & \phi_y & \delta_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}$$

Or for short:

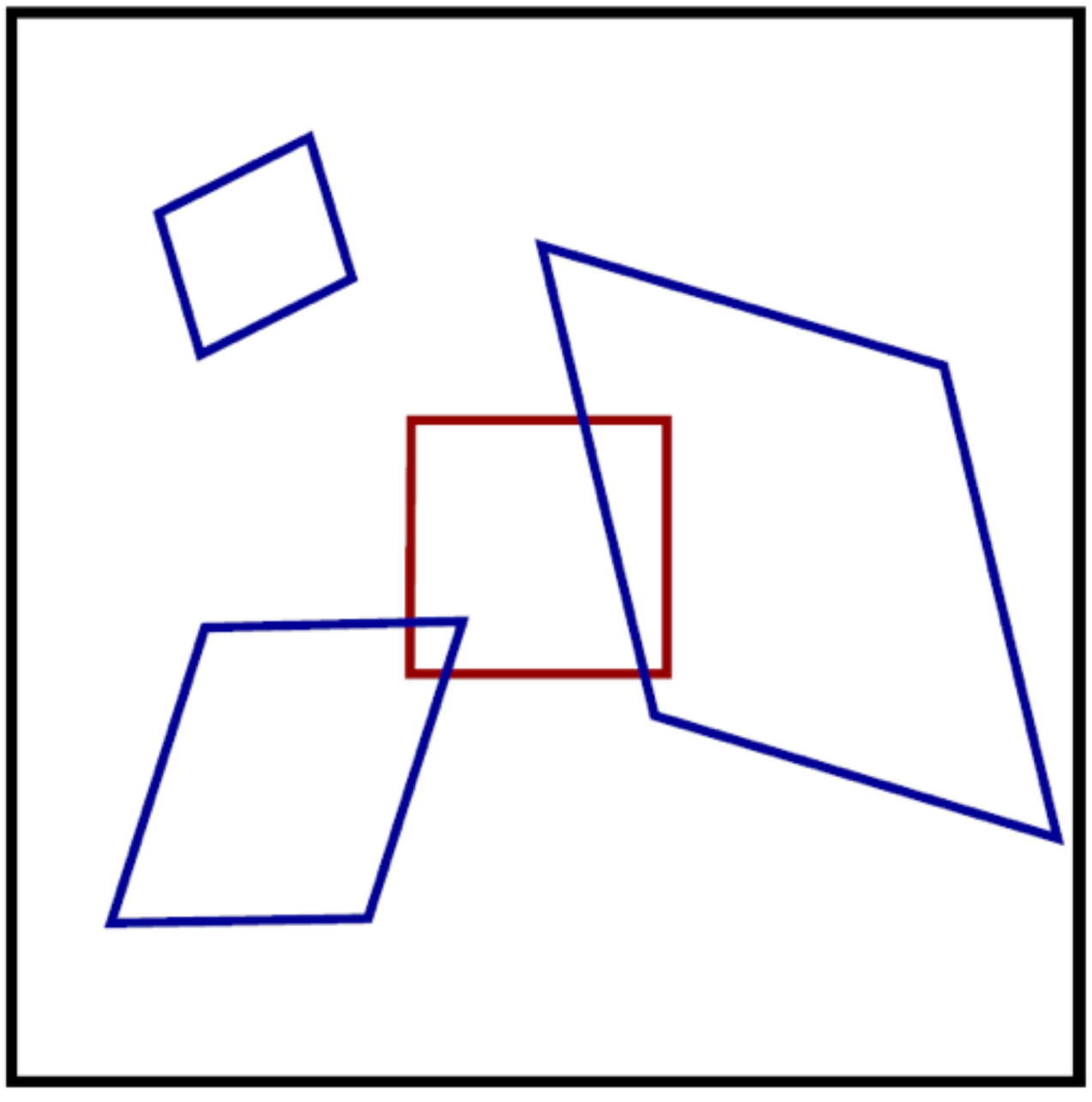
$$\lambda \tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{\Lambda} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{\Omega} & \boldsymbol{\tau} \\ \mathbf{0}^T & 1 \end{bmatrix} \tilde{\mathbf{w}}$$

Or even shorter:

$$\lambda \tilde{\mathbf{x}} = \mathbf{\Lambda} \begin{bmatrix} \mathbf{\Omega} & \boldsymbol{\tau} \end{bmatrix} \tilde{\mathbf{w}}$$

Adapted from: Computer vision: models, learning and inference. Simon J.D. Prince

# Review: Affine warp



Homogeneous:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \tau_x \\ \phi_{21} & \phi_{22} & \tau_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Cartesian:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} \tau_x \\ \tau_y \end{bmatrix}$$

For short:

$$\mathbf{x}' = \mathbf{aff}[\mathbf{w}, \mathbf{\Phi}, \boldsymbol{\tau}]$$

**How many unknowns?**

# Affine and Homography warps


Affine transform describes mapping well when the depth variation within the planar object is small and the camera is far away.


When variation in depth is comparable to distance to object then the affine transformation is not a good model. Here we need the homography.

Adapted from: Computer vision: models, learning and inference. Simon J.D. Prince

# Review: Estimating the Affine Warp

- Rearranging:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} u & v & 0 & 0 & 1 & 0 \\ 0 & 0 & u & v & 0 & 1 \end{bmatrix} \begin{bmatrix} \phi_{11} \\ \phi_{12} \\ \phi_{21} \\ \phi_{22} \\ \tau_x \\ \tau_y \end{bmatrix}$$

- Form system of equations: $\mathbf{x} = \mathbf{A}\mathbf{p}$

- In MATLAB this becomes,

```
>> p = A\x;
```

# Review: Homography

- Start with basic projection equation:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x \\ 0 & \phi_y & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \end{bmatrix} \begin{bmatrix} u \\ v \\ 0 \\ 1 \end{bmatrix}$$

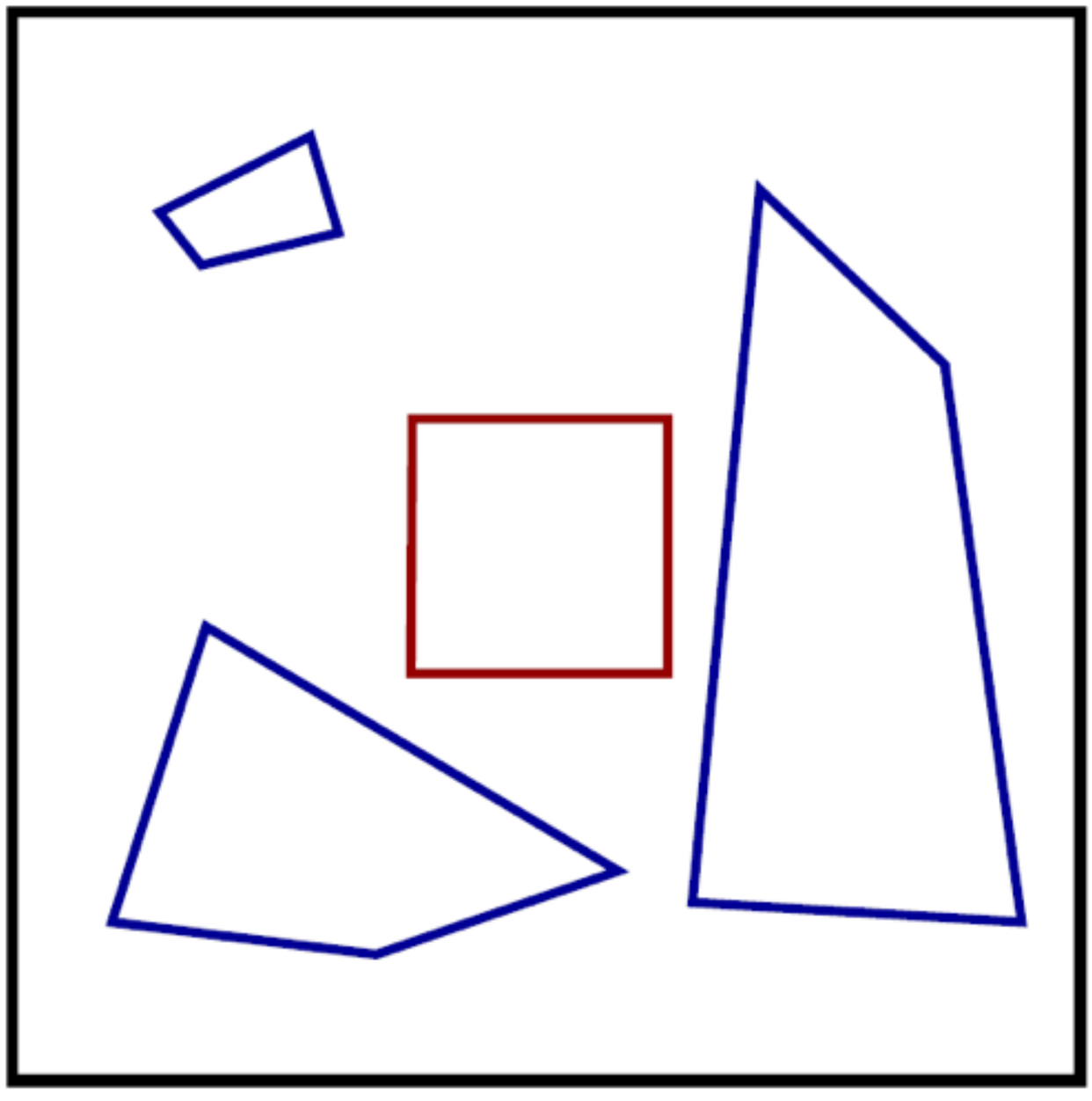
$$= \begin{bmatrix} \phi_x & \gamma & \delta_x \\ 0 & \phi_y & \delta_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \tau_x \\ \omega_{21} & \omega_{22} & \tau_y \\ \omega_{31} & \omega_{32} & \tau_z \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

- Combining these two matrices we get:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Adapted from: Computer vision: models, learning and inference. Simon J.D. Prince

# Review Homography



Homogeneous:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Cartesian:

$$x = \frac{\phi_{11}u + \phi_{12}v + \phi_{13}}{\phi_{31}u + \phi_{32}v + \phi_{33}}$$

$$y = \frac{\phi_{21}u + \phi_{22}v + \phi_{23}}{\phi_{31}u + \phi_{32}v + \phi_{33}}$$

For short:

$$\mathbf{x} = \mathbf{hom}[\mathbf{w}, \mathbf{\Phi}]$$

**How many unknowns?**

# Homography Estimation

- Re-arrange cartesian equations,

$$x(\phi_{31}u + \phi_{32}v + \phi_{33}) = \phi_{11}u + \phi_{12}v + \phi_{13}$$

$$y(\phi_{31}u + \phi_{32}v + \phi_{33}) = \phi_{21}u + \phi_{22}v + \phi_{23}$$

- Form linear system $\mathbf{A}\boldsymbol{\phi} = \mathbf{0} \;\; \text{s.t} \; ||\boldsymbol{\phi}||_2^2 = 1$

$$\begin{bmatrix} 0 & 0 & 0 & -u_1 & -v_1 & -1 & y_1u_1 & y_1v_1 & y_1 \\ u_1 & v_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -x_1v_1 & -x_1 \\ 0 & 0 & 0 & -u_2 & -v_2 & -1 & y_2u_2 & y_2v_2 & y_2 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -x_2v_2 & -x_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & -u_I & -v_I & -1 & y_Iu_I & y_Iv_I & y_I \\ u_I & v_I & 1 & 0 & 0 & 0 & -x_Iu_I & -x_Iv_I & -x_I \end{bmatrix} \begin{bmatrix} \phi_{11} \\ \phi_{12} \\ \phi_{13} \\ \phi_{21} \\ \phi_{22} \\ \phi_{23} \\ \phi_{31} \\ \phi_{32} \\ \phi_{33} \end{bmatrix} = \mathbf{0},$$

Adapted from: Computer vision: models, learning and inference. Simon J.D. Prince

# Homography Estimation

- In MATLAB this becomes,

```
>> [U,S,V] = svd(A);
>> Phi = reshape(V(:,end),[3,3])';
```

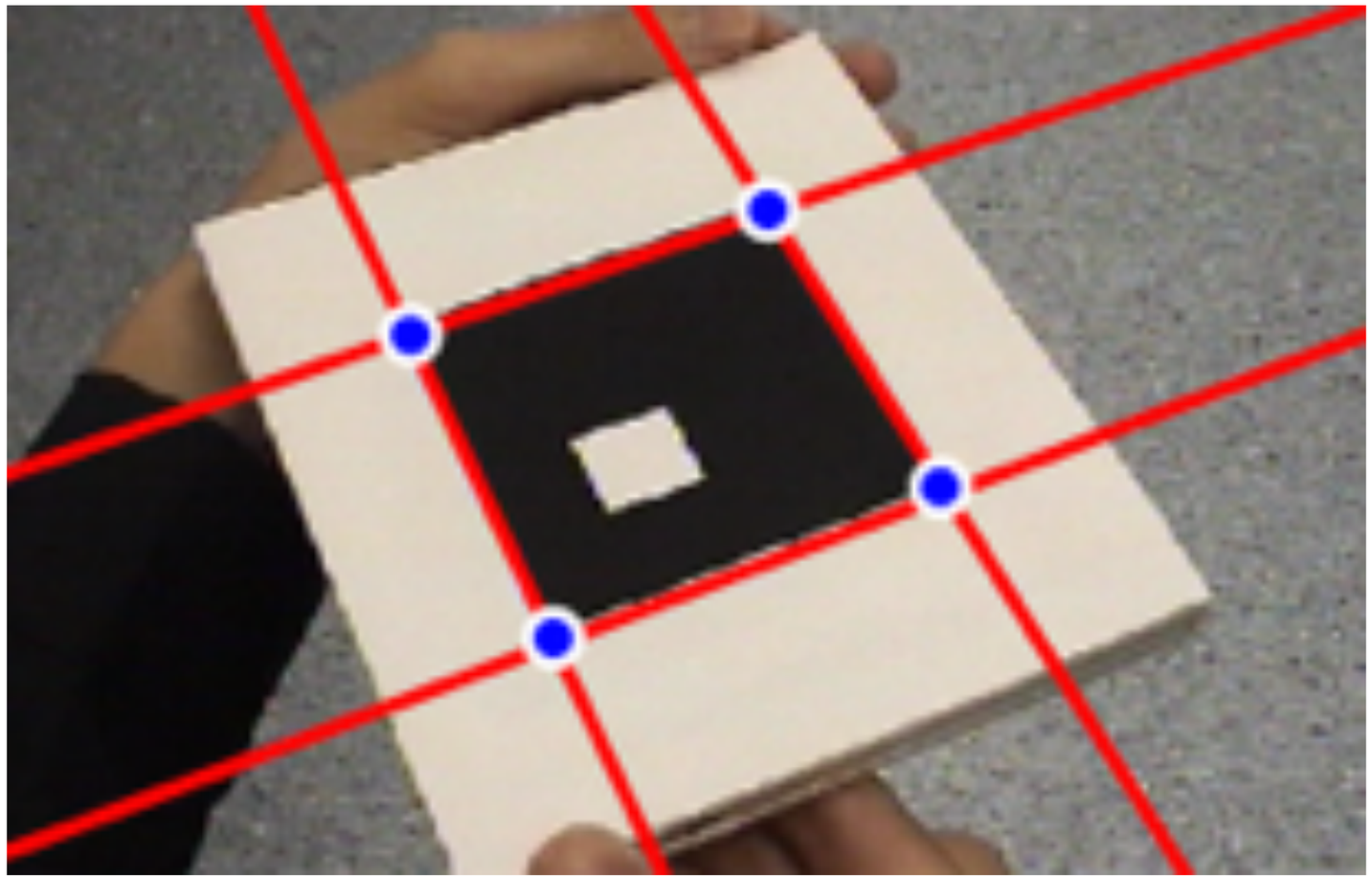
- Both sides are 3x1 vectors;  should be parallel, so cross product will be zero

$$\tilde{\mathbf{x}} \times \mathbf{\Phi}\tilde{\mathbf{w}} = \mathbf{0}$$

- For you to try MATLAB,

```
>> x = [randn(2,1);1]; cross(x,4*x)
```

# Estimating Extrinsics



$$\hat{\Omega}, \hat{\tau}$$

# Estimating Extrinsics

- Writing out the camera equations in full

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \phi_x & \gamma & \delta_x \\ 0 & \phi_y & \delta_y \\ 0 & 0 & D \end{bmatrix} \begin{bmatrix} \omega_{11} & \omega_{12} & \tau_x \\ \omega_{21} & \omega_{22} & \tau_y \\ \omega_{31} & \omega_{32} & \tau_z \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

- Estimate the homography from matched points
- Factor out the intrinsic parameters

$$\begin{bmatrix} \phi'_{11} & \phi'_{12} & \phi'_{13} \\ \phi'_{21} & \phi'_{22} & \phi'_{23} \\ \phi'_{31} & \phi'_{32} & \phi'_{33} \end{bmatrix} = \lambda' \begin{bmatrix} \omega_{11} & \omega_{12} & \tau_x \\ \omega_{21} & \omega_{22} & \tau_y \\ \omega_{31} & \omega_{32} & \tau_z \end{bmatrix}$$

Adapted from: Computer vision: models, learning and inference. Simon J.D. Prince

# Estimating Extrinsics

- Find the last column using the cross product of first two columns

- Make sure the determinant is 1. If it is -1, then multiply last column by -1.

- Find translation scaling factor between old and new values

$$\lambda' = \frac{\sum_{m=1}^{3}\sum_{n=1}^{2}\phi'_{mn}/\omega_{mn}}{6}$$

- Finally, set $\boldsymbol{\tau} = [\phi'_{13}, \phi'_{23}, \phi'_{33}]^T / \lambda'$

Adapted from: Computer vision: models, learning and inference. Simon J.D. Prince

# Augmented Reality

# Today

- Augmented Reality
- Review: Homographies & Pinhole Cameras
- ARToolkit in iOS

# ARToolkit Example

- Good example of using ARToolkit in iOS can be found on ARToolkit website.
- On your browser please go to the address,

http://www.artoolkit.org/dist/artoolkit5/5.3/ARToolKit5-bin-5.3.2-iOS.tar.gz

- **Careful**: Xcode project is a little different to what you are used to at the moment.
- Project has multiple Apps combined together you need to select which one you want to use.

# ARToolkit Example

- To start with go and print off the Hiro pattern from,

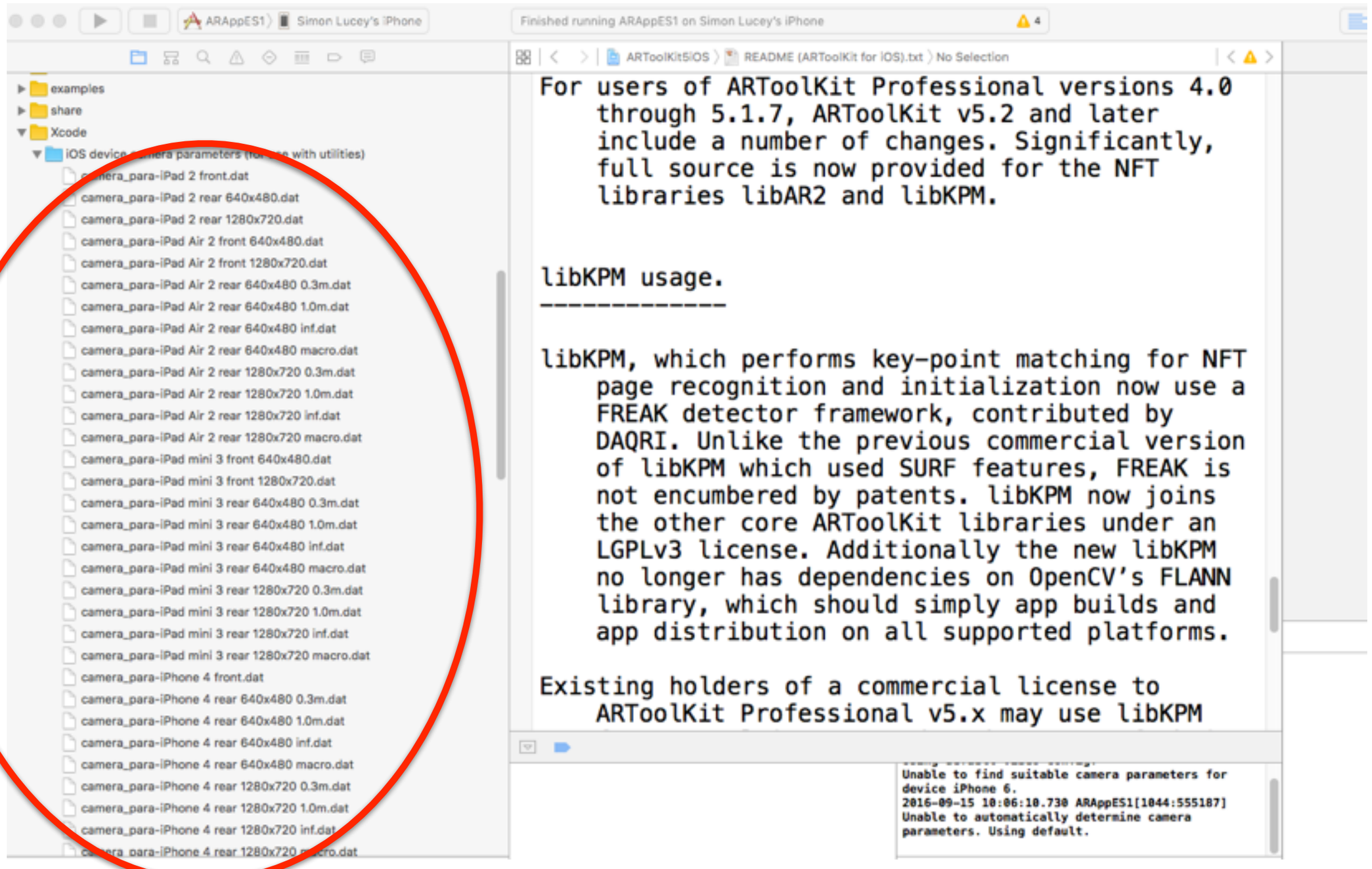
https://github.com/artoolkit/artoolkit5/blob/master/doc/patterns/Hiro%20pattern.pdf

- Load the Xcode project `ARToolKit5iOS.xcodeproj`

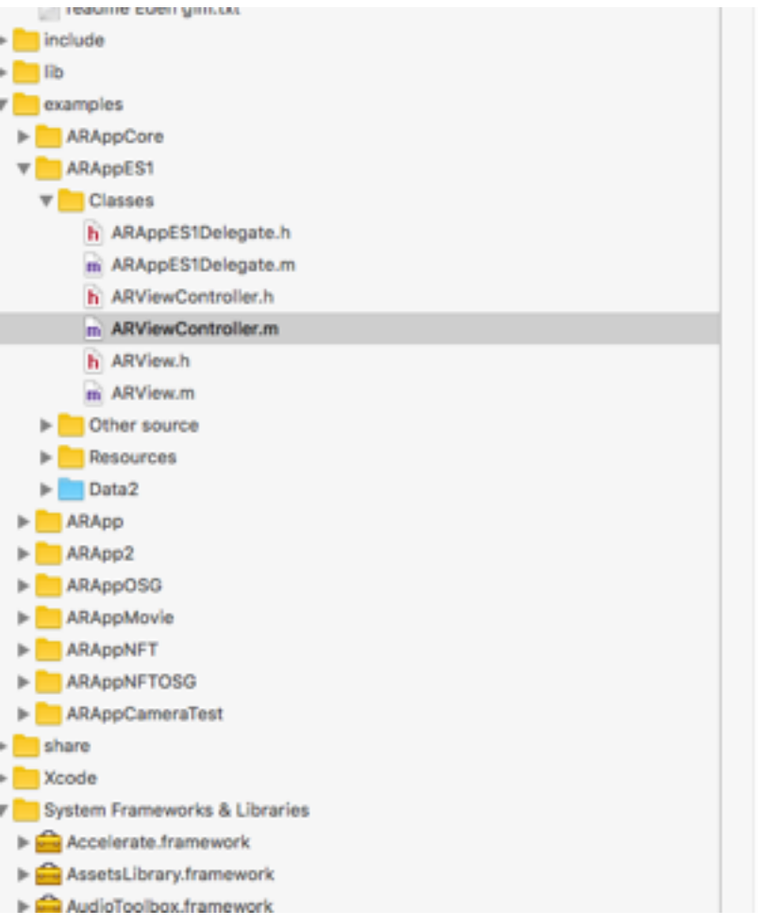
- From there attach your iOS device and select the `ARAppES1` to build and run.

# Intrinsics & ARToolkit

- Cool thing is that they have the intrinsics estimated for nearly all iOS devices.



# Reading in Intrinsics




```
    if (frontCamera == AR_VIDEO_IOS_CAMERA_POSITION_FRONT) flipV = TRUE;
}

// Tell arVideo what the typical focal distance will be. Note that this does NOT
// change the actual focus, but on devices with non-fixed focus, it lets arVideo
// choose a better set of camera parameters.
ar2VideoSetParami(gVid, AR_VIDEO_PARAM_IOS_FOCUS, AR_VIDEO_IOS_FOCUS_0_3M); //
    Default is 0.3 metres. See <AR/sys/videoiPhone.h> for allowable values.

// Load the camera parameters, resize for the window and init.
ARParam cparam;
if (ar2VideoGetCParam(gVid, &cparam) < 0) {
    char cparam_name[] = "Data2/camera_para.dat";
    NSLog(@"Unable to automatically determine camera parameters. Using default.
        \n");
    if (arParamLoad(cparam_name, 1, &cparam) < 0) {
        NSLog(@"Error: Unable to load parameter file %s for camera.\n",
            cparam_name);
        [self stop];
        return;
    }
}
```

# Uses OpenGL ES Heavily

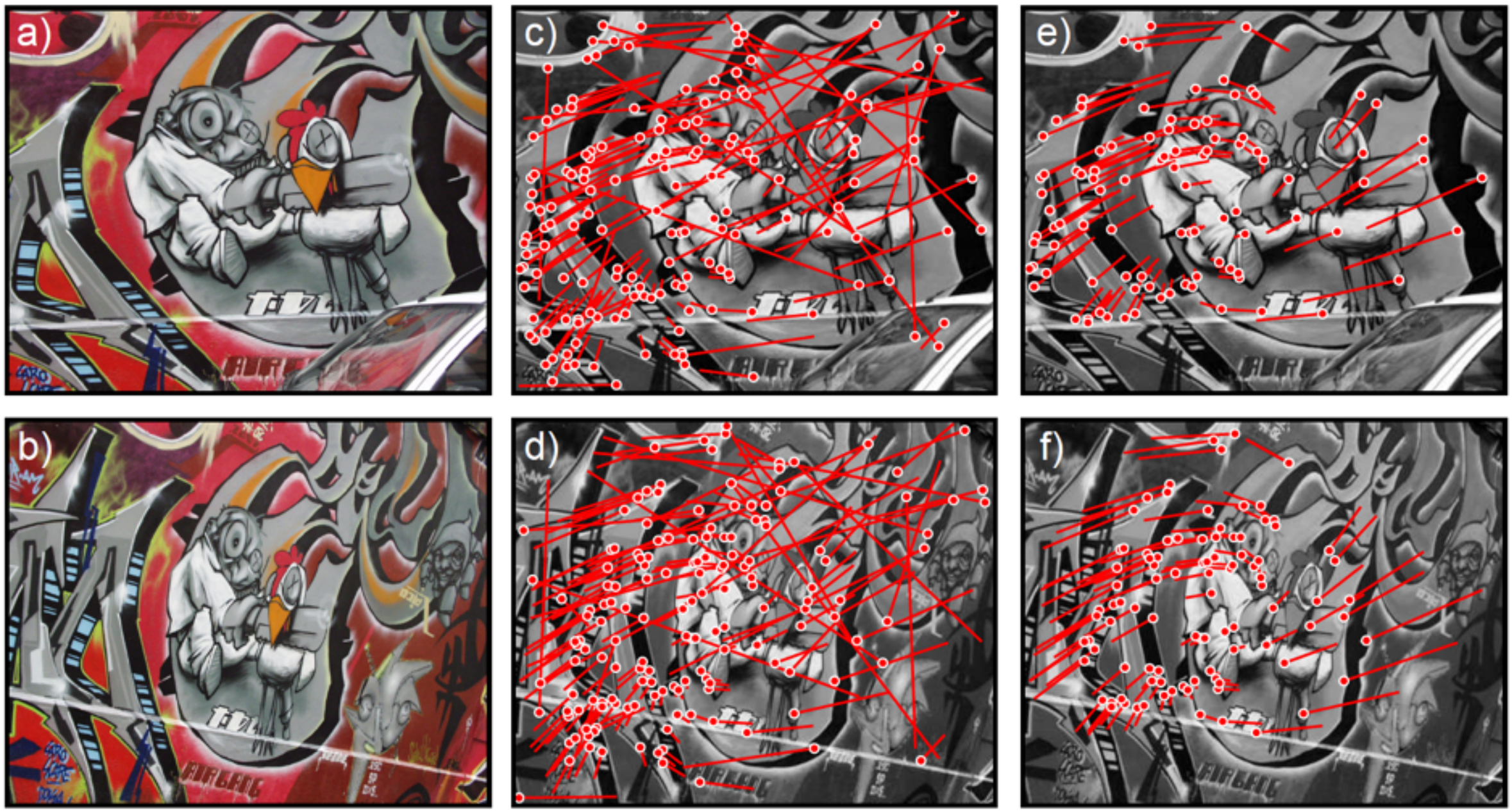
- All drawing and rendering is done through OpenGL ES.
- Useful resource for anyone thinking of project involving AR.

```
// Allocate the OpenGL view.
glView = [[[ARView alloc] initWithFrame:[[UIScreen mainScreen] bounds]
    pixelFormat:kEAGLColorFormatRGBA8 depthFormat:kEAGLDepth16 withStencil:NO
    preserveBackbuffer:NO] autorelease]; // Don't retain it, as it will be
    retained when added to self.view.
glView.arViewController = self;
[self.view addSubview:glView];

// Create the OpenGL projection from the calibrated camera parameters.
// If flipV is set, flip.
GLfloat frustum[16];
arglCameraFrustumRHf(&gCparamLT->param, VIEW_DISTANCE_MIN, VIEW_DISTANCE_MAX,
    frustum);
[glView setCameraLens:frustum];
glView.contentFlipV = flipV;
```

# How does it work?

- Vision tracking component uses a combination of RANSAC and the FREAK detector framework.




Original images | Initial matches | Inliers from RANSAC

# Things to think about??