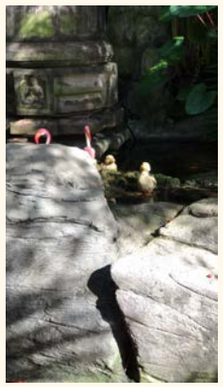

I’m here to talk about 3D photography…

In other words, technologies that enable anyone to capture a place using a camera that they already own. Converting the place into a digital representation that allows anyone to later revisit it in virtual reality.

- As shown here.
- In this talk, we focus on armchair VR experiences, where you can sit comfortably and lean form side to side to peek behind objects.

## Single photograph

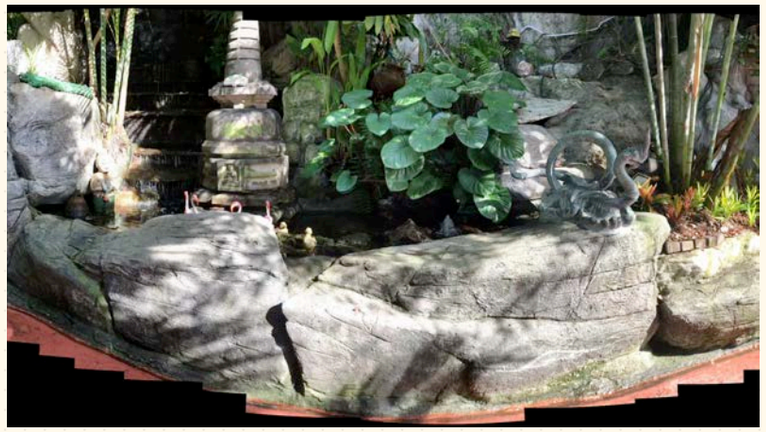


But before we delve into the details, let’s briefly talk about alternative representations for this type of content.

A normal photograph is not enough, as it has a limited field of view.

# Panorama stitching

APAP [Zaragoza2013]
Parallax-tolerant [Zhang2014]
SEAGULL [Lin2016]
...



You see more of the scene by stitching many images into a seamless panorama.

However, the result lacks depth cues and looks flat in VR. We'd like to provide a more immersive experience

## Omnistereo

### Binocular parallax

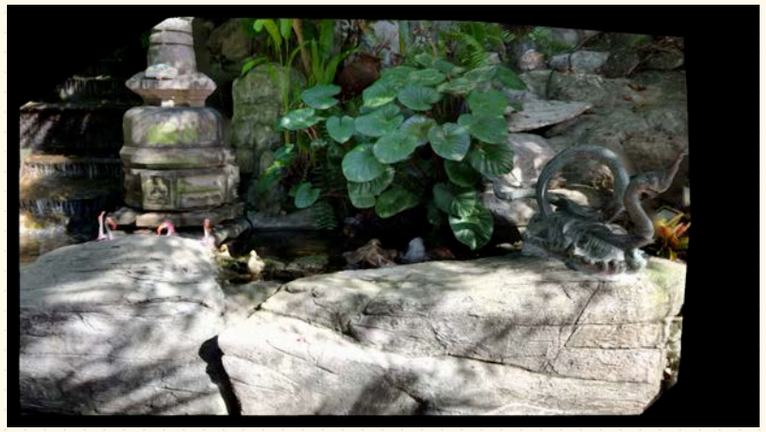
Stereo panoramas [Peleg1999]
Megastereo [Richardt2013]
Google Jump [Anderson2016]
SpinVR [Konrad2016]
...

The omnistereo representation stitches two separate panoramas —one for the left eye and one for the right eye. This provides depth perception through binocular parallax, but the perspective is still fixed in the scene.

# 3D Interpolation

## Binocular & motion parallax

Concentric mosaics [Shum1999]
Parallax360 [Luo2018]
MegaParallax [Bertel2019]
...



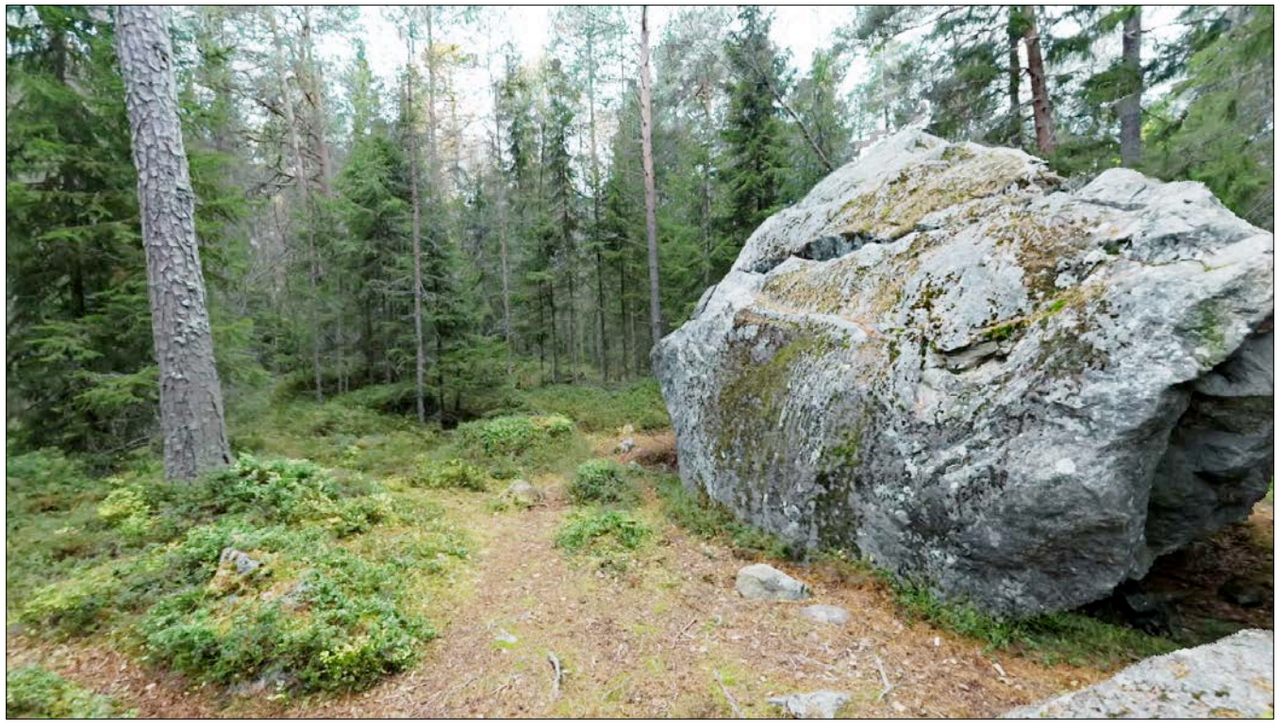
In the previous talk, we saw representations that allowed for 3D interpolation between camera positions. This is more immersive — and provides both binocular and motion parallax. Unfortunately, these representations impose constraints on the range of motion when viewing the scene.

# 3D Extrapolation

## Binocular & motion parallax

MVE [Goesele2007,Fuhrmann2014]
PMVS [Furukawa2010]
COLMAP [Schoenberger2016]
Casual3D [Hedman2017]
DeepView [Flynn2019]
Local Light Field Fusion [Mildenhall2019]
...

In this talk, we'll be discussing representations that allow for slight viewpoint extrapolation —which enable viewers to freely move their heads around and peek behind objects.



... Specifically, we'll be talking about the "3D photo" representation. It is similar to a panorama, ...

**<depth appears>**

... but with extra channels for depths, ...

**<normal map appears>**

... and sometimes normals.

**<Color re-appears, camera starts swinging with parallax>**

It also has multiple layers, so we can move the camera in 3D, experience motion parallax and peek behind objects.

**<camera flies up>**

The illusion breaks if you move really far away, ...

**<...and back down>**

…, but 3D photos look great when staying close to the original views.

Because we have explicit 3D geometry we can interact with the 3D photo in ways that are not possible with normal photos.

**<water effect appears>**

For example, we can do things like flood the scene half-way with water.

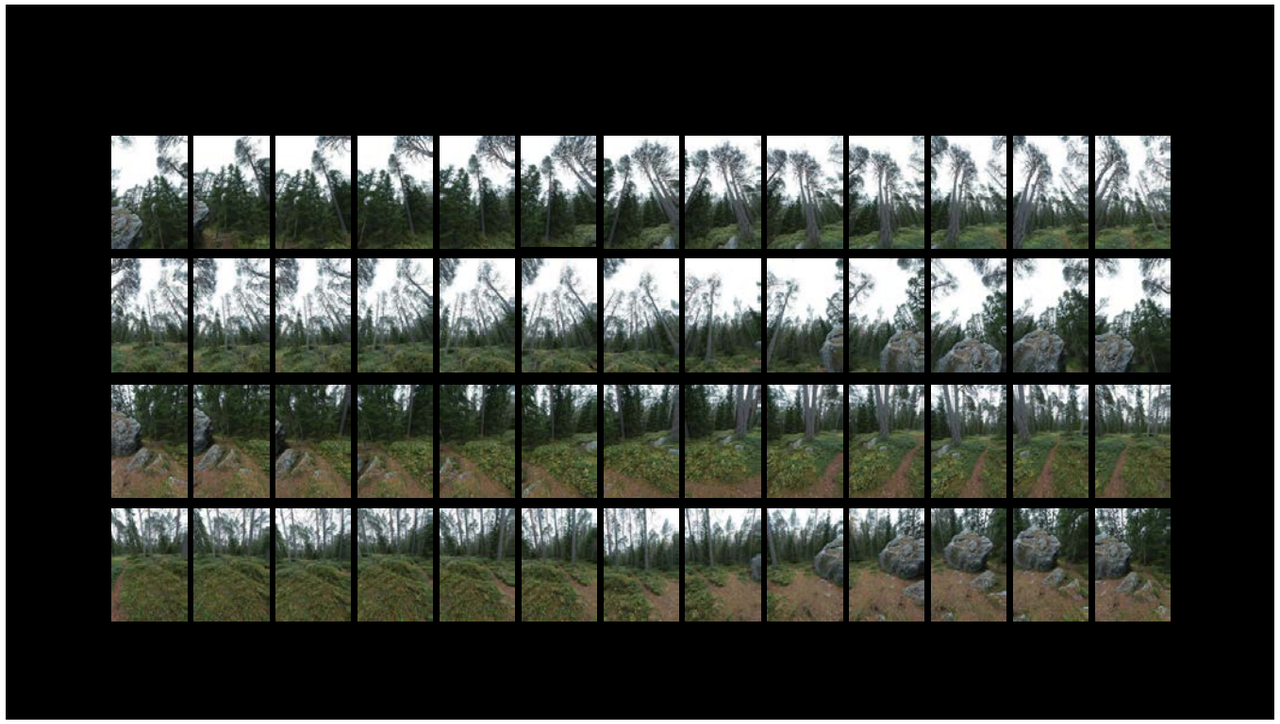
**<light effect appears>**

And our if we have normals, we can also play around with the lighting. We can turn day into night and shoot laser beams out of the ground!

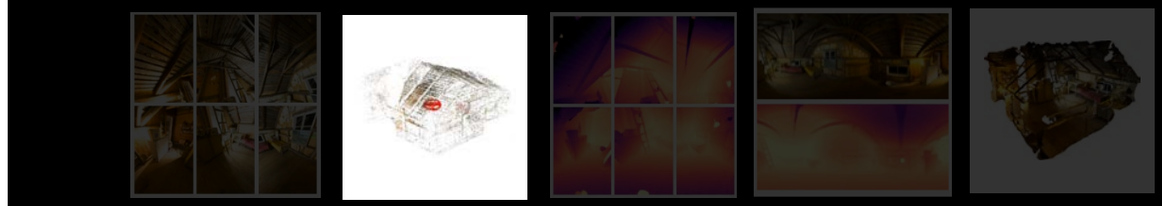
# Casual 3D Photography

Peter Hedman, Suhib Alsisan, Richard Szeliski, Johannes Kopf
SIGGRAPH Asia 2017


Let’s first talk about a pipeline which builds 3D photos from images captured with any off-the-shelf camera.



- In order to make the reconstruction work, we need some overlap in the images.
- Every point in the scene should be seen in 3-5 images so we can reconstruct its depth reliably.
- So, all in all, with a DSLR and fisheye you need about two rings of images with 20 - 25 images each to capture a complete 360 x 180 panorama that covers all directions.



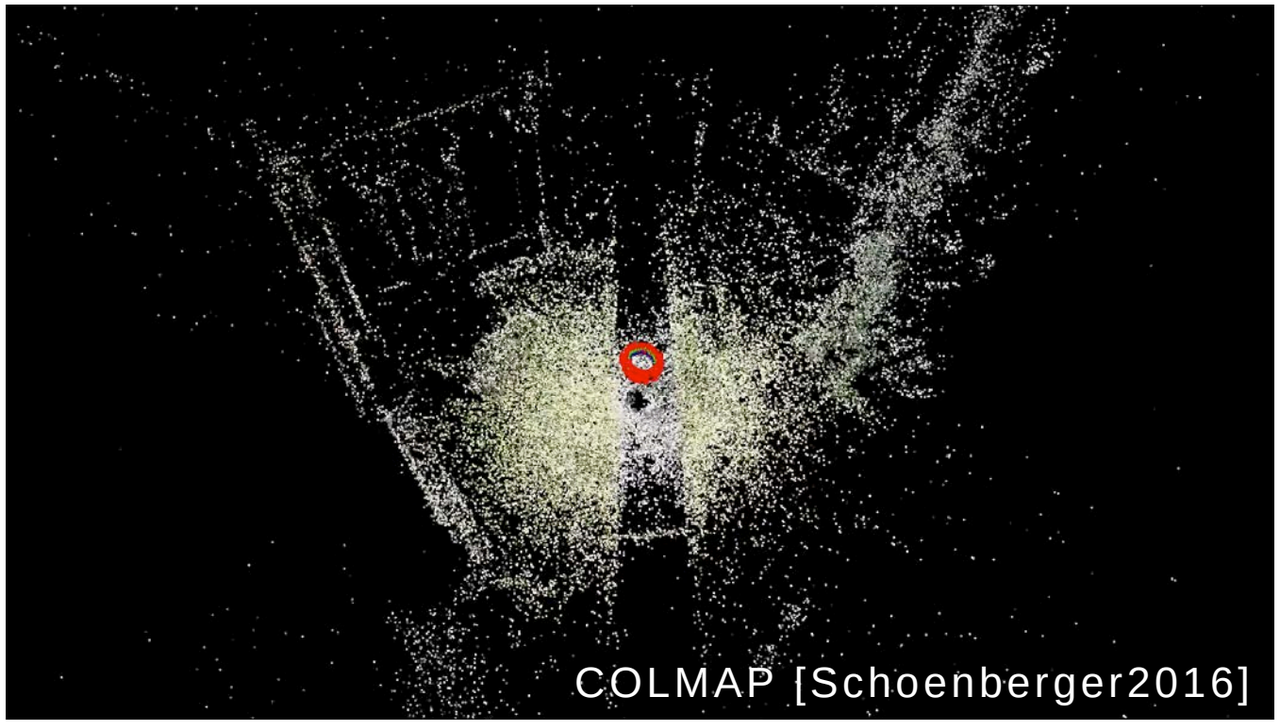
Input: Pictures

Sparse reconstruction

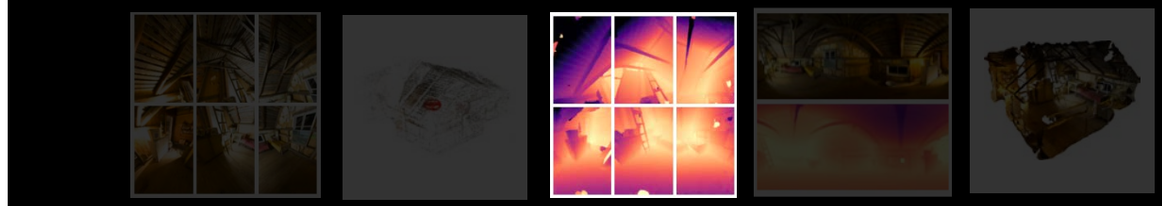
Dense depth

Stitching

Output: 3D Photo




- After we've captured the images, we need to answer the questions "where were the images captured from"?
- This is a fairly standard problem in computer vision, and we use a state-of-the-art structure-from-motion algorithm to do this.
- The algorithm triangulates the camera locations: things, just from the image data:
  - This is visualized here with red crosses. You can nicely see how the algorithm recovered the structure of the rings on which I captured the images.
  - A side-effect of this triangulation is a sparse point cloud of the scene.



Input: Pictures

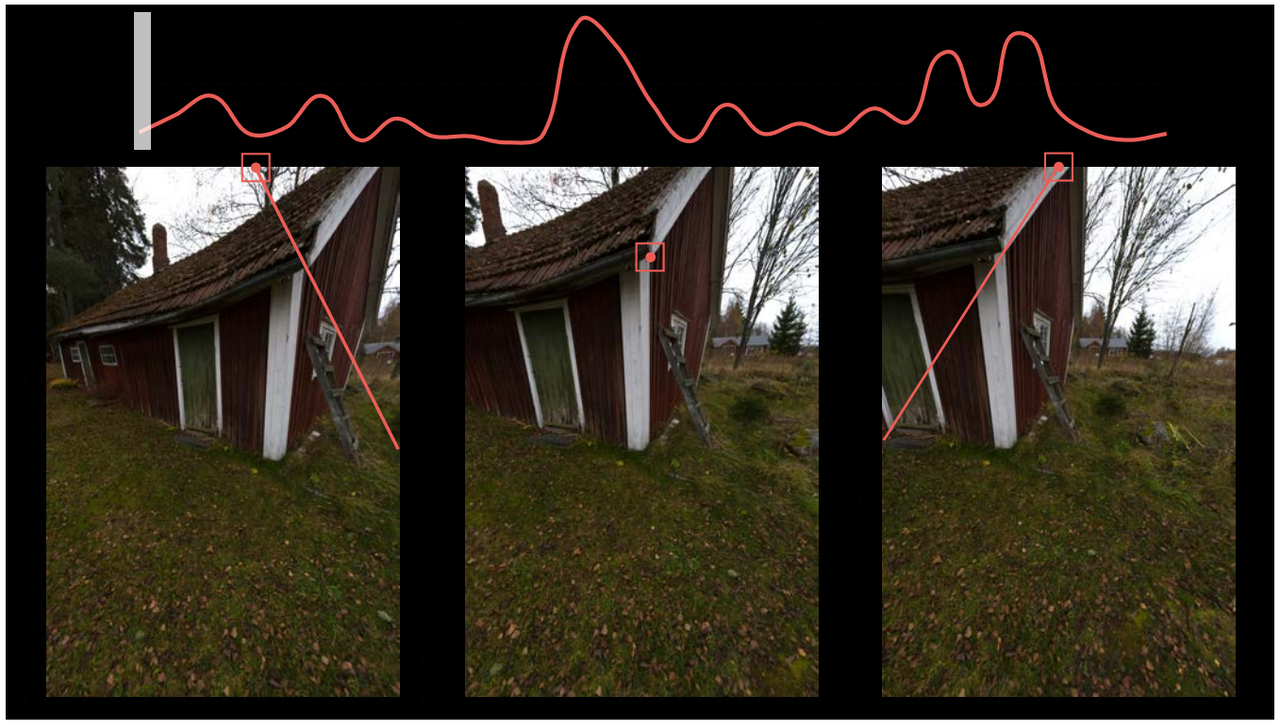
Sparse reconstruction

Dense depth

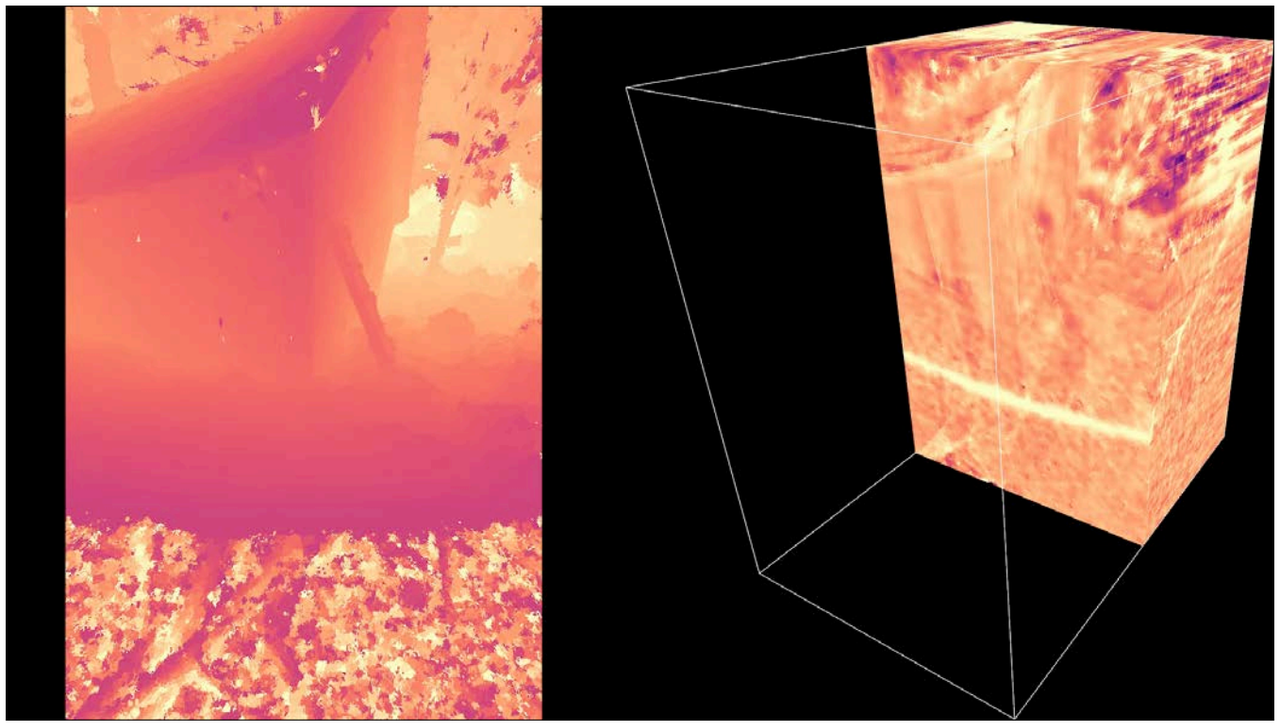
Stitching

Output: 3D Photo

- The next step estimate dense depth maps for every single image in the scene. In other words, we want to estimate the 3D location of every single pixel.
- Here we visualize this information with depth maps, where bright pixels are far away, and dark pixels are closer to the camera.
- This is also a well-known problem in computer vision called “multi-view stereo”.



- Say we want to reconstruct the depth of the corner of the roof seen in the central image.
- The sparse reconstruction tells us how the cameras are position with respect to each other. Based on this, we know that the roof corner will land somewhere along these lines in the neighboring images.
- So we can search along these lines, and look for patches which look similar to the corner in the central image.
- Based on how similar these patches are, we can compute a matching confidence.
- We start our search far away, and move closer to the camera. And hopefully, the confidence reaches a maximum at the correct location, where both patches lie on the corner.
- However, as we try to match patches closer to the camera, it is often likely that we get false positive regions of high confidence.



- Let's visualize this confidence score for every single pixel in the image. I will show you an example of "winner-takes-all" optimization, where we keep track of the best depth we've seen so far.
- We start off by assuming that the depth is infinitely far away, as you can see with the very bright depth map on the left.
- On the right you can see a confidence image, which tells us how confident we are that this far-away depth is correct for each pixel.
- As we move closer to the camera, you can see how the depth map gradually forms.
- If we pause the process half-way, you can see how the depth map is not yet complete, as the ground in the foreground is missing.
- But look at the confidence volume on the right! You can see a bright white line, which tells us that we're very confident that these pixels should have this exact depth.
- As we continue the sweep, we fill in the ground and end up with a complete depth map.

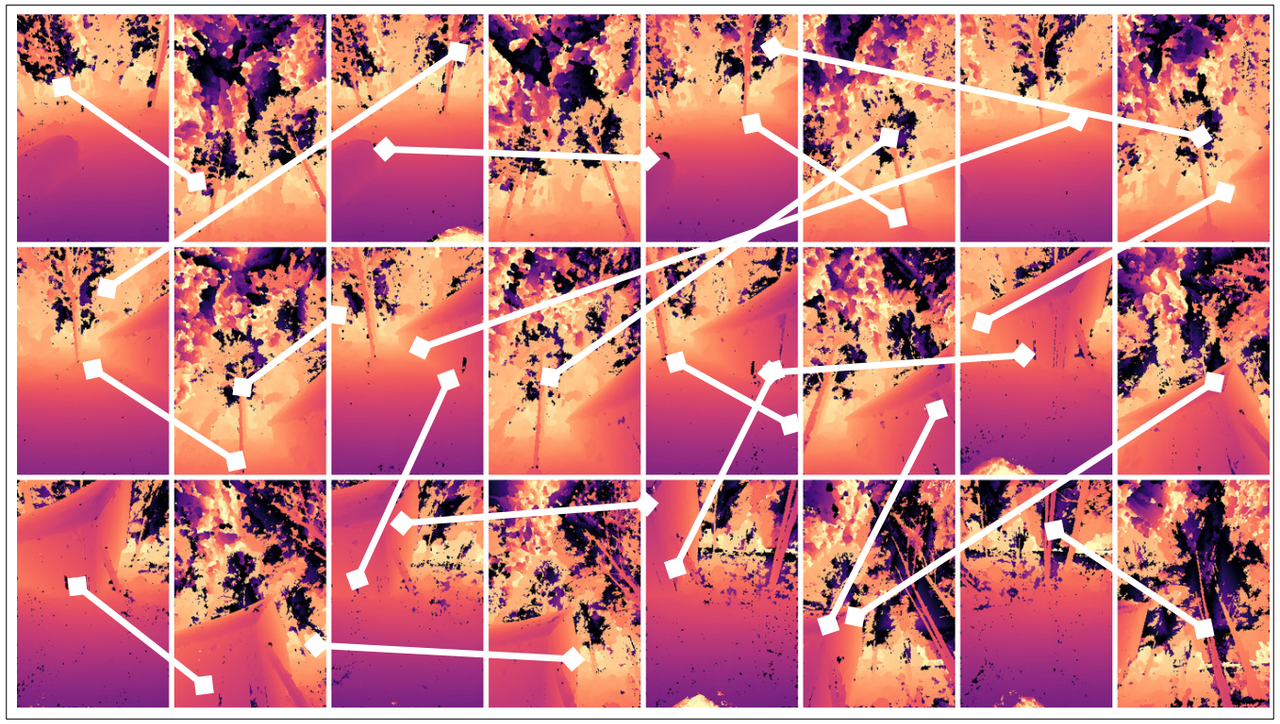
However, this depth map contains artifacts. For example, the dark regions in the top-right corner.

If we take a look at the confidence volume, you can see how these regions correspond to noisy false-positive regions.

These are caused by our capture conditions, since we captured our images one at a time, and the scene is not completely static (for example, trees can be swaying in the wind) it's possible that certain regions will never match completely between images.

Furthermore, as you get closer to the central camera, most of the other images will not be able to see this location. Since we have to compute the confidence score using fewer images, this means that the confidence volume is inherently less reliable nearby.

If we visualize this depth map in 3D, you can see how the dark regions create the so-called "flying pixels" effect, which is very distracting.



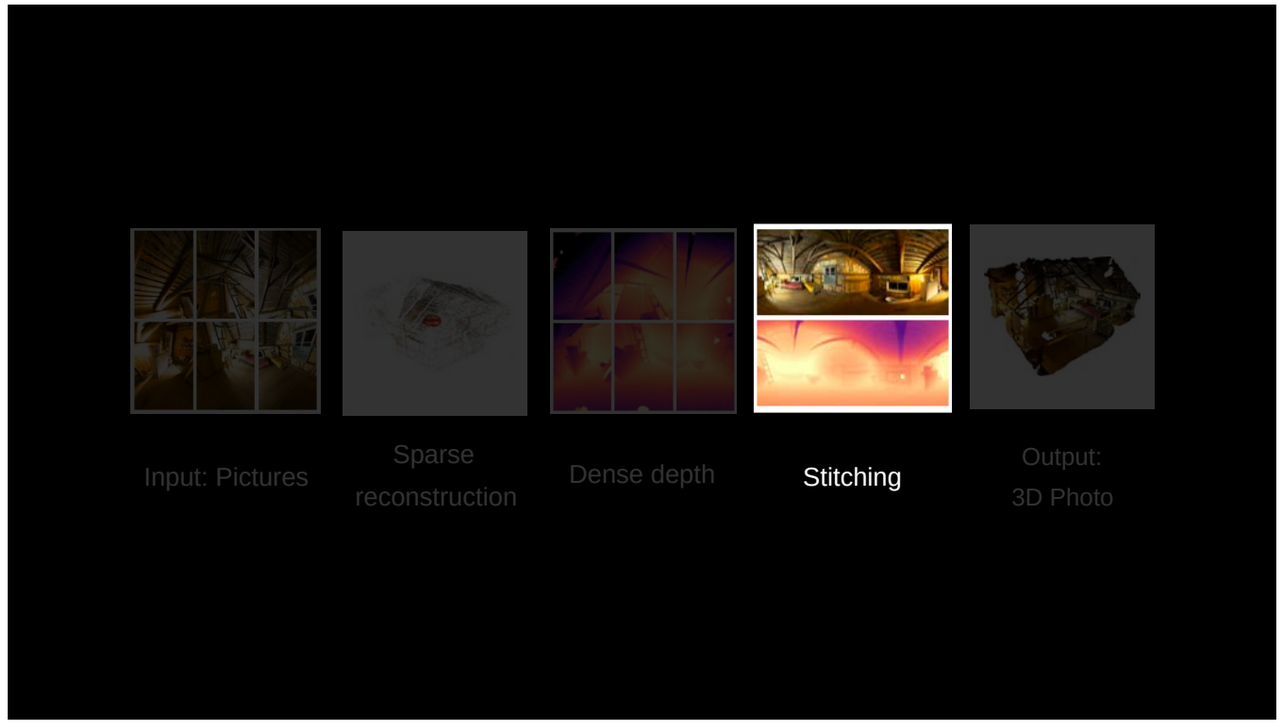
There are many different ways of alleviating these artifacts.

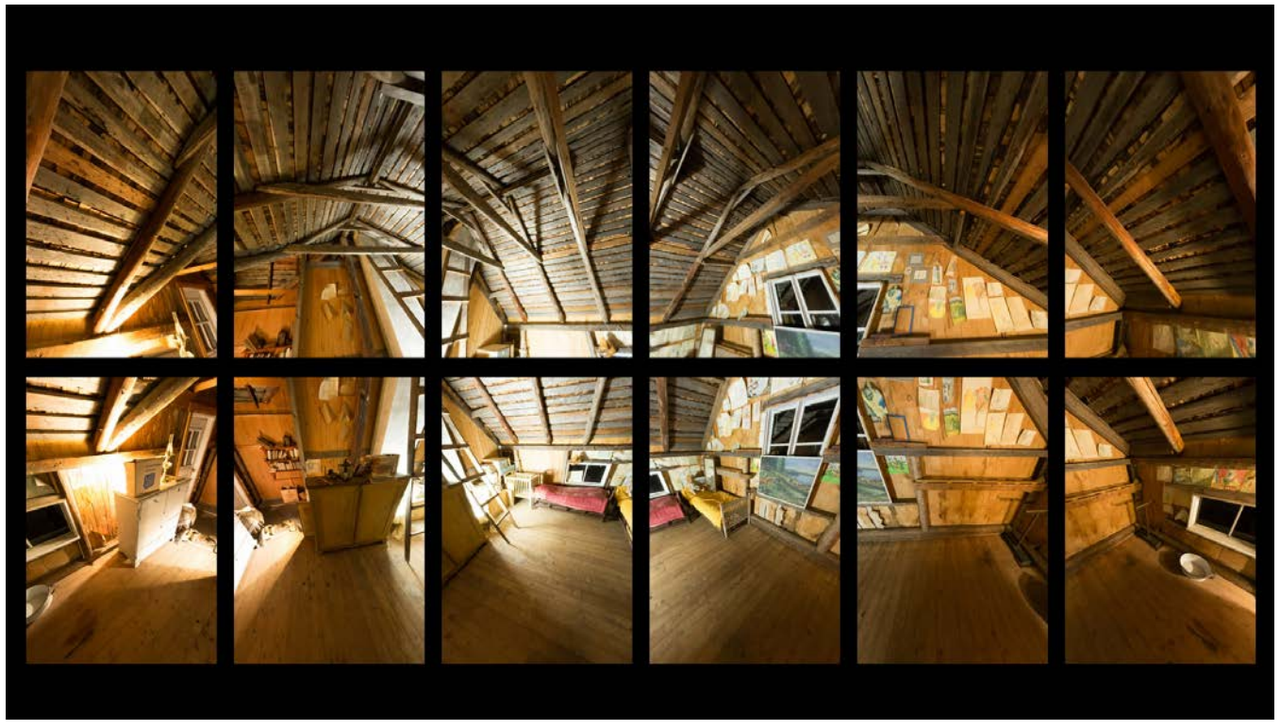
One way I found particularly effective was to first use this cheap winner-takes-all strategy to quickly compute noisy depth maps for each input image.

Then we look for consistencies and inconsistencies between all depth maps. Eventually forming a notion of the free space in the scene.

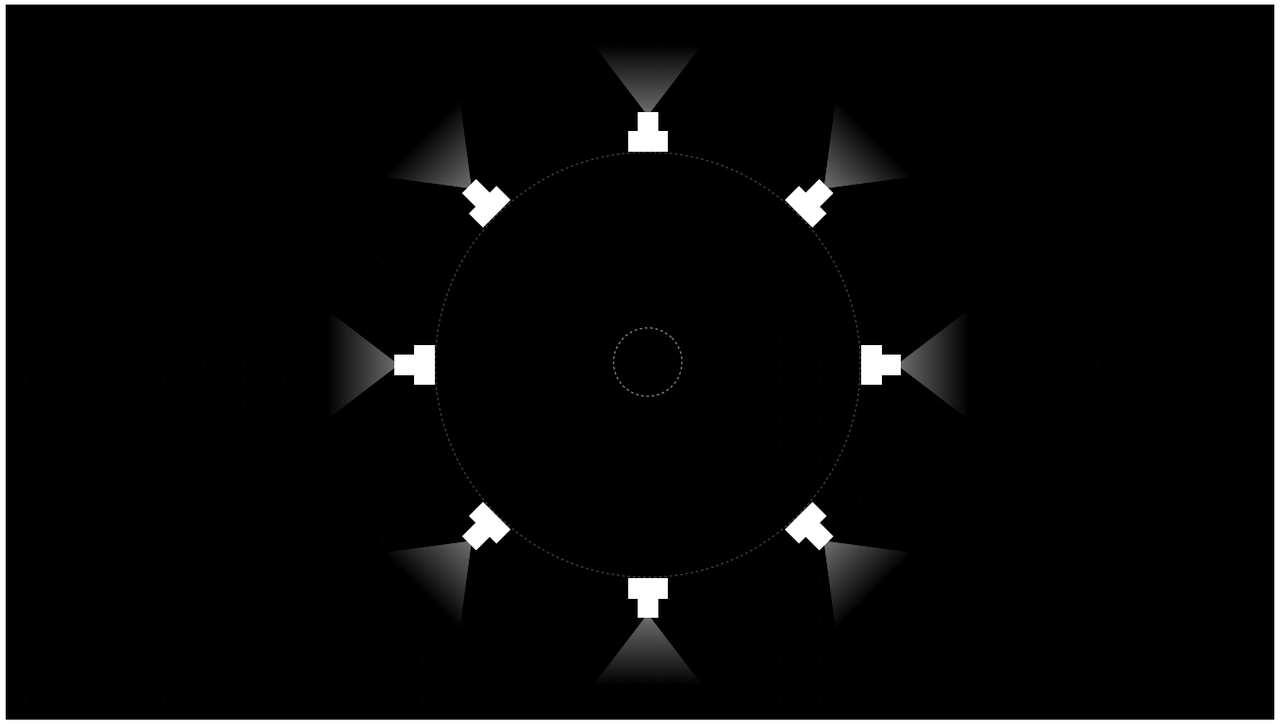
By simultaneously respecting free space constraints in the scene, and also regularizing the depth map to be smooth, it’s possible to extract a better looking depth map without artifacts.

Let's take a look at it in 3D, as you can see it looks much more believable.

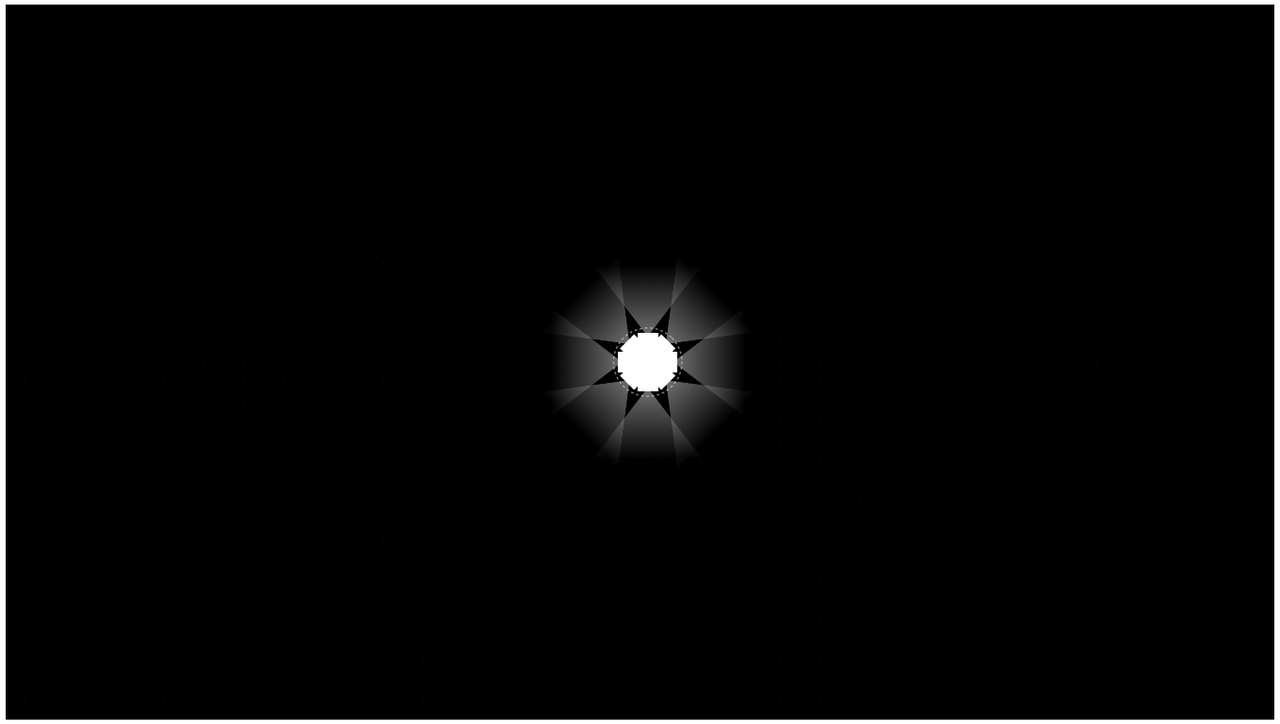

Input: Pictures
Sparse reconstruction
Dense depth
Stitching
Output: 3D Photo



- Now we can compute depth maps that are relatively artifact free, and we're done with the analysis of our images. The depth maps tell us almost everything there is to say about the scene, at least in terms of geometry. So, the question is how we can stitch them together into a panorama. put them together to create a 3D photo.
- However, standard panorama stitchers will not work as the images were captured from different locations. So they will not align, because of parallax.

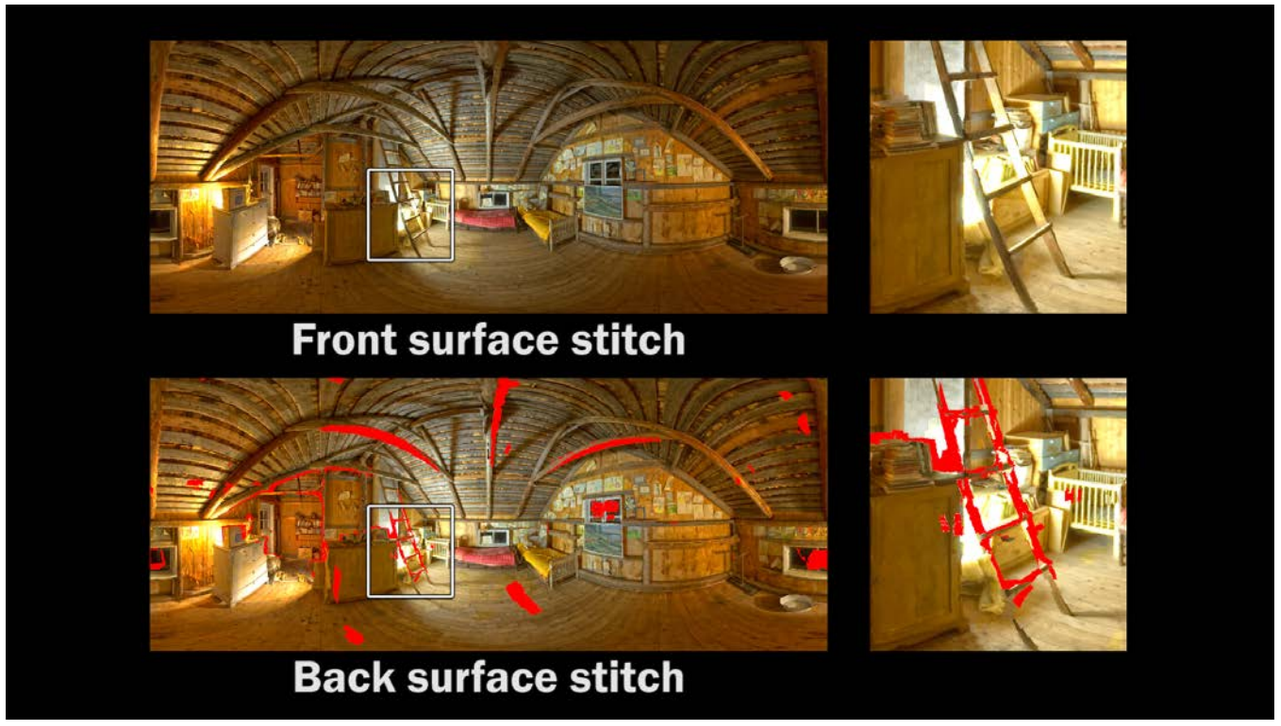


- But thanks to the depth maps, we can warp the images and **re-render** them as if they were all taken from a common central viewpoint.



- Let's take a look at how the images look in the central panorama.
- Take note of how well they are fitting together now.
- For each pixel in the panorama, we need to know from which input image we should fetch depth and color.
- We formulate this as a labeling problem, and use discrete optimization to produce a panorama with both depths and colors.

- Just producing a pano with depth is not enough. As you can see, this only represents the foreground objects in the scene and we cannot peek behind corners.




- We use a cool trick where we invert the depth-test to produce a back-surface stitch. Take a look at the paper if you're interested in the details.
- This second stitch looks very similar to the front-surface stitch, except it shows unique content that is not visible in the foreground layer.
  **<Click... details appear>**
- Focus your attention on the red highlights here. This is where the back surface stitch contains unique content.
- See how the ladder appears eroded away, and we see some reconstructed background that we can use in the 3D photo.

- Let's take a closer look at this. This is the foreground stitch. I will in a moment switch over to showing the back stitch. Watch what happens to the tree...
  **<Click... Back stitch appears.>**
- Now it is gone. This is the back background layer.
- We can fuse both of these layers together.
  **<Click... Connect front/back stitch appears.>**
- And now we have a representation where you can peek behind objects.

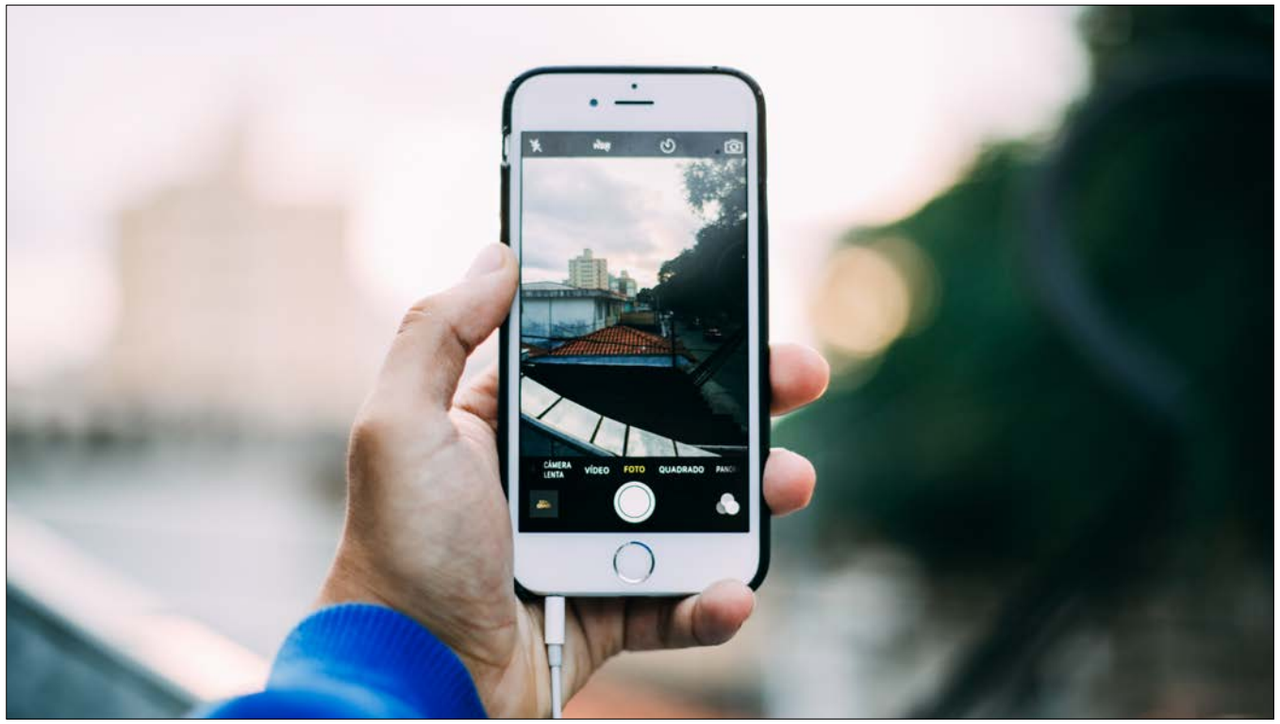
**<Click... Expanded back layer appears.>**

- To allow for even more camera motion, we extend the background layer and smoothly fill in the colors.
  **<Click... Parallax animation appears.>**
- This is all we need to display our scene in VR and enable free viewpoint changes.

- Let's now take a look at our results.
- I should be telling you that I captured a large variety of scenes to show the robustness of our approach and compare it with other approaches: Indoor scenes, outdoor scenes, thin structures, reflective objects and so on.
- However, I really just wanted to go back to Finland and capture my home town in 3D. You'll see the attic in my old summer cottage, the church in the town center and even my old high school.
- So lean back, and enjoy the beautiful views of Jakobstad.
- If you're interested in more details and results, please take a look at our supplemental material.

However, it takes a WHILE to process these 3D reconstructions. It takes up to 4 hours to process a scene!

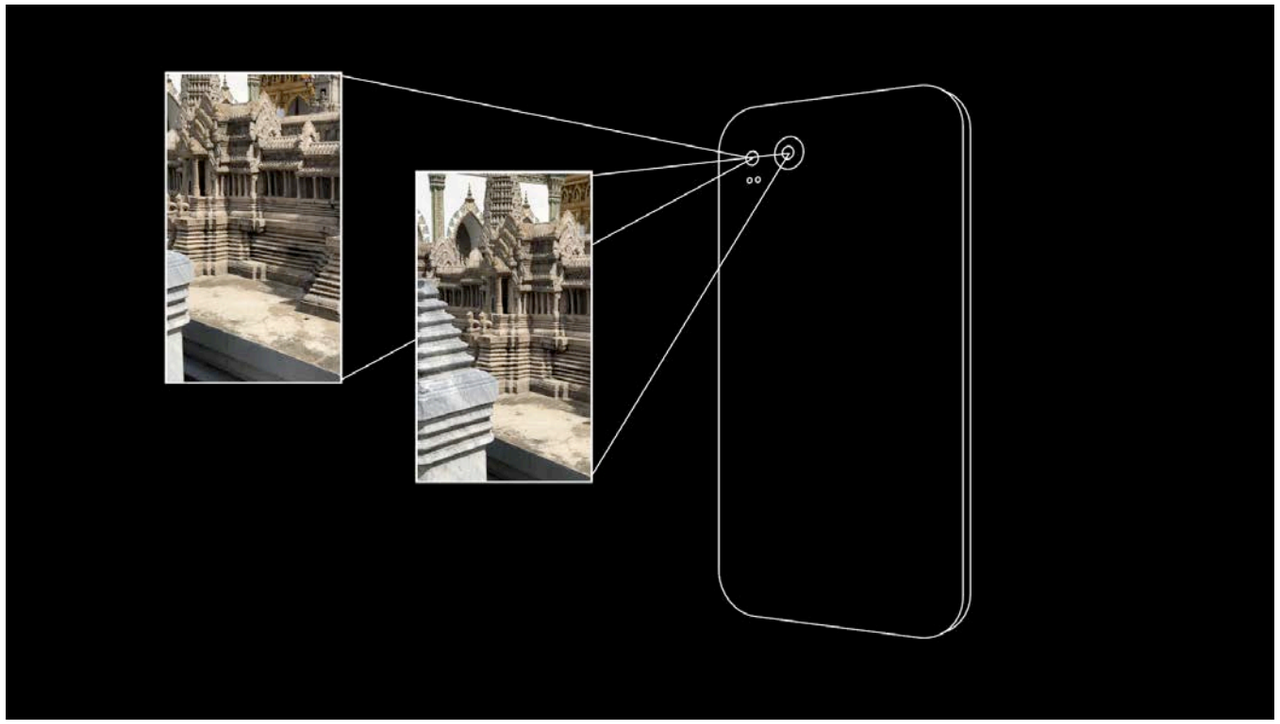
This is like the good old days of film photography: It's easy to take pictures, but you have to wait for the film to be developed before you see the result.



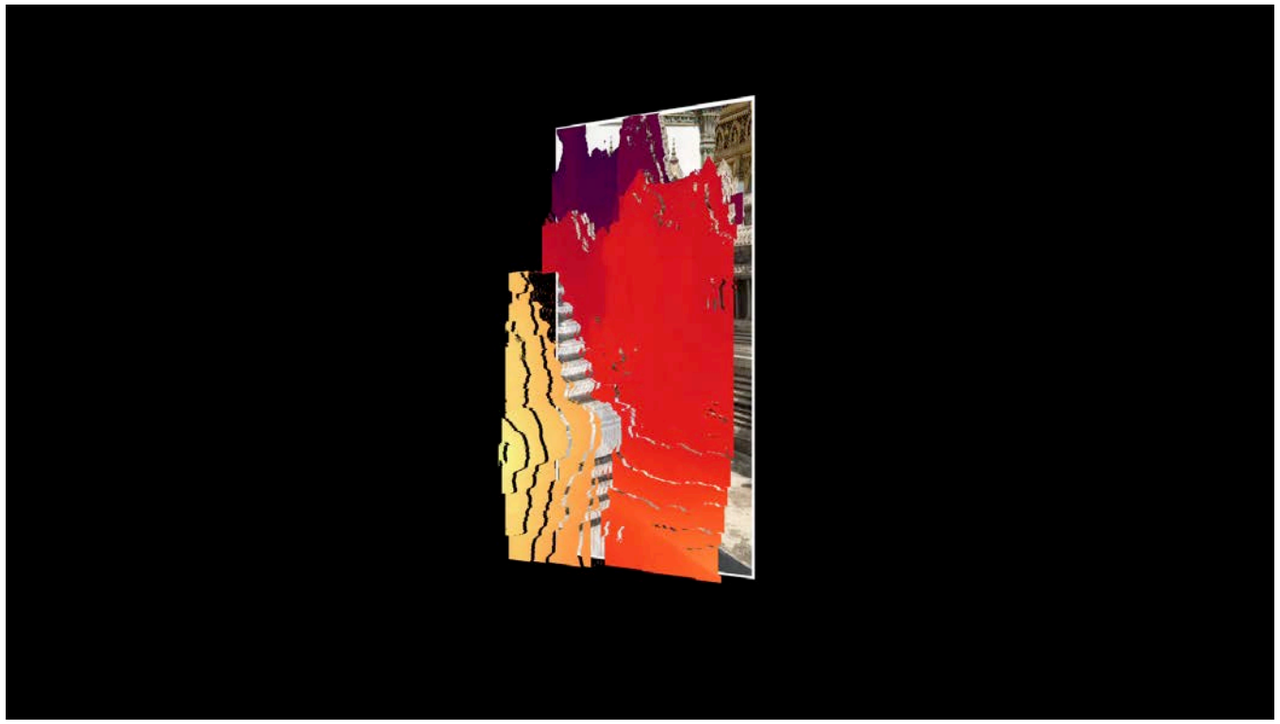

Capture becomes easier with instant feedback, much like taking a picture with a digital camera or a cell phone.

With this in mind, we designed a very fast approach to 3D reconstruction.

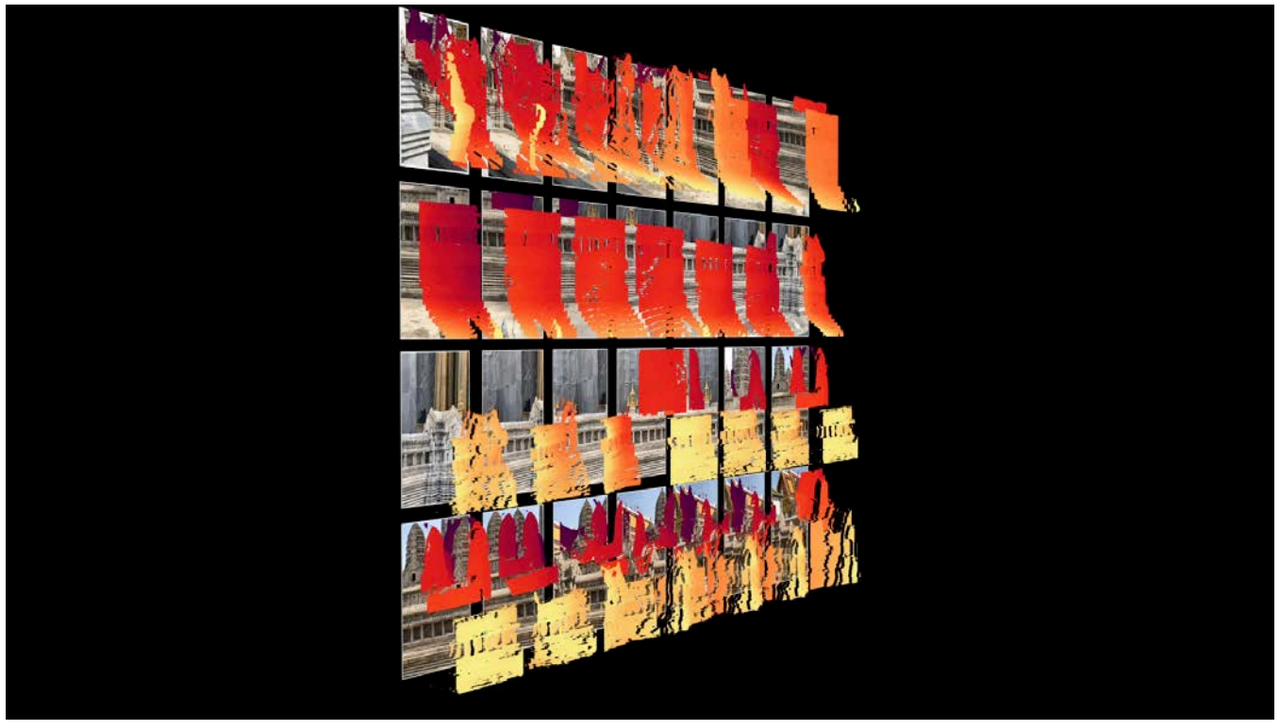
The first design decision was to use a dual camera cell phone which takes two images at every single location.



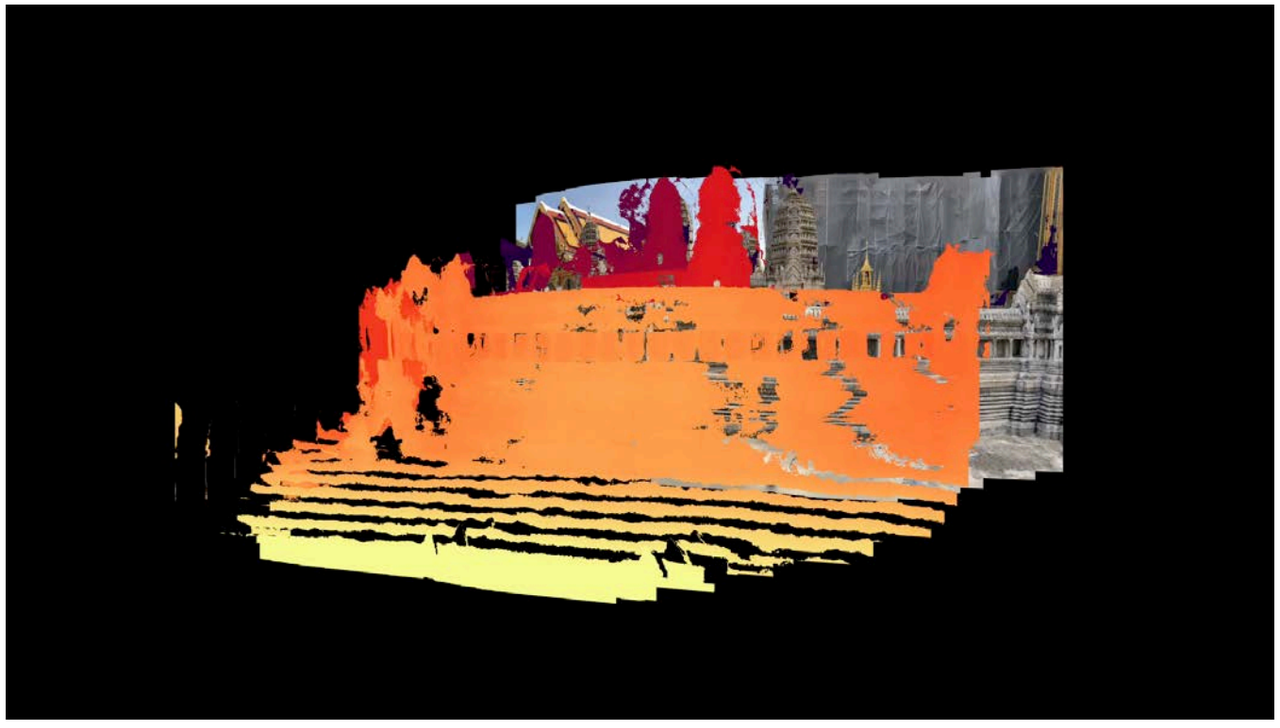
These cell phones also provide fast depth estimation algorithm, which quickly computes a depth map at each location.



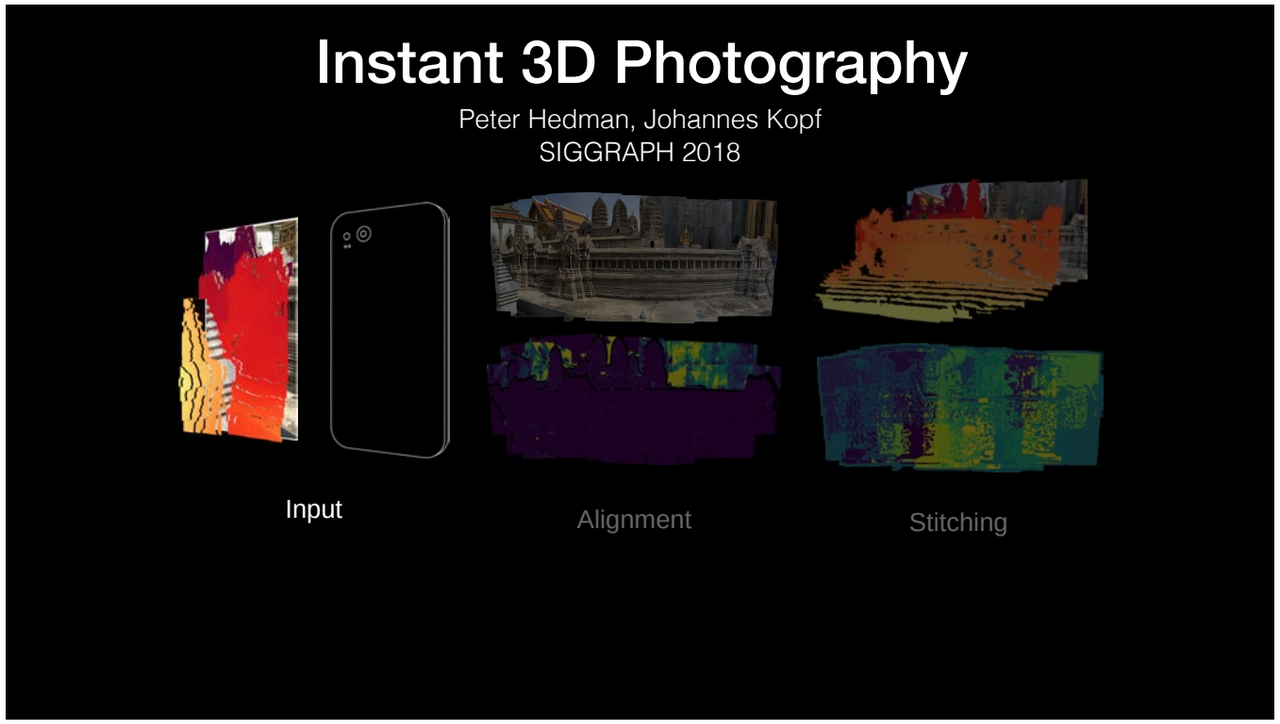
We developed an application that captures a burst of these color-and-depth photos.



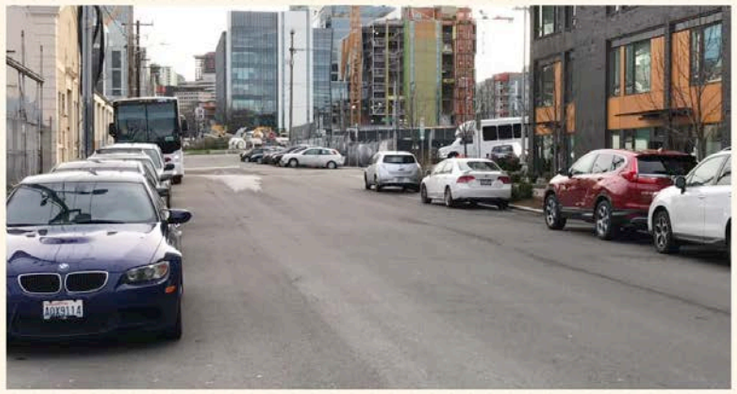
And the main technical contribution is a fast approach to align these photos in 3D, combining them into a color-and-depth panorama.



And then, with a meshing approach we talked about earlier, we turn this panorama into a 3D mesh that enables viewpoint extrapolation in VR.



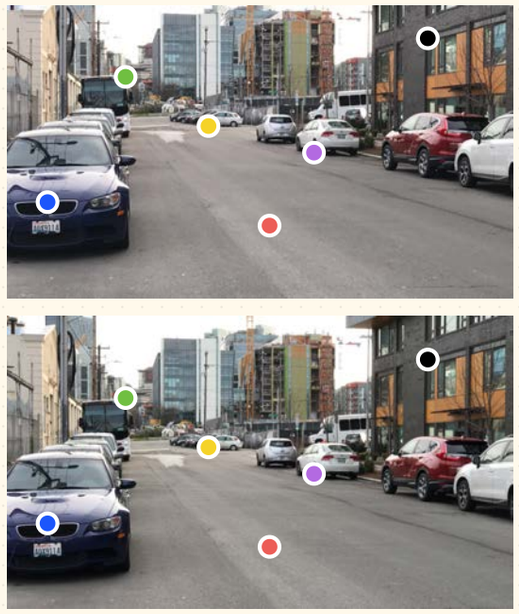

Let’s first discuss the characteristics of our input data.



This is the image pair you get from a dual-camera phone.

Since the distance between the two cameras --- the baseline --- is so small, the images are barely different.

This makes stereo matching fast, as you can keep the search region small.



Optical stabilization

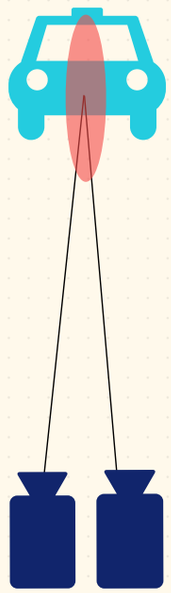
Auto-focus

Gravity

Temperature

Rolling shutter

Edge-aware filters



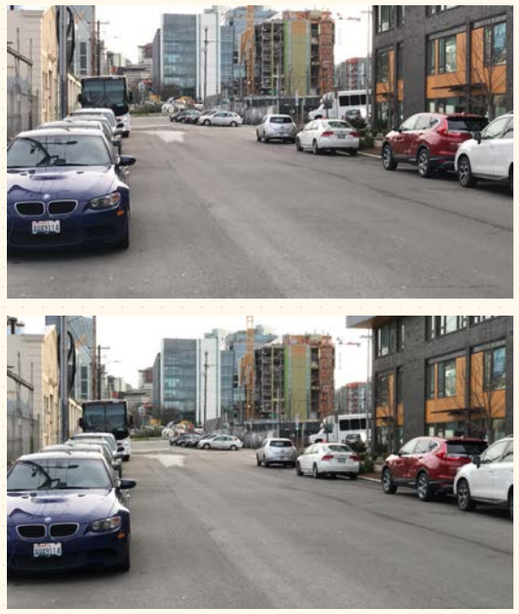
However, the triangulation angle is so narrow, which often makes the reconstructed depth unreliable.

And it gets worse: In practice the cameras move and rotate independently of each other.

This happens because of otherwise desirable features in cell-phone cameras:

- Auto-focus
- Optical stabilization
- Rolling shutter

In fact, even gravity plays a role here.



iPhone 7 Plus

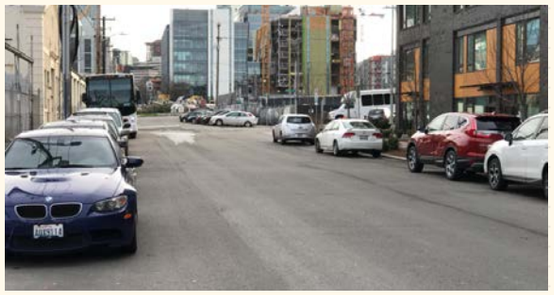
This is why most algorithms use aggressive edge-aware filtering

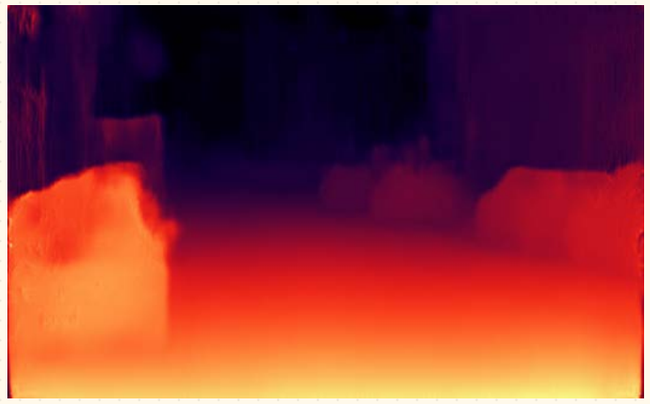
resulting in smooth depth maps that also respect object boundaries.

Unfortunately, this introduces a large low-frequency error on the estimated depth values.

We found that other small-baseline depth estimation methods also suffer from similar degradations

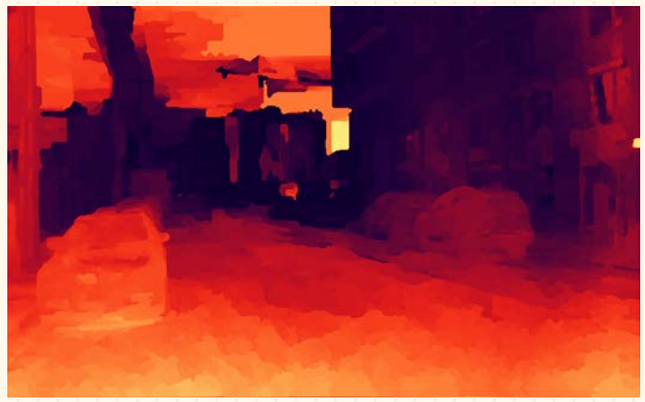
With the help of aggressive edge-aware filtering





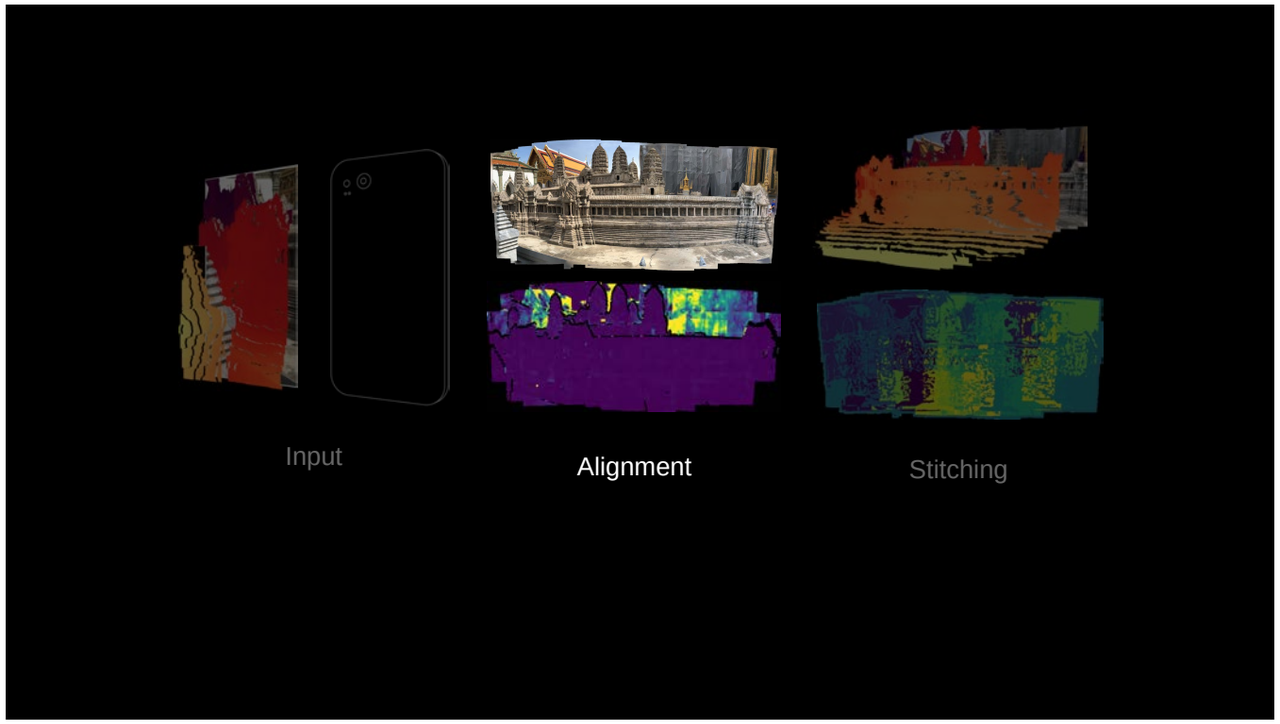
Single view depth [Godard2017]

For example: single-view depth estimation using convolutional neural networks



Small motion clip [Ha2016]

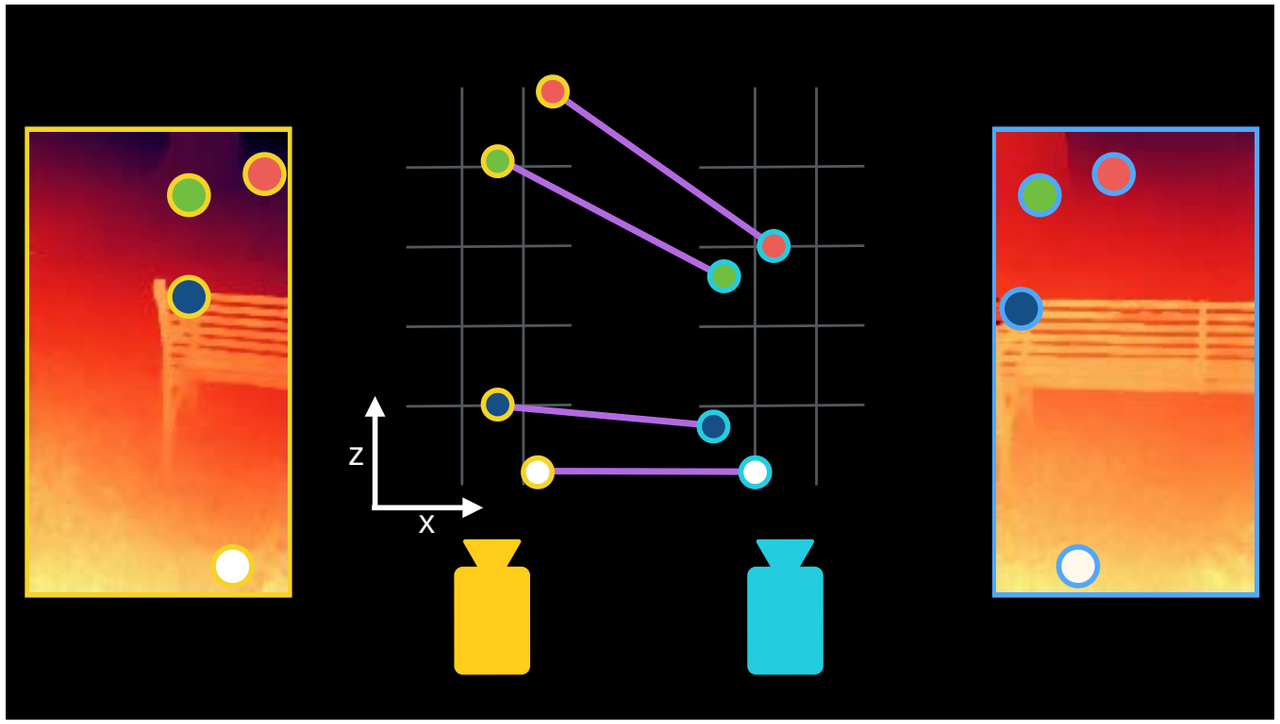
Or: Depth estimation from a  short video clip




So how can we get rid of these deformations?

Remember that we have an application which captures a burst of these color-and-depth images.

This provides redundant information, and we can establish feature point correspondences between the images to reason about the 3D structure in the scene.

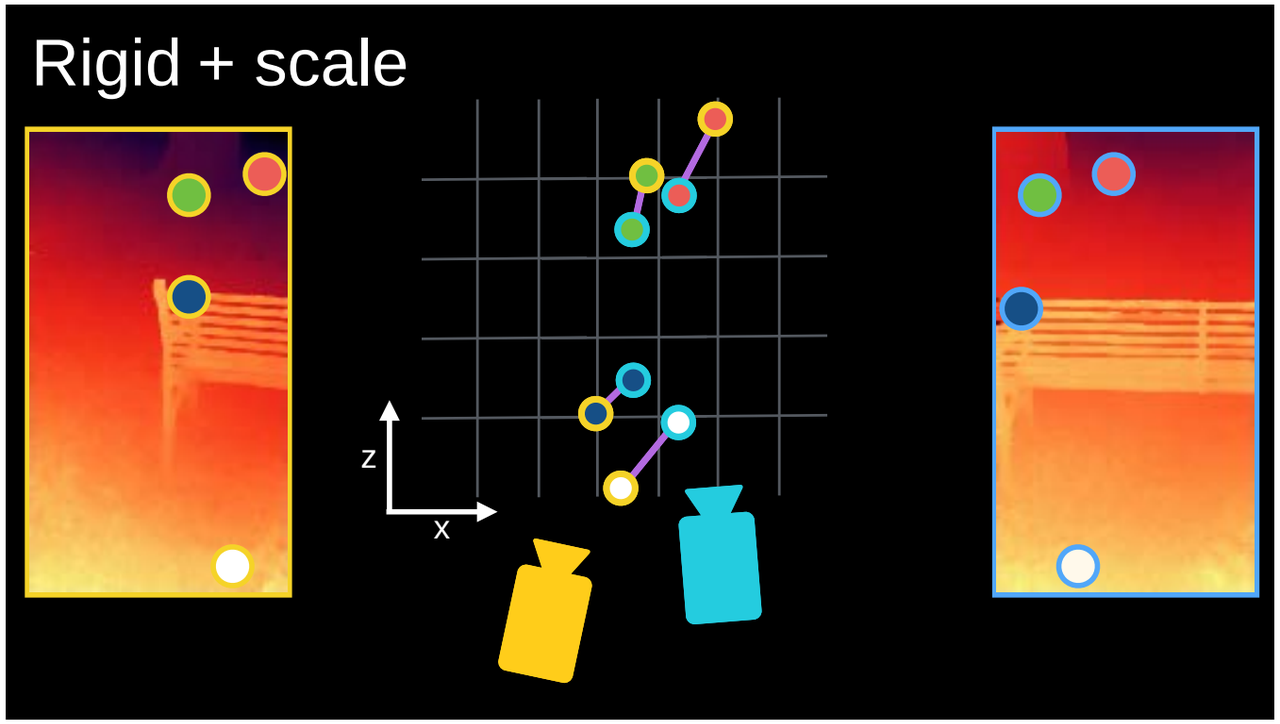



Let’s take a look at two images and a few matching feature points.

Since we know the camera intrinsics and have depth maps, we can project these points out into 3D, forming two point clouds.

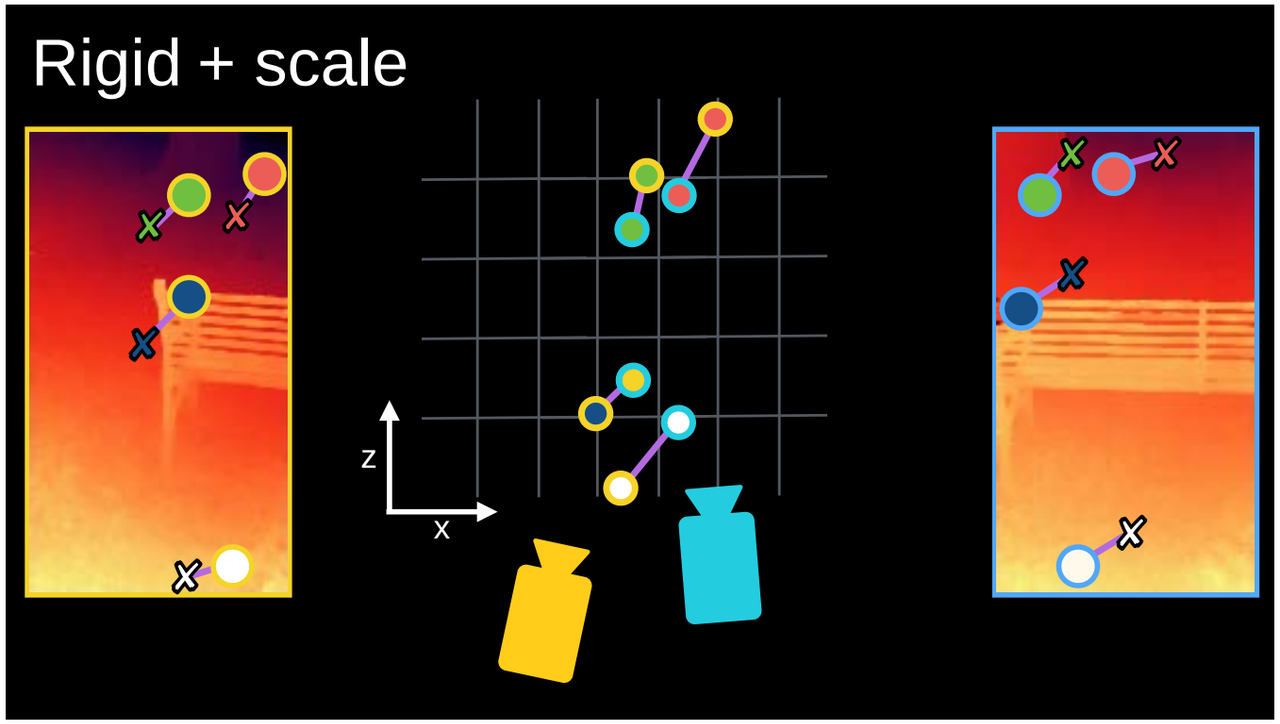
Already here we can see that there’s a scale difference between the two depth maps.

We can now try to better align these two images, optimising for a rigid transformation that minimises the 3D distance between matching feature points.

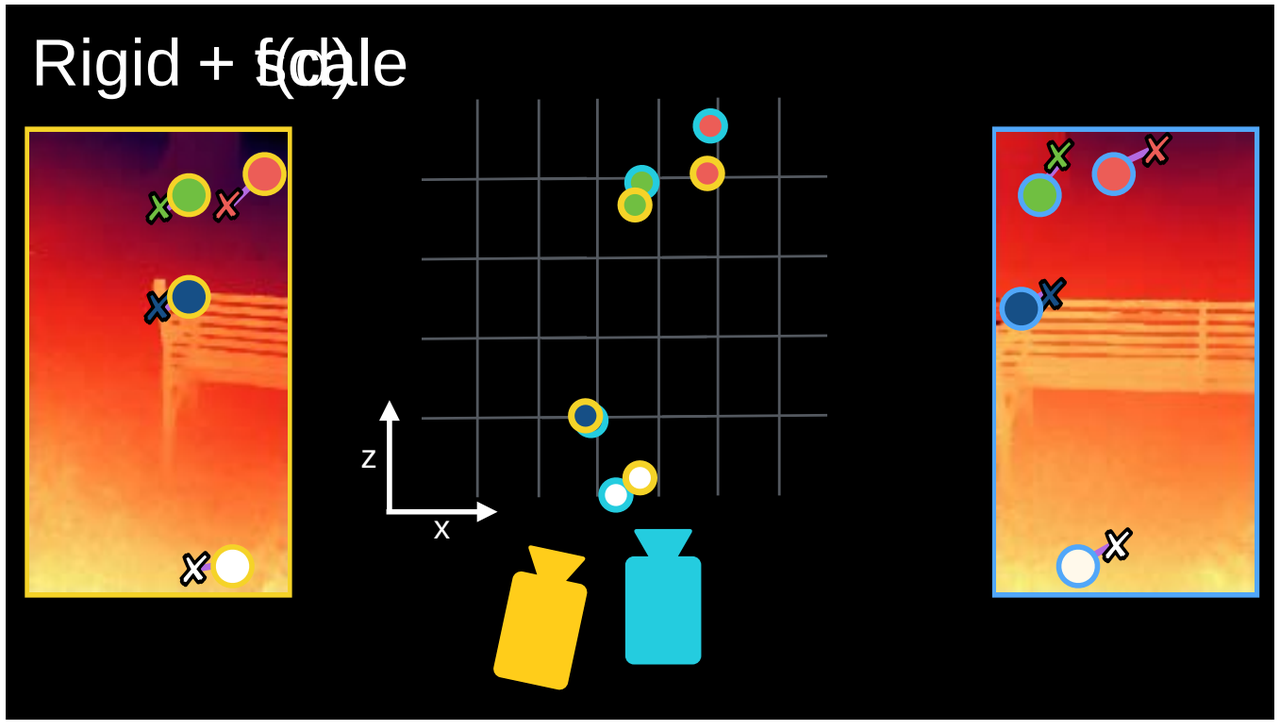



But this is not quite enough, as there is a scale difference between the images.

If we also optimize for scale, we run into trouble —we can easily reduce the error to zero by shrinking the scene!

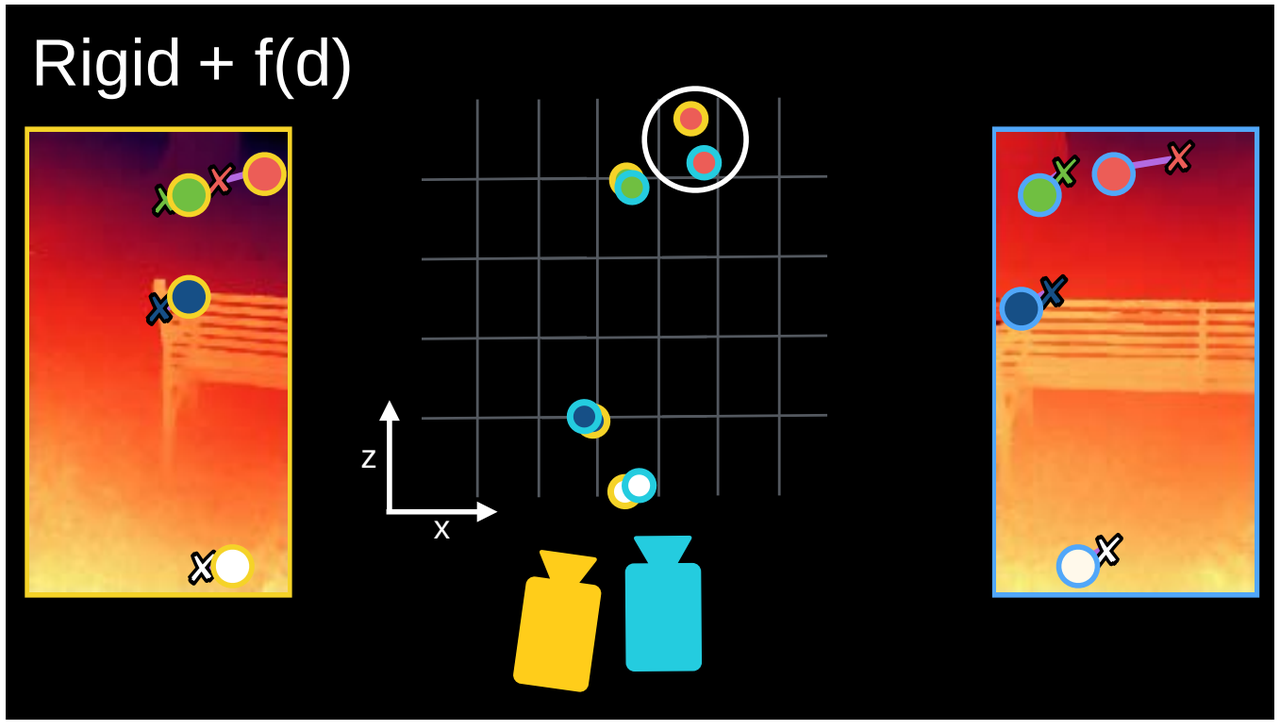



Instead, we found it better to measure the 2D distance between feature points. We use the depths to project each corresponding point into the other image. This allows us to optimize for scale without having the trivial solution of shrinking the scene.




Unfortunately, the deformations on depth cannot be explained by just a per-image scale factor.

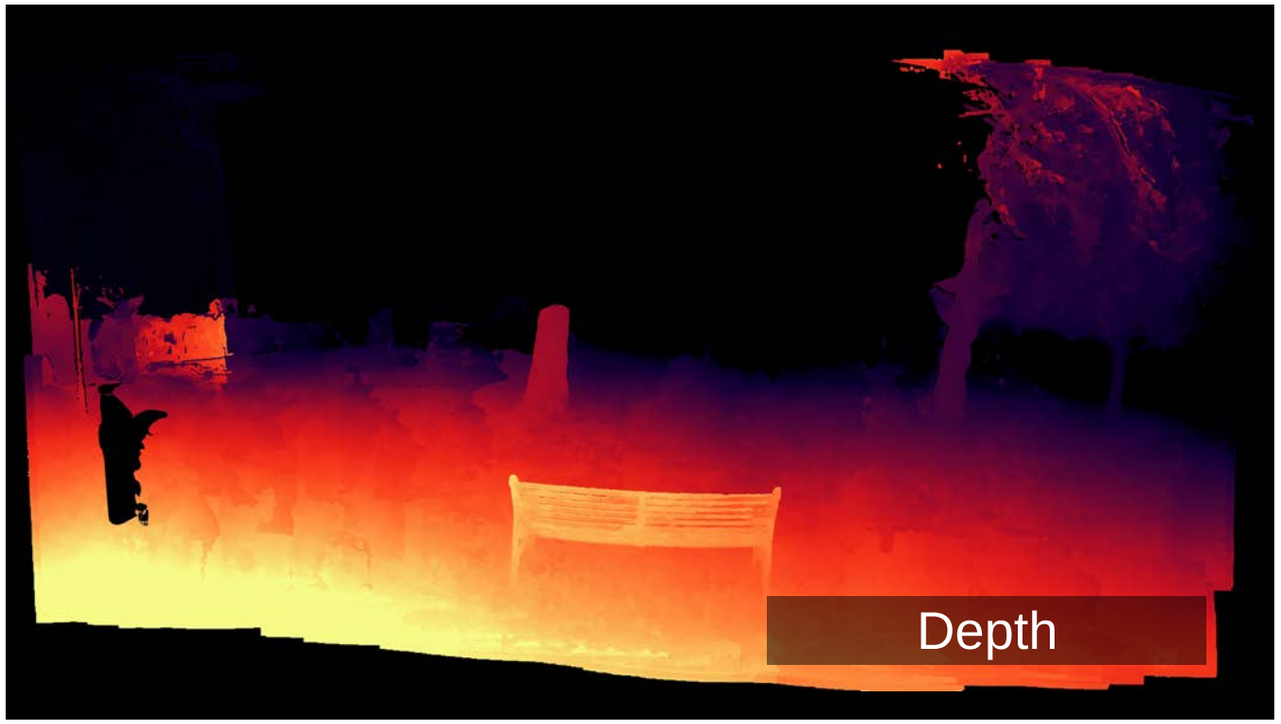
We experimented with more general depth correction functions, and found that an affine transformation on disparities worked best.




But there is still quite a bit of residual error.

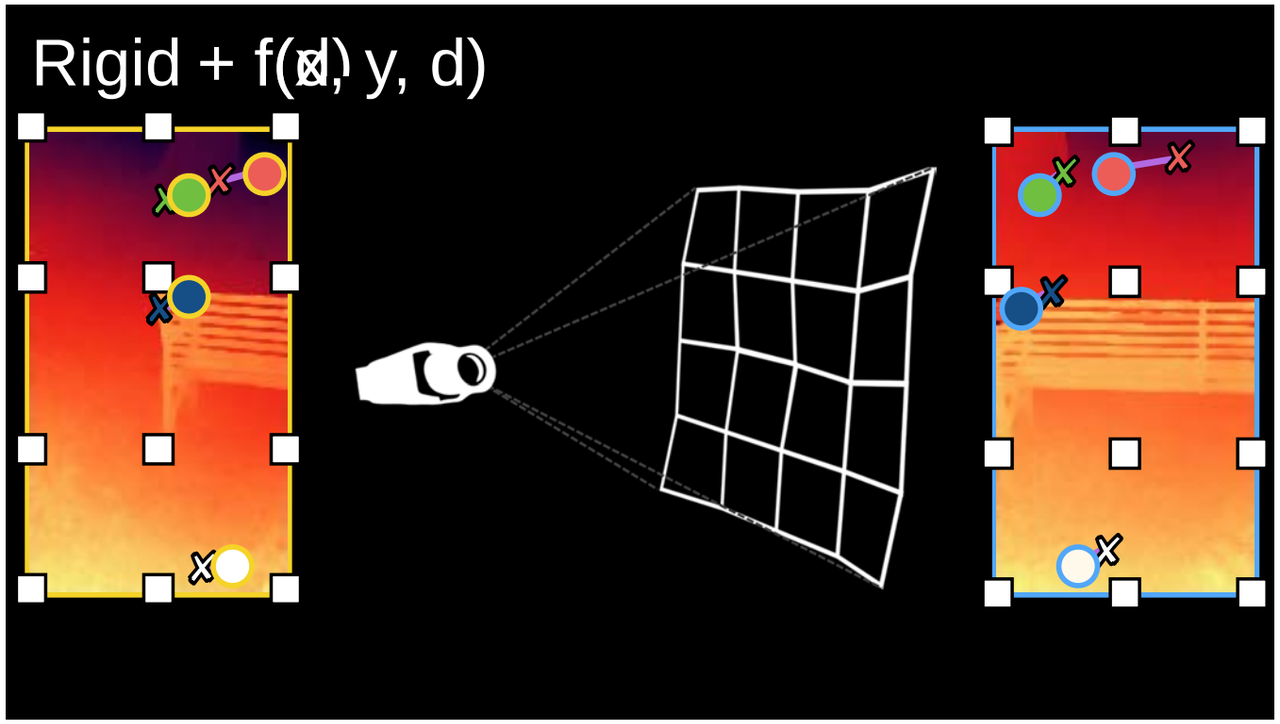
… But maybe this isn’t too big of a deal. What if we simply used this alignment to stitch a panorama?


The color panorama looks great.




But unfortunately the depth channel isn’t doing quite as well.

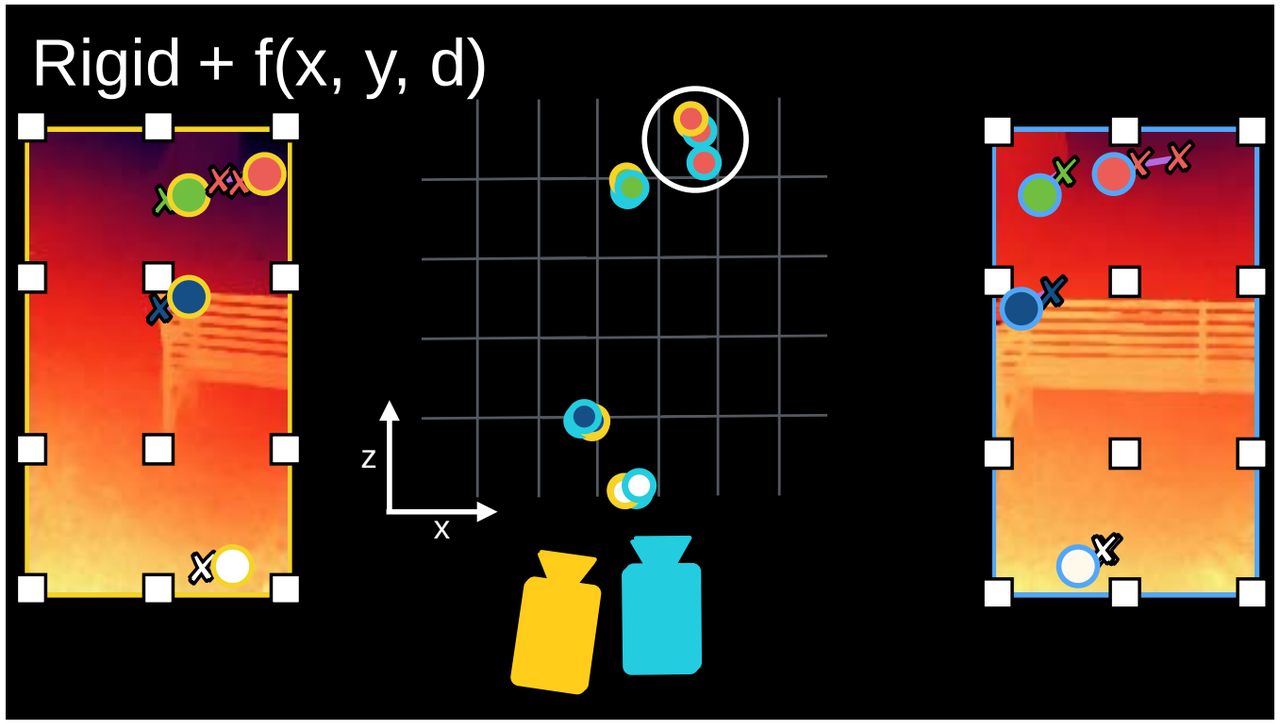
There are a lot of misalignments on the ground, and there are big black regions, where the only feasible solution was to push these parts of the scene out to infinity.




To solve this, we use a depth correction function that varies across each image. We do not want to introduce new structures into the depth maps, so we want it to be smooth.

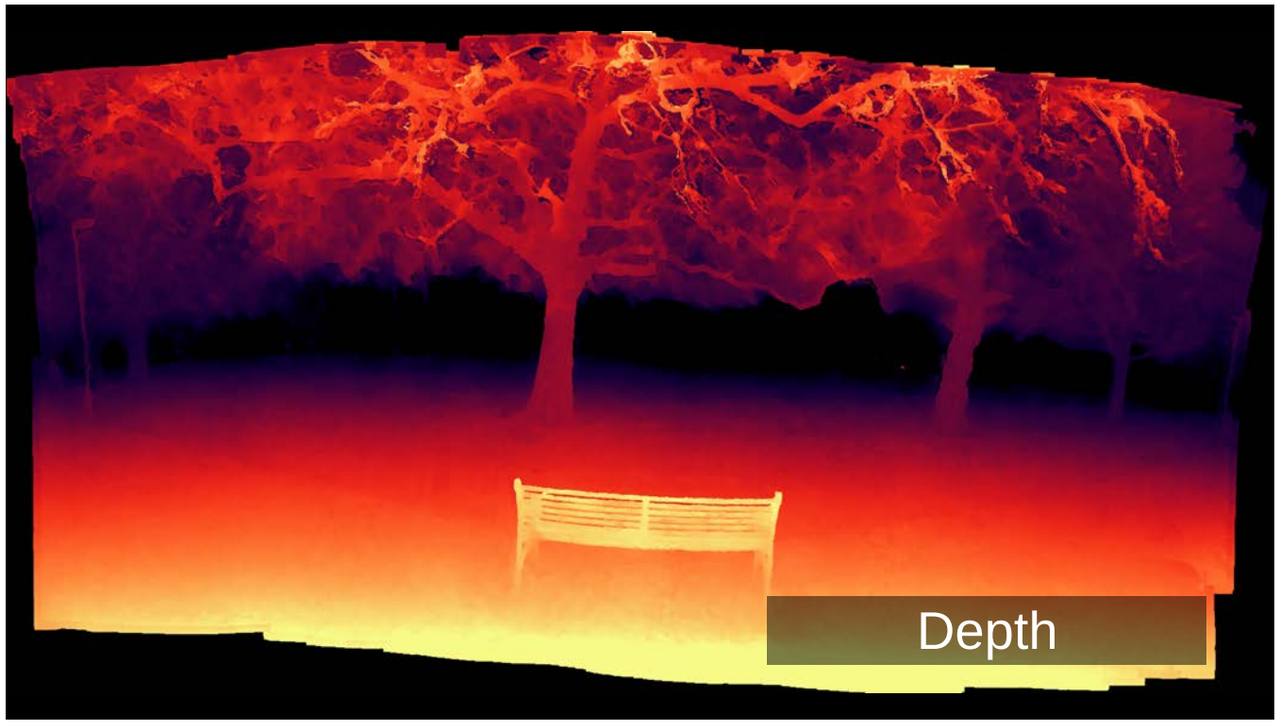
We optimize for coefficients at a regular grid of locations across the image, and use bilinear interpolation to form this smoothly varying depth correction function.

With this approach, we can isolate the feature points in the top right corner of each image, and use a slightly different depth correction there.



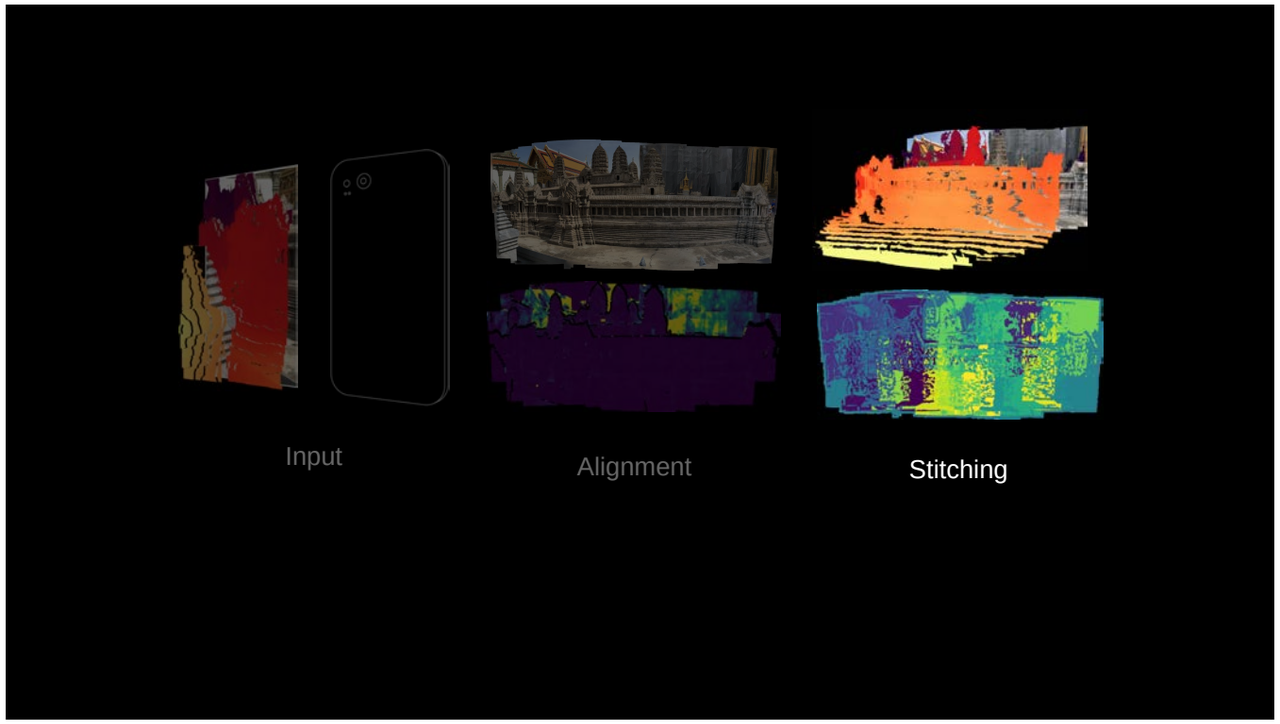

This brings everything into much better alignment.


And if we take a look at the resulting panoramas, the colors still look good.




But there’s a huge difference in the depth channel.

The ground is much smoother, and there are no big black regions anymore.



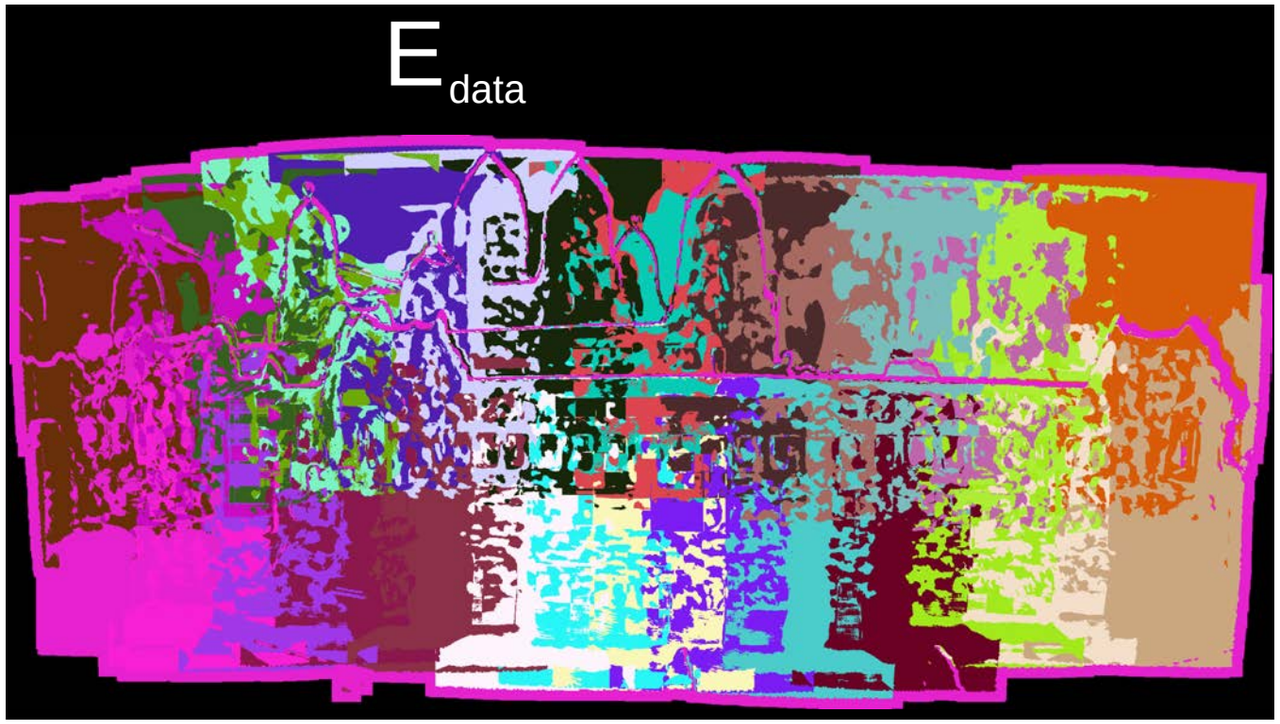

This allows us to quickly align the images in 3D. But how can we speed up stitching?


Earlier in this talk we used a seam-hiding stitching approach, which optimizes for a per-pixel data cost and a seam-hiding smoothness cost.

The data cost tells us how well each image works at each pixel.

The smoothness cost reduces the number of image transitions in the panorama, which reduces the opportunities to make mistake. It also tries to hide these transitions in regions where they are hard to spot.

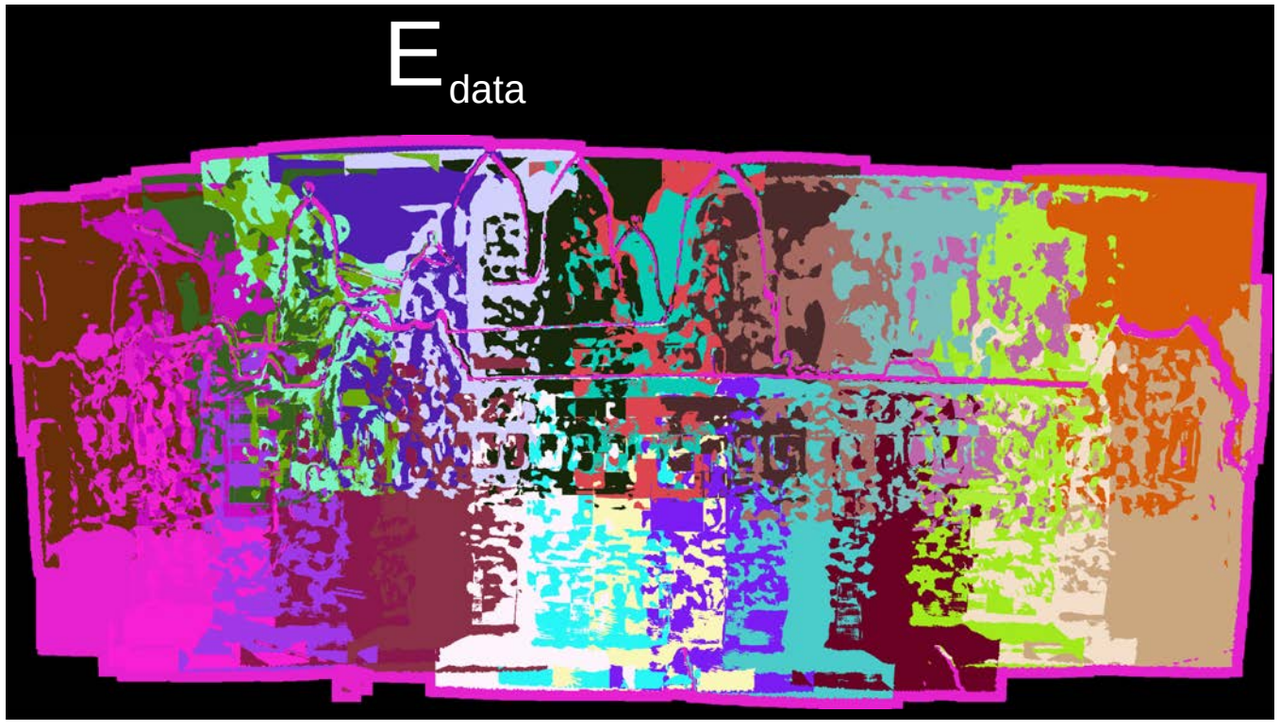
Unfortunately, it is very expensive to optimize for this type of smoothness —using algorithms such as alpha expansion.



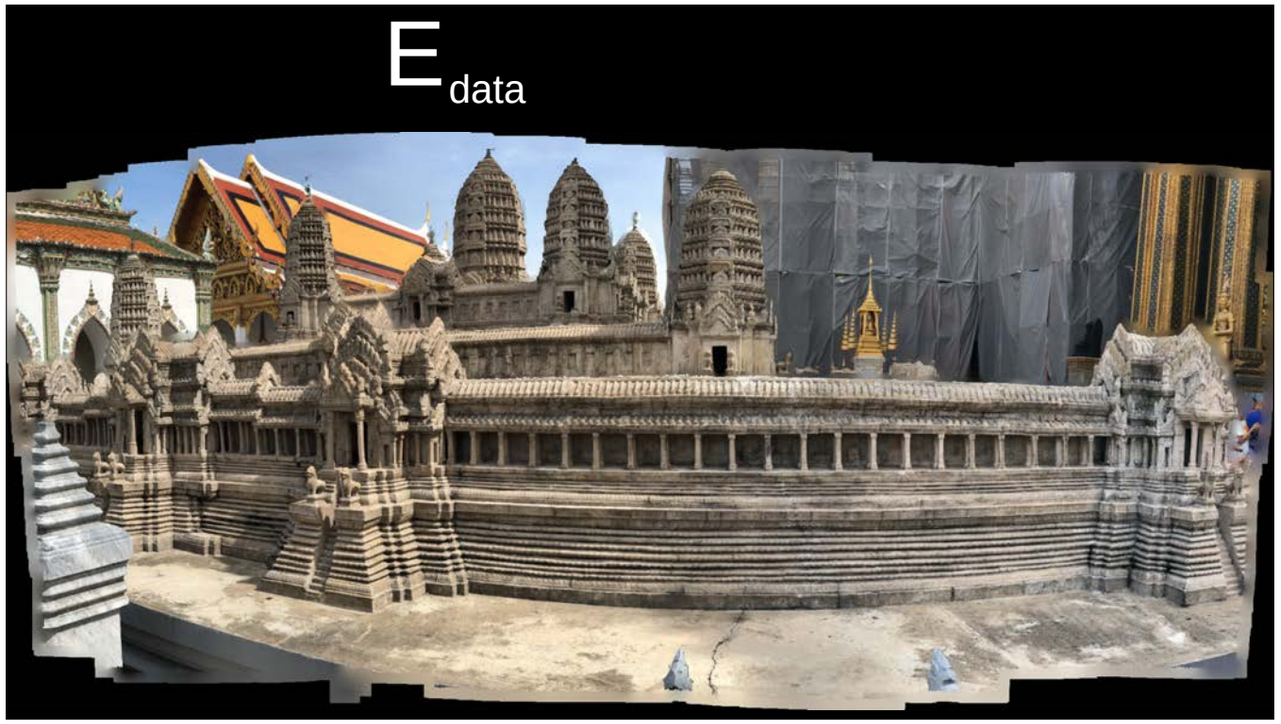

It would be much faster if we could only consider the per-pixel data cost. Which can be quickly obtained for each pixel independently with parallel computation.

However, as you can see, this introduces a lot more image transitions —providing ample opportunity to make mistakes and introduce visible seams.

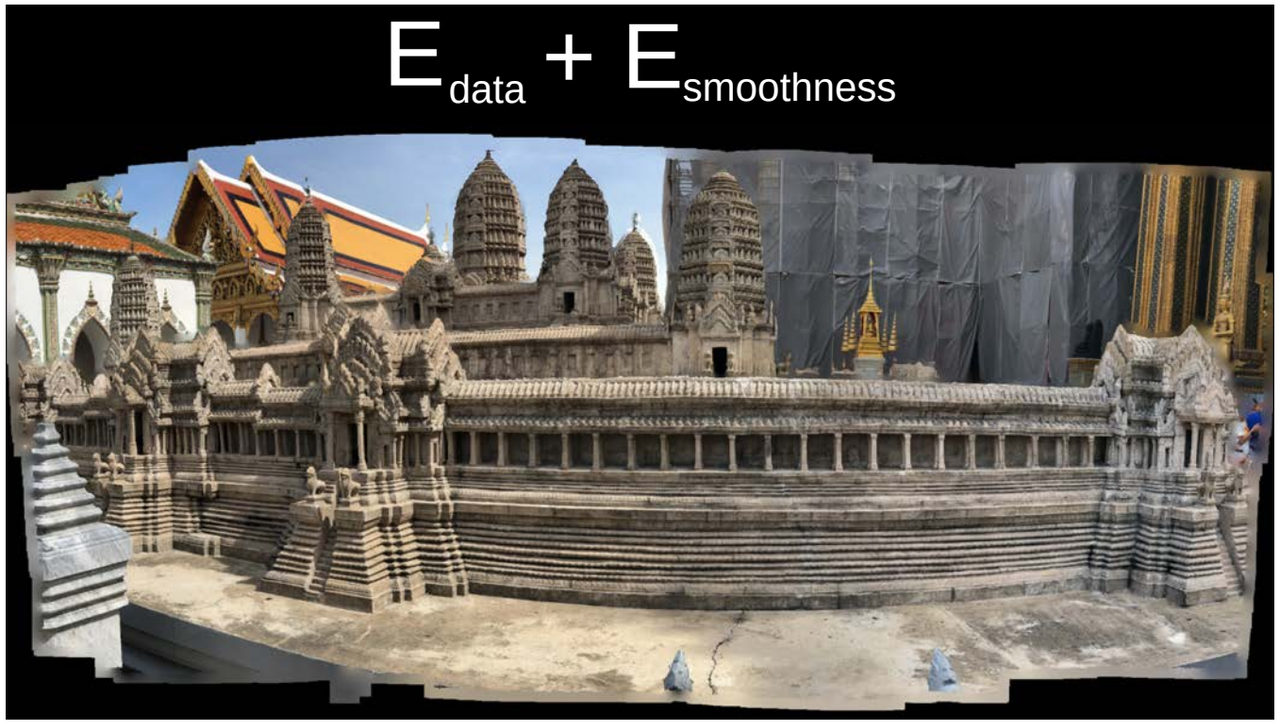
Thankfully our alignment is very precise!



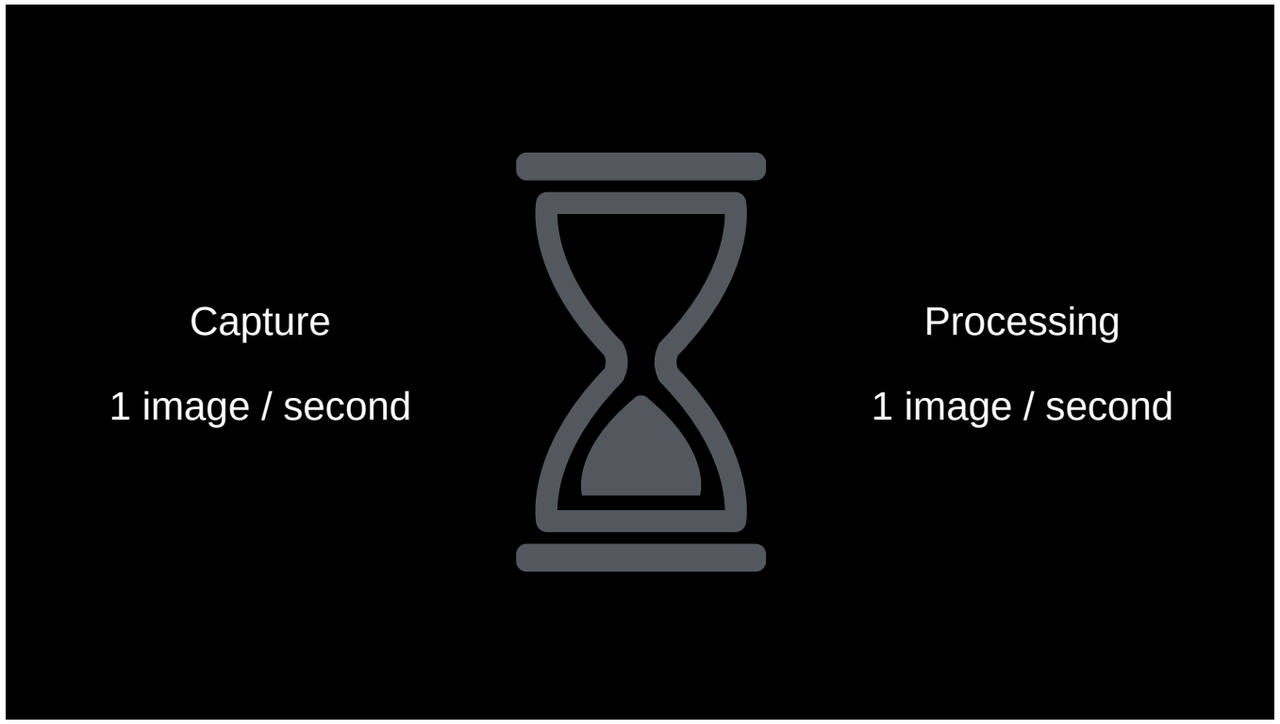

So if we optimize only the data cost




We still get an output panorama that looks great




And is very hard to distinguish from the result which uses slow seam-hiding stitching.

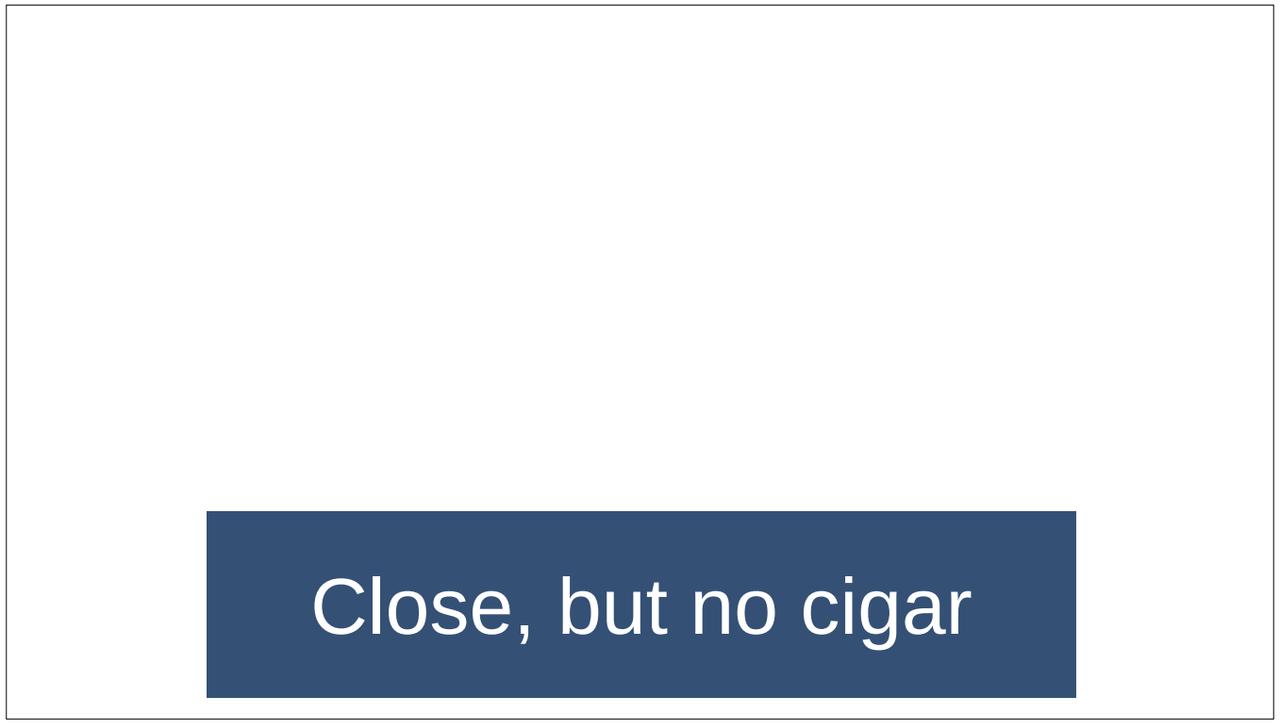



Putting everything together we obtain a very fast pipeline.

The capture application captures roughly one image per second, and the processing takes about one second per image!

Full disclosure, for now, we've only run processing on a desktop PC. But this was using unoptimized CPU code, and most bottlenecks can trivially be ported to the GPU.

With these optimizations in mind, and also by interleaving processing with capture, I am confident that you could produce a result almost immediately after capture.



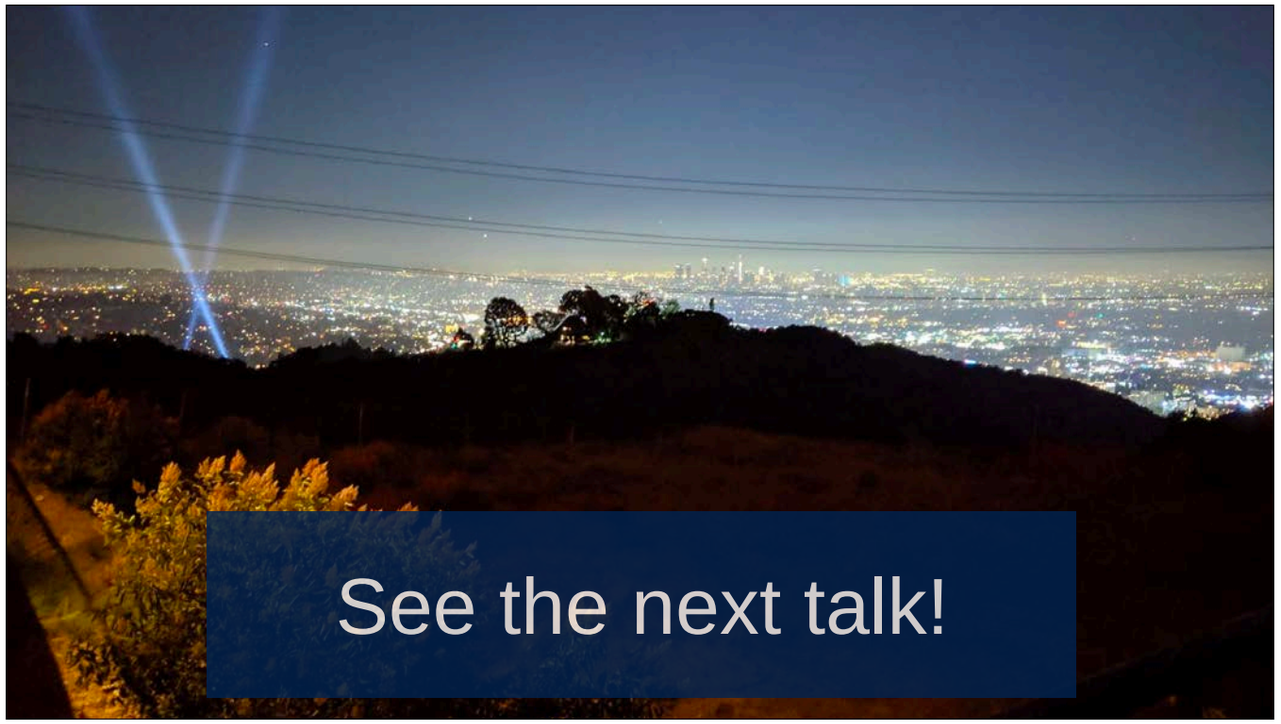

- Overall, the easy capture process, and the fast processing pipeline changed the way I captured 3D photos. I became more opportunistic, and started experimenting more — capturing scenes whenever I felt like it.
- Most of the scenes you'll see here were captured during my vacations. You'll see Bangkok, and some snowy scenes that I captured in Finland over Christmas.

But let's take a step back. In this talk I've talked about technologies that focus on casual capture — enabling anyone to easily capture 3D photos for VR. However, when it comes to quality I can only say: "Close, but no cigar"

- The 3D photo representation discussed in this talk does not support moving highlights. Instead it bakes them into the texture map, resulting in a “painted on” appearance.


These systems also share the limitations of most multi-view stereo approaches. We're unable to reconstruct partially transparent objects such as glass.

Our scene representation also contains only two-layers, so we wouldn't to be able to represent the multiple layers of glass you can see here.




- Finally, we’re unable to represent volumetric effects, such as the beautiful Hollywood lights shining through the fog.

For to reach a higher level of realism and quality, we need to bring out the big guns and use a custom built capture rig to obtain full light fields.

Stick around for the next talk where Ryan Overbeck describes exactly this!