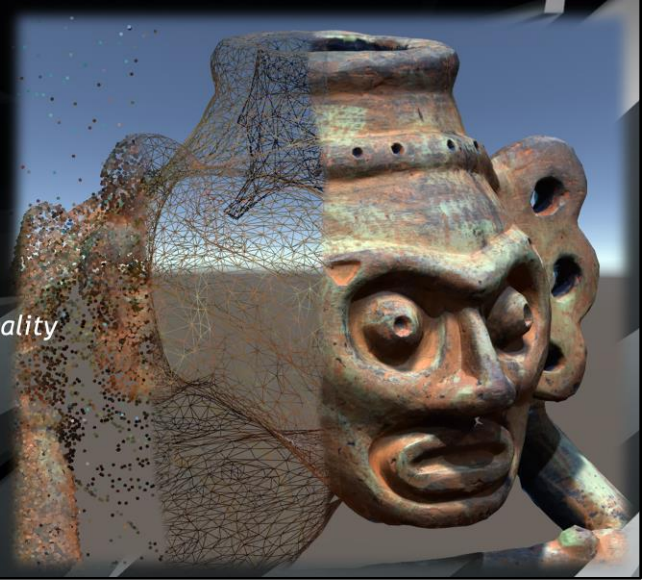
Photogrammetry for Games

Art, Technology and Pipeline Integration for Amazing Worlds

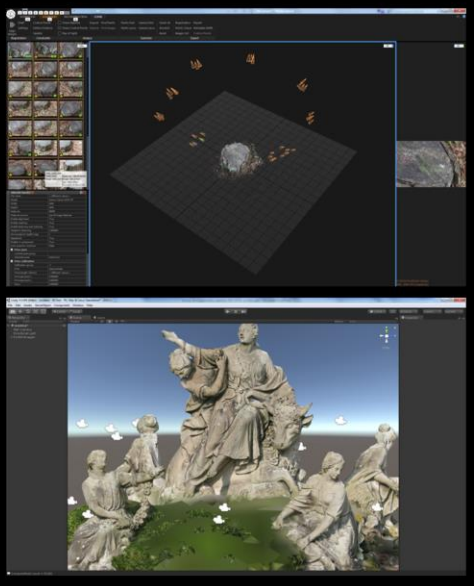
1 March, 2017

Lars M. Bishop *Senior Engineer, NVIDIA*
Chris Cowan *Artist, NVIDIA*
Michal Jančošek *Managing Partner, Capturing Reality*



Where we're going…

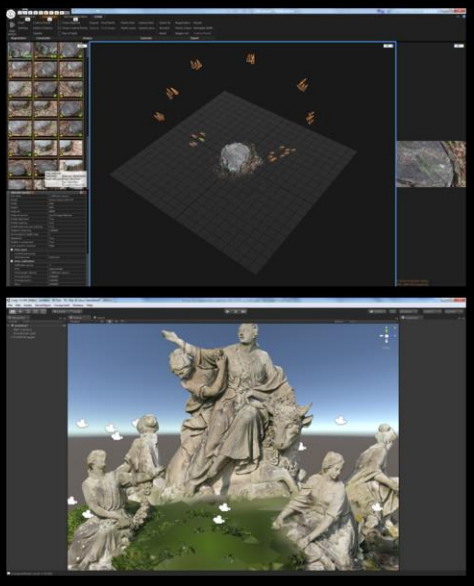
- Defining Photogrammetry (PG) as we'll discuss it
  - The challenges
- An experienced artist's view
  - Hardware, Tools, Pipeline
- Reality Capture
  - New features, licensing, pipeline integration
- Scripting, SDKs and integrations
- Near-term future pipeline options



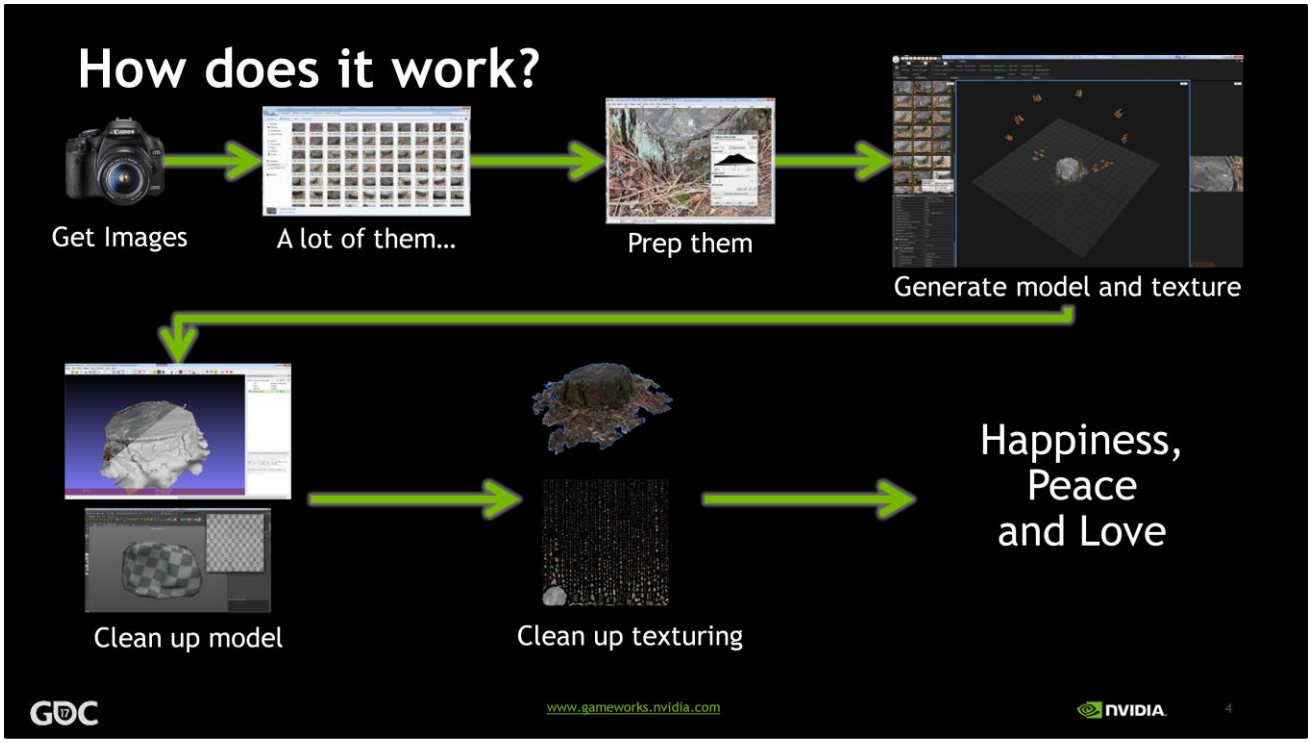
Good afternoon – my name is Lars Bishop, and I'm an engineer with NVIDIA's Developer Technologies group. This afternoon, I'm joined by two of my colleagues, Chris Cowan from NVIDIA and Meekal Janshoshek from Capturing Reality, and we will be discussing a range of topics of interest to Photogrammetry in games. Specifically, Chris will start with an artist's view of applying photogrammetry to a games pipeline, and some guidelines that have worked for him in practice. Then Meekal will focus on reconstruction software itself, showing how his company's tool, Reality Capture, can generate detailed models from photos.

Where we're going…

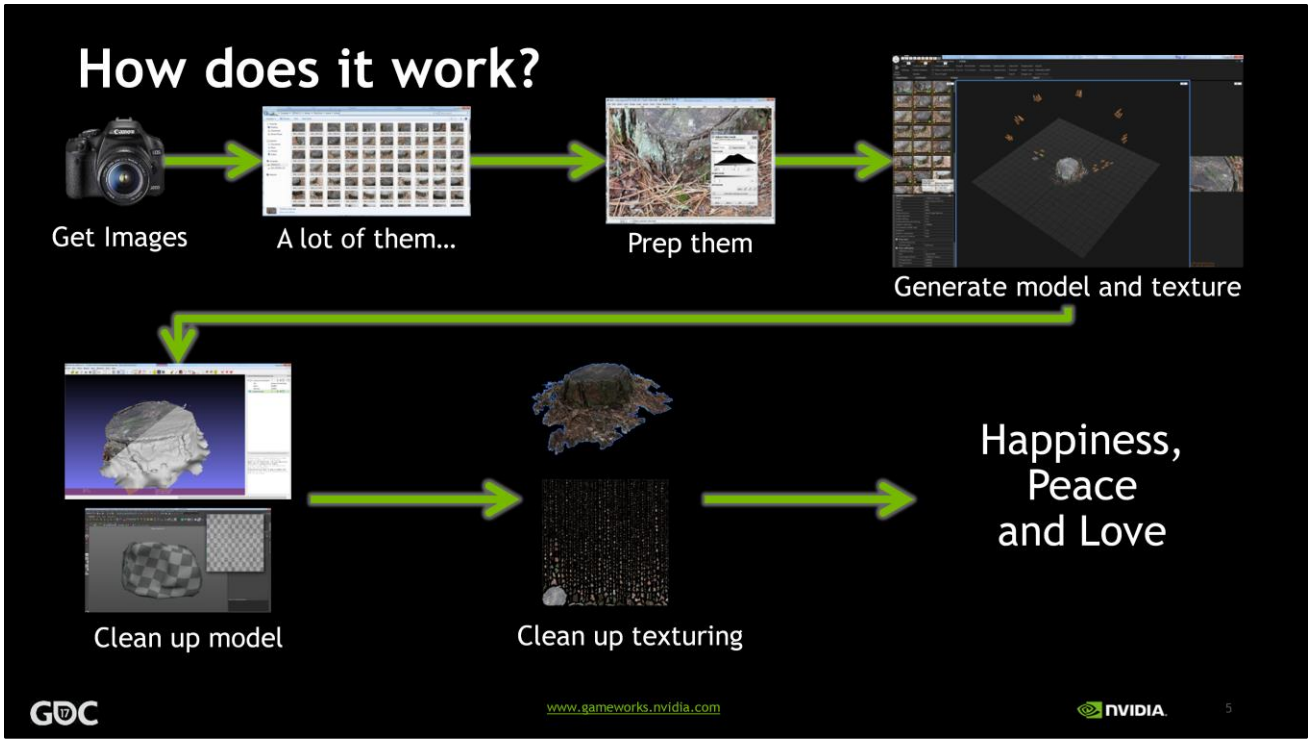
- Defining Photogrammetry (PG) as we'll discuss it
  - The challenges
- An experienced artist's view
  - Hardware, Tools, Pipeline
- Reality Capture
  - New features, licensing, pipeline integration
- Scripting, SDKs and integrations
- Near-term future pipeline options



He'll also provide some tips on using reconstruction software more effectively, based on a better understanding of how each step actually works in the tool. He'll close with how Capturing Reality is working to make photogrammetry easier to integrate into game pipelines. Then, I'll be coming back to discuss a programmer's view of the pipeline and how to choose integration methods to connect the various tools. I'll finish up by showing examples of pipeline integration, integrating Reality Capture directly into the Unity game engine's editor in a few different ways. So let's get started by defining a few things…



For many of you, this part is extreme review, but it will set a framework for what we consider “the pipeline”. Basically, we start with taking photos; a whole lot of them, most likely. Anywhere from a few dozen to thousands upon thousands depending on the object being reconstructed, and the detail level we want in the end. Those photos generally need a little cleanup or adjustment.



Then we feed them into a photogrammetry tool, which matches features in the images, figures out where the cameras were and what they were looking at, and generates a very high density geometric model. It also generates either per-vertex colors or an even higher density set of textures. Now, for non-gaming uses, that may be the end of the pipeline, but in games we're picky about the models and textures, so we'll generally end up decimating the model, cleaning up the mesh and the textures, and then passing the asset off to the team for use in the game. Each of these steps will get some more attention several times in this session.

Who uses it?

- Museums / Archaeologists
- Cartographers
- Intelligence Agencies
- Film / Video Production
- Our focus: Game Developers!

Studio 727 http://www.727.sk/

A key thing to consider when approaching any software tool is to know who forms the CURRENT main market for that tool: they naturally drive the feature sets and expectations for that tool. We have to remember as game developers that we're relative newcomers to a field that, in all honesty, isn't all THAT new. I actually did some work with stereo reconstruction in grad school in the mid 1990s and even then it was well established. The classic drivers of this market are remote sensing for cartography and intelligence analysis; stereo photography was used in world war two as an early form of 3D reconstruction.

Who uses it?

- Museums / Archaeologists
- Cartographers
- Intelligence Agencies
- Film / Video Production
- Our focus: Game Developers!

Capturing Reality

Studio 727 http://www.727.sk/

Ten24 http://ten24.info/

Much more recently, film studios have used it for scene reconstruction for about two decades. <CLICK> Also, museums and archaeologists use it to non-invasively recreate and catalog fragile or immovable treasures for research and exhibition purposes. Most of these results are focused on dense geometric quality, rather than real-time visual results. Obviously, we're focusing TODAY on the NEWcomer, gaming; so we're still going to have to pre- and post-process the data.

Why now?

- **Demand for realism in games:**
  - More detail, realism, organic look
  - On schedule…
- **Available GPU compute power:**
  - Requires lots of memory (images)
  - Requires tons of computation
  - Massively parallel
  - Reality Capture and AgiSoft support CUDA:
    - Several pipeline stages use optimized, multi-GPU CUDA code.

Ten24 http://ten24.info/

So, if other markets have been using photogrammetry for quite a while, why the push for games applications now? We see it as the intersection of several trends. First, the demand for realism, detail and EXPANSE in games has continued unabated for several decades now. In addition, games continue to require more and more organic, and real-world locales. And game schedules and budgets can't keep growing. Those are major drivers of desire.

Why now?

- **Demand for realism in games:**
  - More detail, realism, organic look
  - On schedule…
- **Available GPU compute power:**
  - Requires lots of memory (images)
  - Requires tons of computation
  - Massively parallel
  - Reality Capture and AgiSoft support CUDA:
    - Several pipeline stages use optimized, multi-GPU CUDA code.

Ten24 http://ten24.info/

On the other end is the advancing technology. Thanks to GPU computing, the fact that photogrammetry is computationally expensive, image-based, data-heavy, and massively parallelizable means that leaps in performance on the order of MAGNITUDES rather than FACTORS is possible. Finally, the availability of compute-capable GPUs everywhere means that NVIDIA's CUDA and other languages are available on any game dev PC. Reality Capture and AgiSoft, among other tools have taken advantage of that, optimizing for, or even solely focusing on CUDA-based GPU algorithms.

What do *we* call "Photogrammetry"

- On-site capture
- Photo-based
- Single, independent (movable) camera
- GPU/CPU-hybrid model/texture extraction, e.g. Structure-from-Motion
- Simplification/Post-processing into game assets

While we'll discuss other methods in passing today, our concept of photogrammetry for this session involves mainly; on-site capture, rather than in-studio, scanner-style capture. Visible light capture, i.e. photos, not laser scans or special IR projector systems. The registration of cameras via image processing, and not via fixed cameras screwed into pre-calibrated rigs. And of course, we'll focus on the generation of high-res models in GPU-based software, along with simplification and processing to focus on real-time models.

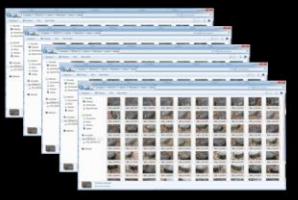
Photogrammetry and games – the dream/goal

- Take an easily portable camera (cell phone?) on site
- Take a walk in the sunshine…
- Shoot some photos and know when you have "enough"
- Upload them to your PC and…
- Out pops a perfect game model!

Just about any of us who have tried photogrammetry personally for the first time have this dream; I know I did when I first attempted it. We figured we'd take our favorite camera, maybe even just a cell phone, <CLICK x2> walk around the desired location, lackadaisically shooting pictures from what seemed like the right locations. We'd stand up normally like we were shooting vacation photos until we got bored, and then <CLICK x2> take the photos back to our desk and <CLICK x2> expect to have a nice, textured game-ready model pop out.

Photogrammetry and games – the fear…

- Lots of gear
- Hope for the "right" weather
- Tons of photos; "**F**ear **O**f **M**issing **O**ut"
- Tons of artist time in PS
- Wait for PG tools to run. And wait… And wait…
- Jury-rig a dozen tools manually
- Cry.



The dream isn't quite here yet…

But the fear need not be realized…

Then, there's "the producer's nightmare" of photogrammetry. The fear that photogrammetry means a big crew carrying expensive gear to some distant location. Upon arrival, they still forgot stuff. <CLICK x2>They sit in the hotel waiting for the right weather and lighting. Finally, they give up and shoot anyway because they have to fly home. <CLICK x2> They spend long days down in the mud, hoping they're getting the right photos. And on the other end of the return flight<CLICK x2> , there's days of photoshop work, hundreds of useless photos, and long waits for reconstructions to finish. Finally <CLICK>, more work to fix it up. The reality, from what we've seen, is <CLICK> that the DREAM is not yet realized, but the NIGHTMARE is very much avoidable even with today's tools.

Our speakers

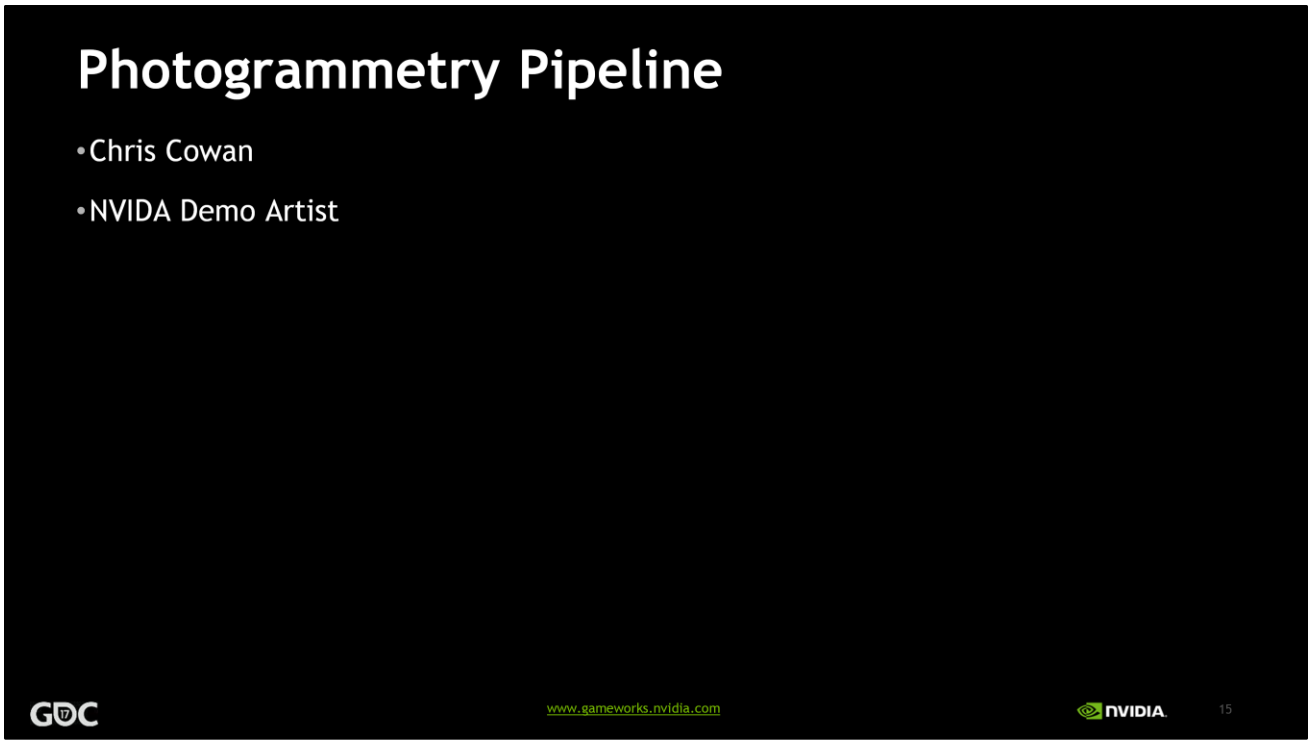
- **Chris Cowan:**
  - 20 years of experience using photogrammetry since before it was a “thing” (sigh – hipsters)
  - Walt Disney Feature Animation, Sony Pictures Imageworks, NVIDIA
- **Michal Jančošek:**
  - PhD, Computer Vision, Czech Technical University (Prague)
  - Years of research into better models from photos
  - Co-founder at Capturing Reality
- **Lars M. Bishop:**
  - 10 years in game engine middleware
  - 12 years (and counting…) in mobile devices, 3D rendering, cameras, Android apps/OS

So, having set the stage, I’ll yield it to our other speakers for a bit. Our next speaker, Chris, is an artist on the demo team at NVIDIA. He’s been a user of photogrammetry for about twenty years, starting in the film industry.

Our speakers

- **Chris Cowan:**
  - 20 years of experience using photogrammetry since before it was a "thing" (sigh – hipsters)
  - Walt Disney Feature Animation, Sony Pictures Imageworks, NVIDIA
- **Michal Jančošek:**
  - PhD, Computer Vision, Czech Technical University (Prague)
  - Years of research into better models from photos
  - Co-founder at Capturing Reality
- **Lars M. Bishop:**
  - 10 years in game engine middleware
  - 12 years (and counting…) in mobile devices, 3D rendering, cameras, Android apps/OS

Following Chris will be Meekal of Capturing Reality; he's made a successful transition from cutting-edge photogrammetry research in his doctoral dissertation to applying that work in the commercial package Reality Capture that he'll be demonstrating today. Finally, I'll come back at the end; I've split my career between 3D games middleware and mobile devices. Chris?



Hello, My name is Chris Cowan, I am a demo artist at NVIDIA

I have been working in the film and CG industry as a character modeler and a cg generalist for approximately 20 years.

I use a lot of tools to build a wide variety of objects and Photogrammetry makes that process faster and more accurate.

I started experimenting with photogrammetry about 20 years ago at Disney on the movie dinosaur.
There were many shots in the movie that required accurate reconstruction of real world locations.

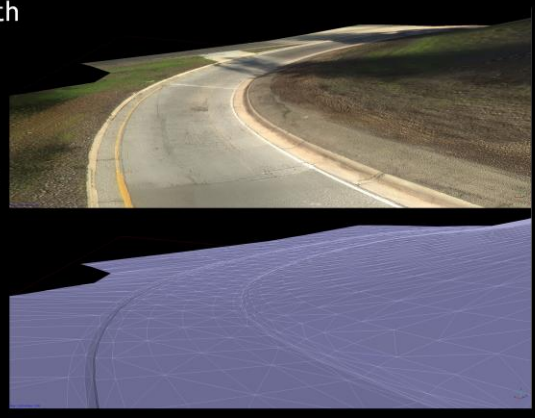
We ultimately used a combination of lidar and photogammetry to retrieve data.
At the time lidar was very new and very expensive, also the data was prohibitively heavy.
Photogrammetry was also new but the idea seemed very promising.
I was using photomodeler to place what would be *control or alignment* points now but were actually poly vertices of the handmade scan at the time.

Even with the limitations of early photogrammetry software it was apparent how valuable the information from just imagery could be…

Now with improvements like- gpu acceleration, High resolution digital cameras and especially software like Reality Capture, scans can achieve the detail and quality of extremely expensive laser scanning solutions.

Why is Photogrammetry important?

- The Who:
  - Anyone can become a high precision scanner with
    - Cheap hardware
    - Very little instruction.
- The Where and When:
  - Images can recreate scenes long after:
    - You leave
    - The location or object has changed or disappeared
- The How:
  - Reduced creation time
- The Why:
  - Opens up techniques from numerous industries:
    - VR, Film, VFX, etc.
  - Tools like Reality Capture are blazingly fast
    - Getting faster with newer GPUs
    - Automation and SDKs streamlining the pipeline.



GDC www.gameworks.nvidia.com NVIDIA 16

So what makes such a game changer..

With photogrammetry anyone can become a high precision 3D scanner, requiring relatively cheap hardware and simple basic camera techniques.

A basic camera instruction video and some practice with your camera of choice can produce amazing results…

Photogrammetry allows you to capture an object or place that might change or be destroyed at a later time… *Movie sets, crime scenes, or objects that decay for instance.*

The quality of the images will not improve after being taken, but the software will.. therefor you can expect to be able to reproduce even better results in the future from the same images.

With streamlined workflows asset creation time for real-world objects can be dramatically reduced. Dice & EA's work on Star Wars Battlefront is an amazing achievement and a first blueprint for real-world capture. One important slide in their GDC 2016 presentation asked - how much faster can an asset be made using photogrammetry. And the answer was - a first gen estimation of Approximately 50% faster and an estimated 70 % faster with future refinements in pipeline and process.

Photogrammetry allows an incredible amount of freedom... Image sequences from almost anywhere can be used to reconstruct real world objects. Vr walkthroughs of remote inaccessible locations, Set reconstruction and camera tracking for visual fx.

Reality Captures incredible speed and accuracy allows you to experiment with settings without tying up workstations for huge amounts of time.

And as newer and faster generations of gpus become available, results will be achieved even faster.

Also with The new sdk studios will be able to get under the hood and customize RealityCapture to their specific needs and pipeline.

Do you need expensive equipment or training… Absolutely not…

Any camera will work, but the higher the resolution and quality the better the results will be.

Prime lenses tend to be the best because of their consistency, and usually produce higher quality images.

Star Wars Battlefronts incredible results were achieved with just 20 MegaPixel cannon 6Ds and 24 mm lenses.

With newer cameras reaching 50 MegaPixels, incredible detail can be captured.

Video cameras are being used more and more as well. Obviously motion blur and rolling shutter are issues, but shot correctly individual frames can be used for construction.

Spherical cameras are also improving daily, and captures within the reconstruction area can record the environment for delighting purposes.

If all you have is a smartphone, excellent results are possible as well. Especially with the latest phones reaching 12 MegaPixels.

I have seen very detailed scans done with just a smartphone.

Drones with very good cameras are now common place.

And allow you to reconstruct objects and locations you could not reach any other way.

Drones produce very clear, stabilized images perfect for reconstruction..

And quality will improve rapidly as the drone industry matures.

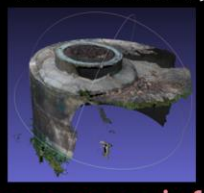
RealityCapture also allows you to incorporate Lidar scans into the photogrammetry solution, allowing you to get the best of both worlds.

Image Capture: Object/Scene

- #1 Goal: Don't make *unrecoverable mistakes*
- Coverage of the target object
  - Too many photos are better than too few
- Lighting Environment:
  - Sharp shadows can be a big pain later
  - Shoot on overcast day, under tree cover, etc
- Camera Settings: Full Manual (Exp/Focus)
  - Shot-to-shot consistency, but if you get it wrong…
- Camera Settings: Auto-exposure
  - Fool-proof; images can be adjusted later
- Sharpness is king!
  - Watch your shutter speed, focus and depth-of-field
  - But, but – DoF and shutter speed are at odds… Higher ISO?

Blown highlights are forever

Blurry photos are forever
(PG SW lives and dies by details)



Missing coverage is forever

On to the process… Image Capture..

Coverage and overlap are very important. You should have at least 50% overlap to allow for good camera alignment.

Lighting is important..

Good ambient lighting is best. Overcast days outside and Diffused Light boxes inside.
It is best to shoot raw to retain higher bit depth, and future proof your data. No in camera processing except white balance if necessary and only mechanical image stabilization.

As for camera Settings:

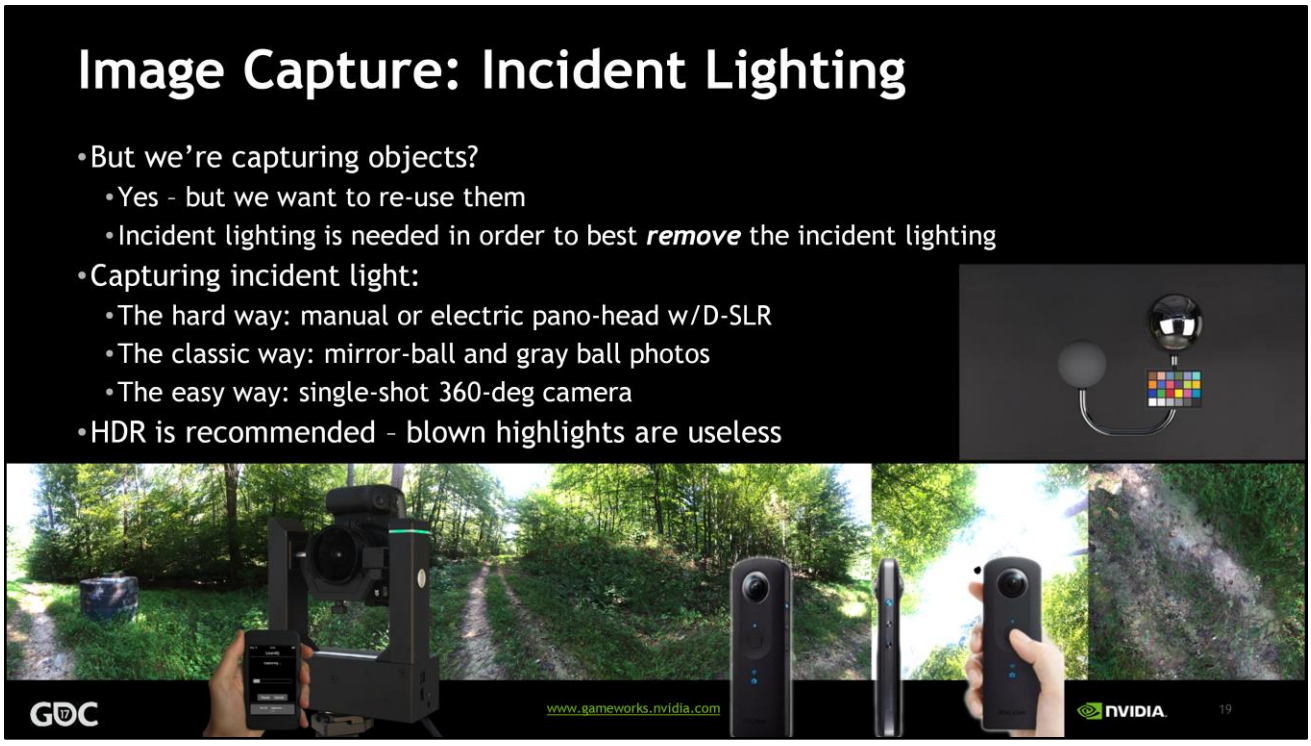
Manual gives you the most control of the image but is the easiest to screw up.

Automatic will usually always get the shot but can introduce a lot of noise in challenging lighting situations.

Higher shutter speeds should be used to reduce shake or blur.
But every person has a different threshold for how slow a shutter speed they can use, and still get crisp shots.

Always use the lowest possible ISO setting and as much depth of field as possible without sacrificing quality.

Ultimately It's all about light, and how you trade off between shutter speed, ISO and aperture, to get the best shot.



Why worry about the environment when you are capturing objects?

Captured objects will most likely end up in scenes that have completely different lighting conditions than they were captured in.

This means the environmental light contribution and shadows at the time of capture will need to be removed from textures.

An HDR panoramic image can be very useful for determining the lighting conditions at the time of capture..

along with a color calibration chart.

Unfortunately HDR Panoramas can very time consuming to make on location, but with the advent of new panoramic cameras, the process can be speeded up considerably. Also without the time consuming stitching process needed for traditional hdr panoramas when you get back to the studio.

Cameras like the Ricoh Theta S are very portable, (even shirt pocket sized) and apps make the capture of multiple exposures for HDR effortless and fast.

Image Prep

- Should you use two sets of images?
  - Tools like RC let you flag images as “use for geometric mesh only”
- One set for mesh construction:
  - Masking of the target object
  - Sharpening of detail without introducing noise
  - Structured Light / projected pattern images
- One set for texture extraction:
  - Removal of shadows/highlights
  - Color/exposure equalization

What to do with your images??

Should you use two sets of images?

Two sets of images can be used to produce a final result… one geared towards the best possible reconstruction… and another for the best texture result.

For reconstruction –
Masking can be done on the Images helping reduce model cleanup. Masking is often used for objects being captured on a turntable.

If you find some reconstruction images are soft and are absolutely necessary to get good coverage, some sharpening can help. Contrast can be increased as well.

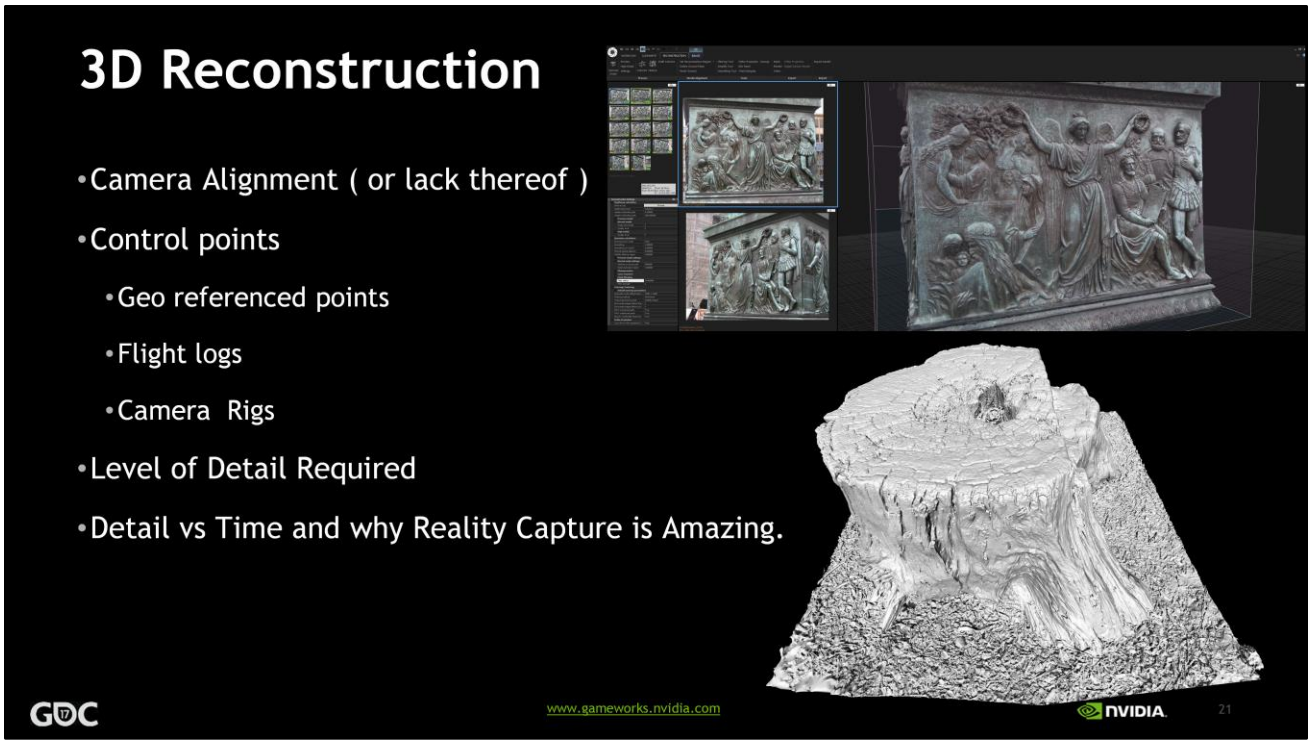
Reconstruction images might also use *a structured light pattern*. The pattern is projected onto the object, then turned off before a second set of texture images is taken in quick succession.

For the texture Image set -

The images for texture extraction should have shadows and highlights removed.

Color and exposure should be equalized taking into account the incident light contribution - possibly using a panoramic hdr image taken at the time of capture.

The goal should be a final texture with just the pure color of the target object.



Now you are ready to start building something…

Images for reconstruction are imported into Reality Capture and the alignment process is run.
Hopefully, all your careful camera work and image overlap will give you one component.

If you end up with more than one component..
Control points can be placed in each component that have visible matches.
The separate components can then be aligned and merged

Ground control points can be used to geo-reference an object, and give scale to constructions as well.
Flight logs are also helpful in alignment and geo-referencing constructions.

Camera rigs, such as the ones used by TEN 24 with a 100 plus cameras are also going to be supported.
So all camera positions can be predefined for highly accurate repeatable results.

It is a good idea to know what level of detail an asset is going to require...
Like lidar, photogrammetry data can get heavy quick.
Reality Capture makes things a bit easier with it's incredible speed and quality.

The normal setting achieves quality comparable to hi on most other photogrammetry options. And at many times the speed.
Allowing you to quickly decide if maximum settings are necessary.

Even at maximum settings results are generally as fast or faster than other solutions on normal settings.

Export and Retopology

- Decimation and export
  - You will likely need to decimate some in PG tool
  - Many external geometry tools cannot handle huge, full-res PG meshes
- Mesh Cleanup
  - Non-manifold, cracks, duplicated tris, etc are common in export
  - PG tools getting better about these…
- Re-topology
  - Animated vs Static Topologies
  - Level of Detail
  - Re-topology Software Options:
    - TopoGun
    - MAYA / 3DS MAX
    - 3D Coat
    - Zbrush
    - Wrap 3 (blend-able topology)

So now you have a highly detailed high poly mesh, but now you need an efficient asset. Time for retopology…

Generally you export a decimated version of the hi poly model from RealityCapture.

One that has all the form but not the hyper (pore level) Detail.

If the Hi Poly mesh has missing areas or needs cleanup it can be brought into programs such as zBrush, meshlab or geomagic and cleaned up there.

For re-topology, The decimated representative mesh is brought into a program such as topogun, maya, 3ds max, zbrush, 3d coat etc then an ordered mesh is snapped to it.

The type of re-topology you perform is most likely dictated by the asset..

Whether it is going to be Animated, Static, a hero or background object.

Animated topologies usually require a fairly ordered layout, while static objects like rocks and trees can be less so.

An optimized hand built mesh will usually be lighter and cleaner.

Although! automated re-topology has come a long way in programs like zbrush and 3D coat.

Texture Generation

- PG tools generate high-visual-quality textures
- But…

- The UVs are… Not. Great.
- So other tools are needed

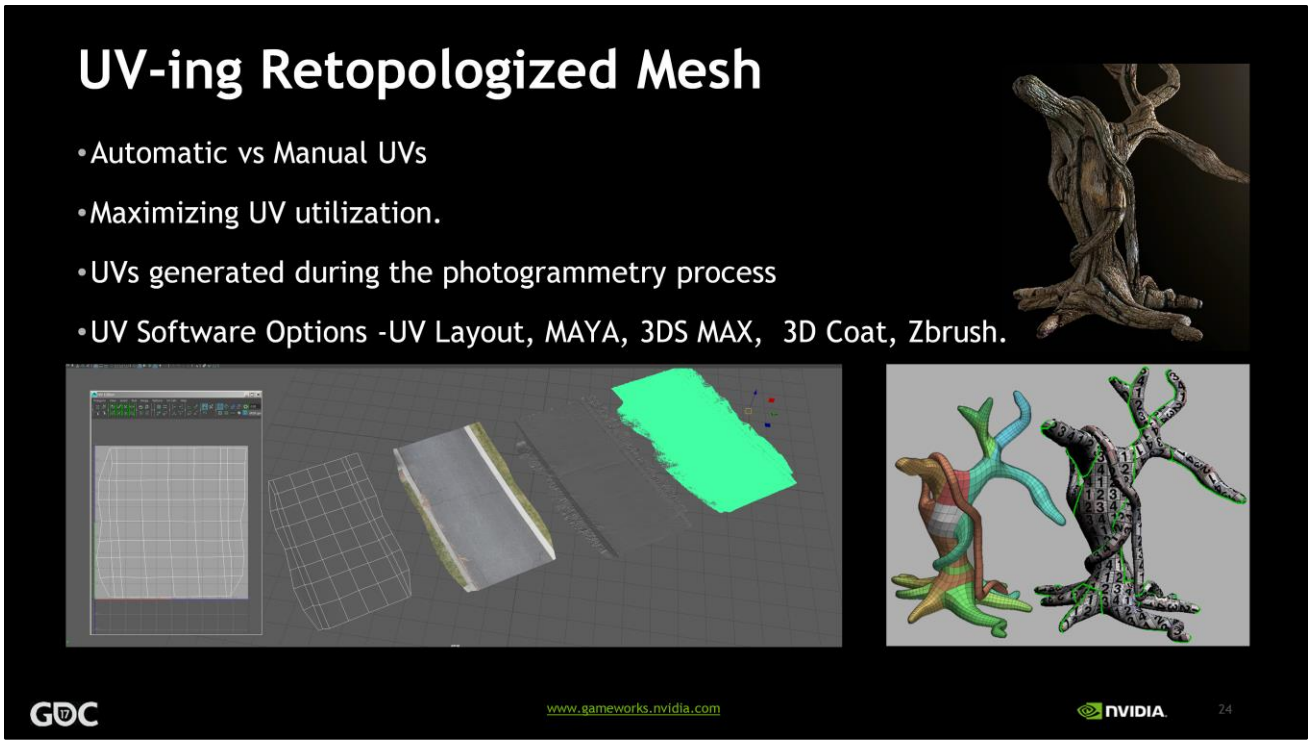
modo
3D COAT
AUTODESK MAYA
AUTODESK 3DS MAX

So.. You have your model now you need textures…

Diffuse textures can be projected onto the High poly mesh immediately after reconstruction but there can be some issues....

The Uvs can be less the optimal.

So we turn to external software options.



UVs can either be generated automatically or laid out manually.

Automatic methods have improved, but, they still tend to waste large amounts of precious map space.

Photogrammetry packages can create uvs, but they tend to be shredded and disorganized as the result of being projected from each image used in the 3d construction.

This haphazard uv layout makes editing of textures in external software very difficult.

There are many third party programs that uvs can be laid out in.

I prefer UV Layout but 3D Coat, 3DS Max, Modo and Maya have many tools and add-ons to get the job done.

Zbrush has a good automatic unwrap that can give acceptable results as well.

Texture Re-projection and Baking to the Re-Topologized Mesh

- Internal re-projection from original photographs to imported re-topo "cage"
- (External) Lighting map baking:
  - Normal map, Displacement map, AO map, etc…
    - Software tools for external baking:
    - xNormal
    - Substance Painter/Designer
    - Modo
    - Maya / 3DS MAX...

Now that you have a efficient asset with optimized Uvs its time for texture projection and baking.

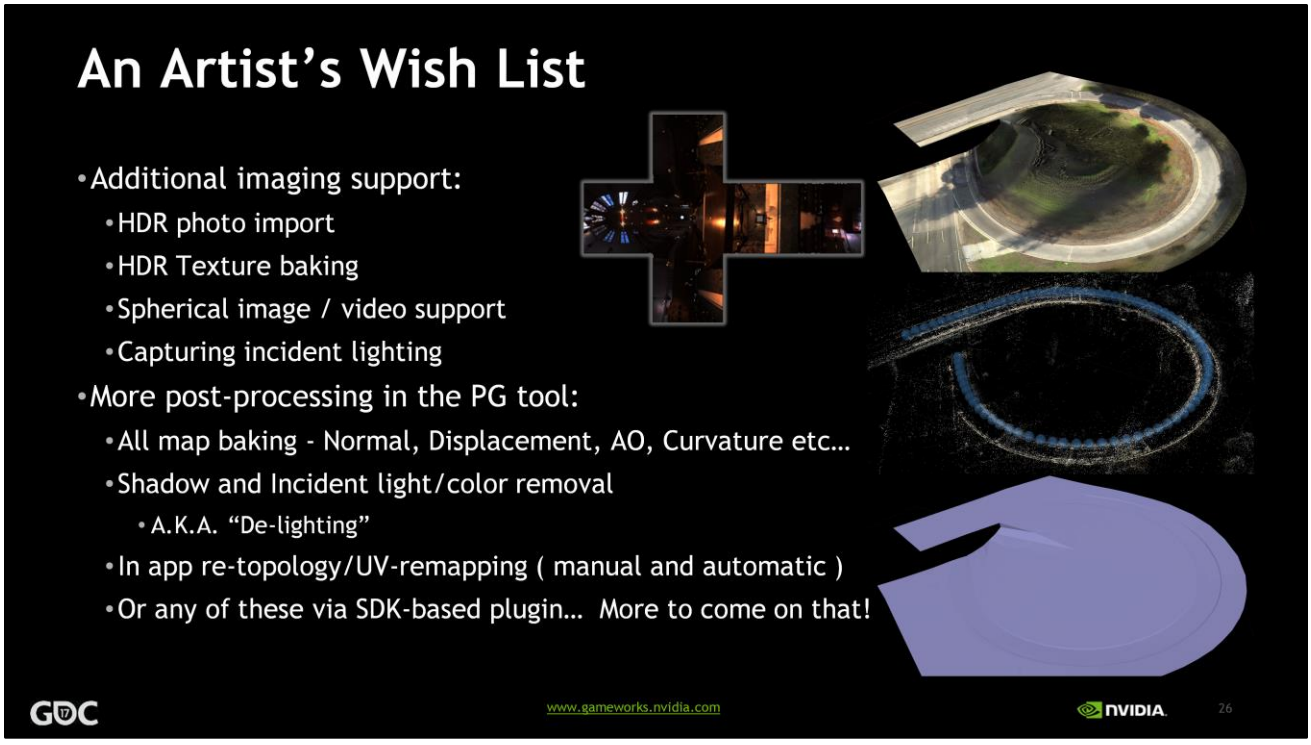
The re-topologized and re-UVed mesh can now be brought back into RealityCapture and the diffuse texture can be projected onto it.

For lighting and detail maps external baking is still necessary.

Finally.

Maps such as: Normal, Displacement, AO, Curvature, Ect.. can be made by exporting the highest resolution model your system can handle from RealityCapture and bringing it and the re-topologized mesh into another program such as xNormal, Substance Designer, Substance Painter, Modo, etc .. etc .. and baking the maps there.

And this completes the asset creation process.



And finally my wish list for new features

Hdr and High bit depth image input and baking support , on RC roadmap

Spherical image support. On RC roadmap

Direct video support. Using frame increments or manual frame choice.

Capture and processing of incident lighting.

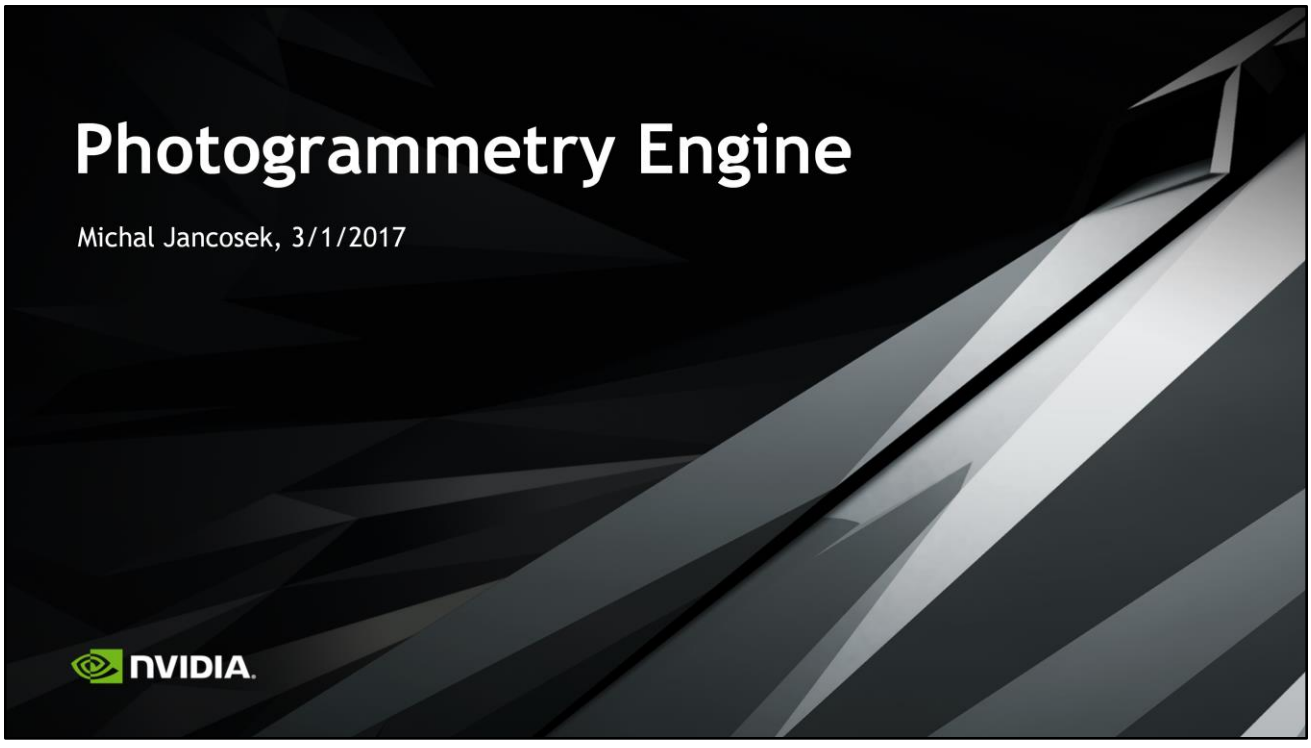
It would be nice to move more of the external processes into RC -

Baking of all the detail maps- Normal, Displacement, AO, Curvature etc... on RC roadmap

Shadow and incident light removal from the diffuse texture..

Pie in the sky … Manual and automatic re-topology (mesh creation)

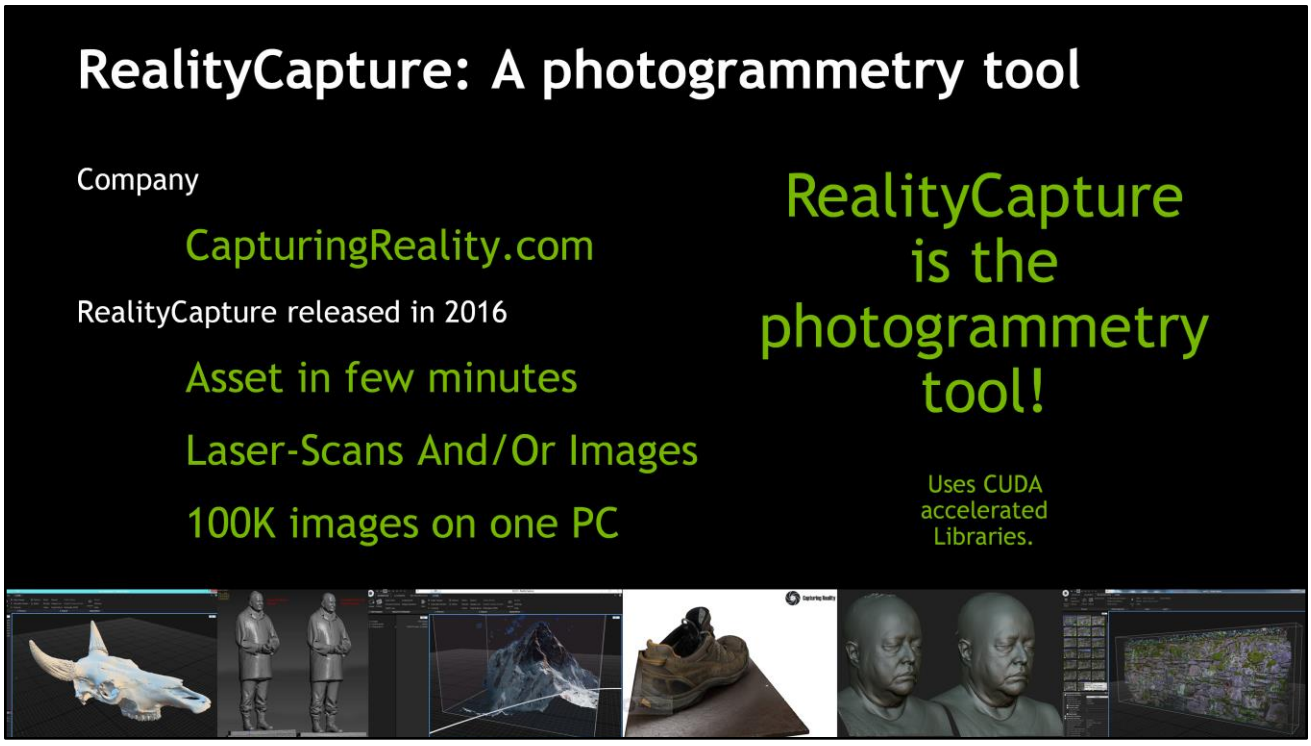
OR – with the use of the new SDK.. Live link to external applications to perform these types of functions.


Photogrammetry Engine
Michal Jancosek, 3/1/2017

Imagine that you can create an asset to your game of a real object or location just from images.

Well, this process has been studied for centuries and is called photogrammetry.

It is clear that you need a photogrammetry tool.



Let me introduce RealityCapture.
A photogrammetry tool produced by us.

We released it publicly more than a year ago.

RealityCapture is fast. You can reconstruct a model in just few minutes.

It is the first software that has been able to process terrestrial lasers scans with images automatically.

And you are not limited. You can process even one hundred thousand images on a PC.

Photogrammetry with RealityCapture

What can be done

100,000-image example

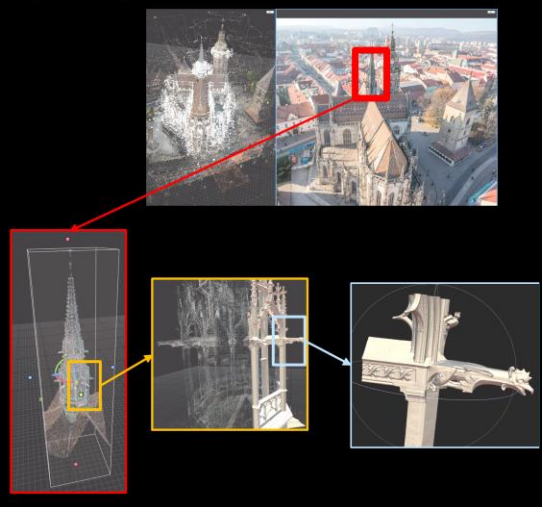
on ONE PC

Most probably

You will not do such a project

However, it shows that

You are not limited

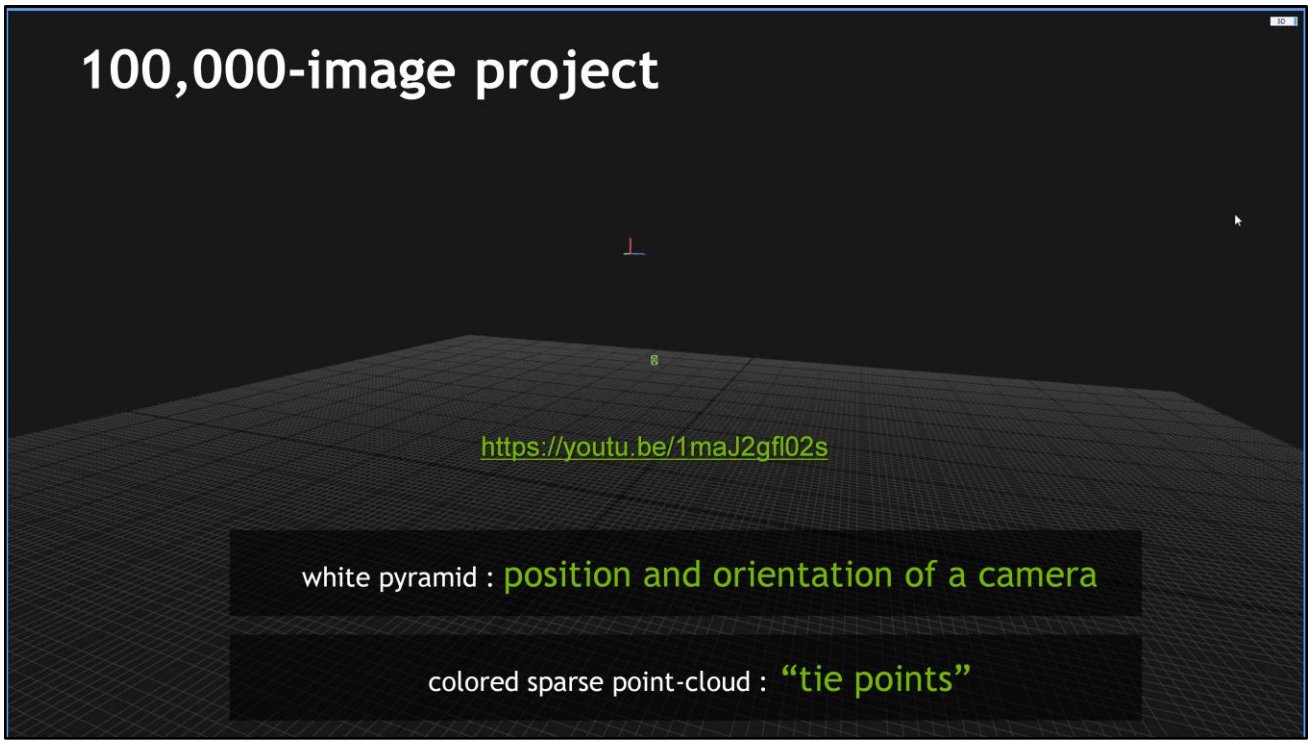


DATA PROVIDED BY STUDIO 727 : www.727.sk

Let me show you a project of one hundred thousand images running on one PC.

You will most probably not do such a project.

But this way I just want to demonstrate
that you are not limited by the number of images
or by size of your project.



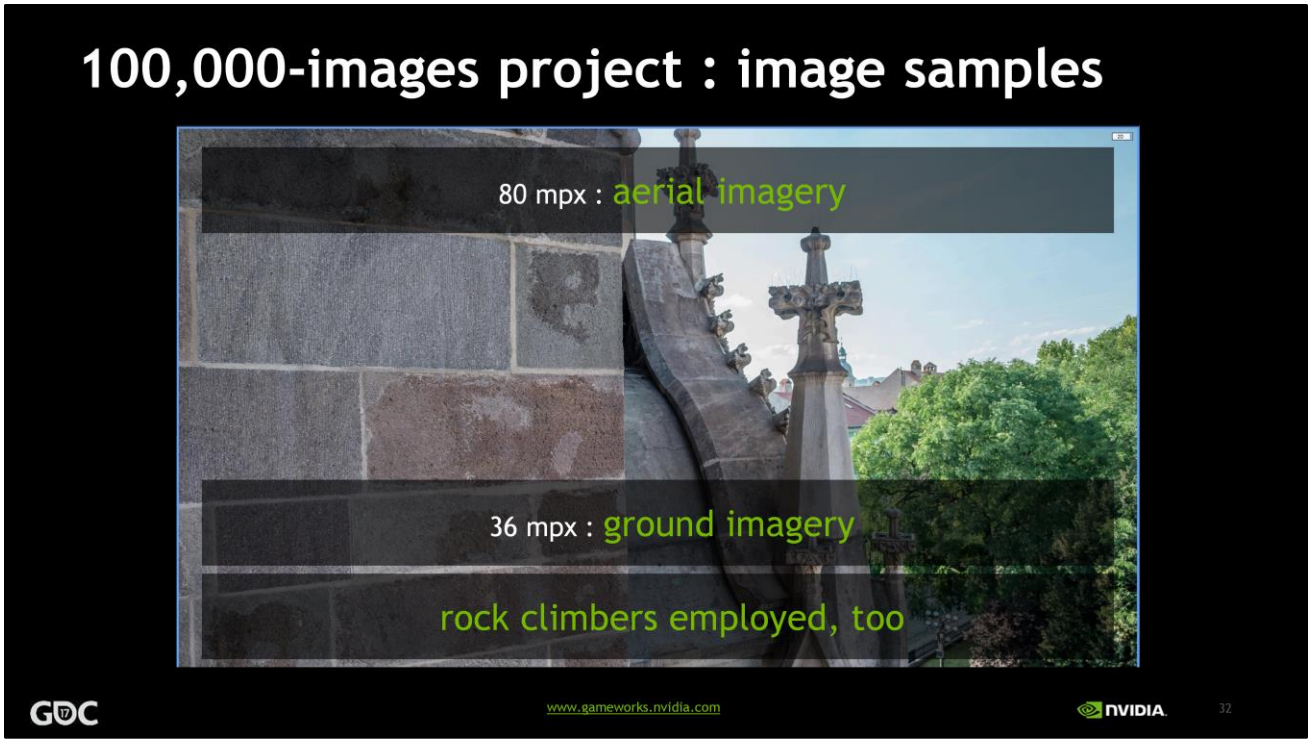
https://youtu.be/1maJ2gfl02s

This is a 3D view of our application.

Each white dot represents one image.

More specifically, it represents a camera position and orientation at the time when the image was taken.

The colored sparse point cloud is something we call “tie points”
and it is a by-product of image alignment process.

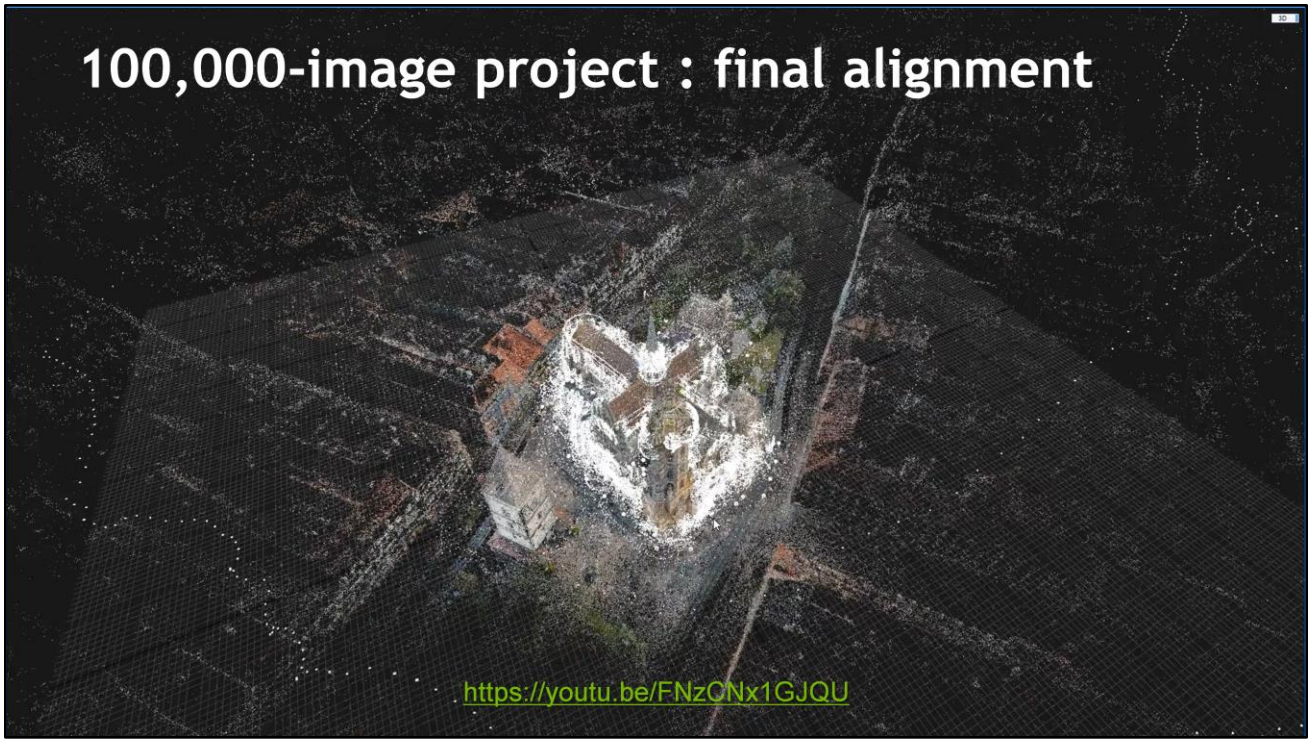


Here you can see few image samples of the project.

The aerial images have been captured from drone.

Then there have been many ground-level images captured from street, windows, roofs etc.

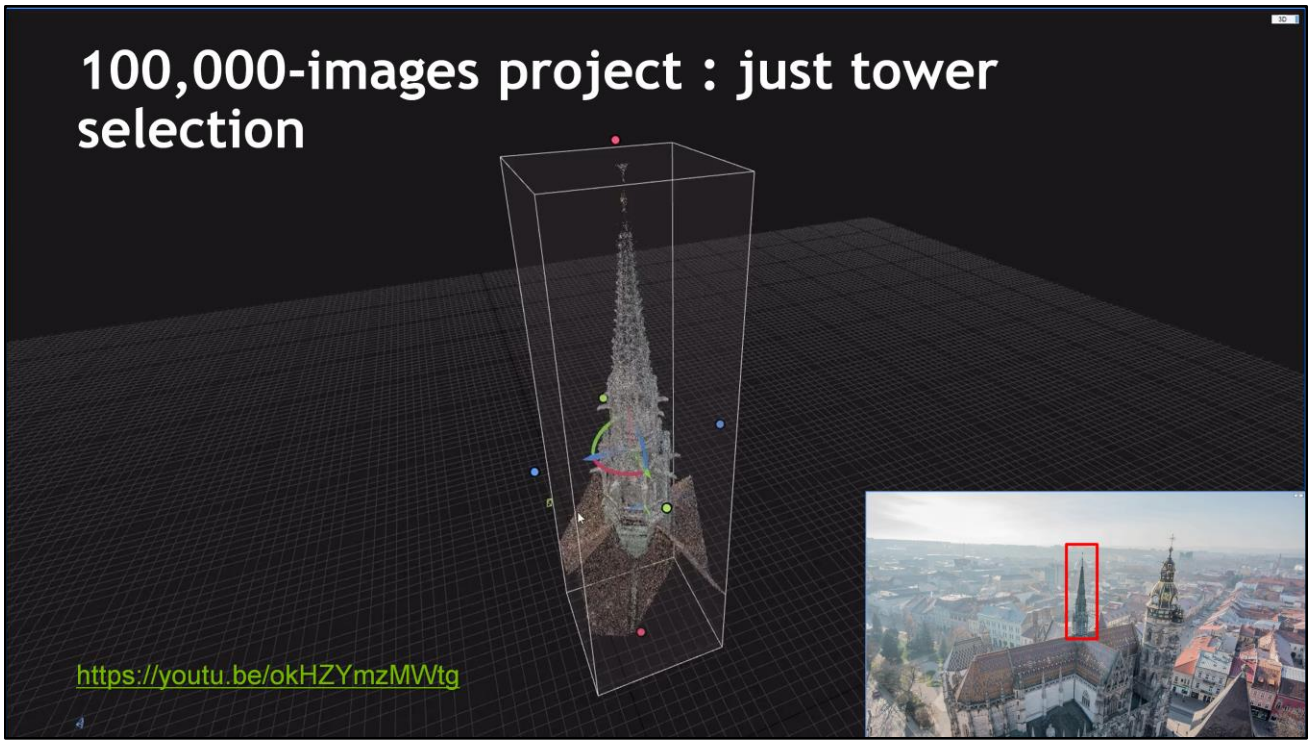
Also
rock climbers
have been employed
to capture some details



https://youtu.be/FNzCNx1GJQU

Let’s have a look at the alignment once again

Each white dot represents an image.



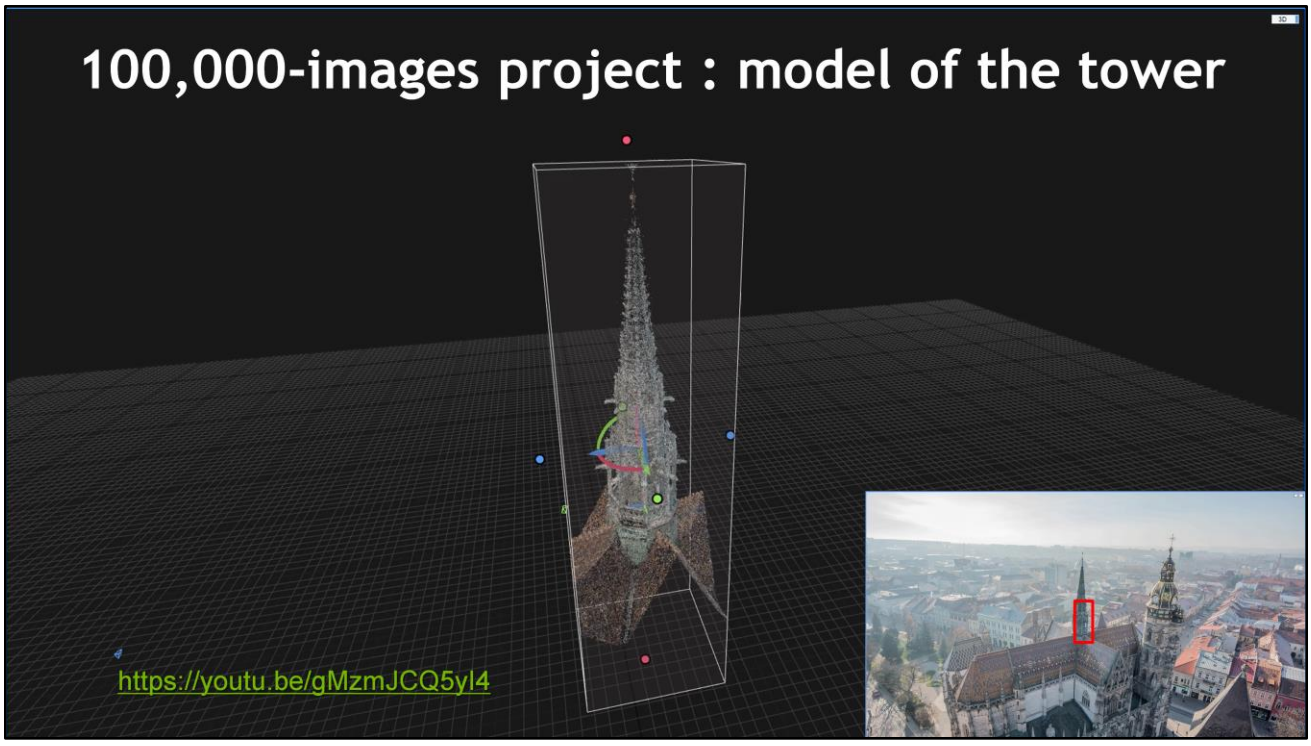
https://youtu.be/okHZYmzMWtg

Let’s take just this tower now.

The tower is 59 meters high, which is 194 feet.

This is just a sparse point cloud now.

It is not a model.



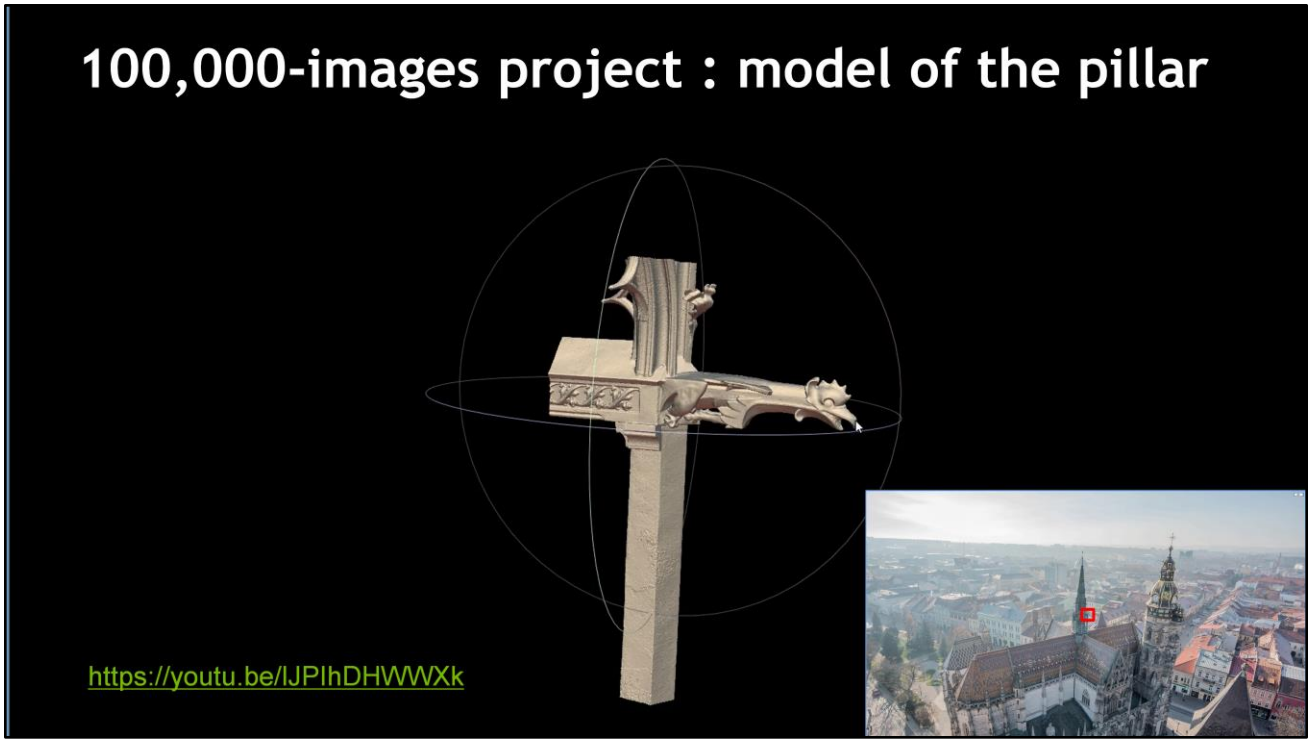
https://youtu.be/gMzmJCQ5yI4

Now let's take a look at the model.

It is in millimeter accuracy.

so you can see all the details …

Even the lightning conductor is reconstructed.



https://youtu.be/lJPIhDHWWXk

Let's take a look at
just one pillar of the tower.

The red rectangle displays its location on the cathedral.

The whole cathedral has been reconstructed in this detail.

This gives you an idea on what size the project itself must be.

It's huge!

You are most probably interested just in assets of this size.

However, here I just wanted to demonstrate that you are not limited with RealityCapture.

Photogrammetry process demo

Small dataset

- 200 images
- captured in 39 minutes

Processing times

- alignment in 4 minutes
- model in 42 minutes

PC used

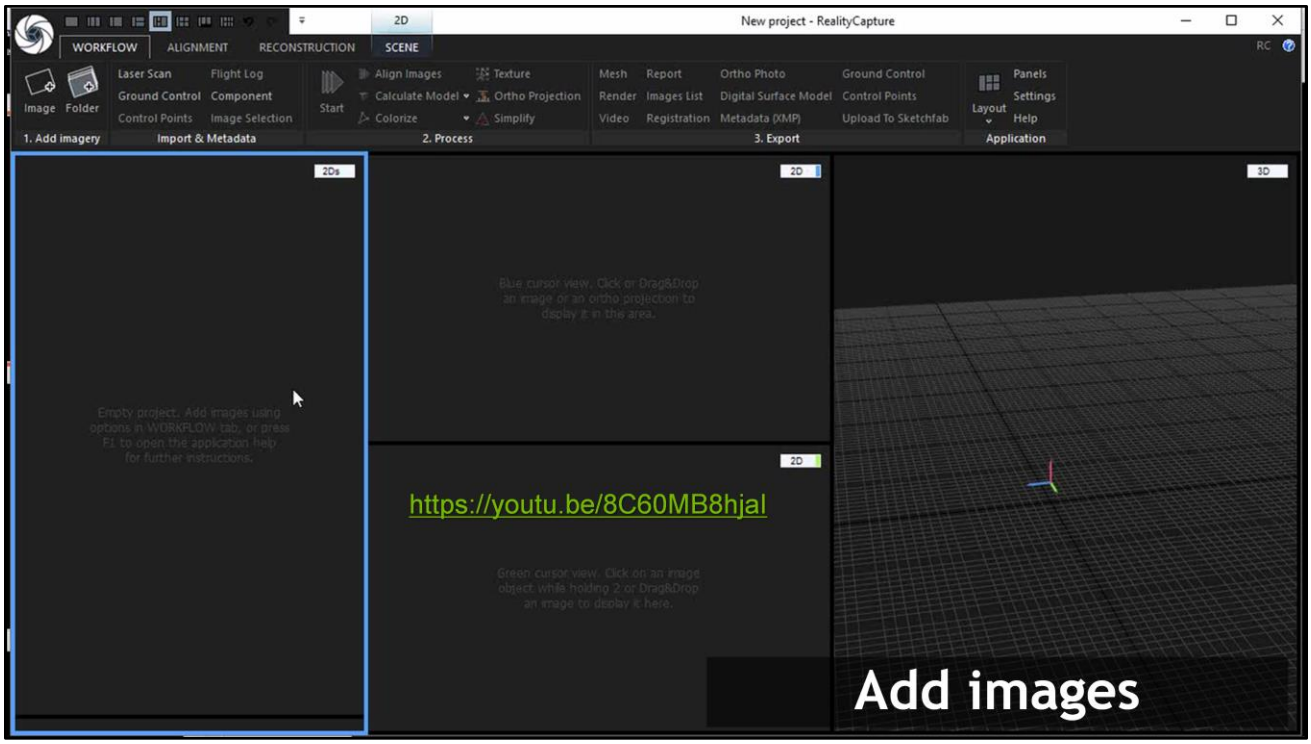
- ordinary gaming PC
  16 GB RAM, i7 CPU, GTX 960

Let’s take a look at a smaller project that you are probably interested in the most.

I will guide you through a basic workflow in our application.

This dataset was captured just in 39 minutes.

There are something like 200 images.

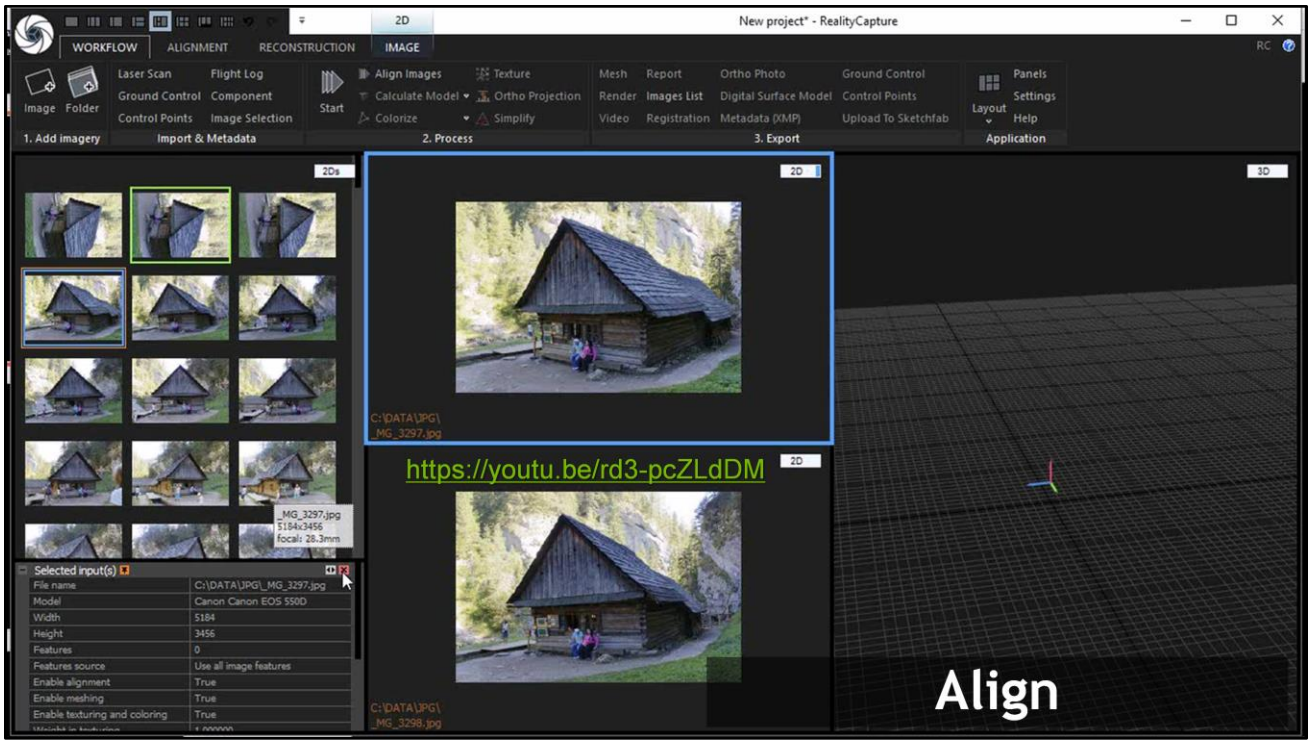


https://youtu.be/8C60MB8hjaI

This is our application.

You can just drag and drop images into the application.

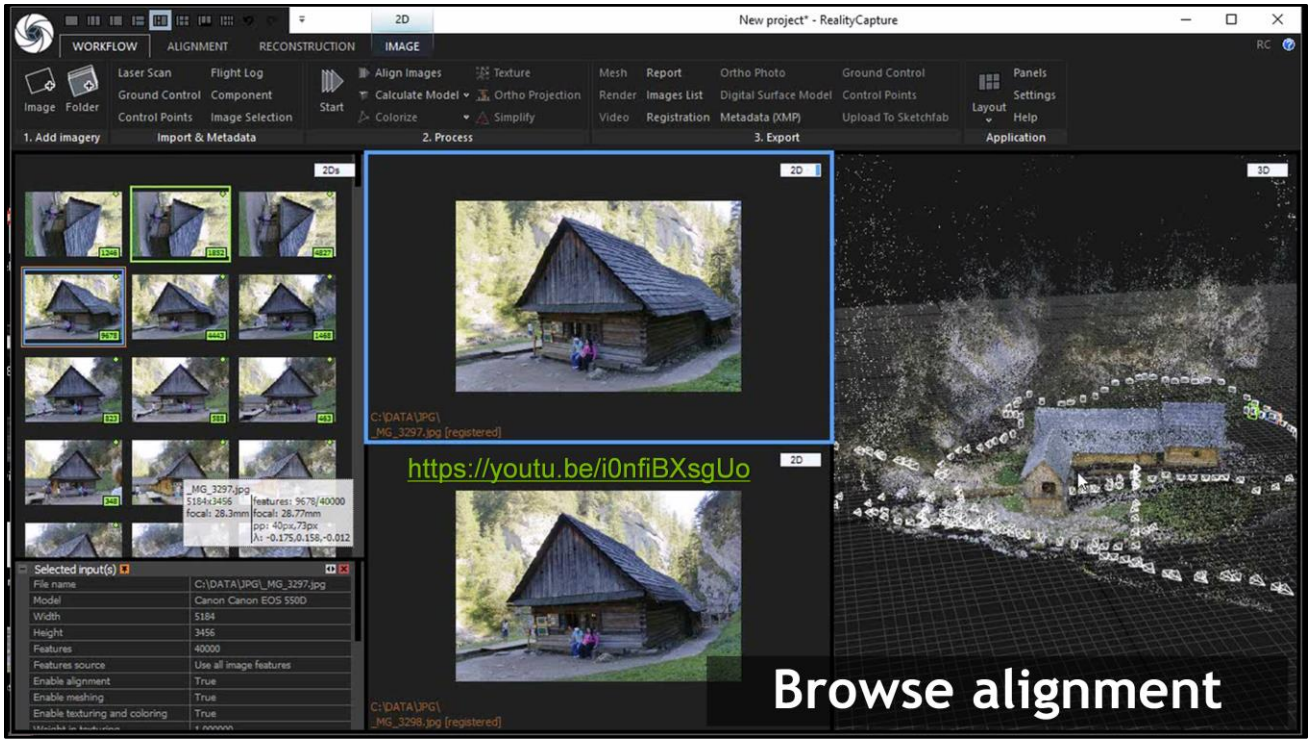
You can browse them in the left panel or in 2D views.



https://youtu.be/rd3-pcZLdDM

Then you just press the Align button.

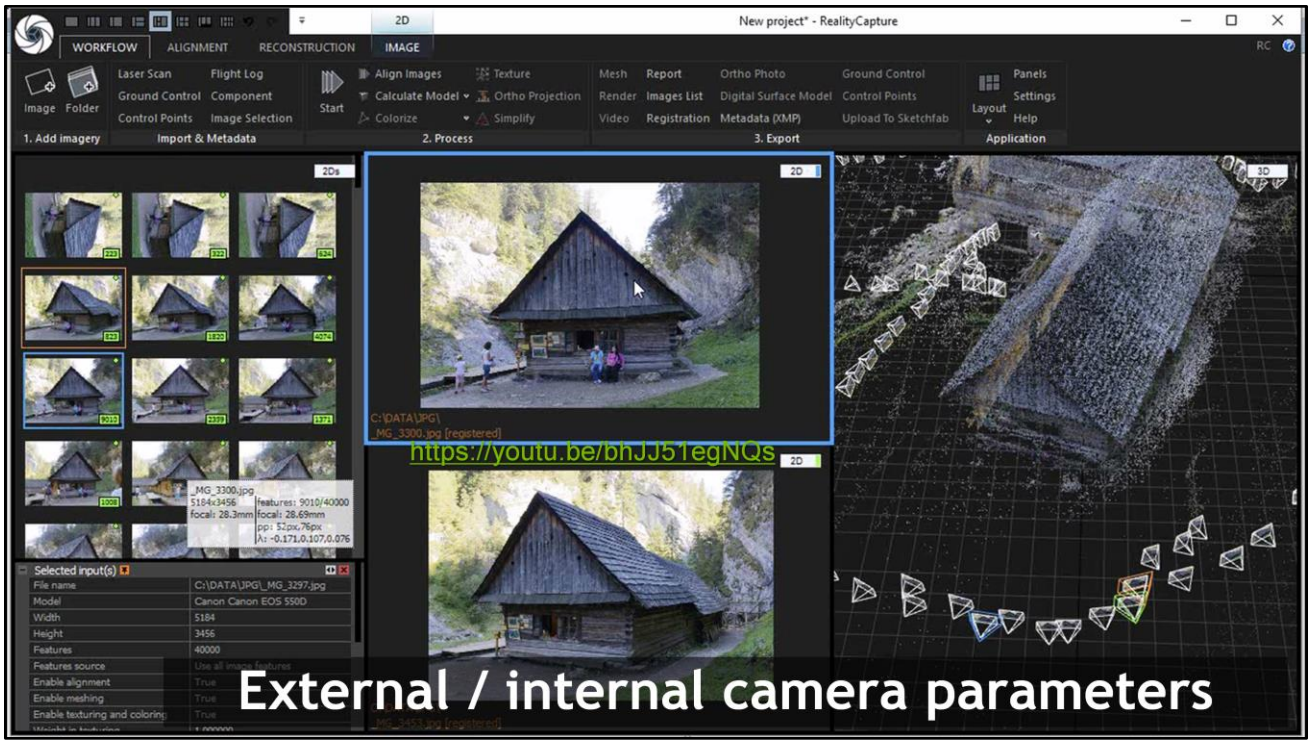
It took just 4 minutes to align these 200 images.



https://youtu.be/i0nfiBXsgUo

And after a while you can see the alignment in the 3D view.

You will not have time for a coffee for sure :)



https://youtu.be/bhJJ51egNQs

Camera poses are computed for each image during the alignment.

Moreover, focal length and image distortion are computed, too.

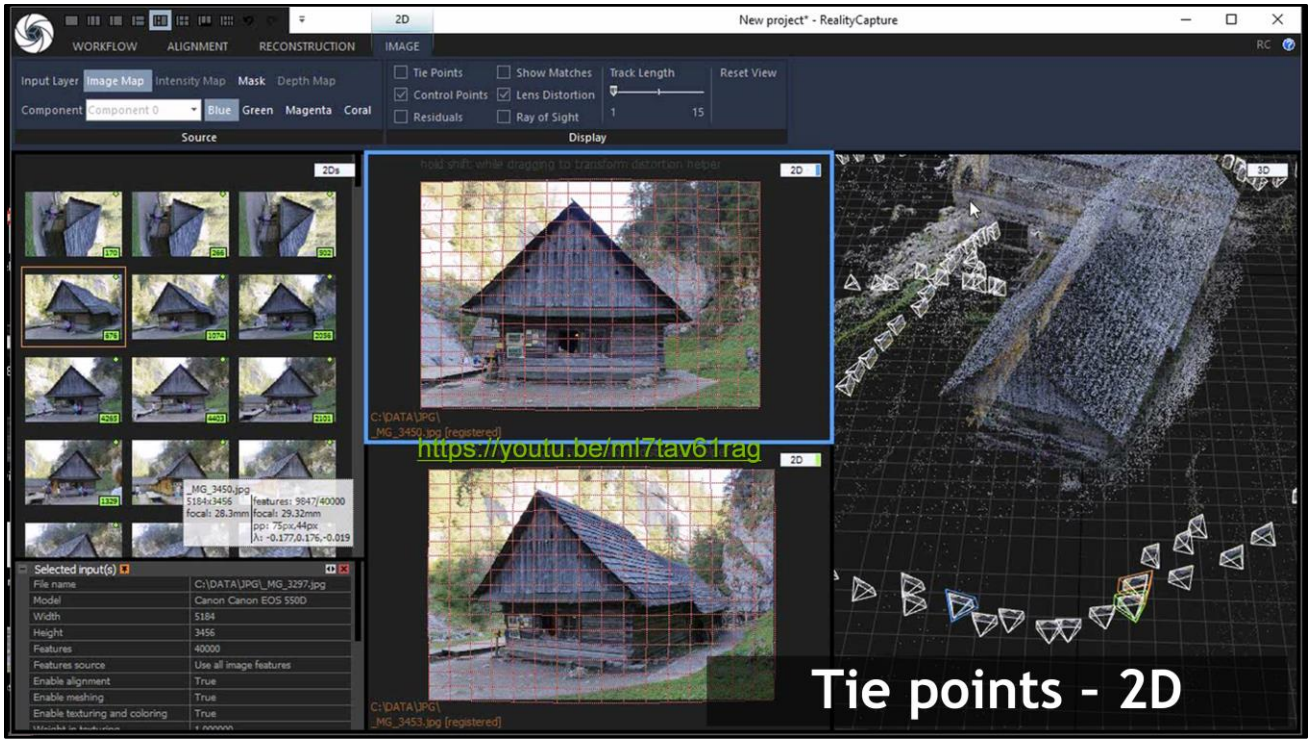
You can check it here.

That means that all you need is just your photos.
No pre-calibrations
No industrial cameras
Just use anything you have.
For example your Iphone, your DLSR, tour GoPro … whatever.

The colored sparse point cloud is a by-product of the alignment process

We call these points as “tie-points”.

You can see them in the 3D view, too.

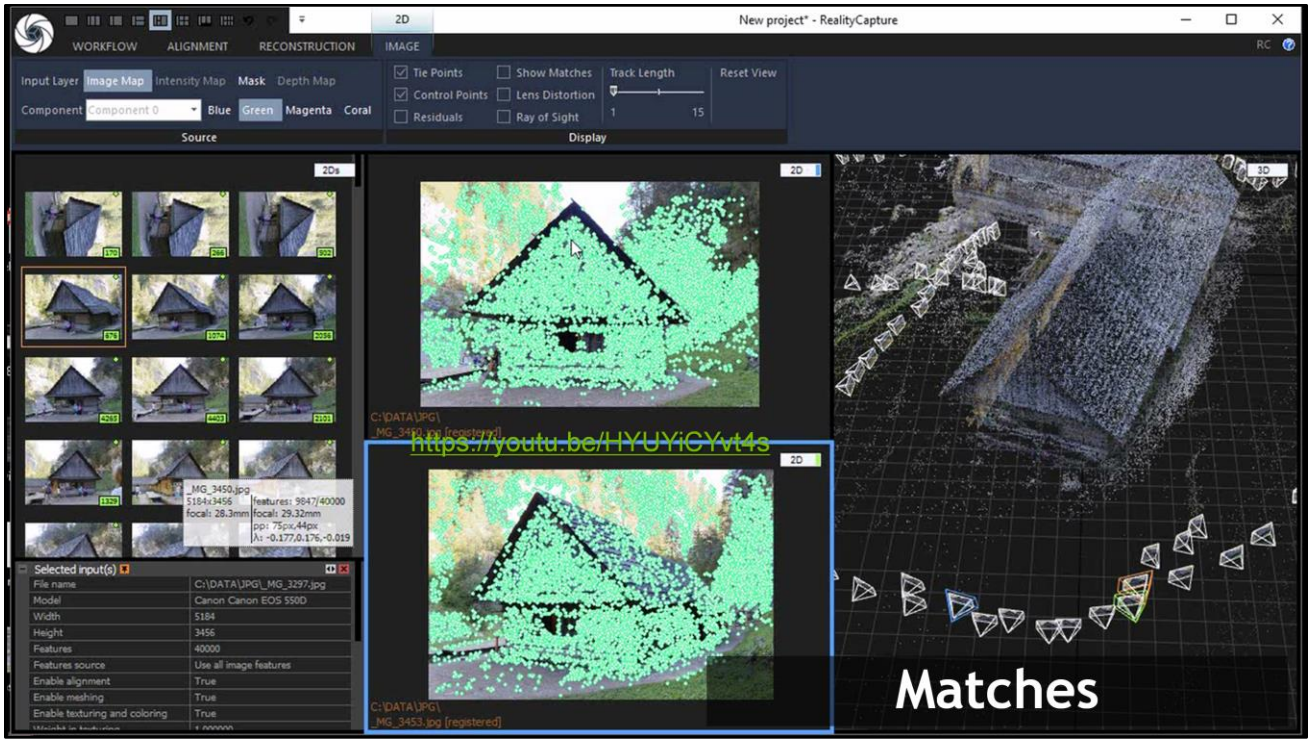


https://youtu.be/ml7tav61rag

So how it works internally?

First, the software detects some unique features in each and every image.

For example :
a window corner
or
spot on a wall

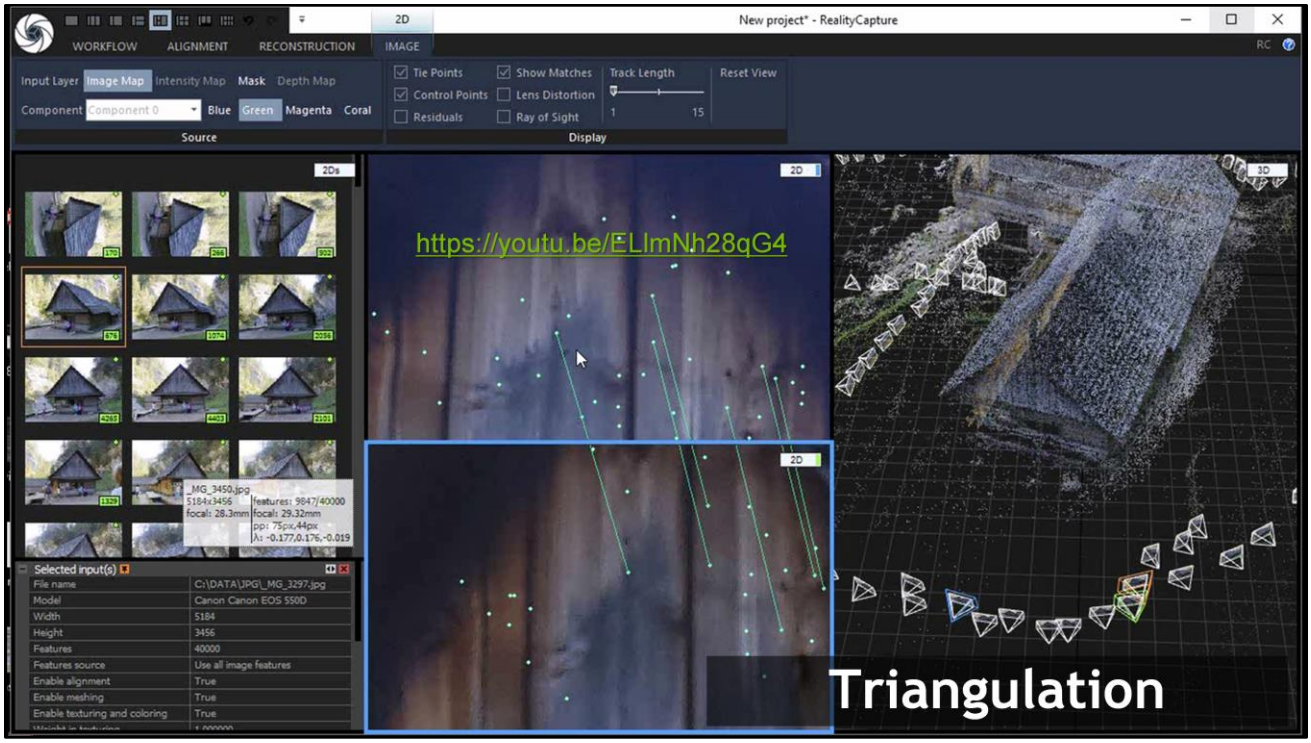


https://youtu.be/HYUYiCYvt4s

Next, the software finds all images that see the same spots

here for example on the walls.

In other words,
it groups all features
that match the same physical point.

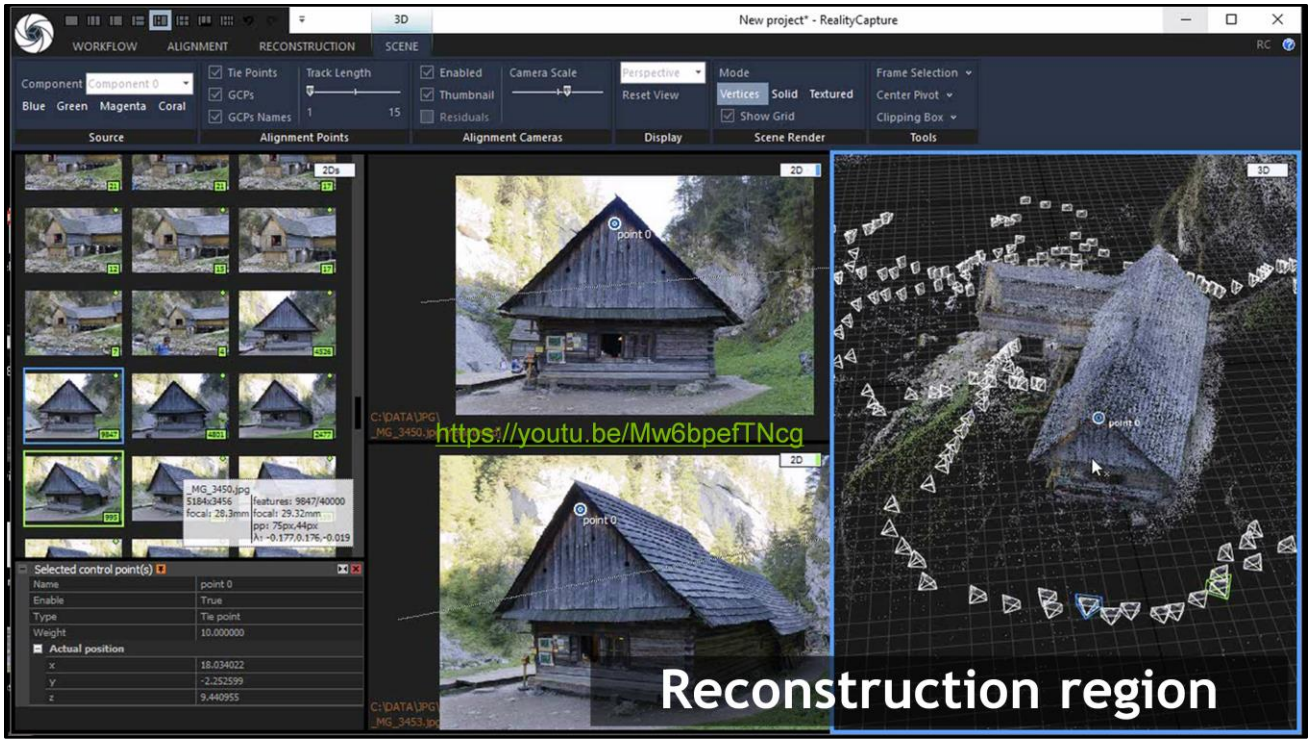


https://youtu.be/ELlmNh28qG4

Once the software computes camera poses and matches.

Then it is easy to compute a 3D point for each match using triangulation.

Here I placed just one control point
to demonstrate
the internal triangulation process.



https://youtu.be/Mw6bpefTNcg

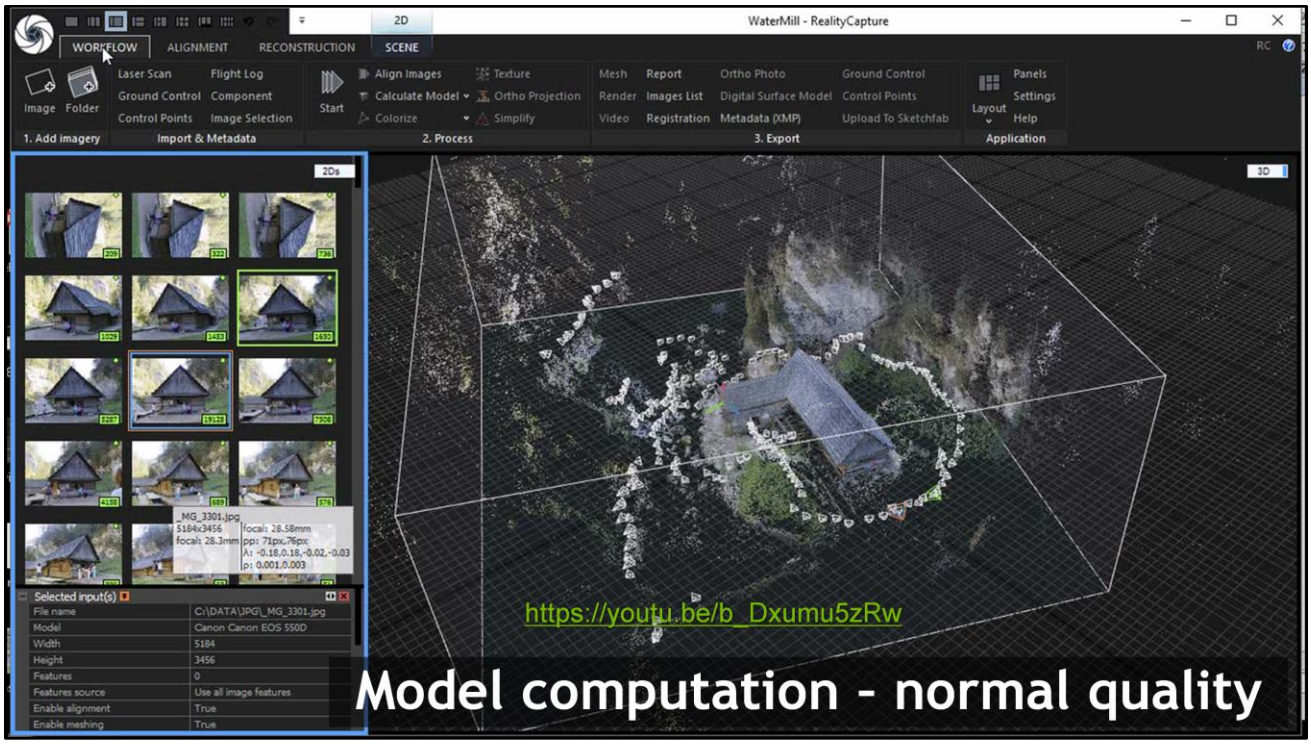
The tie points are very helpful.

They help you to
see the scene structure
so that you can select your object of interest.

Use this reconstruction region tool to select it.

This is the best practice
because the application will not have to reconstruct parts
that you are not interested in
and so it will be faster.

Finally, you can decide what quality you want your model to be in.
Preview
Normal
High



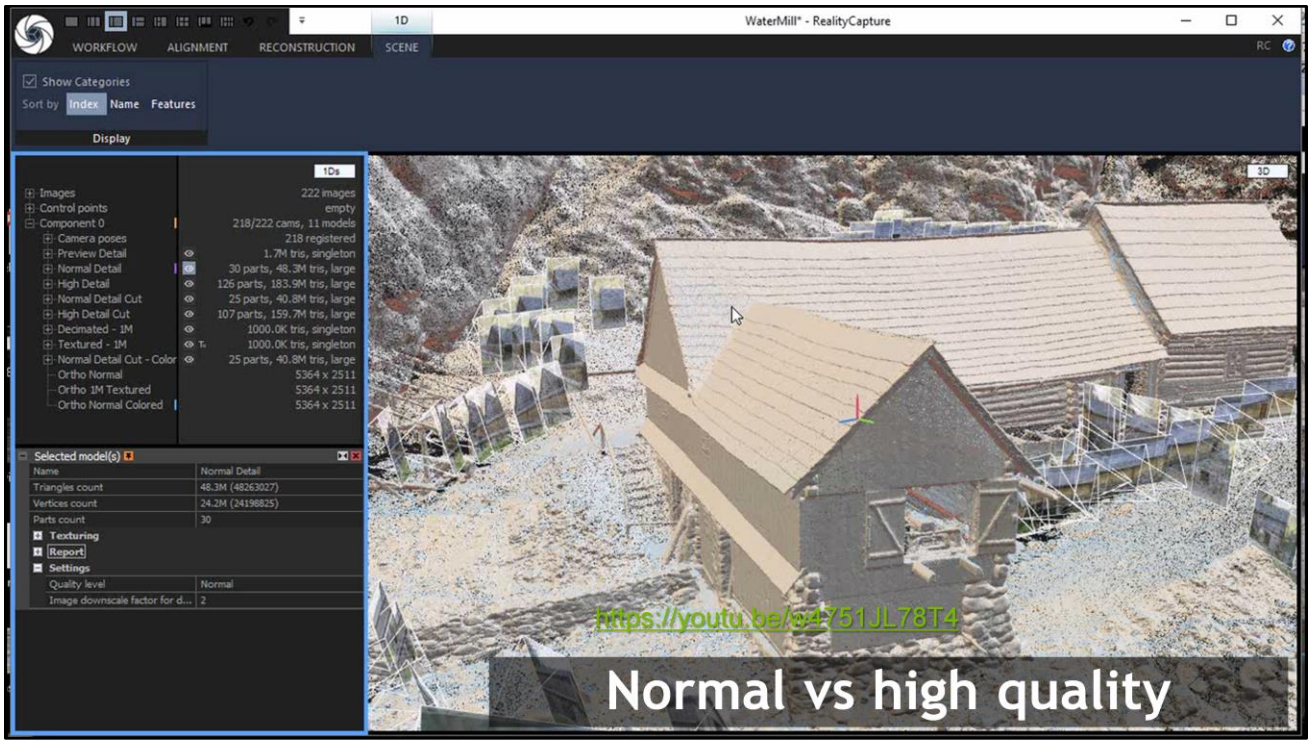
https://youtu.be/b_Dxumu5zRw

Let's take a look at the models.
Here,
in the 1D view on the left,
you can see the project structure including all models.

The preview quality is very low.
It is literally just a preview.
But it is fast since it is just meshing of tie points.

This is the model computed in normal quality.

I like to use the normal quality
it is reasonably fast
and
it gives good results.



https://youtu.be/w4751JL78T4

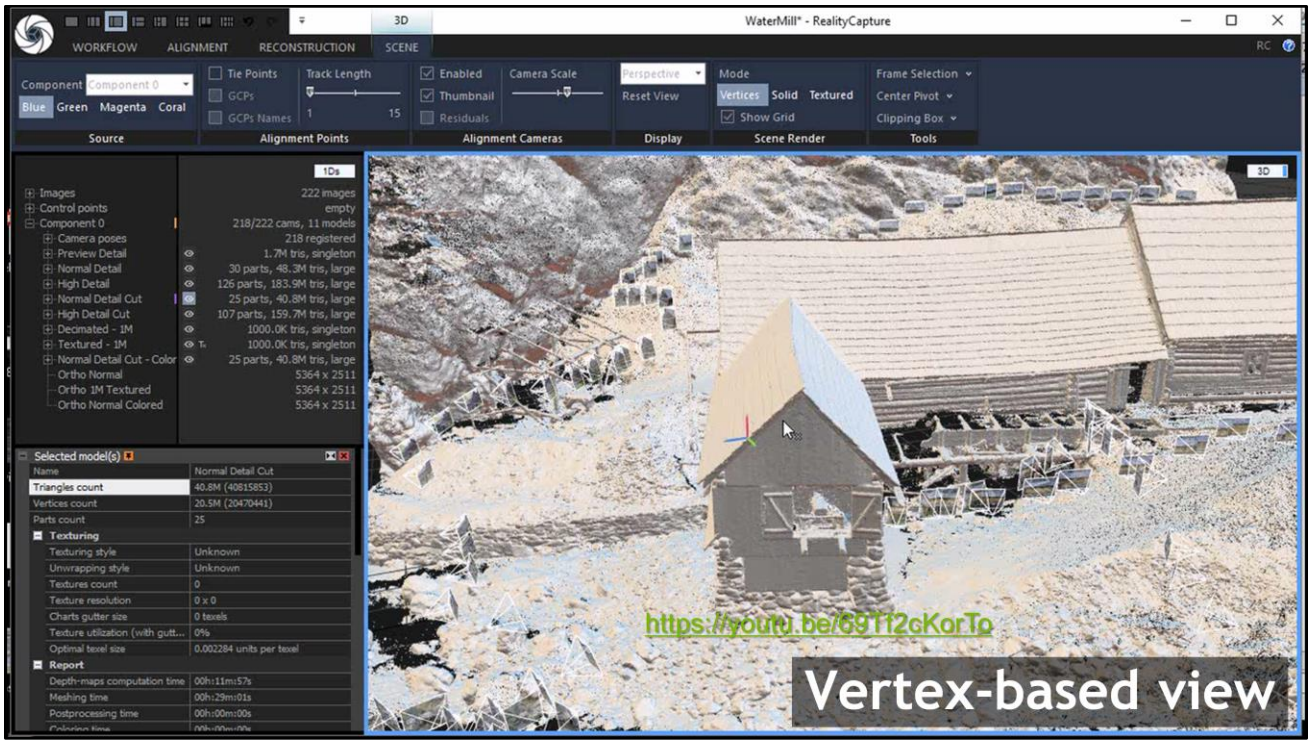
When you select a model,
then a panel with all details of the model is displayed.

This model has 48 million triangles
and has been computed in 40 minutes on normal quality.

If you need the highest possible detail,
then use high quality.

There is usually 4 times more detail
at a cost of 4 times longer processing time.

It was 3 hours for 183 million triangles in this case on high quality.



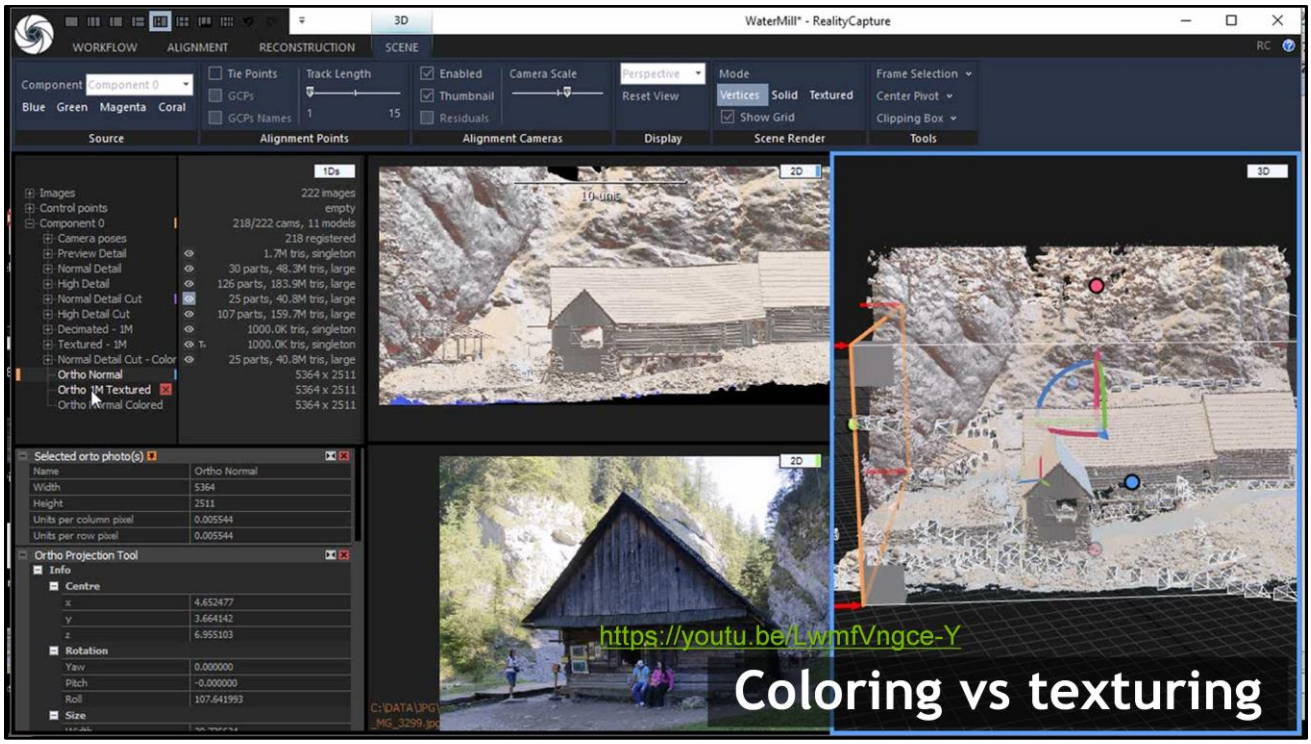
https://youtu.be/69Tf2cKorTo

In the 3D view you can see just vertices of the model.

But if you make an orthographic projection,
then it is computed from the full mesh
that is stored inside.

For example here.

This is the orthographic projection.



https://youtu.be/LwmfVngce-Y

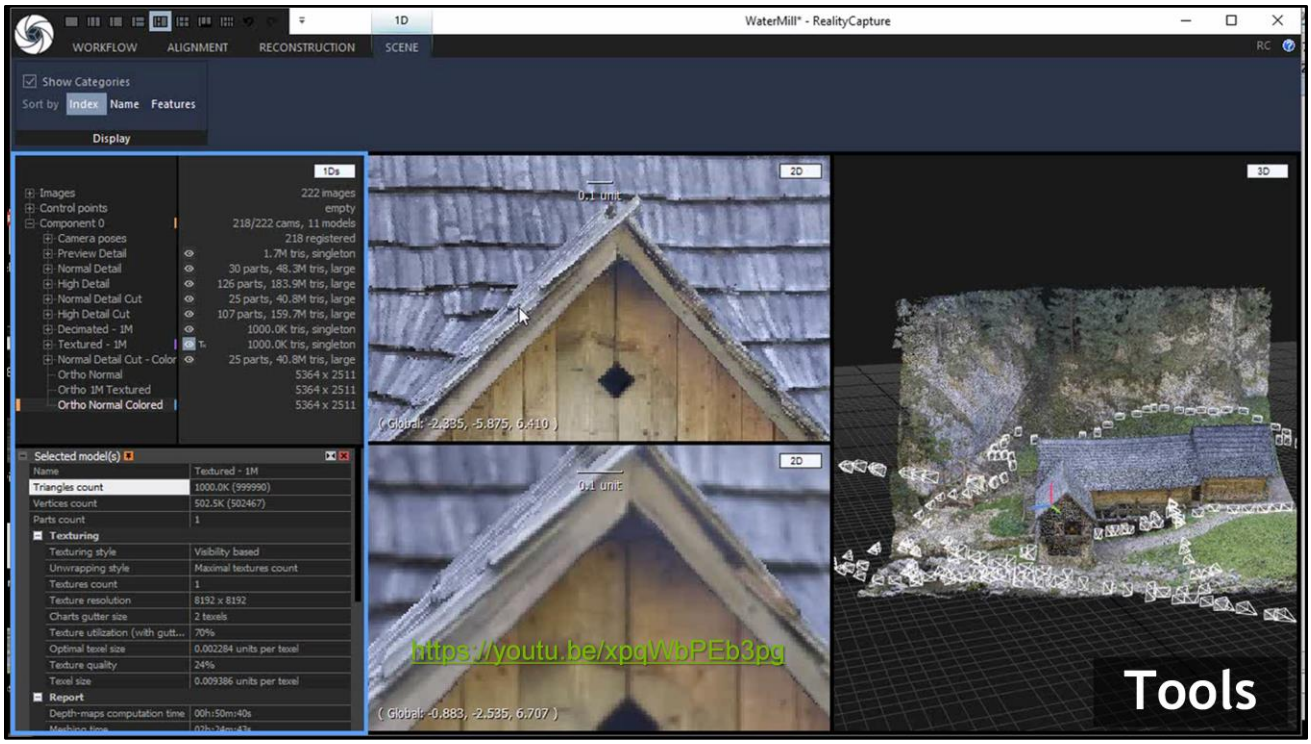
You can do whatever you want with the model after it is computed.
You can compute colors of its vertices
and create orthoprojections
which is important for making a project documentation.

But you, as game developers,
will be interested in simplification and texturing.

Or simplification, export, re-topologization, import, and texturing.

This is,
for example,
the difference between coloring on a full model
and texturing on a simplified one.

The top is texturing.
and
The bottom is coloring.



https://youtu.be/xpqWbPEb3pg

There are many tools inside the application for different tasks.

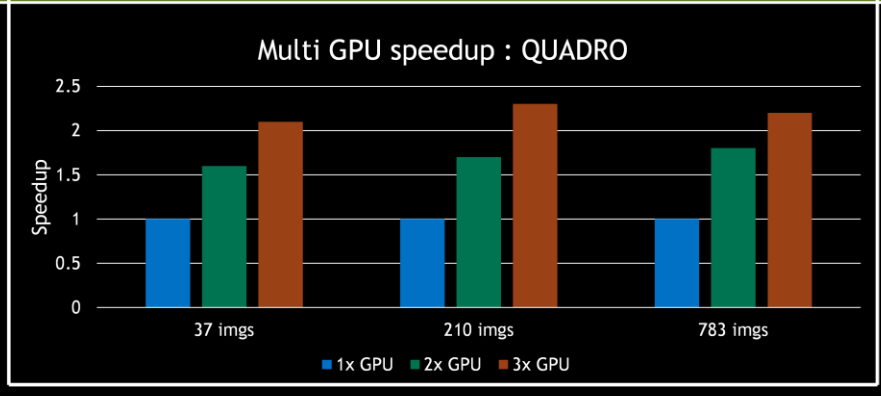
For example,
this is a simplification tool
that I have used to simplify the normal model
to one million triangles.

Then I used the texturing button to compute the texture for it.

And you can see the texturing parameters in the model panel too.

RealityCapture pipeline : speed

Alignment 10% CPU	Model computation 80% NVIDIA GPU (CUDA) + CPU	Texturing 10% CPU



Let me talk a little bit about speed of different parts.

As you can see on the slide the most time consuming part is model computation

Which usually takes 80% of the whole processing time.

Alignment is usually very fast and takes just 10% of the whole processing time.

Texturing takes usually the same time as alignment but it depends on the number of texels.

Therefore if you want to speedup your computation then
it is worth to speedup the model computation part.

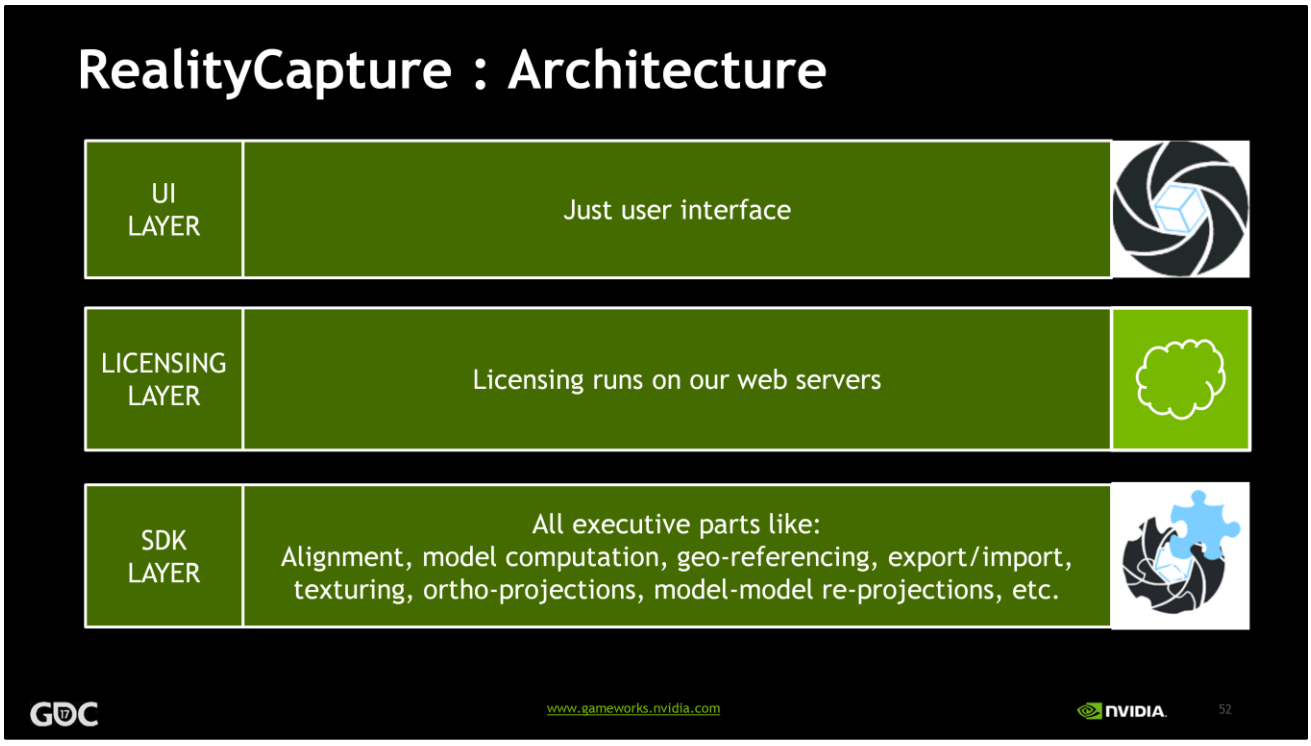
You can do it by using multiple GPUs.

Here you can see how it scales with multiple GPUs on different datasets.

There is a bottle-neck in host to device data transfer which can be partially solved by updating our algorithms.

We work on that.

But still,
you can save a lot of time by using more GPUs.



This is basic architecture of our solution.

The bottom layer is our SDK.

The SDK contains all executive parts like:

alignment,
model computation,
geo-referencing,
export/import,
texturing,
ortho-projections,
model-model re-projections, etc.

The licensing layer runs on our web servers
and unlocks the SDK.

Finally,
The user interface layer
is the top layer and it is just a thin-client.

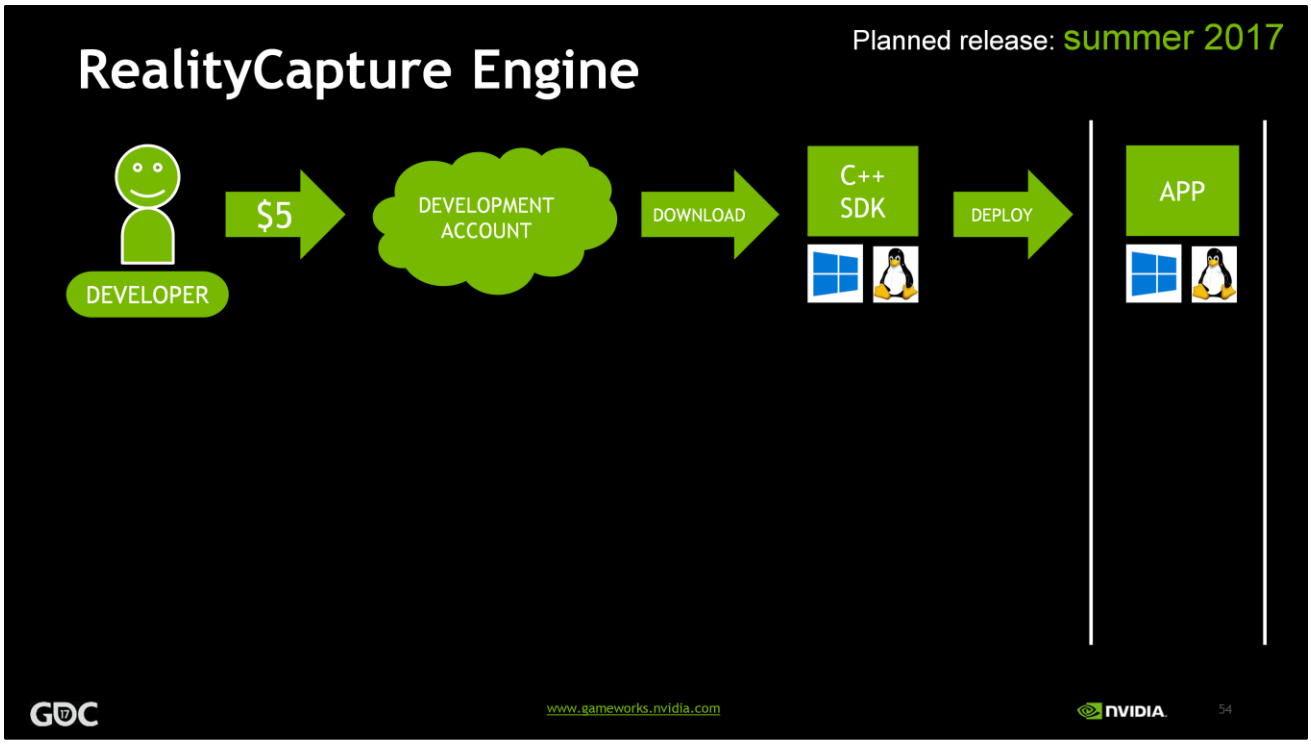
In another words
having our SDK
you can create exactly the same application as we have.

I am thrilled that I can announce here that we will release RealityCapture SDK this summer.

We will call it RealityCapture Engine. It will be accessible for everybody.

Deployment will be for free and you will pay as you go.

Let me explain it in more details.



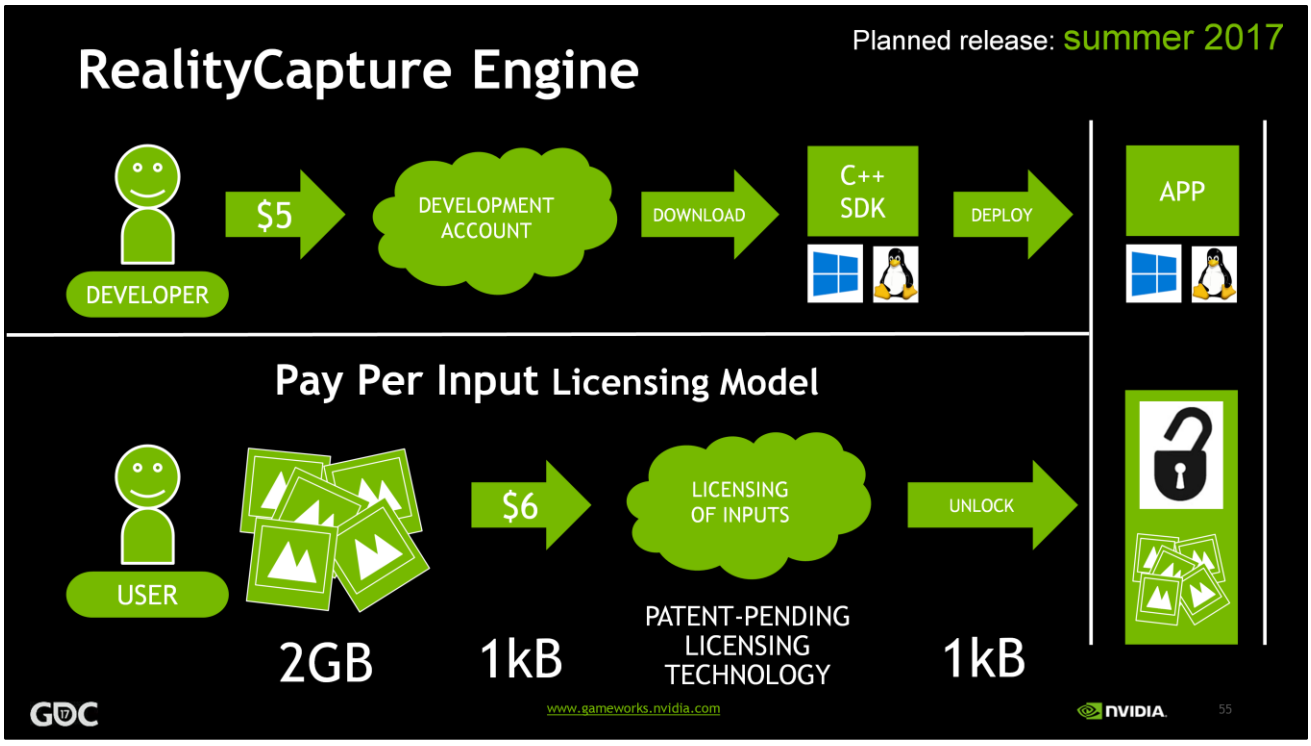
This is a developer

He creates a development account at our webpage, for a small price.

Then he can download and incorporate our SDK to his project.

He can use as many computers as he wants for free.

Then once he is ready he distributes his application to users. The deployment is completely for free.



This is the user. He wants to reconstruct these images. Let's say that they contain 2 Gigabytes of disk space.

All the user needs to do to unlock the application for his input images is that he needs to buy a license for each image.

This is our new licensing model which we call "Pay Per Input".

The user however, does not need to upload all images. He just needs to upload a fingerprint of each image. That, in this case, will not be more than 1kB, for all images together. This way is also privacy of the image data secured.

Then the server sends unlock data to the deployed application and all features will be unlocked for the input images.

This way you are not bounded to a computer so you can process the same data on many computers but pay just once!

RealityCapture Engine

You can use it to

- integrate photogrammetry to your in-house workflow
- create plugins for other tools
- automate your processes
- create cloud-based solutions
- create a one-button solution

It is

- accessible for everybody

I know that you already have thousands of ideas on how to use this engine.

For example :

integrate photogrammetry to your in-house workflow
create plugins for other tools
automate your processes
create cloud-based solutions
create a one-button solution

And so on and so on.

Well …
And it is accessible for everybody

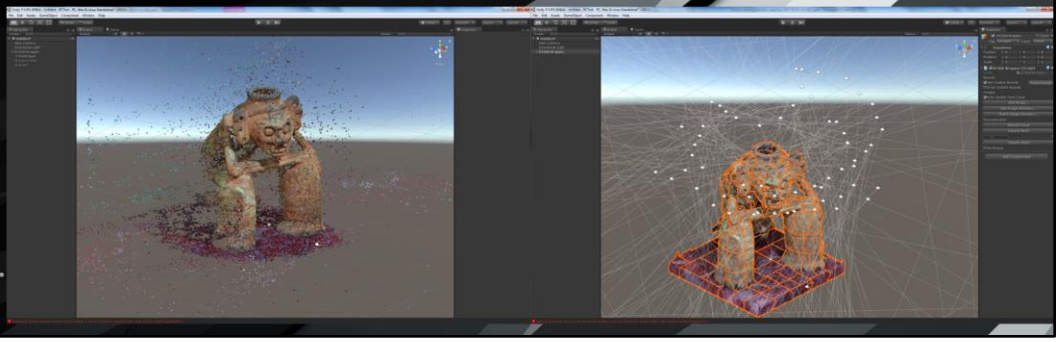
If you would like to try our SDK prior to its official release.

Then please don’t forget to take this flyer.

Thank you for your attention.

Photogrammetry Pipeline Programming

Lars M. Bishop, 3/1/2016

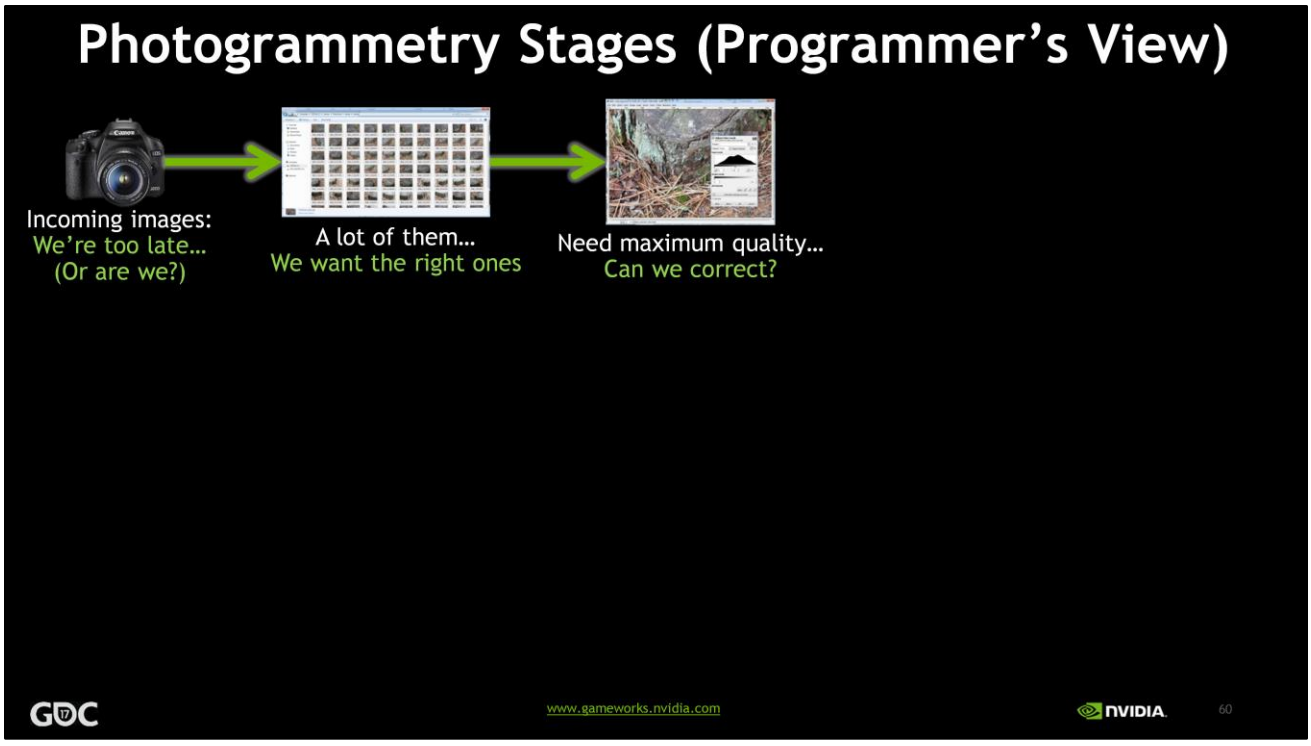


NVIDIA

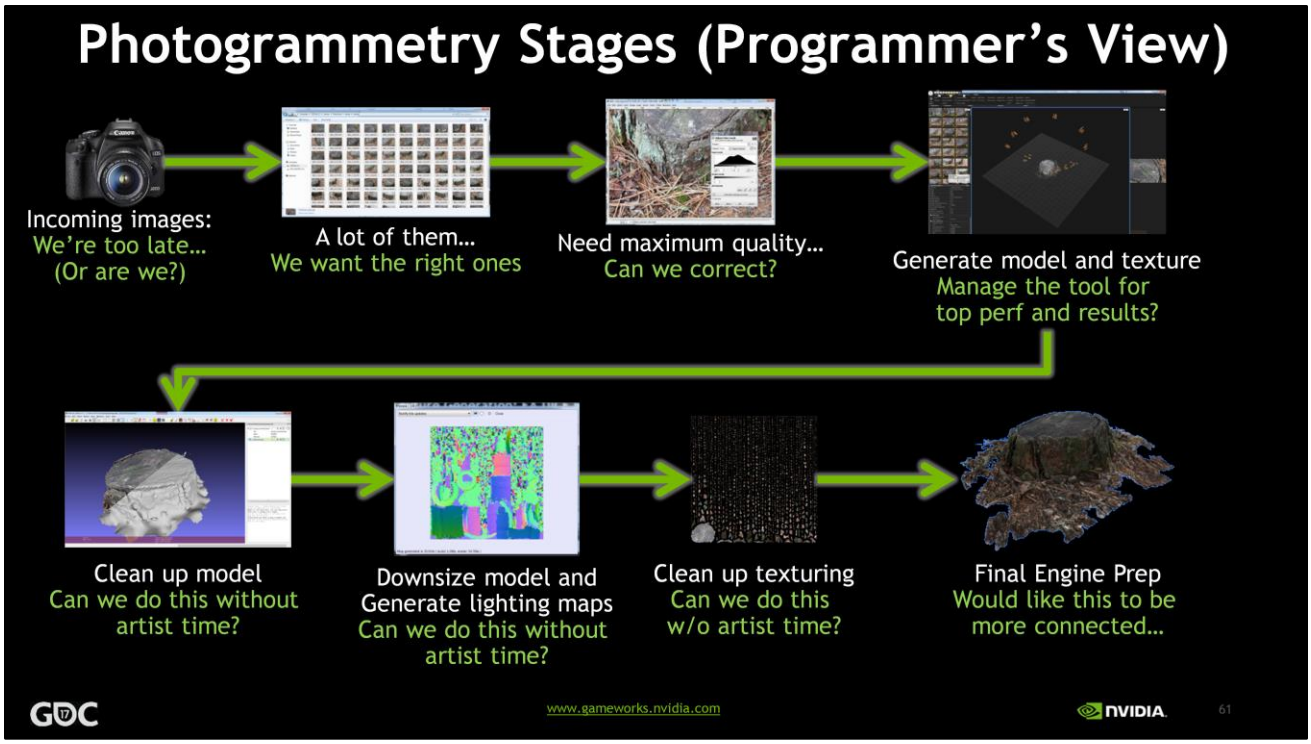
Programming the Pipeline

- A programmer's step through the looking glass
- Very open topic - one most developers are actively discussing/innovating
- A few topics/experiments we've been working on:
  - Easing the use of PG in game pipelines
  - Automation and integration
  - On-site photogrammetry feedback

To close out the session, I'll discuss topics on the programmer's role in the photogrammetry pipeline. This is by no means comprehensive, but is designed to introduce some of the existing integration points, some of the common workflows, and to demonstrate a few experiments we have created to show the power of integrating photogrammetry tools directly into existing game tools. I'll finish with a quick discussion of some other possible applications of pipeline integration.



Referring back to the earlier diagram, let's look at things from a tool programmer's point of view, or at least mine when I began looking into it. First, we get a lot of images, and I figured that there was little I could do to affect the actual capture of those images. However, as I'll show later, that isn't quite true. But the first bit where the programmer is very likely to show up is preparing the images. This can involve <CLICK> triage, automatically skipping the images that are clearly not usable, and it can involve <CLICK> pre-processing, ensuring that the good images have the best chance of generating accurate models and textures.



The photogrammetry package itself is likely self-contained, but with lots of models to be reconstructed, it's also possible that programmed automation can ensure that the assets are reconstructed consistently and at top performance. The next phase <CLICK> is model and <CLICK> texture cleanup, and if possible, it would be great if that could be done wasting as little artist time as possible. Finally, perhaps it would be useful <CLICK> to be able to run as much of this pipeline as possible using the actual game engine or its editor. So let's dive in.

Automation Shall Set you Free?

- Tools have wildly varying automation methods
- Benefits/issues with each (well, most)
- Keep this in mind when choosing tools!

Class	GUI-based	
Automation	None (GUI only)	GUI Record / Playback
Target User	Any Tool User	Any Tool User
Ease of Use	N/A	Easy
Interop	None	None
Example	Too many…	MeshLab*

* MeshLab can be run from the command line with an MLX script, but generally, the script is generated in the GUI. The files are editable, but that is not the main use case in the docs

In order to automate this process, as a programmer, I need a way in. The photogrammetry tool pipeline that Chris laid out involves quite a few tools, and a lot of choices for each tool. One of the important aspects of this choice is whether or not a tool allows automation, and if so, how. In the chart on this slide, I break the levels of integration into three basic categories; First, those that require using the GUI itself, either because they don't support ANY automation, or because they support a macro-like "record and play back" method. While these can be set up by anyone who knows the tool, obviously the reusability of each automation is limited.

Automation Shall Set you Free?

- Tools have wildly varying automation methods
- Benefits/issues with each (well, most)
- Keep this in mind when choosing tools!

Class	GUI-based		Script-based	
Automation	None (GUI only)	GUI Record / Playback	Command-Line	Scripting
Target User	Any Tool User	Any Tool User	Most Tool Users	Programmer / Tech Artist
Ease of Use	N/A	Easy	Easy	Medium
Interop	None	None	None	None
Example	Too many…	MeshLab	Reality Cap old CLI	MeshLab MLX (XML-based)

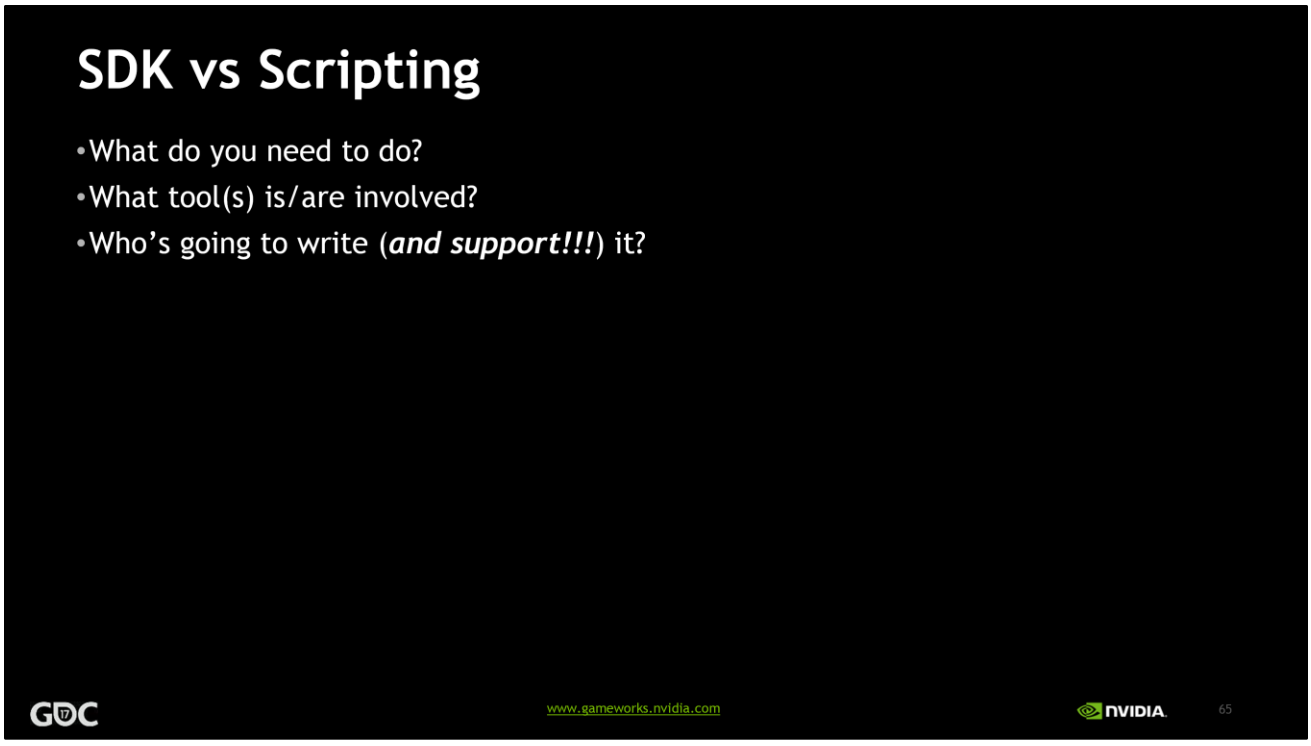
The next level is via some form of tool-specific scripting. This can be as simple as a set of command-line options that let the user specify the operations, sources and options for each step, and can be as complex as a proprietary language for setting those same things. The command-line system has the drawbacks of being pretty ungainly, and likely involving writing to file between each step. The scripting method is limited because it is internal to that single tool. It doesn't help connect other tools together in a single script.

Automation Shall Set you Free?

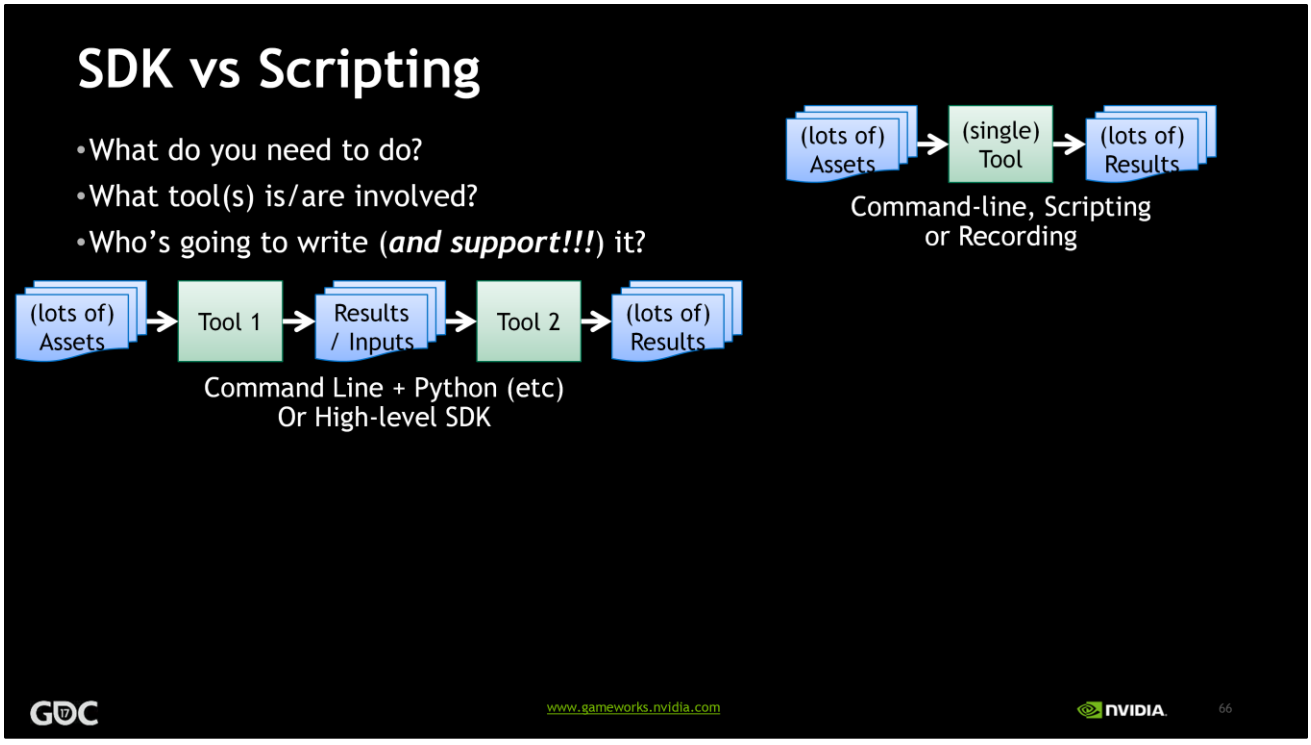
- Tools have wildly varying automation methods
- Benefits/issues with each (well, most)
- Keep this in mind when choosing tools!

Class	GUI-based		Script-based		Code-based	
Automation	None (GUI only)	GUI Record / Playback	Command-Line	Scripting	High-level SDK	Native code SDK
Target User	Any Tool User	Any Tool User	Most Tool Users	Programmer / Tech Artist	Programmer	(Experienced) Programmer
Ease of Use	N/A	Easy	Easy	Medium	Complex	Complex
Interop	None	None	None	None	Good	Best
Example	Too many…	MeshLab	Reality Cap old CLI	MeshLab	AgiSoft Python	Reality Cap SDK C++

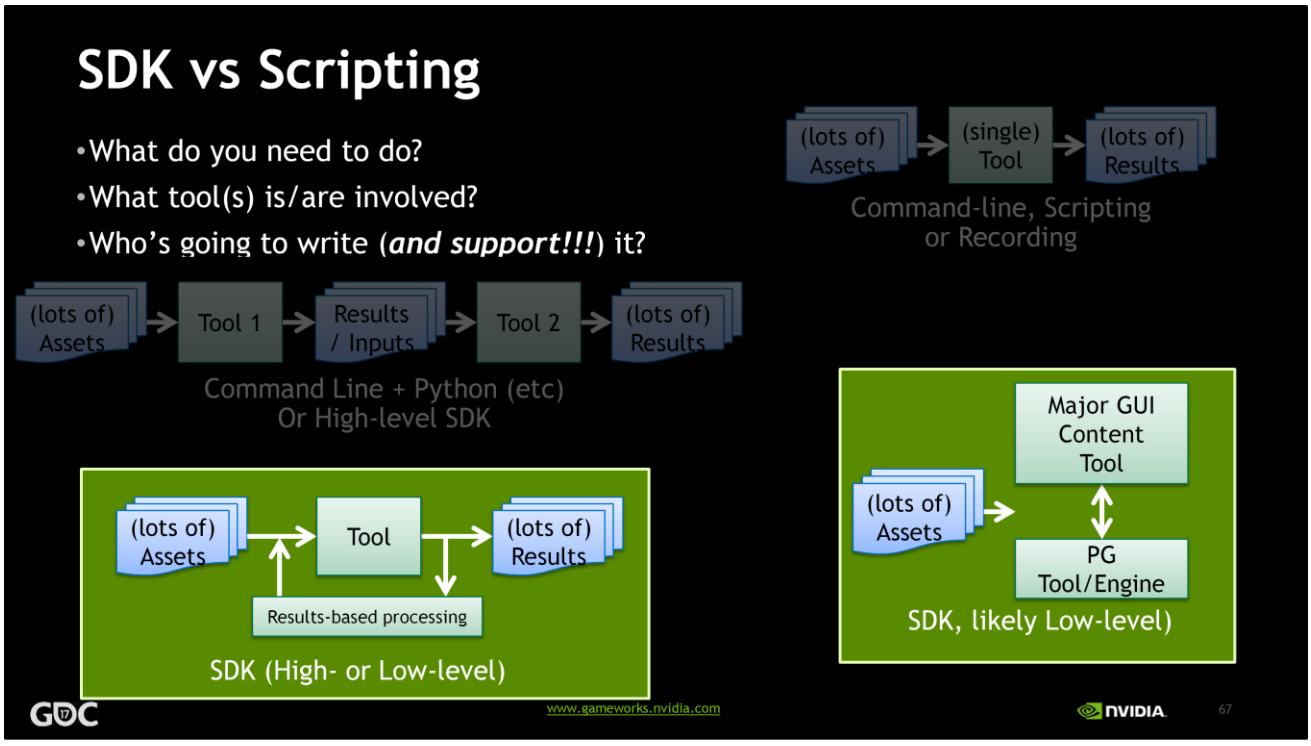
Finally come the Code-based SDKs, either as a high-level API in a language like Python, or as a low-level API representing the tool's data and pipeline via C++ or C#. We'll spend the most time talking about this lattermost case.



Lemme stop and clarify that I'm not saying a low-level SDK is the best method across the board. Like any other tool, there is a balance to be made based on requirements. For example, how complex of an automation do you need? "resize all images" doesn't require a C-sharp integration. Also, what tools are you connecting together? What do they share in common in terms of automation? And most importantly, who is going to support this tool? A C++ integration is great, but if you have to find a C++ programmer for every fix you need, you'll want to consider how often that's gonna happen. Let's consider four common architectural cases.



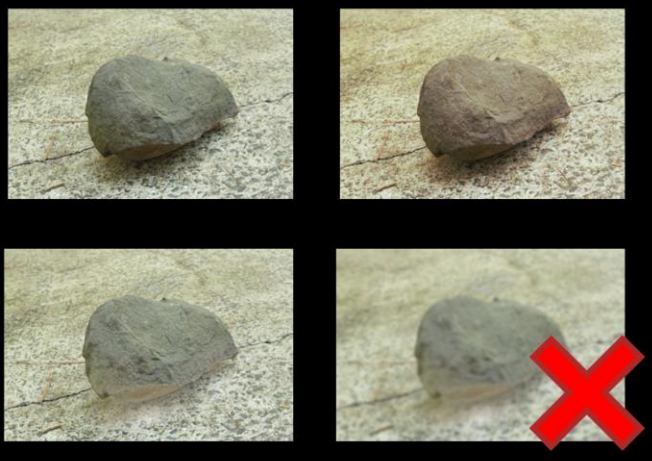
First <CLICK>, the case of applying the same processing on a ton of images in a one-to-one pipeline. Almost any method is fine for that. In a sense <CLICK>, the most easily accessible system possible, like command-line is possibly the best. Next <CLICK>, a similar many-in,many-out pipeline but with multiple tools involved. In this case, the main consideration is one method that can control BOTH tools. So <CLICK>, either command line or a high-level SDK for both tools (assuming they use the same language) are likely the easiest.



A major step forward is feedback, or results-based processing <CLICK>, where the results of a step in a tool control what happens next, or whether we redo a step with different settings. In this case <CLICK>, some form of SDK that gives programming language access to the mid-pipeline data is going to be best. Finally <CLICK>, the complex case of "plugging" a tool into a major GUI like a 3D modeler or game engine editor. The best option here for deepest integration is probably <CLICK> a low-level SDK, one that shares its language with the GUI tool itself and can be treated as an "plugin". My demos today will be examples <CLICK> of these low-level SDK cases.

Image Processing

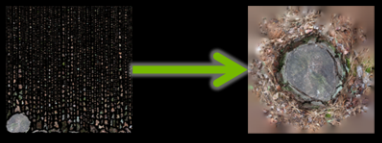
- One of the first steps to automate:
  - Huge number of images to deal with
  - Lots of existing processing frameworks
  - Many operations are “turn the crank”
- Examples:
  - Color matching
  - 2D-only de-lighting/shadow reduction
  - Image pre-selection for PG (sharpness, etc)



Source photos are an obvious place to automate, because most image operations are simple and because of the sheer volume of images. Also, many image processing frameworks already exist for batch work. Before preprocessing, there’s more you could do; the highest performance images are the ones you skip. So if possible, trying to drop bad images before they make it into the pipeline can be a win. Blurry images, images with a ton of blown-out pixels, and other issues should be dropped; if this photo culling can be done automatically, all the better.

Geometry (Post-)Processing

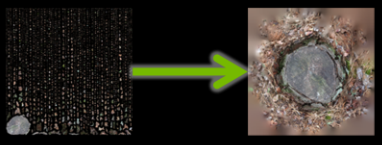
- Harder to automate (completely)
- Geometry tools can take a long time to run
  - Scripting and “farming” could help
  - Geometry cleaning can be heavily automated (topology, etc)
- UV remapping is perhaps the most pivotal
  - But the hardest to completely automate?
  - PG packages reimport external UV-mappings to be re-textured from those cleaned Uvs
  - What if it all happened in the modeling tool…



On the other end of the pipeline, is the post-processing of the reconstruction results. Right now, much of that can be hard to fully automate. Given the current quality of photogrammetry U Vs <CLICK> compared to the desired unwrapping in games <CLICK> U V remapping is the most requested. Currently, the major photogrammetry tools already support some form of this by allowing a mesh to have U Vs generated in an outside tool and then re-imported and the texturing stage done again, projecting into the remapped, cleaner U Vs.

Geometry (Post-)Processing

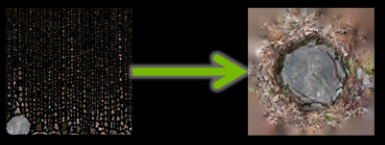
- Harder to automate (completely)
- Geometry tools can take a long time to run
  - Scripting and "farming" could help
  - Geometry cleaning can be heavily automated (topology, etc)
- UV remapping is perhaps the most pivotal
  - But the hardest to completely automate?
  - PG packages reimport external UV-mappings to be re-textured from those cleaned Uvs
  - What if it all happened in the modeling tool…

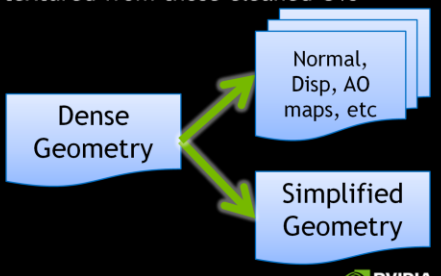


Looking ahead, the ability to use a photogrammetry tool's SDK to automate this step is a very high priority. And it could be done one of several ways; one is to integrate the entire photogrammetry process into a larger tool that includes the UV remapping, like Maya. The other architecture, which we've not touched on here is for the photogrammetry GUI tool to expand ITS SDK APIs to include plugging into the reconstruction pipeline itself, "trapping out" to a plugin to generate the U Vs.

Geometry (Post-)Processing

- Harder to automate (completely)
- Geometry tools can take a long time to run
  - Scripting and “farming” could help
  - Geometry cleaning can be heavily automated (topology, etc)
- UV remapping is perhaps the most pivotal
  - But the hardest to completely automate?
  - PG packages reimport external UV-mappings to be re-textured from those cleaned Uvs
  - What if it all happened in the modeling tool…
- Secondary texture generation

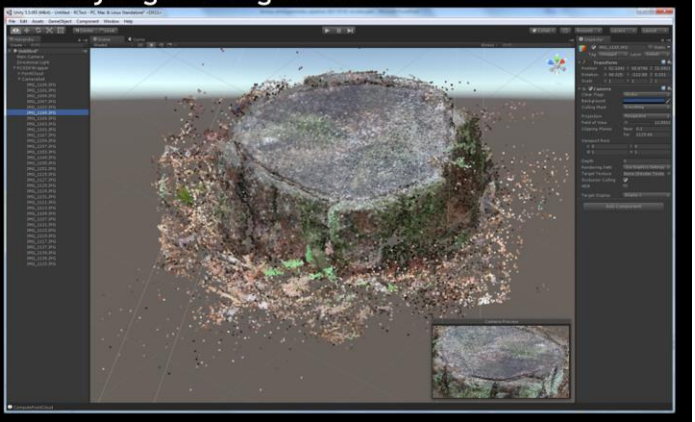




One of the post-processing stages that Chris mentioned is more straightforward to automate. This is the move from <CLICK> dense geometry to <CLICK> sparser geometry along with detail textures like normal maps. Since this stage comes after U V cleanup, there’s no re-import into the photogrammetry tool, and it should be possible within one or two tools; one for generating the low-poly mesh, and one for generating the maps. For example, it could be done via MeshLab and xNormal for the two stages from the command line.

Unity/RC Integration

- An example of integrating a PG tool into a major game engine tool
  - RC SDK (C++) wrapped with C# for Unity
  - Threaded back-end
- Features:
  - Load images or directory
  - Images mapped as Unity cameras
  - Sparse cloud
  - Volume cropping
  - Colored model
  - Triangle simplification
  - Texturing
  - UI functional while computing PG results



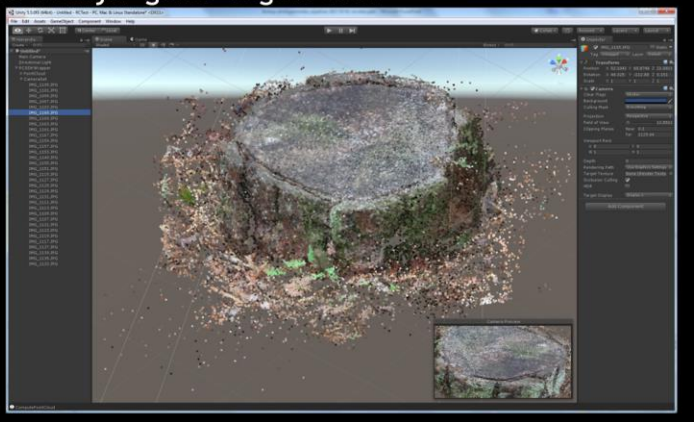
- Caveat - might not be a production use case (Maya/3DS Max might be an option)
  - But it is a great example of integration

GDC www.gameworks.nvidia.com NVIDIA 72

As I mentioned previously, with a low-level SDK for the reconstruction tool, it becomes possible to integrate that tool into a GUI-based game engine editor. This is exactly what I did with Reality Capture's C++ SDK and Unity's C-sharp MonoBehaviors. I created a new MonoBehavior that calls down to Reality Capture to provide Unity's editor with built-in photogrammetry reconstruction. The first step was to build a small C++ interop DLL that did the basic steps of reconstruction using Reality Capture's APIs, and then presented them as a C-sharp-compatible, limited set of APIs. Then, in Unity, I created a C-sharp object that represented this photogrammetry object, along with an editor UI.

Unity/RC Integration

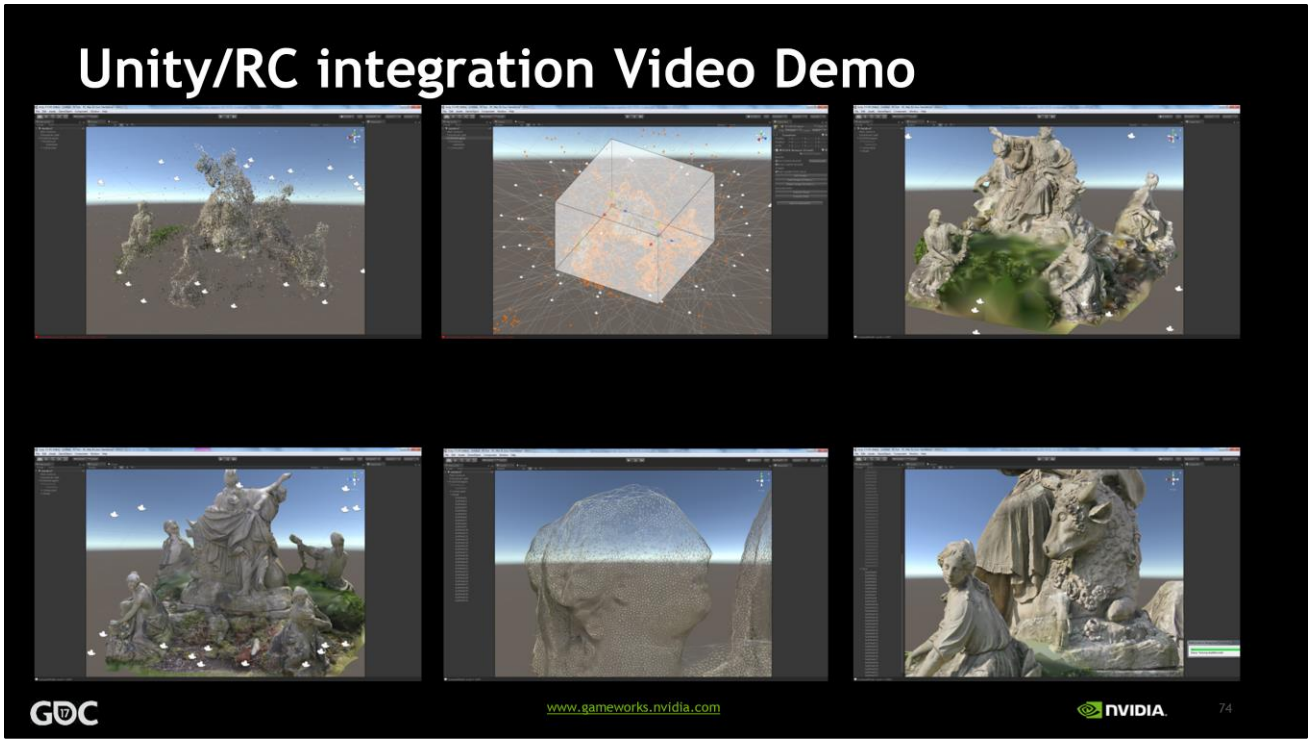
- An example of integrating a PG tool into a major game engine tool
  - RC SDK (C++) wrapped with C# for Unity
  - Threaded back-end
- Features:
  - Load images or directory
  - Images mapped as Unity cameras
  - Sparse cloud
  - Volume cropping
  - Colored model
  - Triangle simplification
  - Texturing
  - UI functional while computing PG results



- Caveat - might not be a production use case (Maya/3DS Max might be an option)
  - But it is a great example of integration

GDC www.gameworks.nvidia.com NVIDIA 73

To ensure that Unity's UI is blocked as little as possible, the C-sharp code calls down to the C++ DLL only in a secondary worker thread. Once the Reality Capture SDK has generated a result in the background thread, it adds a function to Unity's update queue, so that the main Unity thread is called and we can safely make Unity calls to convert the photogrammetry results into Unity geometry and textures.



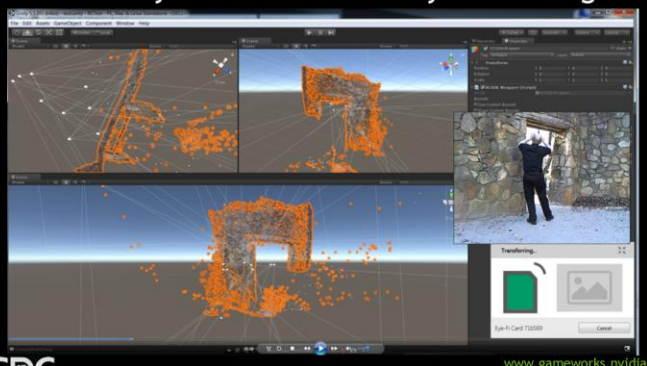
Here, presented as video, is the viewer. We're also giving live demos of the viewer in NVIDIA's booth.

https://www.youtube.com/watch?v=6SD3hZgQpnY

You can see each step here; we select the image directory. All images in that directory are added to the reconstruction, and the system automatically kicks off the computation of the tie points and camera locations. Then, we follow Meekal's recommendation and set a nice, tight bounding volume for the reconstruction. Then we request model reconstruction. We've edited the video, but as you can see, the actual timing is quick, and Unity is usable the entire time. We get back a dense, vertex-colored model. We review it, and switch into wireframe to see the density. Next, we adjust the slider to choose the level of simplification we want for the geometry. Since we will be texturing this model in a minute, we can drop the geometric density. Finally, the system textures the model in the background. Once this is done, we have a fully textured Unity model.

Looking forward: Live pre-viz

- On-site photo sessions are expensive
- Nothing is more expensive than the photo(s) you DIDN'T take
- But overshooting everything is also expensive
  - Especially down the pipeline…
- What if you could see "how you're doing"… As you're shooting?



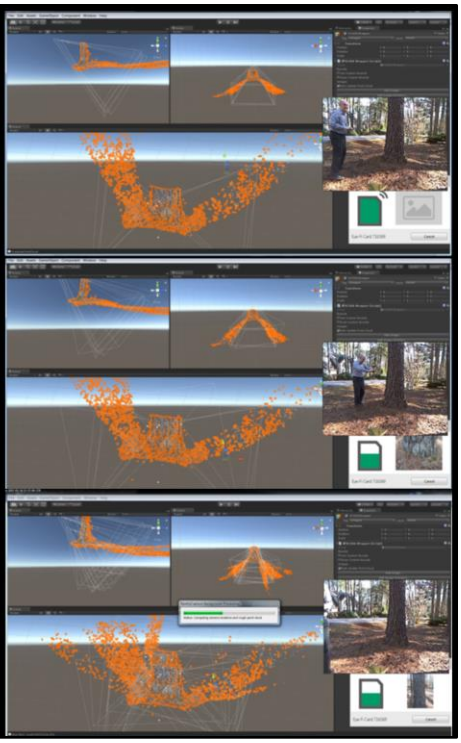
GDC www.gameworks.nvidia.com NVIDIA 76

One of the things I alluded to in my original slide on programming the pipeline was the idea that while the programmer is likely NOT on-site for capture, their CODE could be. We've mentioned several times in the session that ensuring you have the right images is important. Well, in a sense, the best way to ensure that you have the right images is to run at least part of the actual photogrammetry pipeline on site. This was key to our next example…

Almost-Live Pre-viz

- Sparse point cloud is cheap even with many cameras
- Modern D-SLRs often have wifi support
  - And all drones do…
- Using only a laptop and camera
  - Visualize the sparse cloud and camera locations
  - Per photo as the photo is taken

Leave the shoot knowing what you do (and don't) have!



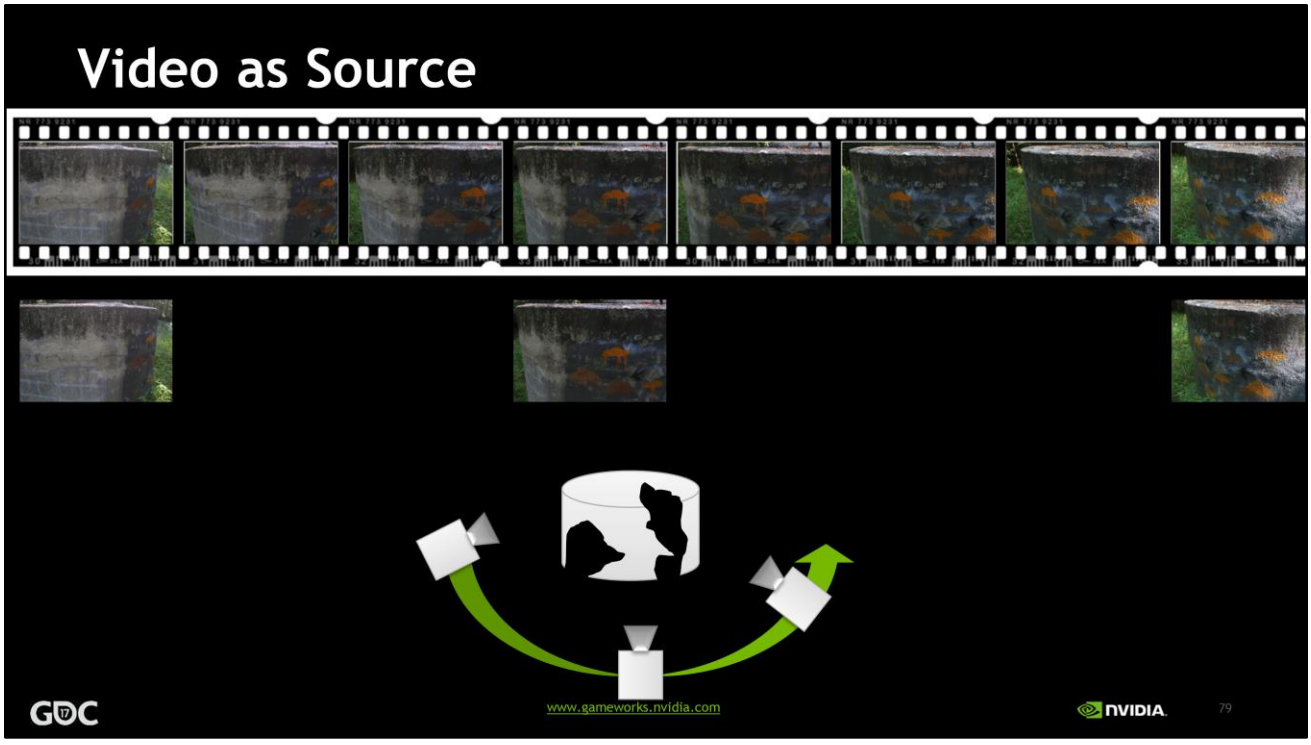
GDC

www.gameworks.nvidia.com

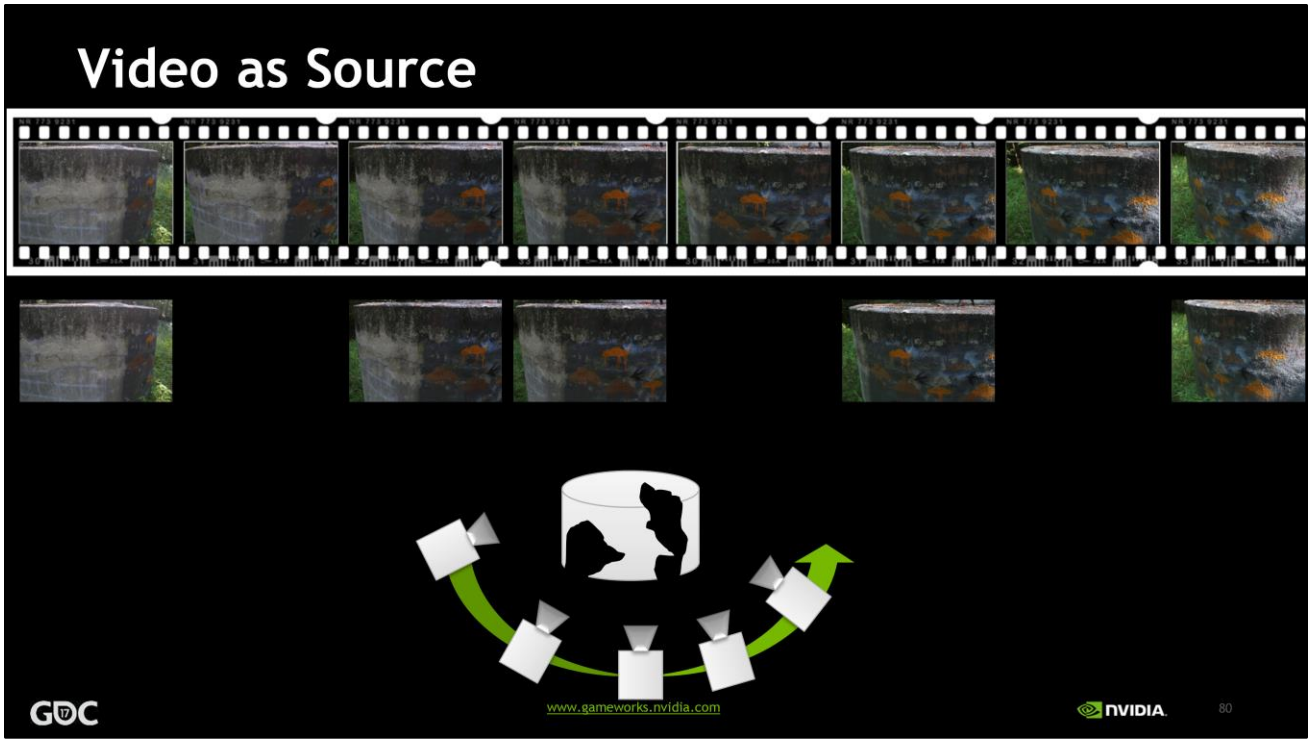
Since the tie points are cheap to compute, even on a laptop, perhaps we update them as new images come in. All modern digital cameras can support wifi one way or another. And basically all drones have wifi cameras. Using only a laptop and a connected camera, we can compute the tie points and registration as each new photo is taken. We can leave the location knowing what we've captured. We want this to be 100 percent automated so the artists will feel comfortable using it on site.

https://www.youtube.com/watch?v=HofnsfF6Qgw

Here is our experiment; extending the viewer I showed earlier, we add a new option: WATCHING a given directory. I connected my camera to my laptop's wifi, and set it to transfer new photos into a selected directory. Now, I just shoot photos, and you'll notice the reconstruction updating as I do. This was all shot live on my laptop. The inset window is the live view of me shooting – I attached a webcam to the laptop. When I think I'm done, I COULD press the model button and even see what it would look like. But mainly, I can just look over at the laptop while shooting, without touching it and see how I'm doing. Or, if I had coworkers there, they could watch the screen and direct me.



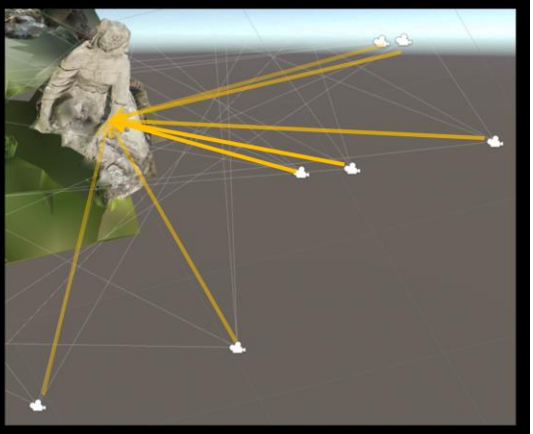
Some tools are starting to allow videos as sources for reconstruction, others support video-related features like rolling shutter correction. However, even a photogrammetry tool without video support can work with a video codec library to do some very interesting things via an SDK. Specifically, the iterative addition of photos we used in the live pre viz can be very powerful. First, we use our video decoder SDK to automatically decode key frames <CLICK>, either by regularly sampling in time, or perhaps decoding only the I-frames. We run these images through the photogrammetry tool's SDK <CLICK> and analyze the location of the cameras. We'll get a base model or cloud from that <CLICK>.



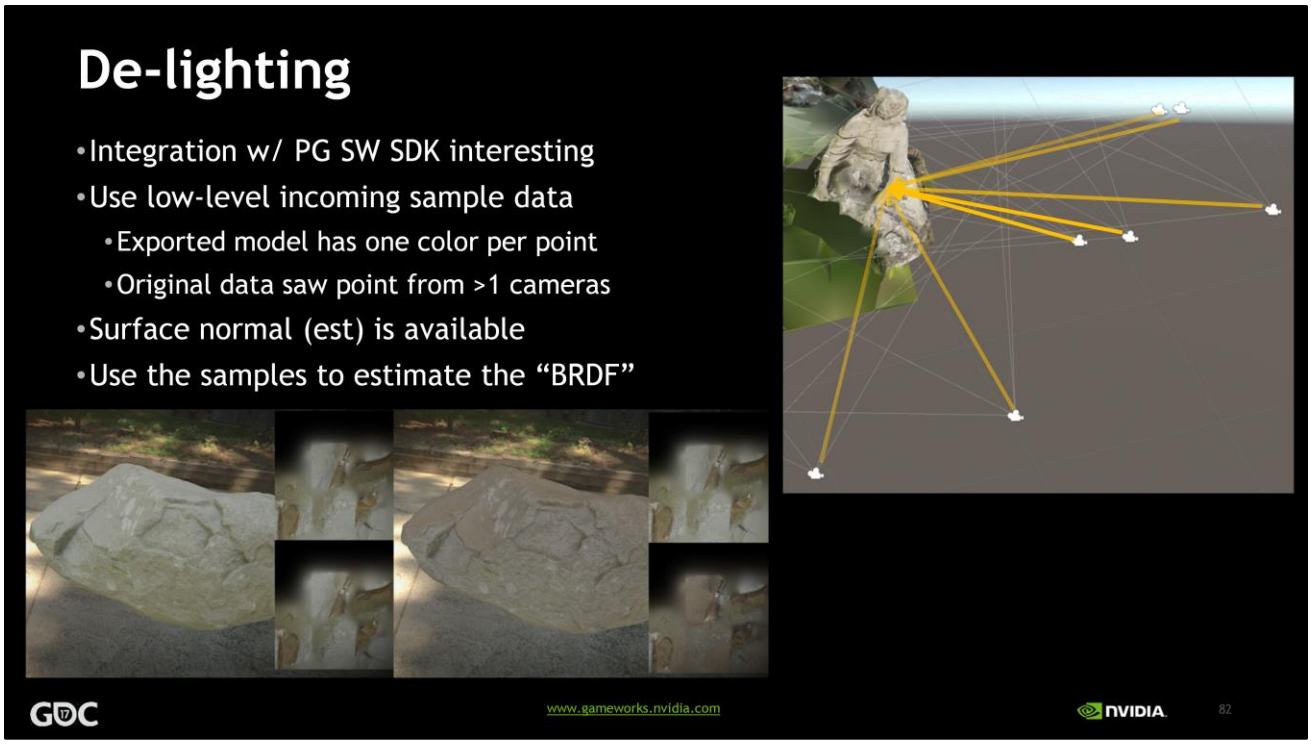
We likely know the frames of video represent samples of a smooth sweep of the camera, so in areas where we see large gaps between camera locations, we grab MORE video frames in the gap in the timeline <CLICK>, which will fill in a camera location <CLICK>, and fill the point cloud <CLICK>. Our code can continue adding subdivision frames <CLICK>, computing the camera locations <CLICK> and updating the cloud <CLICK> in an automated fashion, quickly, until we have good coverage. At which point it can spend the time generating the detailed model from just the right set of images. All of this is possible today with a system like Reality Capture's SDK.

De-lighting

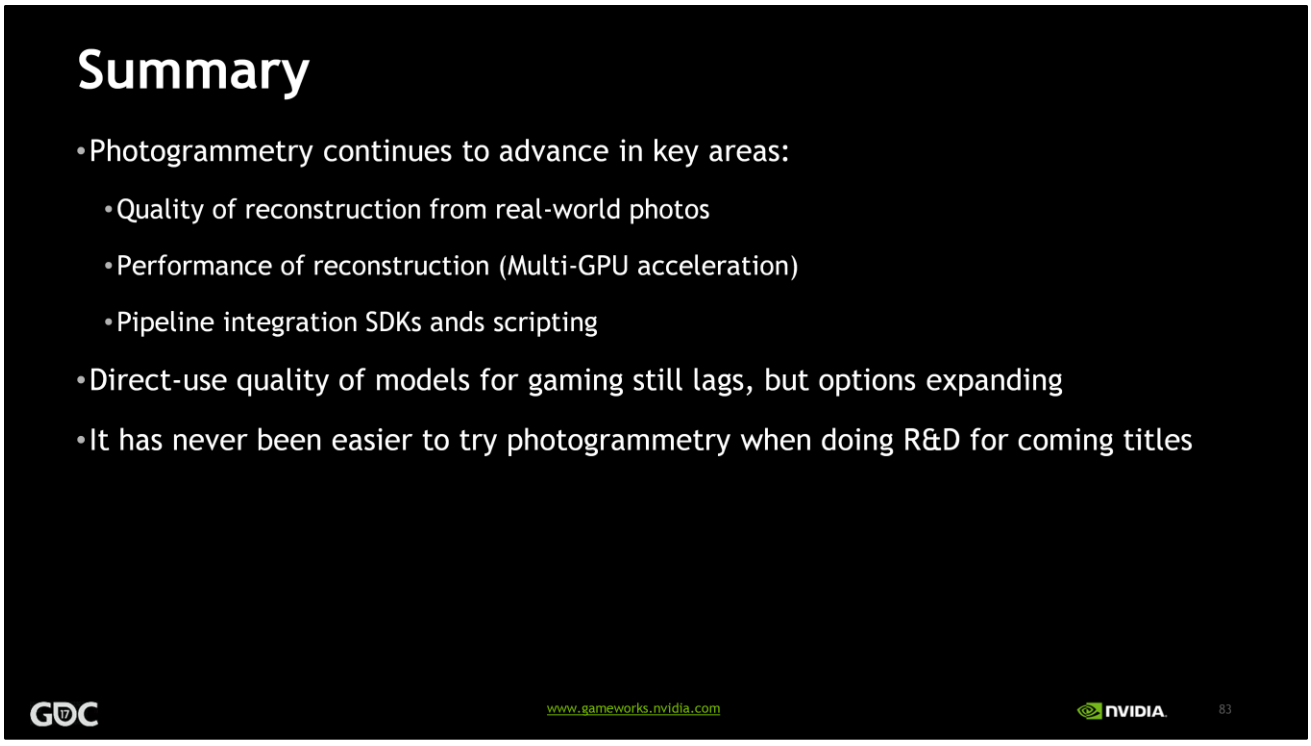
- Integration w/ PG SW SDK interesting
- Use low-level incoming sample data
  - Exported model has one color per point
  - Original data saw point from >1 cameras
- Surface normal (est) is available
- Use the samples to estimate the “BRDF”



Another very hot area of research is de-lighting. We want to be able to re-light our captured objects in different orientations, for different seasons and times of day, and possibly even in a completely different environment. This means removing the incident lighting, or ”extracting the albedo” from the capture. Photogrammetry tools contain data used during the computation of the model and texture that they generally do not export. Specifically, by definition any point on the reconstructed model <CLICK> had to be seen from at least two <CLICK> and hopefully more <CLICK x 5> camera locations. In the more advanced tool SDKs, EACH point in the reconstruction is linked to ALL of the image points where it was seen.



Now, when the tool exports a model with color, it must compute a single color from these samples somehow. But in the SDK, the programmer has access to all of the views of that point, the ray to the camera that shot it, and the color seen by that photo. This can give a de-lighting algorithm far more information than just a single color and a normal; it can allow the de-lighting model the possibility of estimating the material or BRDF of the surface. There are a lot of complexities here, but the possibilities here are quite open and interesting.



Thanks for joining us today; we hope the session has given you an idea of some of trends in photogrammetry pipelines. Both photogrammetry tools AND integrations around them are improving at a high rate. The performance of reconstruction is also improving AT or ABOVE the rate of GPU hardware improvement, as photogrammetry tools optimize their use of CUDA and multi-GPU. The output of the photogrammetry tools are still not game-ready, but automation is allowing programmers to treat these tools as a starting point for their OWN innovation. Finally, as Meekal has discussed with respect to Reality Capture, it has never been easier to try out photogrammetry for yourself with a professional pipeline and the GPUs you already have in your PC.

What's Coming!

GPU Technology Conference (Q2 2017):

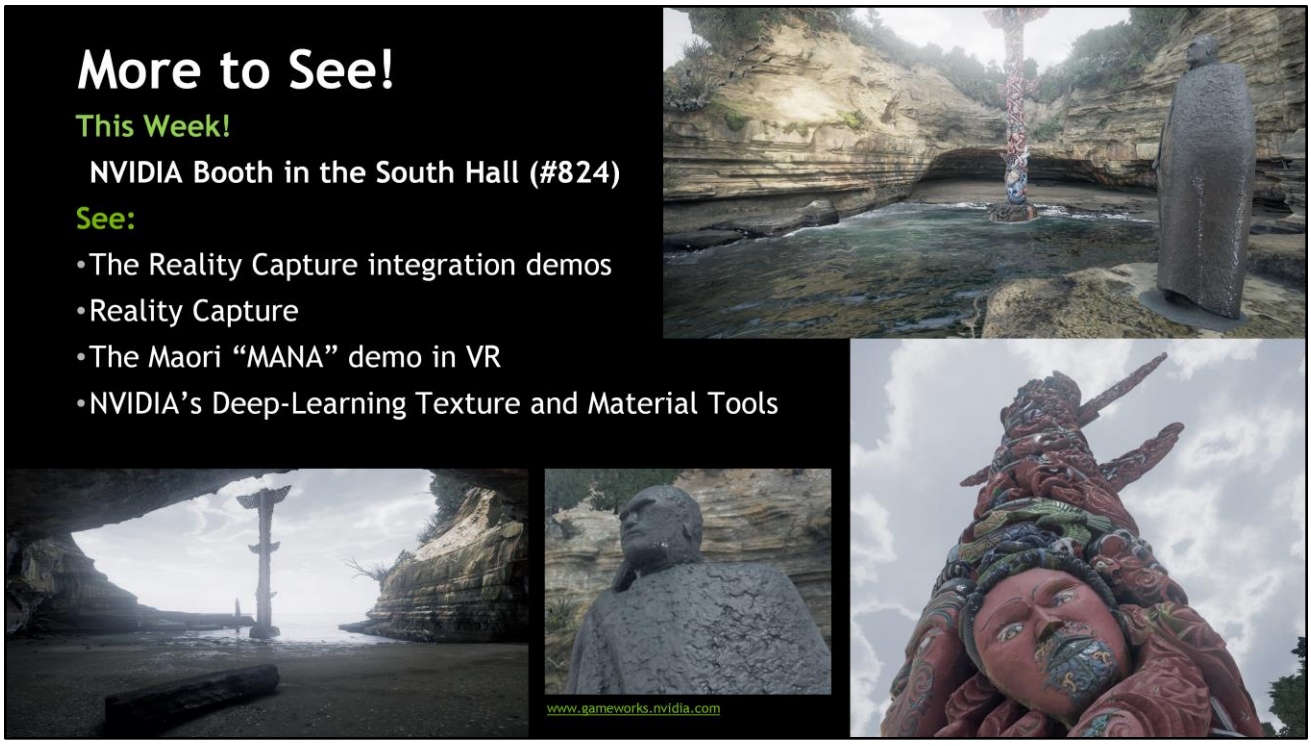
Reality Capture SDK Release

NVIDIA's Reality Capture SDK samples (including Unity viewer)

Q3 2017:

Reality Capture Engine and Licensing

The Reality Capture SDK we demoed here and in the booth will be released publicly in Q2, likely around NVIDIA's Graphics Technology Conference, as will some of the source examples from NVIDIA, including the Unity viewer. And Capturing Reality will be releasing their new Engine licensing model in Q3 of this year.



Please do stop by the NVIDIA booth on the show floor, number 824; we'll be showing the Reality Capture Unity viewer plugin and Reality Capture as well as NVIDIA's new Deep-learning-based texture and material tools all week in our Content Creation demo pod. We also have a VR experience of a large-scale Maori wood carving, set in a beautiful location, all captured and reconstructed using Reality Capture and rendered live in UE4. Thanks to my team presenters, Meekal and Chris, and let's open things up for questions.