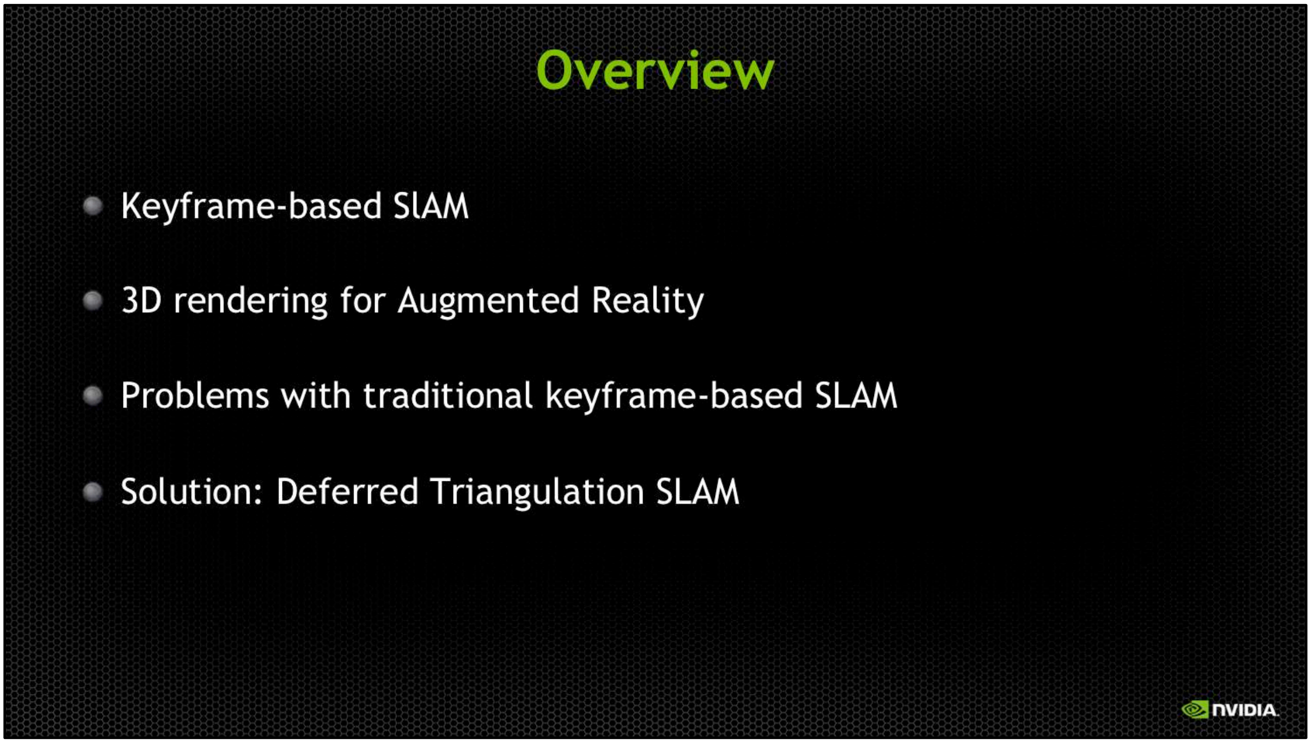
SLAM@NVIDIA

Kari Pulli| Senior Director of Research

Hi, my name is Kari Pulli, I head a team at NVIDIA Research working on Mobile Visual Computing.

I’m going to talk about our work on SLAM, Simultaneous Localization and Mapping.



Overview

- Keyframe-based SlAM
- 3D rendering for Augmented Reality
- Problems with traditional keyframe-based SLAM
- Solution: Deferred Triangulation SLAM

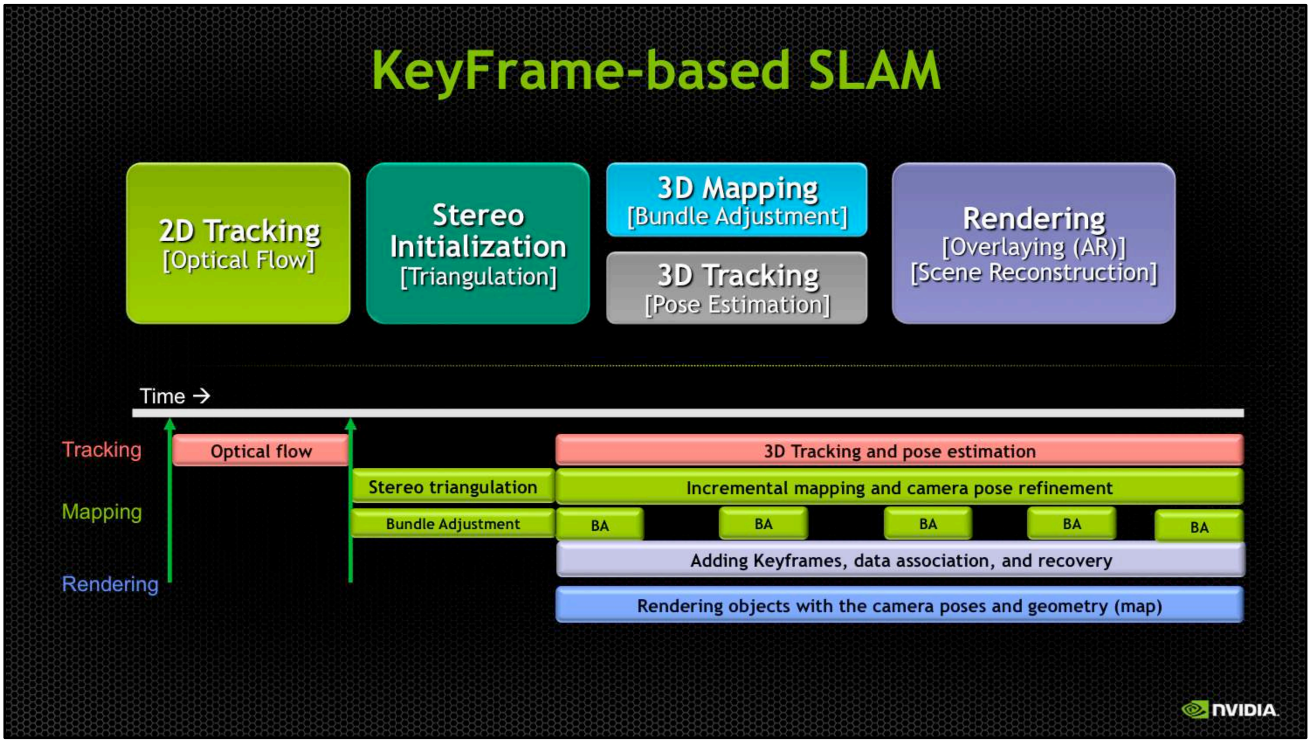
I'll first give an overview of keyframe-based SLAM.

Then we'll see some 3D graphics rendering for Augmented Reality applications.

We'll demonstrate some of the typical problems with using a traditional keyframe-based SLAM system,

and finally propose a solution,

where we first track features in 2D and later promote them to 3D model features.



KeyFrame-based SLAM

2D Tracking [Optical Flow]

Stereo Initialization [Triangulation]

3D Mapping [Bundle Adjustment]

3D Tracking [Pose Estimation]

Rendering [Overlaying (AR)] [Scene Reconstruction]

Time →

Tracking

Mapping

Rendering

Optical flow

Stereo triangulation

Bundle Adjustment

3D Tracking and pose estimation

Incremental mapping and camera pose refinement

BA BA BA BA BA

Adding Keyframes, data association, and recovery

Rendering objects with the camera poses and geometry (map)

NVIDIA.

Here are the typical components for keyframe-based SLAM.

<click> We initialize the system by moving camera and by tracking image feature motion in 2D.

<click> Once there’s enough motion, we can triangulate some of the features and find their 3D location with respect to the camera.

<click> After this initialization, we start doing several things at once.

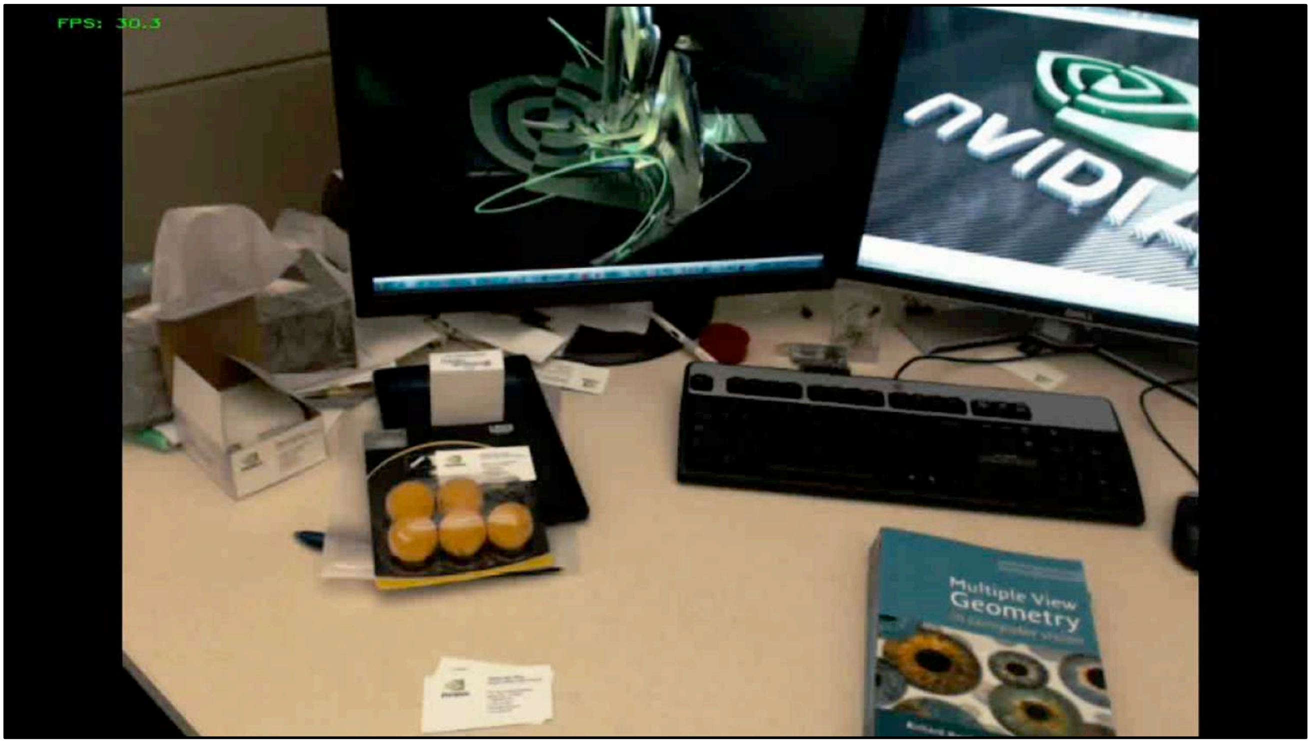
The first is to track the camera motion in 3D, with respect to the 3D model we just built. This runs in one thread.

<click> At the same time, a mapping thread optimizes the 3D model and camera pose further using a small subset of keyframes

<click> and another thread, every now and then, performs a global bundle adjustment to refine the 3D model and poses.

<click> If the current frame is sufficiently different from the other keyframes, it’s added as a new keyframe.

<click> Finally, the application uses the camera pose and 3D model to draw some application-dependent content.

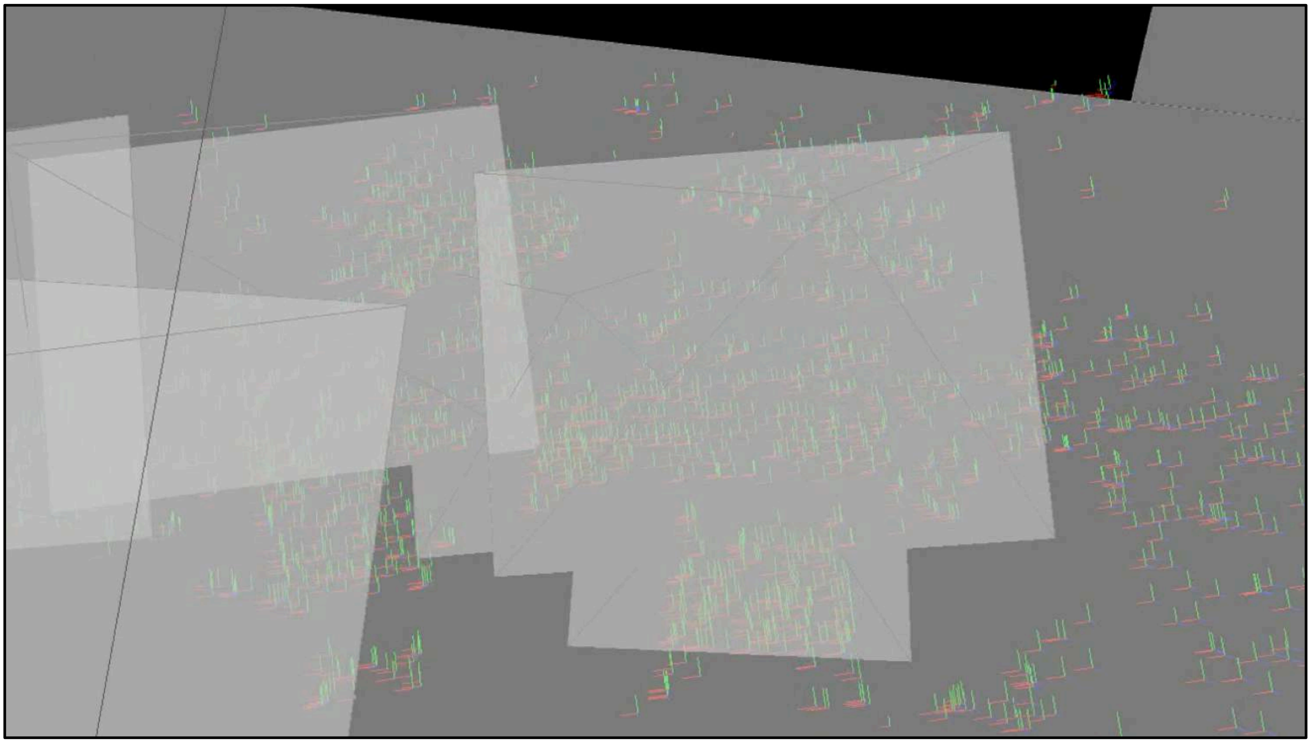


Here we first start by finding some 2D image features, in blue,

We track their motion, and once the baseline is long enough, we triangulate them into 3D points.

We then define a 3D coordinate system on the table and place a green cube on it.

As we move the camera and new features are seen, the model is expanded.

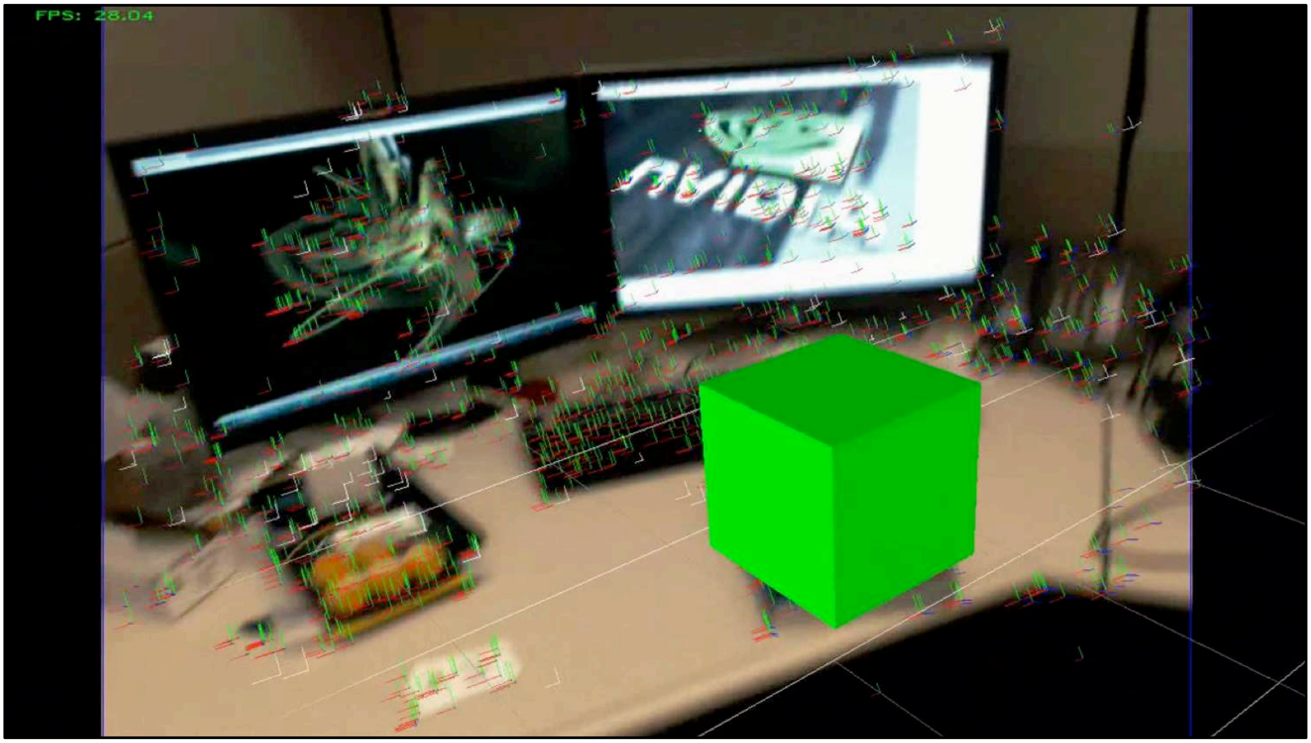


Now we show how a partial model looks like.

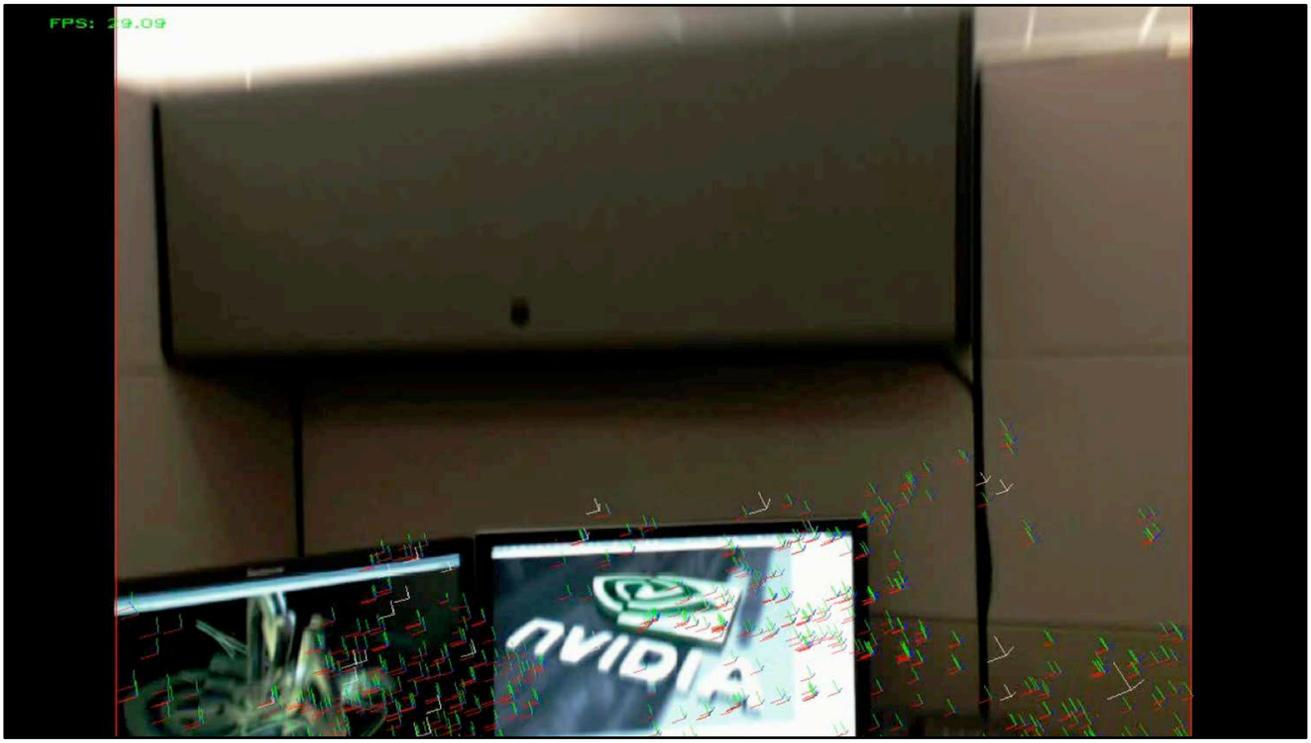
The gray boxes show the keyframes that have been captured.

The small RGB coordinates show the 3D scene points that have been modeled.

The large grid shows the global coordinate system.



This is a stress test.
The camera is shaken violently, so that there is motion blur,
and the whole image wobbles a lot due to the rolling shutter effect.
Tracking is still robust, the green cube stays on the table.



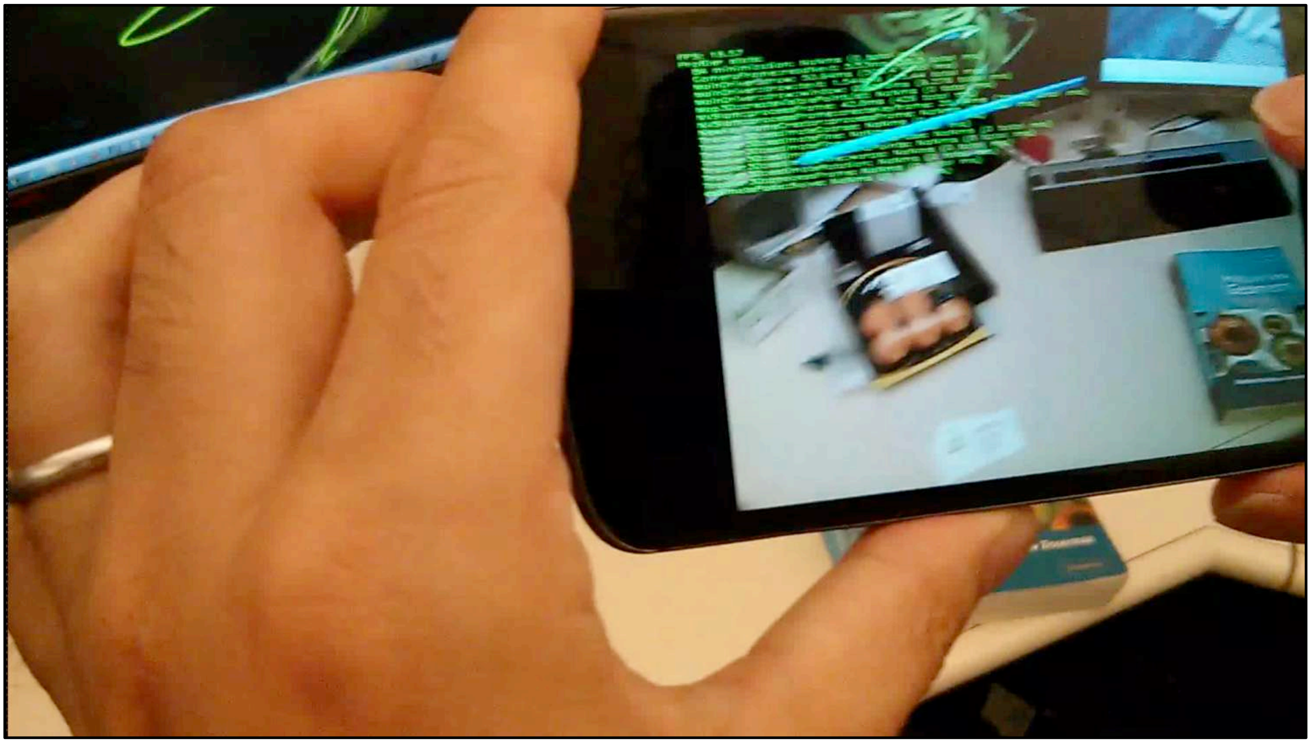
If you move the camera quickly so that you don’t see anything you have already modeled, you lose the track.

But when the camera points back towards the scene, the system can relocalize itself.

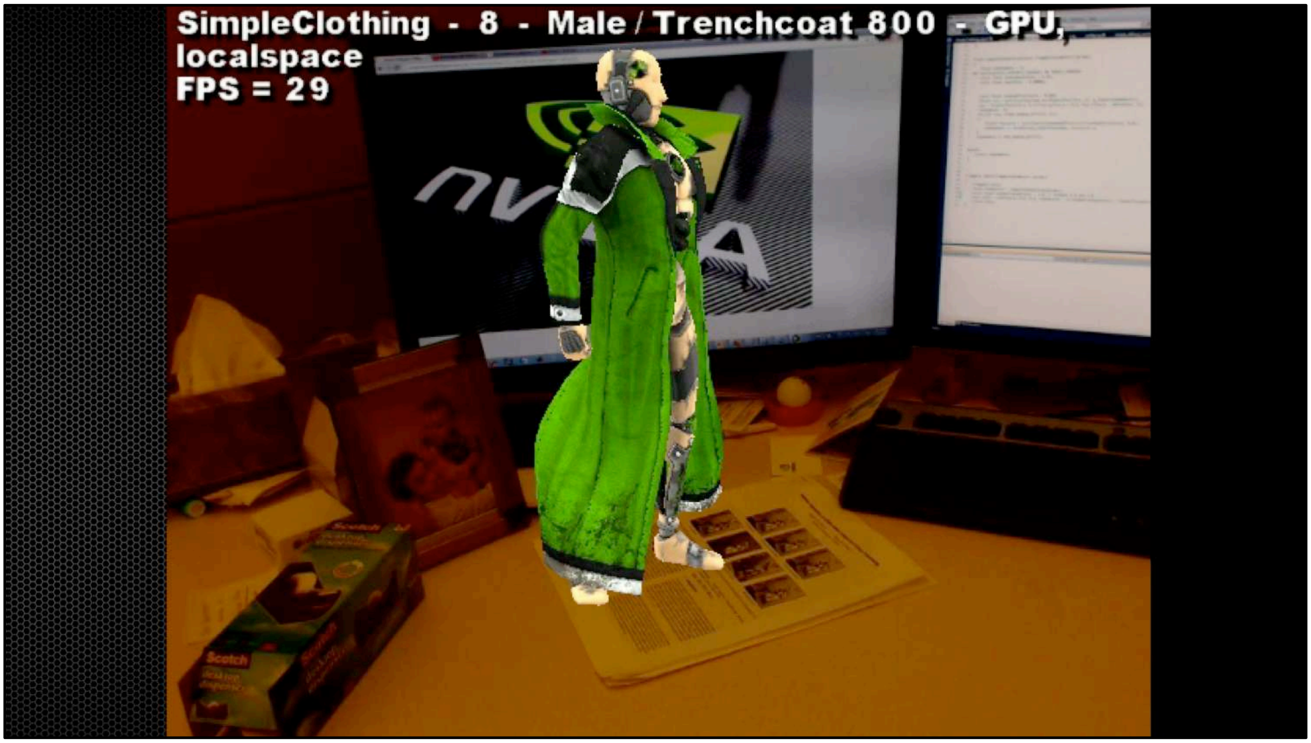
If we are lost, we analyze every new frame, and see if it looks similar to any of the previous keyframes.

And then try to estimate the camera pose and continue tracking.

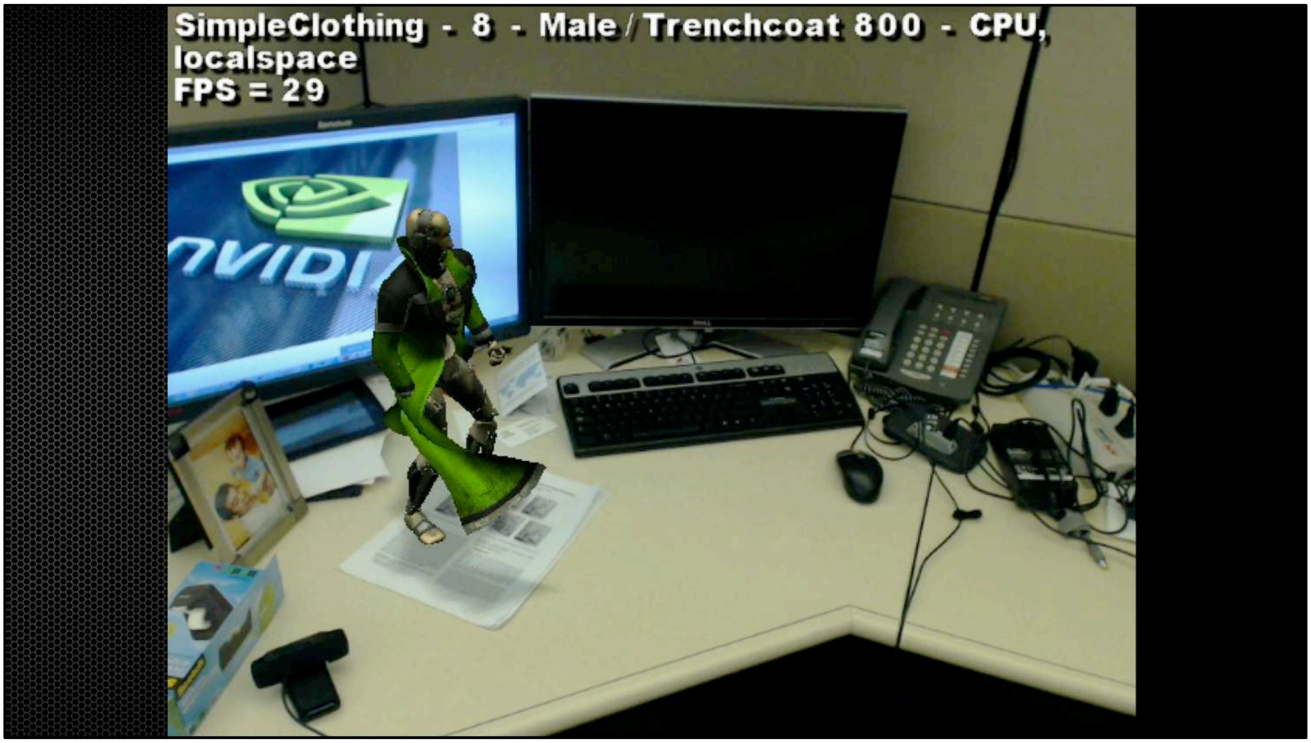
Here it works pretty well.



The system is so light-weight that it can even be run on a Nexus 4, though the frame rate is a bit slow.


SimpleClothing - 8 - Male / Trenchcoat 800 - GPU, localspace
FPS = 29

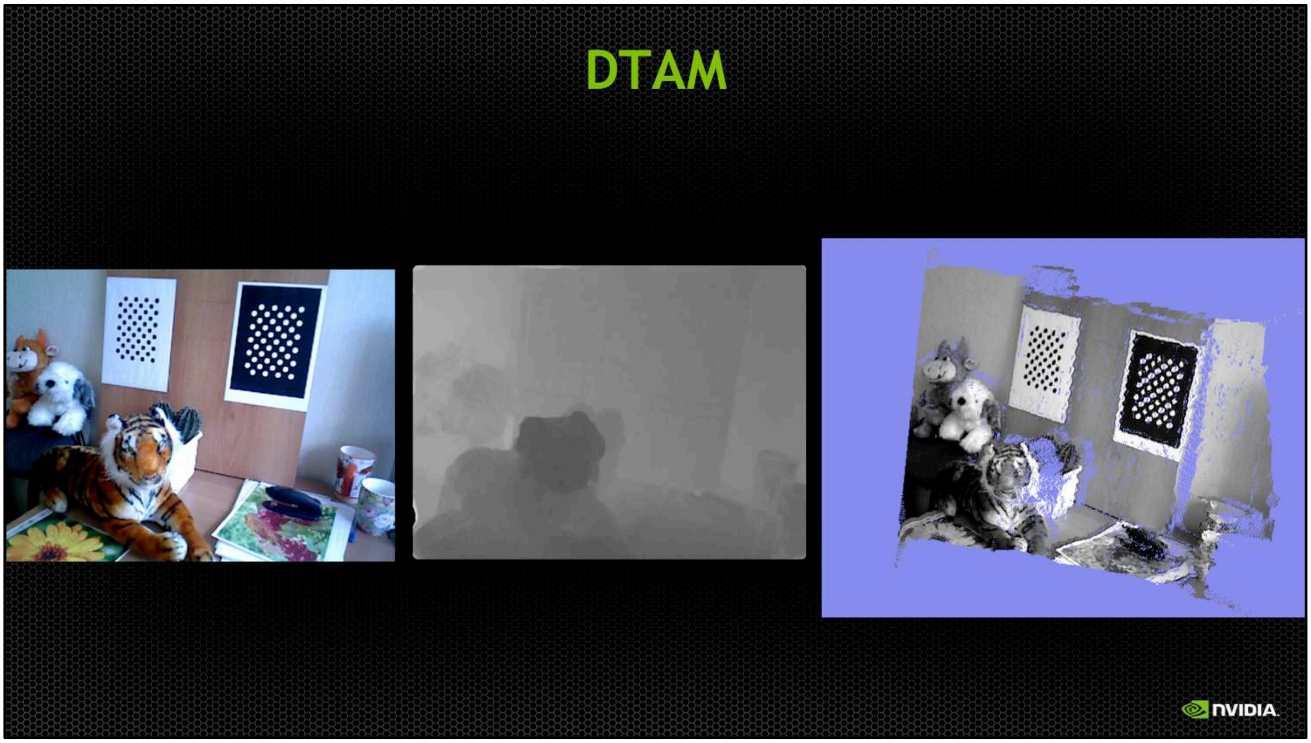
We can now render more complex content than just a green cube.
But it is very obvious that this character is synthetic and not really part of the scene.


SimpleClothing - 8 - Male / Trenchcoat 800 - CPU, localspace
FPS = 29

Once we estimate the color of the illumination in the scene,
and add a shadow,
the augmentation becomes more compelling.

The same technology can be also used for parking assistance,
so that the car can fit itself to the parking square
without hitting the other cars.

In this test sequence we do have a human driver, it’s part of the training sequence we use to evaluate performance and feasibility of this application.

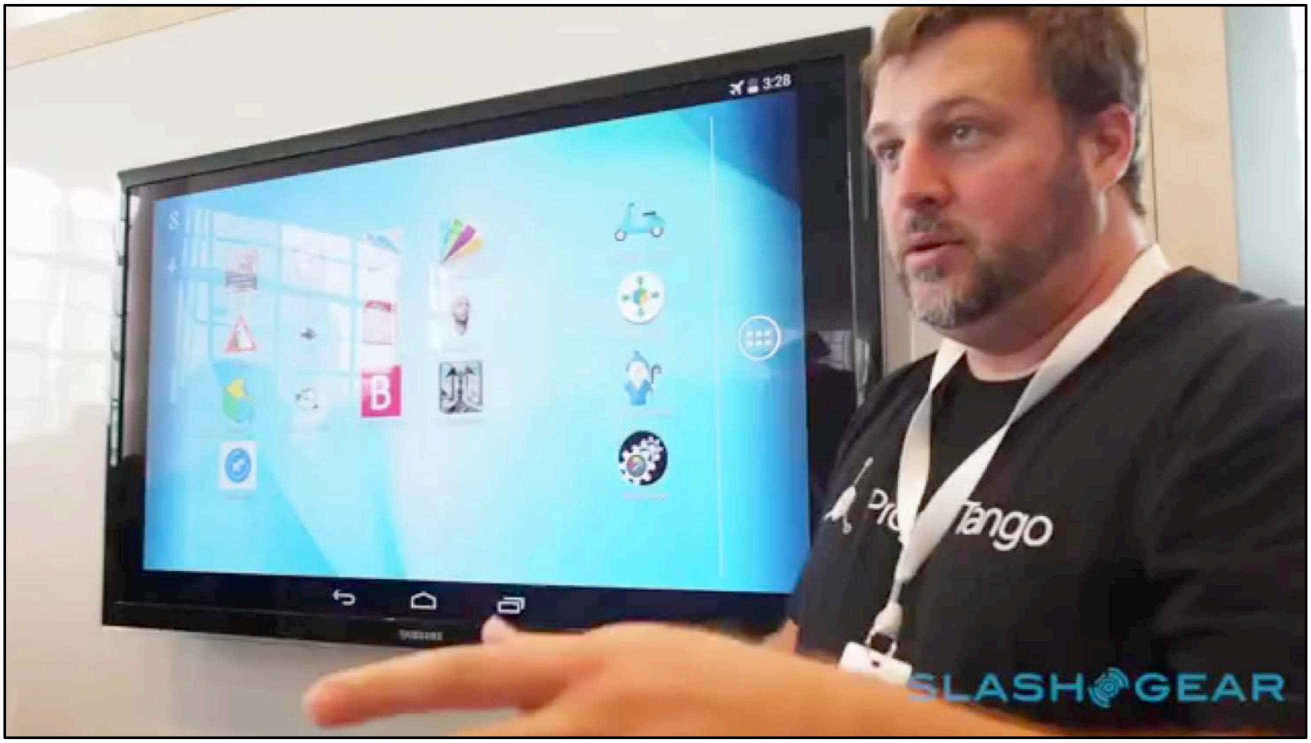


DTAM

These keyframe-based systems only give a sparse set of 3D points.

There are other methods that obtain a dense 3D description of the environment, just from a regular 2D camera.

Here’s one called DTAM, or Dense Tracking And Mapping, that we experimented with, but since the processing effort is much too high for a mobile device, we didn’t go too far in this direction.



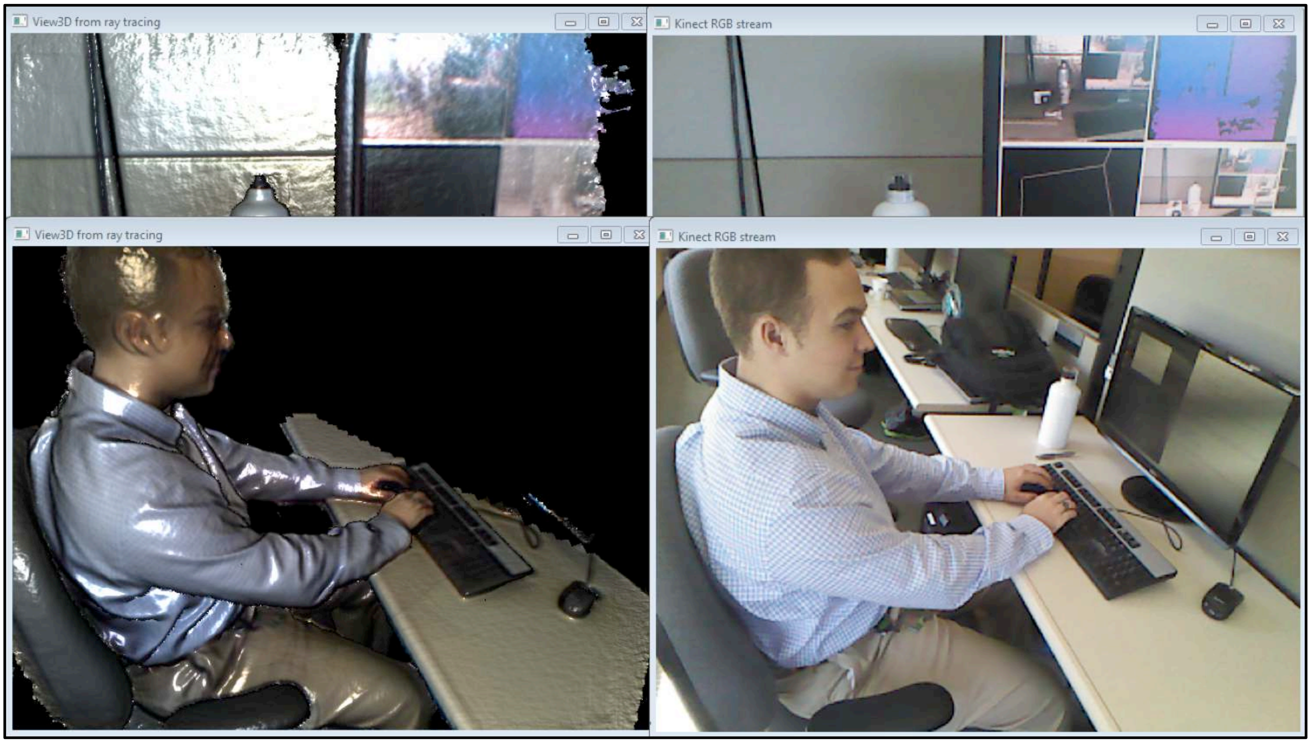
But if you add a range camera, either using triangulation or time-of-flight,

like Google did in their Tango project,

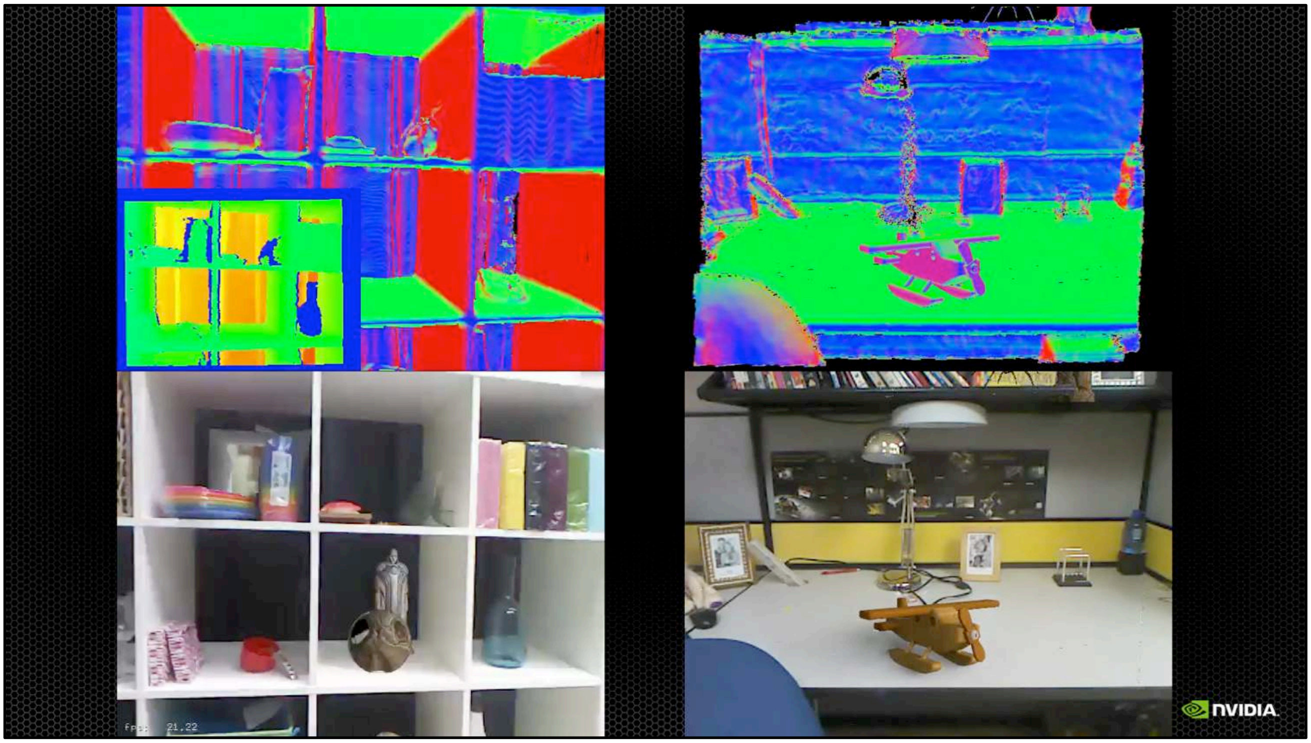
you can get dense depth, and still have it work in real time on a mobile device.

This video is from Google IO where Tango was demonstrated.

This version of Tango devices, codename Yellowstone, is based on an NVIDIA Tegra K1 embedded processor.



We have also done Kinectfusion-type of processing using SoftKinetic range scanners.



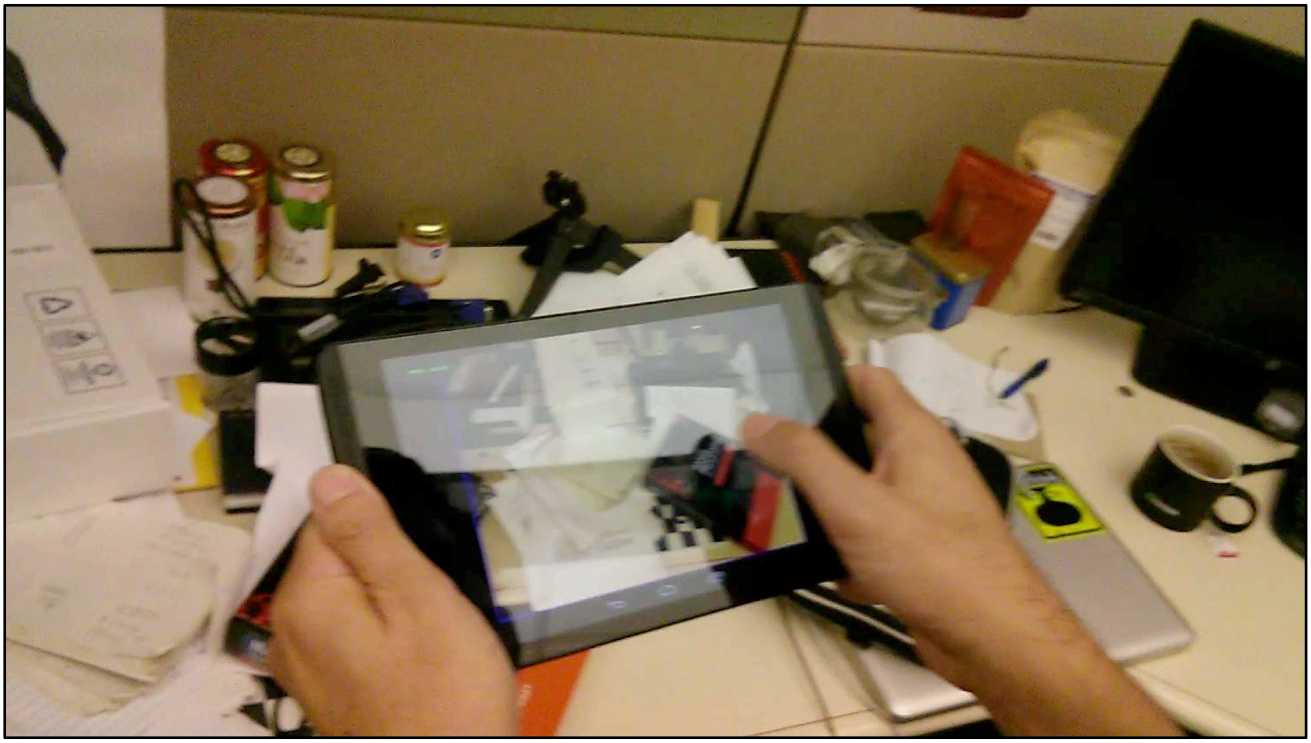
In an on-going project, we are experimenting on real-time realistic rendering of synthetic content

in a dense depth-camera based SLAM system.

Some of the objects you can see are not really there, but have been inserted into the scene.

We model the interaction of the light between the real scene and the synthetic objects.

This is unfinished work, so I won’t dive into details.



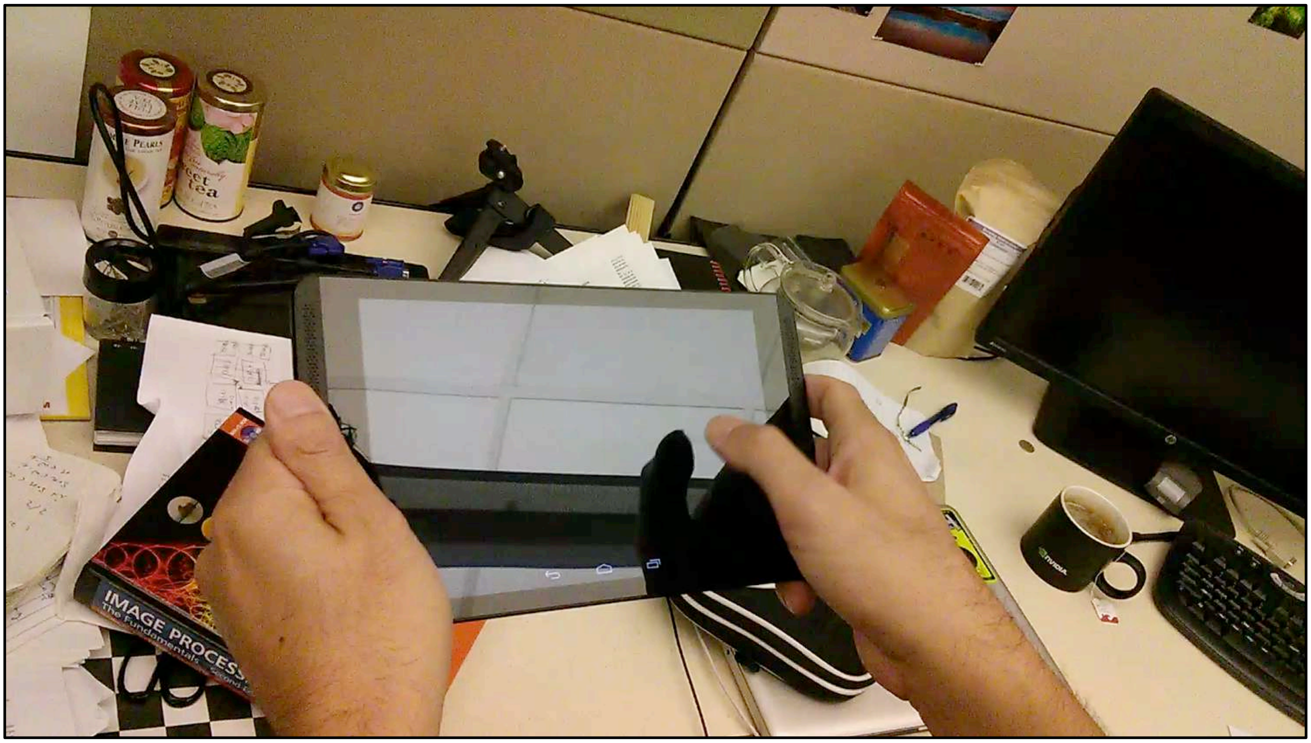
But let’s go back to monocular SLAM, without range cameras.

Here the system is running on a Tegra tablet,

at a much higher framerate than what we earlier saw on the phone.

This user is an experienced one and knows how the system works can avoid any pitfalls.

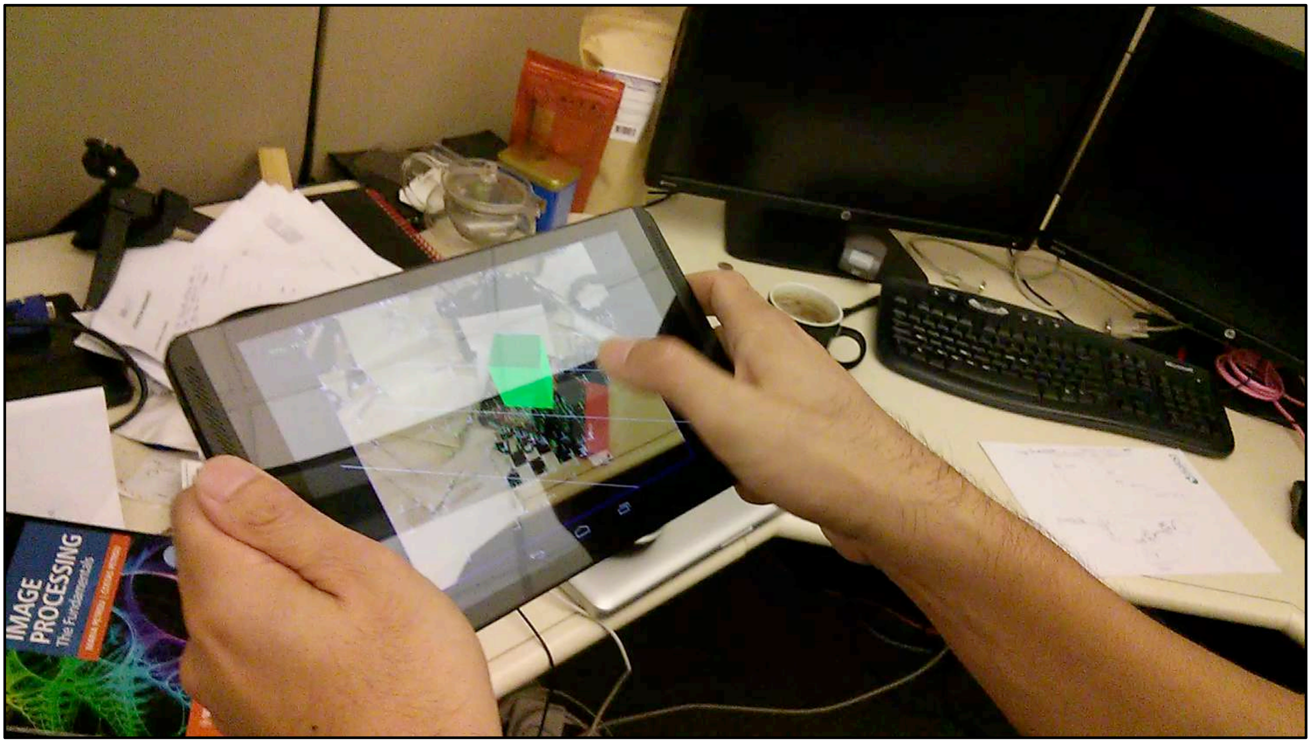
Mapping is initialized well, the coordinate system behaves nicely, and the tracking is continuous.



This user is less experienced, and experiences a typical set of problems.

If the system is not properly and carefully initialized,
the first model that is triangulated can be completely wrong.

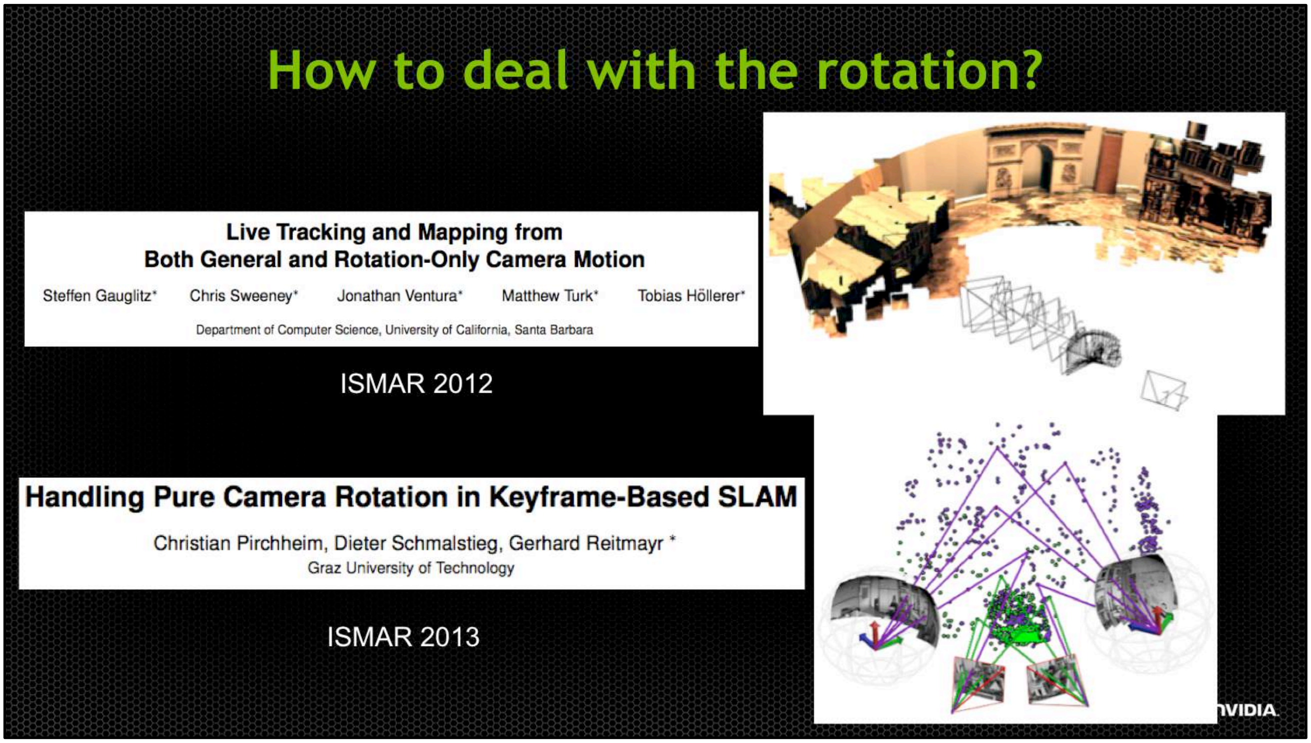
Now when you try to place a cube,
the coordinate system is ill-defined,
and the system behaves erratically.



This system has been initialized well.

But if the user is not aware that modeling only works well
if the keyframes are separated by some translation.

If the camera is rotated in place,
there is zero baseline between keyframes,
and triangulation of new image features fails.



How to deal with the rotation?

Live Tracking and Mapping from Both General and Rotation-Only Camera Motion

Steffen Gauglitz* Chris Sweeney* Jonathan Ventura* Matthew Turk* Tobias Höllerer*

Department of Computer Science, University of California, Santa Barbara

ISMAR 2012

Handling Pure Camera Rotation in Keyframe-Based SLAM

Christian Pirchheim, Dieter Schmalstieg, Gerhard Reitmayr *

Graz University of Technology

ISMAR 2013

There is some previous work addressing this problem.

This work from UC Santa Barbara, published 2 years ago at ISMAR, has two modes.

When the camera moves, it does normal 3D tracking,

and when it rotates, it switches to a new homography / panorama based tracking mode,

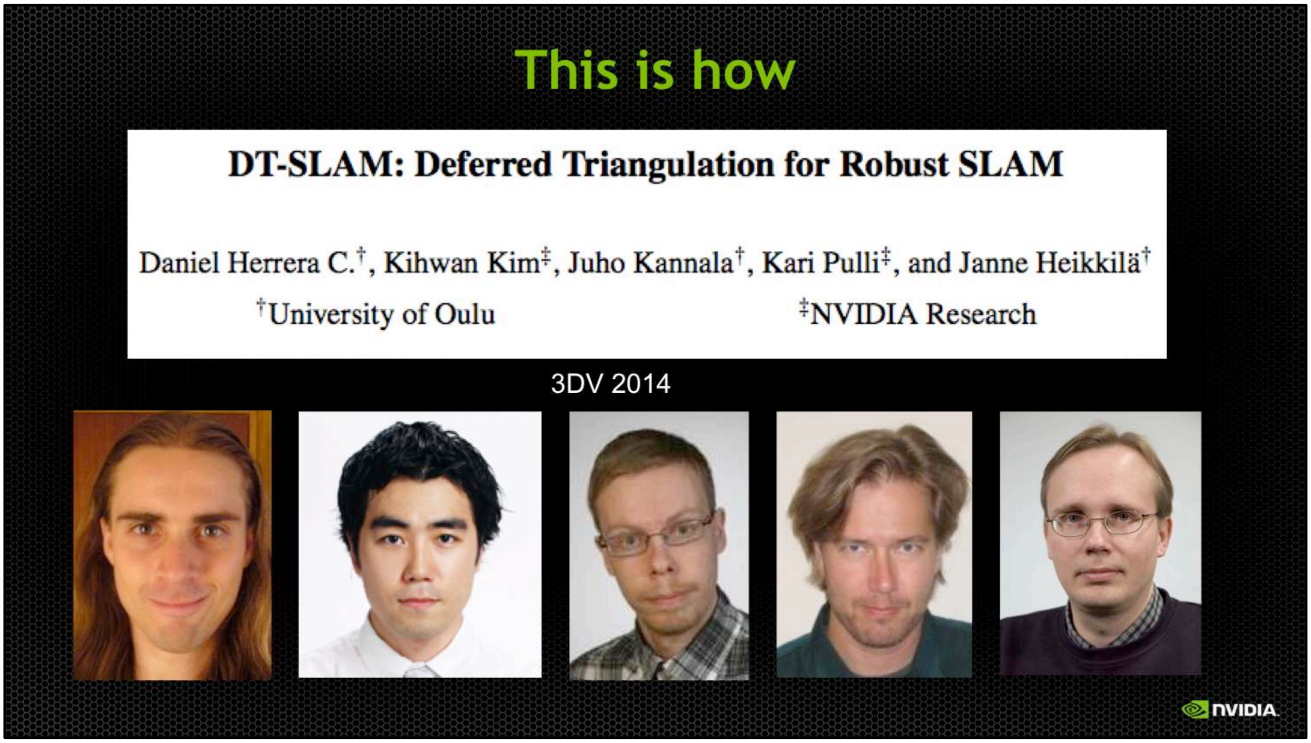
where it does no model building

Last year at ISMAR a group from TU Graz addressed the same situation better.

This system, called Hybrid SLAM, allows the 2D features to become “3D” points at infinity.

But they still have two tracking modes, rather than a single unified mode,

And points at infinity do cause some problems.



This is how

DT-SLAM: Deferred Triangulation for Robust SLAM

Daniel Herrera C.[†], Kihwan Kim[‡], Juho Kannala[†], Kari Pulli[‡], and Janne Heikkilä[†]

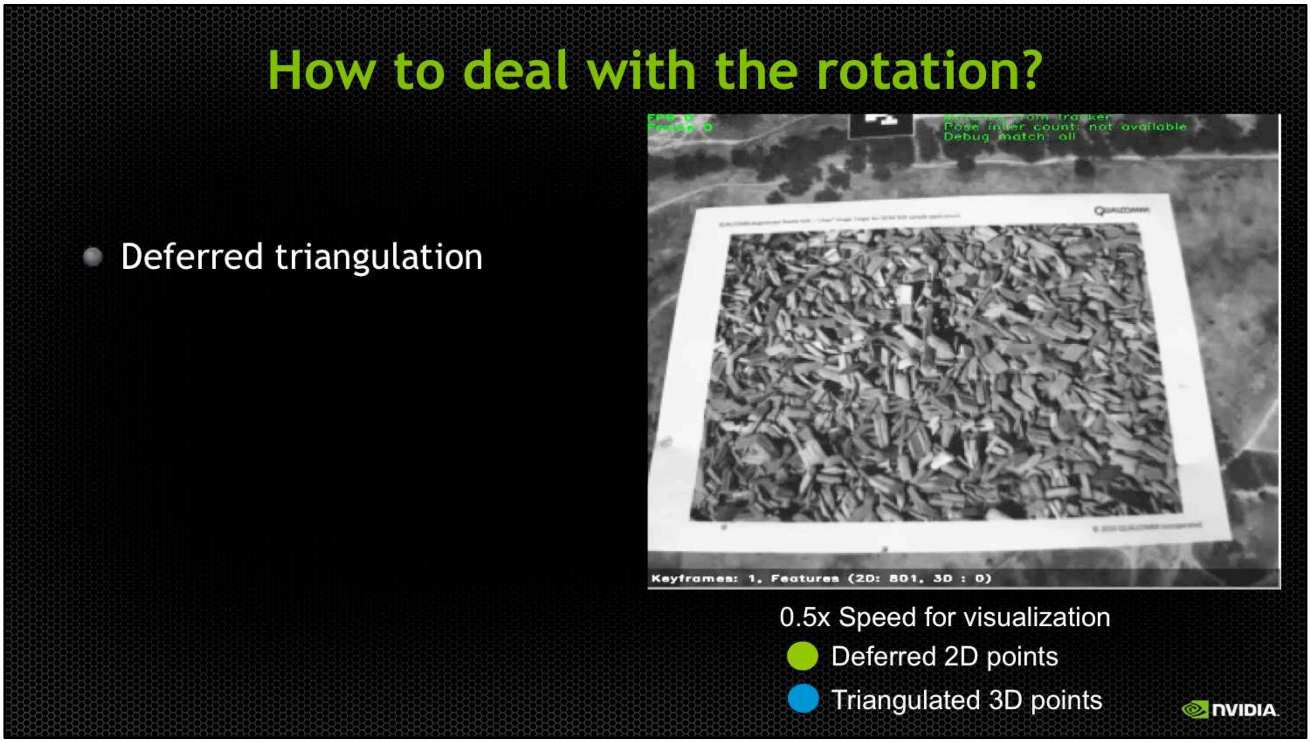
[†]University of Oulu [‡]NVIDIA Research

3DV 2014

We built on this work and got a paper accepted to 3D Vision conference, in December this year.

The work is collaboration with our ex-intern from U. Oulu in Finland and his advisors and colleagues.

Kihwan Kim is the project leader at NVIDIA and was Daniel’s mentor.



How to deal with the rotation?

- Deferred triangulation

```
FPS 0
Frame 0
Matches from tracker
Pose inlier count: not available
Debug match: all
Keyframes: 1, Features (2D: 801, 3D : 0)
```

0.5x Speed for visualization

● Deferred 2D points

● Triangulated 3D points

The main points in our system are the following.

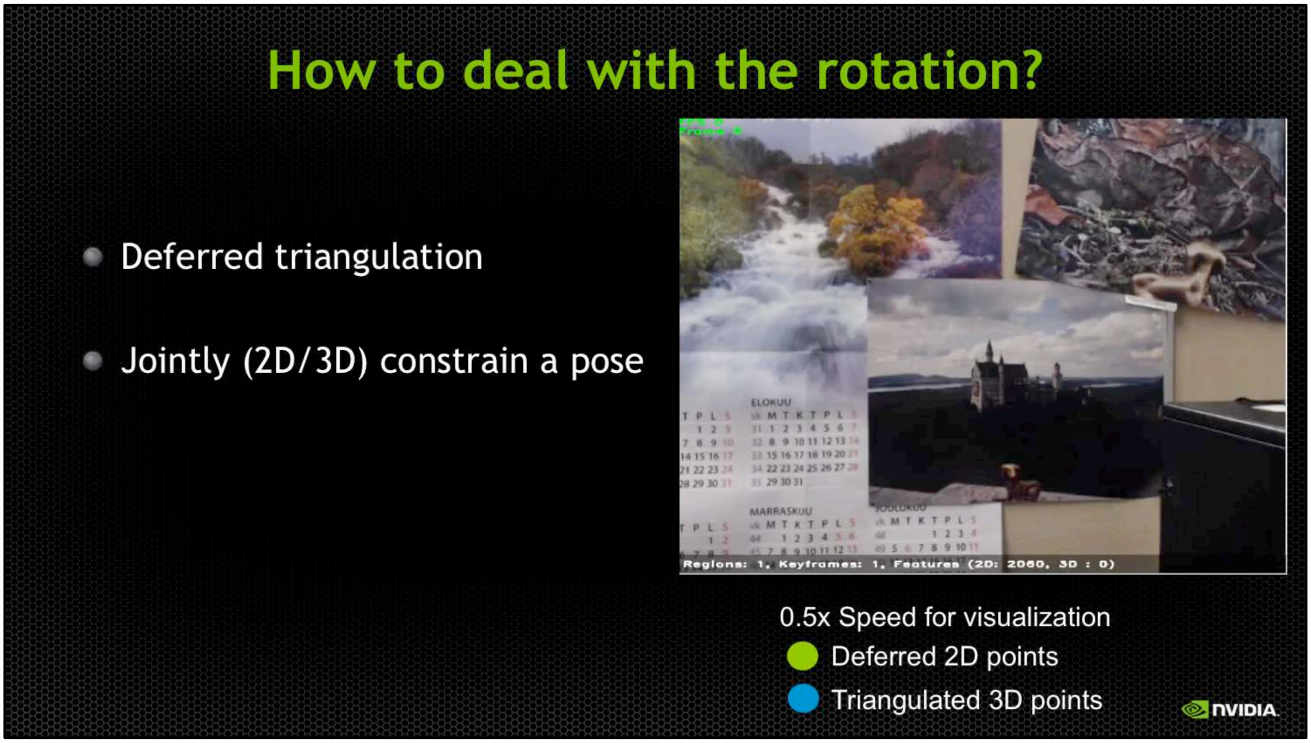
First, we start tracking each point in 2D, in image space, shown in green, using optical flow.

Only once we get at least two views that are not at the same spot,

that have a sufficiently long baseline,

the 2D features are promoted into 3D points, these are shown in blue.

(File name : deferred_slow.mp4)



How to deal with the rotation?

- Deferred triangulation
- Jointly (2D/3D) constrain a pose

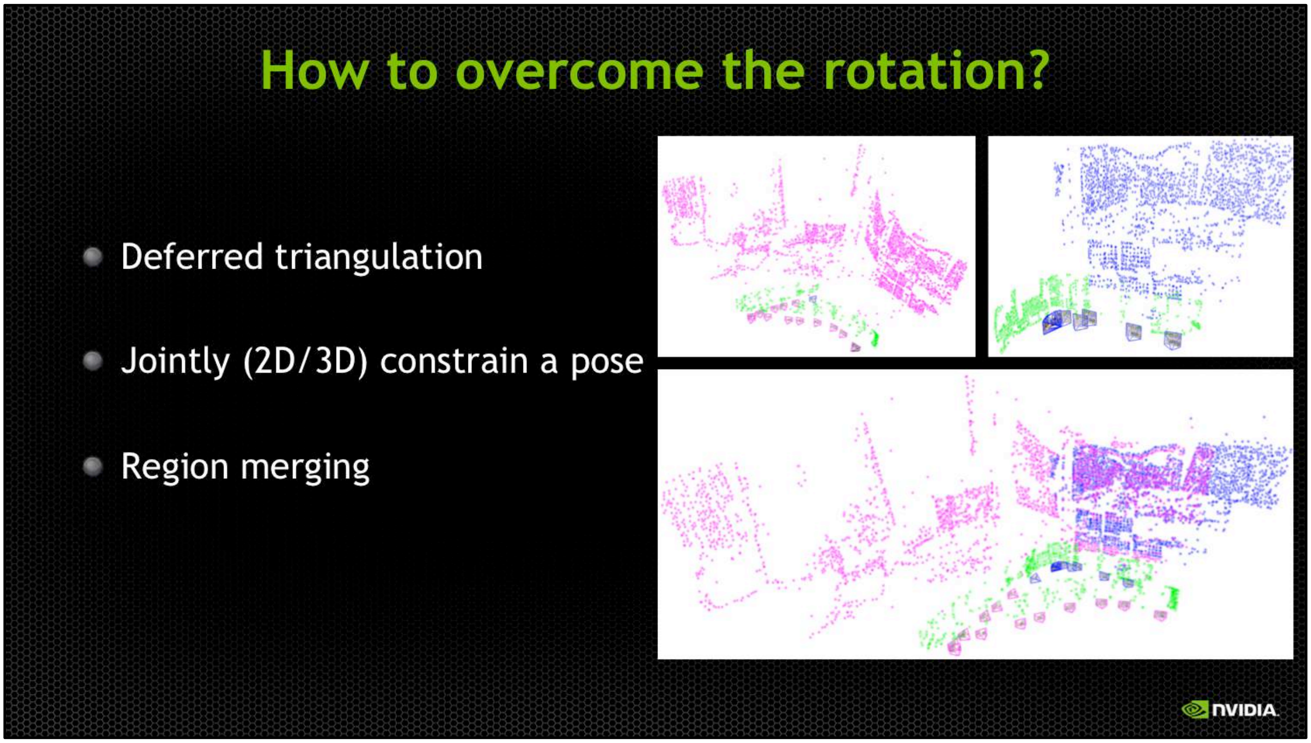
```
Regions: 1, Keyframes: 1, Features (2D: 2060, 3D : 0)
```

0.5x Speed for visualization

- 🟢 Deferred 2D points
- 🔵 Triangulated 3D points

When we track the camera motion,
We constrain the pose using both the 2D and 3D matches.
I’ll give a bit more details how we do that shortly.

(file name : joint_pose.mp4)



How to overcome the rotation?

- Deferred triangulation
- Jointly (2D/3D) constrain a pose
- Region merging

The third point is how we deal with disjoint model regions.

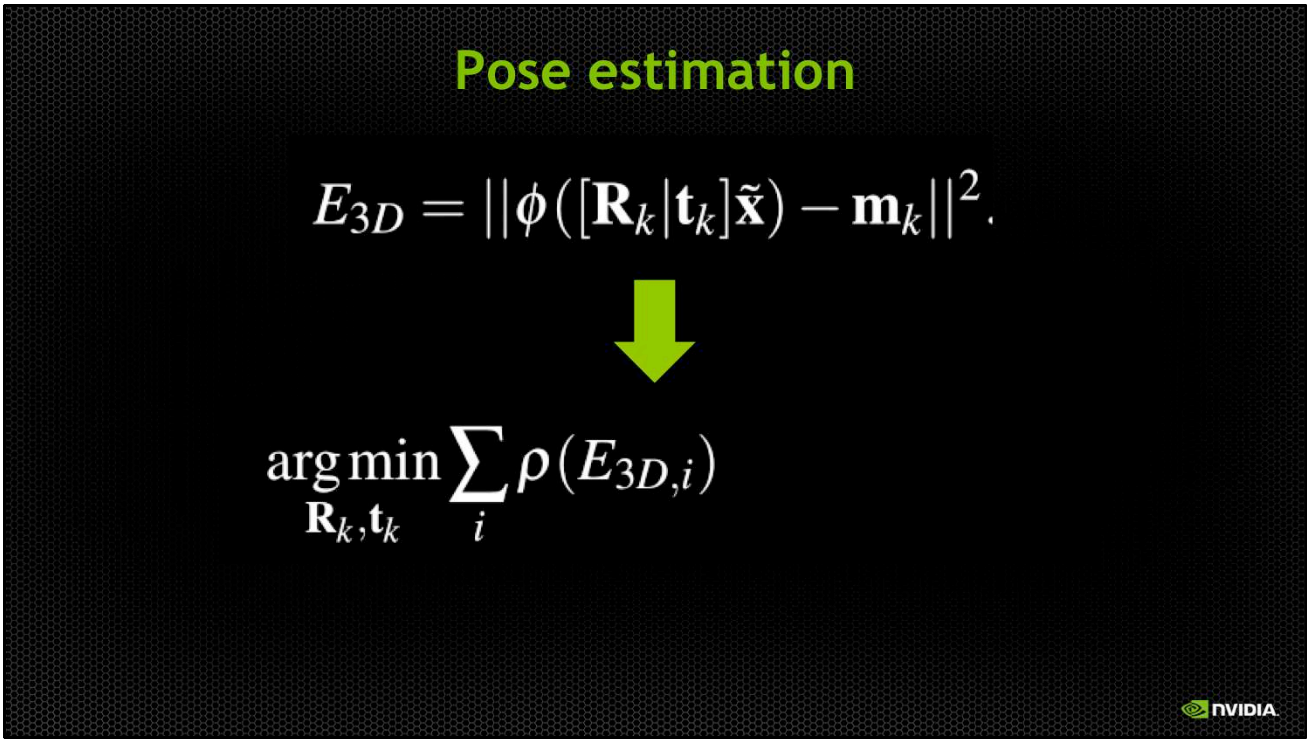
Here’s one part of the scene that we have already modeled.

<click> Now the camera was moved so fast that tracking failed.

The camera sees a different part of the scene, and has to start building a new model, shown in blue.

The problem is that the scale of the model is pretty arbitrary, and we don’t know how the new model relates to the old one.

<click> Later, if we get a keyframe that can see part of both the red and blue models, we can merge the models and match the scales.



Pose estimation

$$E_{3D} = ||\phi([\mathbf{R}_k|\mathbf{t}_k]\tilde{\mathbf{x}}) - \mathbf{m}_k||^2.$$

$$\downarrow$$

$$\arg\min_{\mathbf{R}_k,\mathbf{t}_k} \sum_i \rho(E_{3D,i})$$

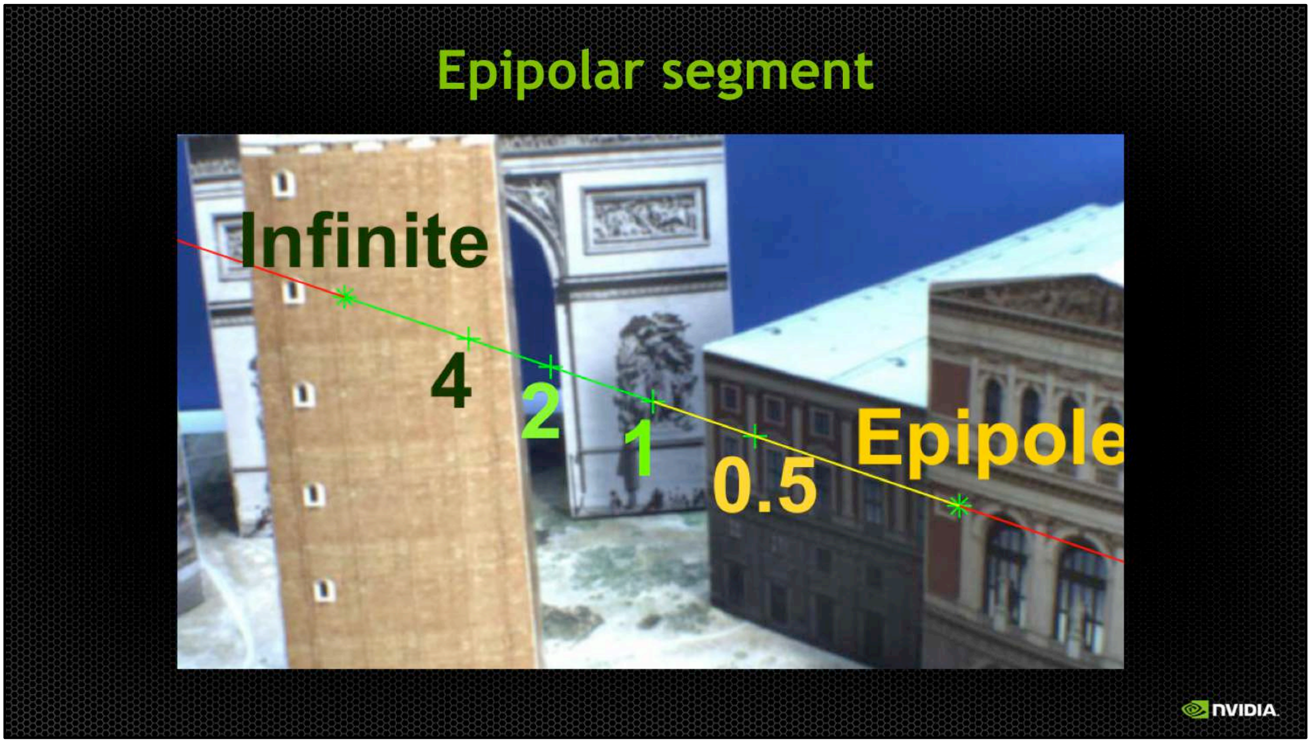
We do pose estimation as follows.

First, we minimize the error between 2D feature location, and 3D model point, with the current pose, projected to the image plane.

<click> We find the rotation and translation that minimize this error.

This is how PTAM and SFM does it too.

<click> Then we add a 2D term for those points that are being tracked but haven’t been triangulated yet.



Epipolar segment

Let’s see how the 2D error is defined.

Given a point that is visible in one of the previous views,

and given that we have a good estimate of the relative camera poses of the two views,

we can project to the current image all the potential locations where that pixel could project to.

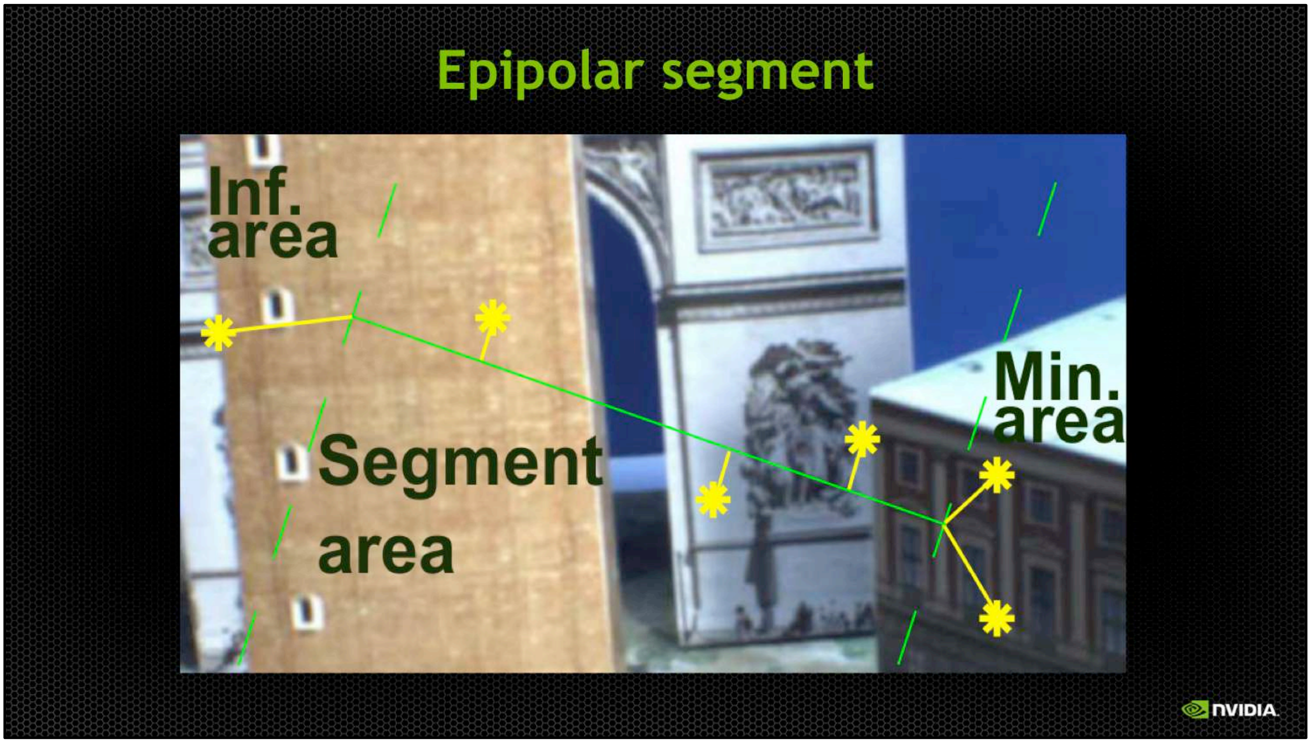
This set of points is called the epipolar line.

Additionally, we know that the match can’t be before the epipole,

which corresponds to the center of the other camera,

as those locations are behind the other camera.

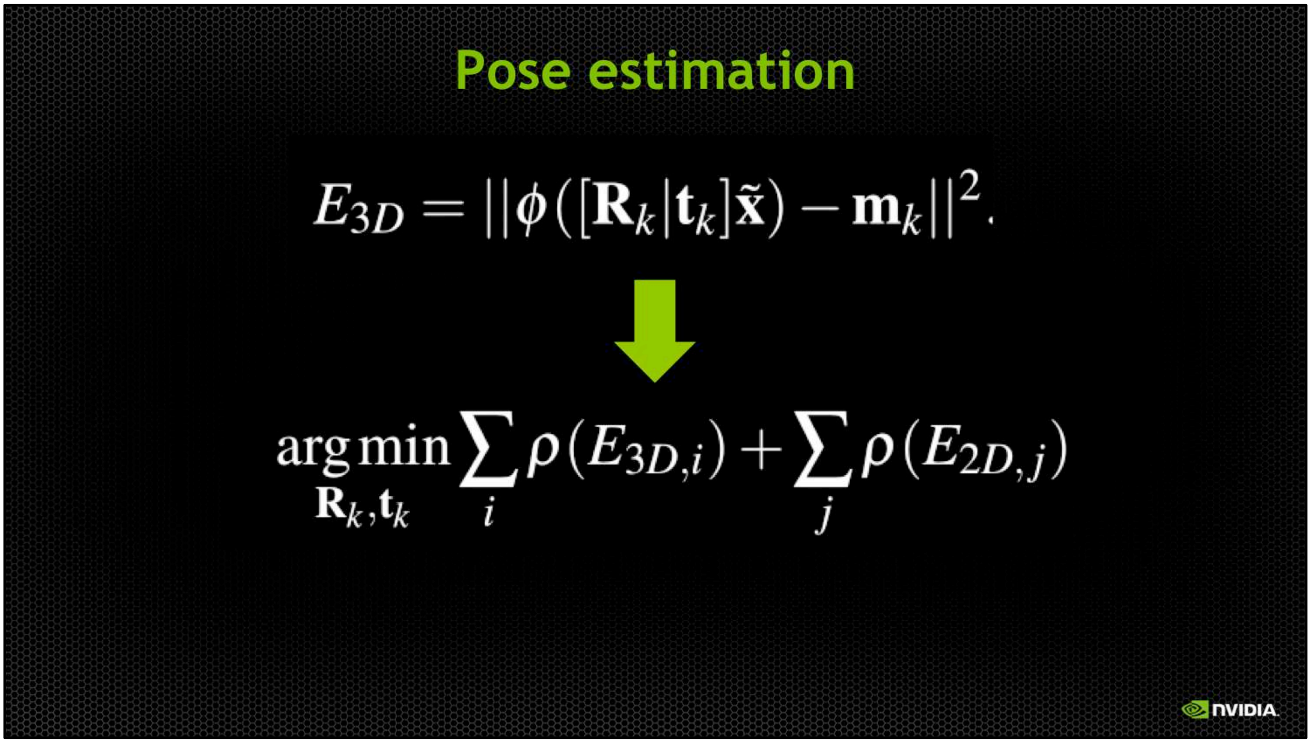
Similarly, it can’t be beyond the vanishing point.



Epipolar segment

Now the 2D penalty is defined like this.
We first find a candidate location in the current image
that matches the image feature in a previous frame.

We then determine the epipolar segment corresponding to that feature.
The 2D penalty is simply the Euclidian distance in the image plane to the epipolar segment.



Pose estimation

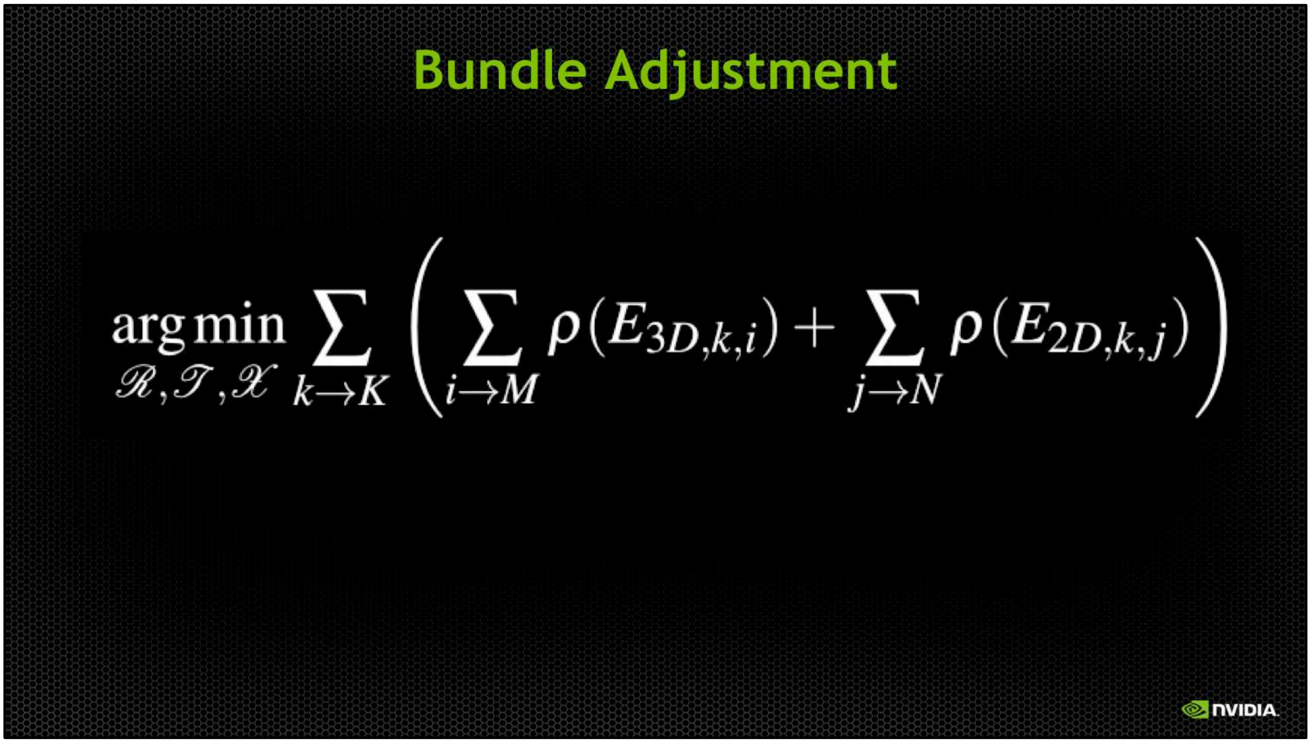
$$E_{3D} = ||\phi([\mathbf{R}_k|\mathbf{t}_k]\tilde{\mathbf{x}}) - \mathbf{m}_k||^2.$$

$$\downarrow$$

$$\underset{\mathbf{R}_k,\mathbf{t}_k}{\arg\min} \sum_i \rho(E_{3D,i}) + \sum_j \rho(E_{2D,j})$$

Again, here’s the combined matching cost.

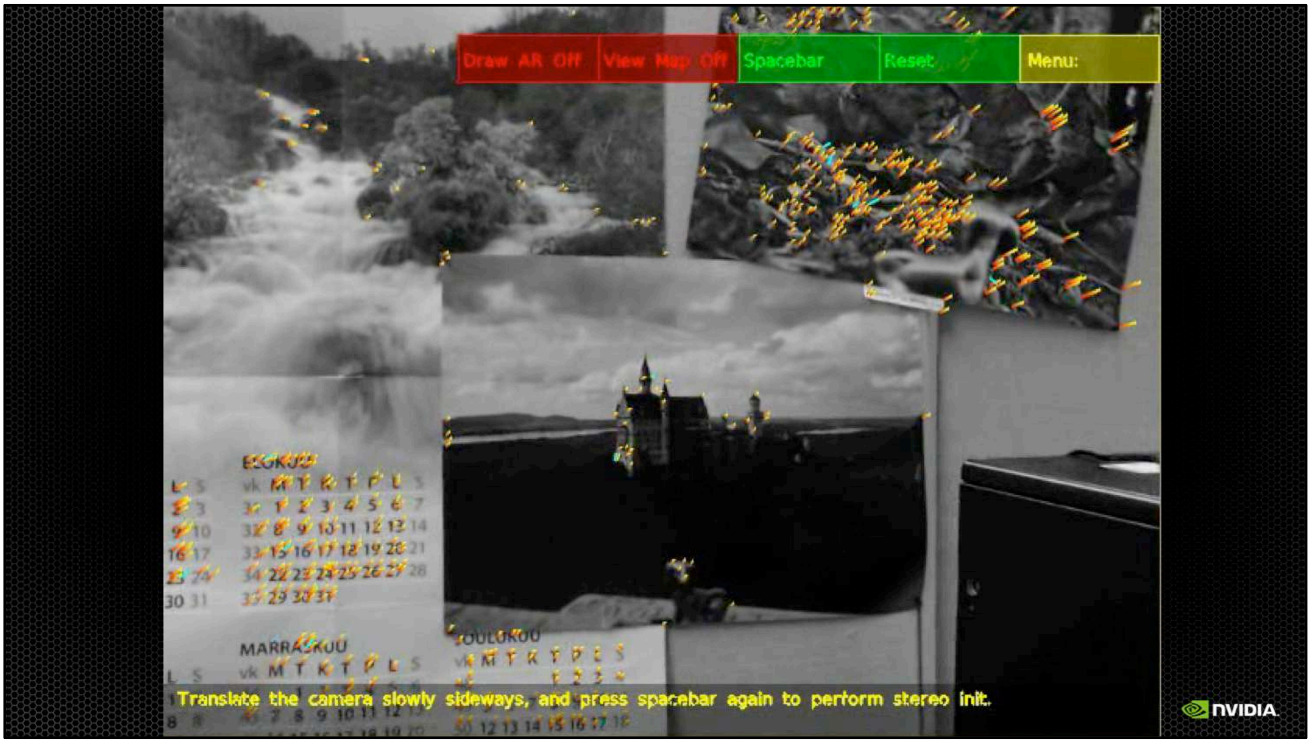
The function rho is a more robust cost function than just squaring the error, and reduces the impact of outliers.



Bundle Adjustment

$$\underset{\mathscr{R},\mathscr{T},\mathscr{X}}{\arg\min} \sum_{k\to K} \left(\sum_{i\to M} \rho(E_{3D,k,i}) + \sum_{j\to N} \rho(E_{2D,k,j}) \right)$$

Then we repeat this over all keyframes

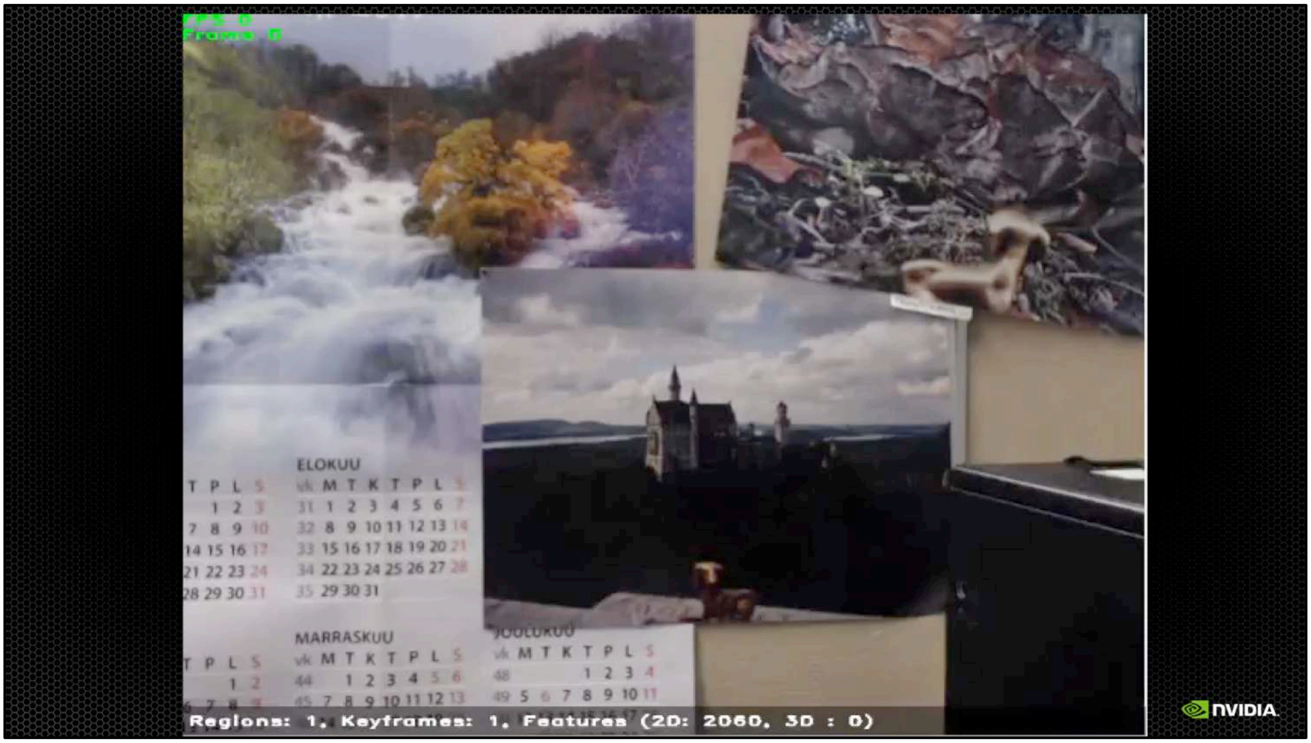


Translate the camera slowly sideways, and press spacebar again to perform stereo init.

Here is an image sequence that contains a lot of pure rotations.

This system is a traditional PTAM like SLAM system.

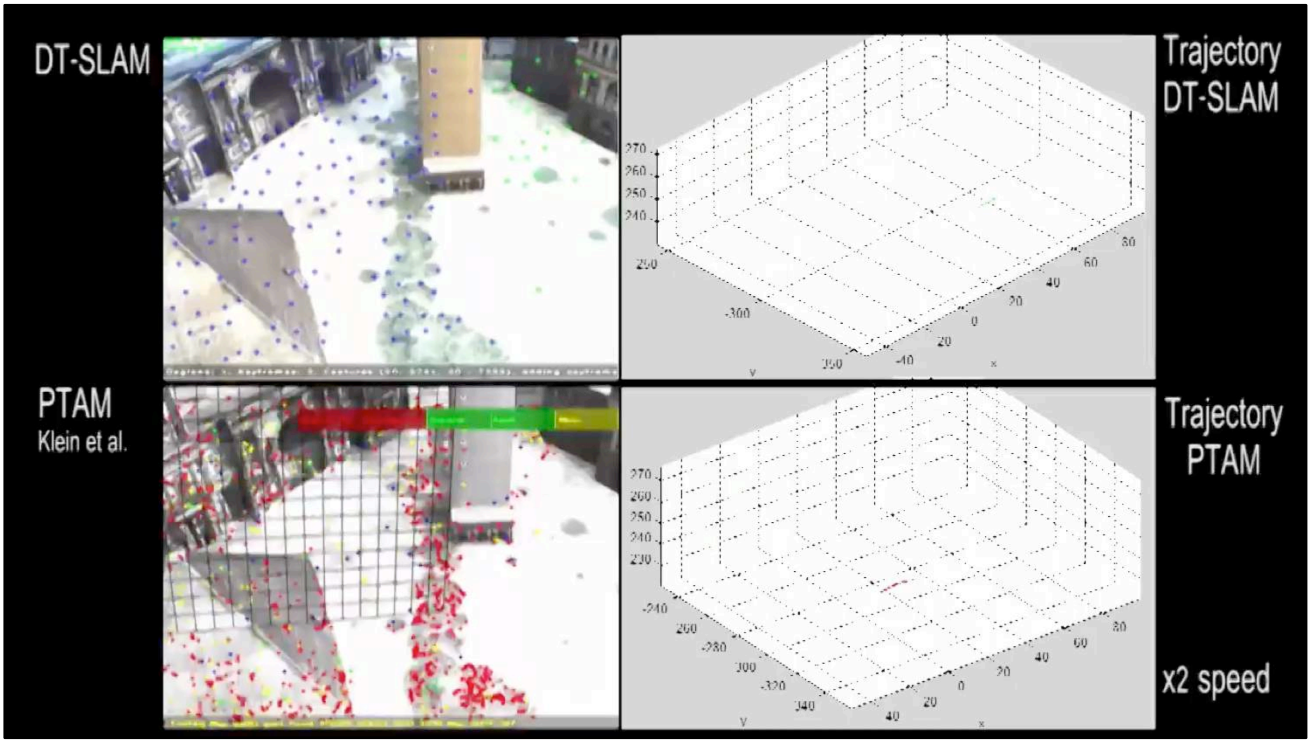
The tracking starts well, but once the pure rotations start, the system loses track, and the coordinate system jumps erratically.



```
FPS 0
Frame 0
Regions: 1, Keyframes: 1, Features (2D: 2060, 3D : 0)
```

Here’s our result with the same input video.
The system behaves nicely.

(File : rotation_test_new.mp4) : I changed this video!



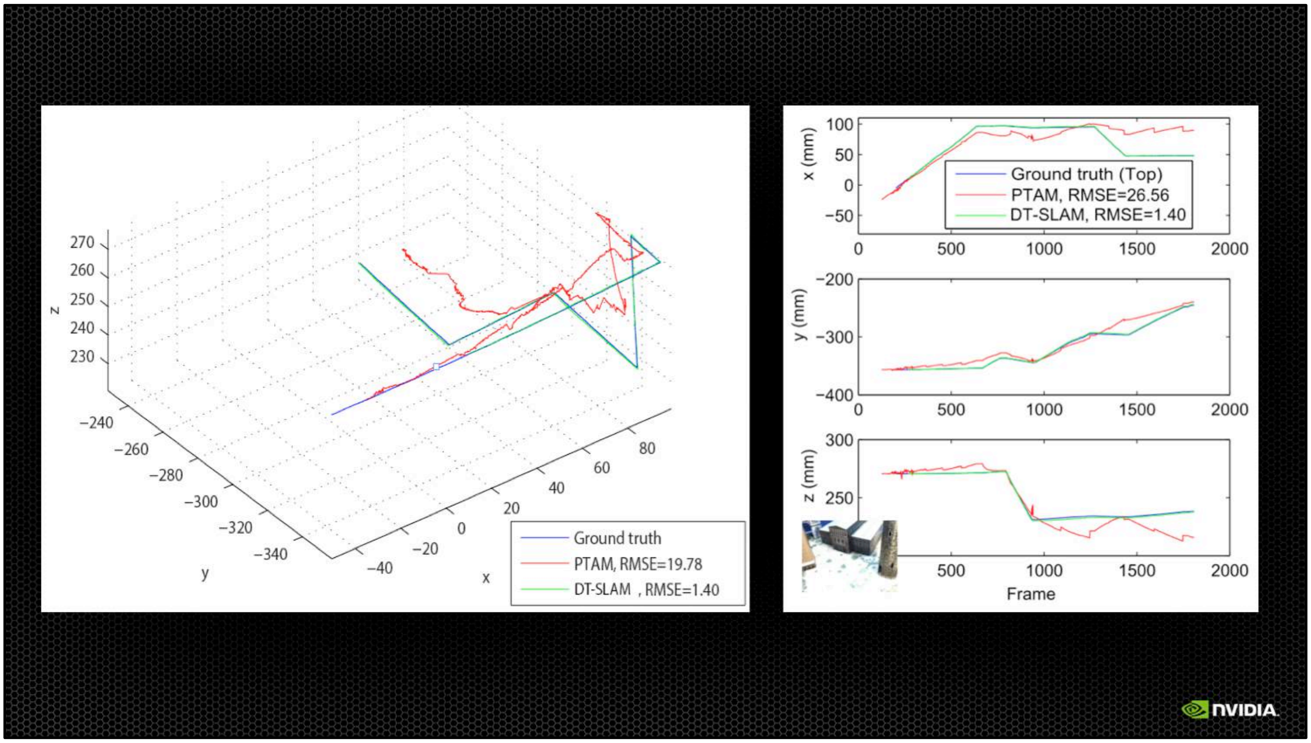
We evaluate the accuracy of the pose estimation using a popular City of Sights dataset,

which has ground truth captured with a robot arm.

As you see, the path of DT-SLAM is much more stable.

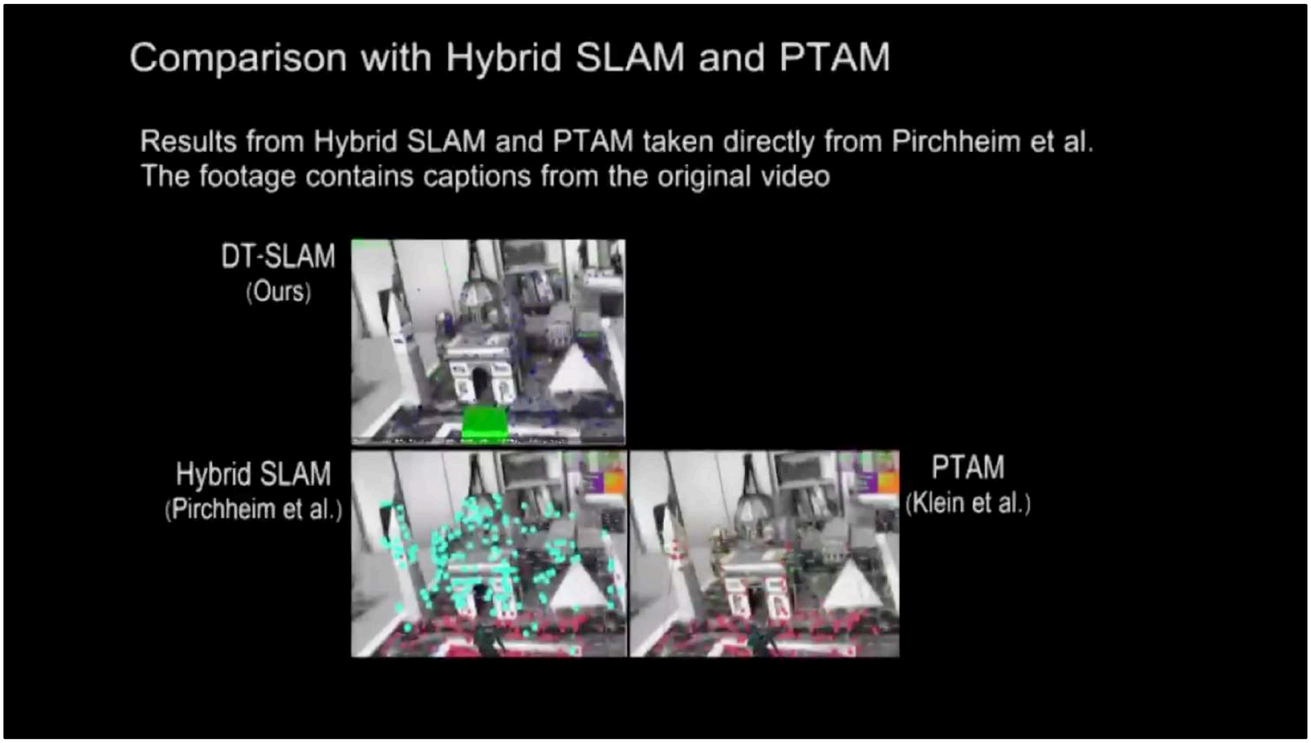
In the next slide, we see a comparison with the ground truth data

File : dtslam-ptam-quanti.mp4



Here are the results against the ground truth.
DT-SLAM is quite close the the true solution,
While PTAM deviates a lot.

More results can be found in our paper when it’s out shortly.



Comparison with Hybrid SLAM and PTAM

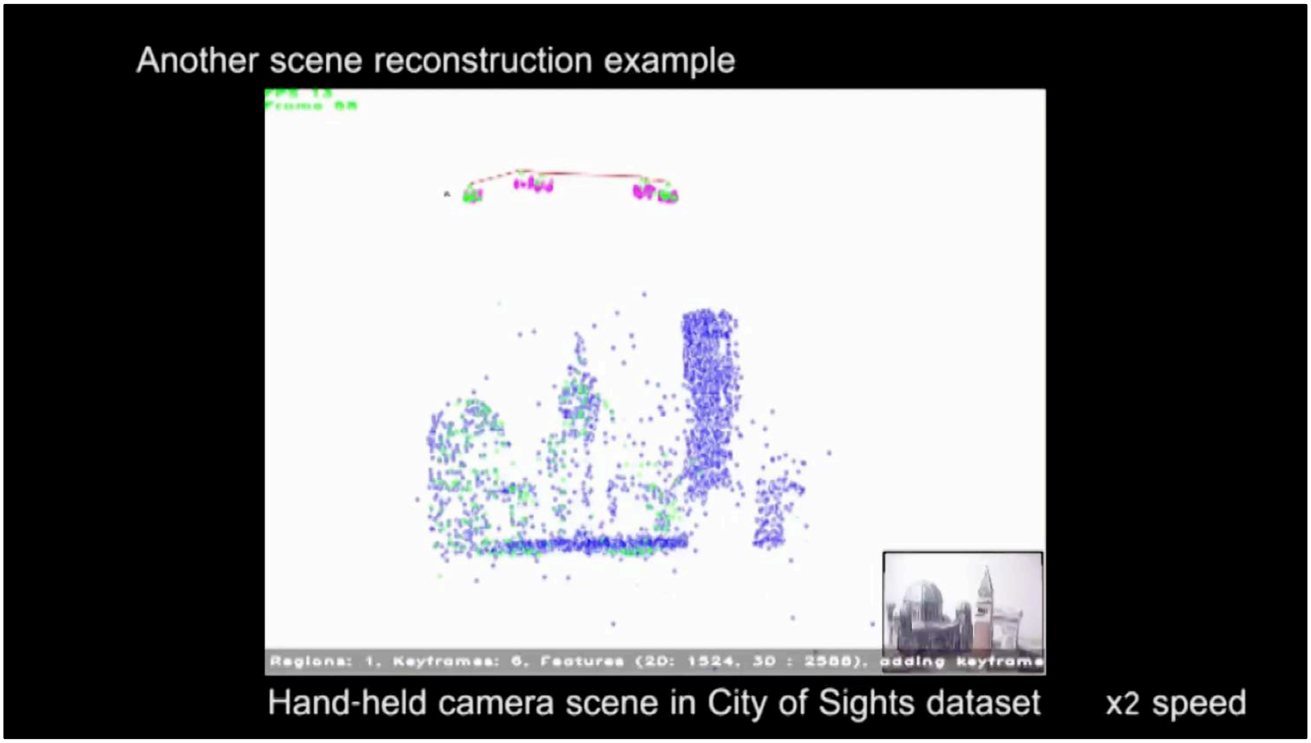
Results from Hybrid SLAM and PTAM taken directly from Pirchheim et al.
The footage contains captions from the original video

Here we show a direct comparison with Pirchheim’s hybrid SLAM.

The videos provided by the authors are very challenging as there are many rotations, and classic methods like PTAM simply fail.

The tracked points seem to swim and not really stick to the scene in the Hybrid SLAM system,
especially when it switches the mode
between 3D and 2D tracking modes.

File ; dtslam-vs-hybrid.mp4



Another scene reconstruction example

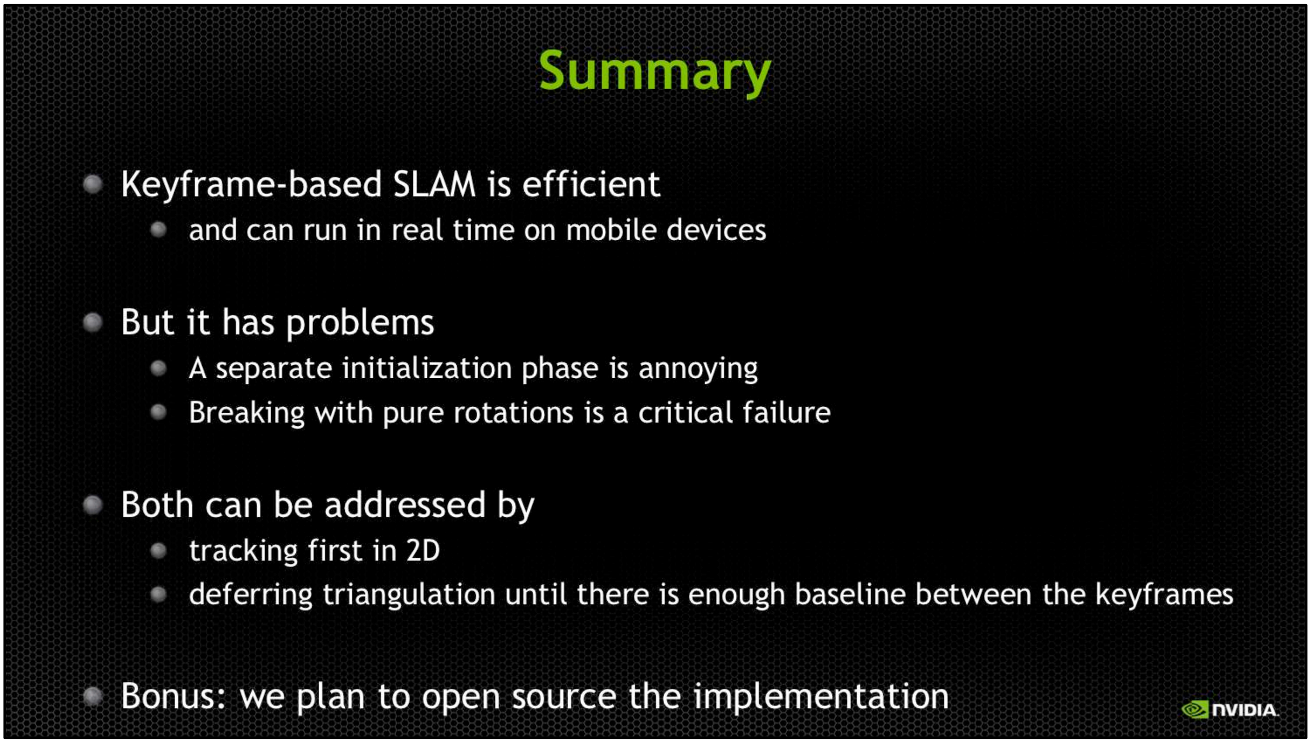
```
Regions: 1, Keyframes: 6, Features (2D: 1524, 3D : 2588), adding keyframe
```

Hand-held camera scene in City of Sights dataset x2 speed

Here is a final example scanning the City of Sights dataset.

The camera view is shown at bottom right,

the rest of the image shows the camera and keyframe locations and the modeled 3D points.



Summary

- Keyframe-based SLAM is efficient
  - and can run in real time on mobile devices
- But it has problems
  - A separate initialization phase is annoying
  - Breaking with pure rotations is a critical failure
- Both can be addressed by
  - tracking first in 2D
  - deferring triangulation until there is enough baseline between the keyframes
- Bonus: we plan to open source the implementation