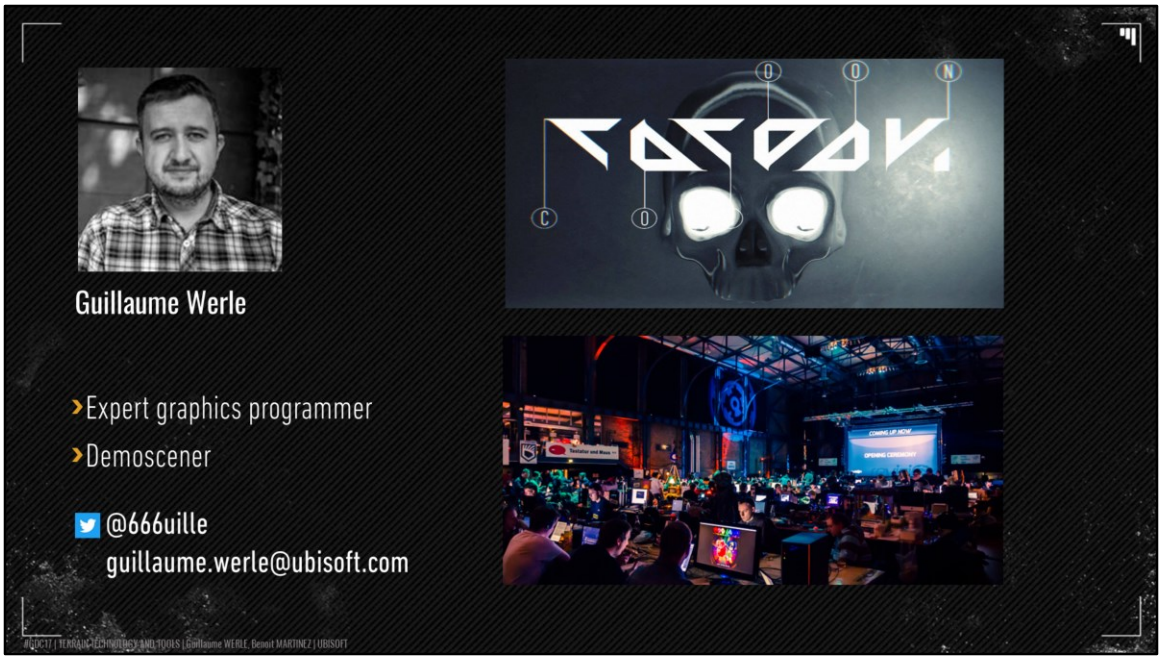
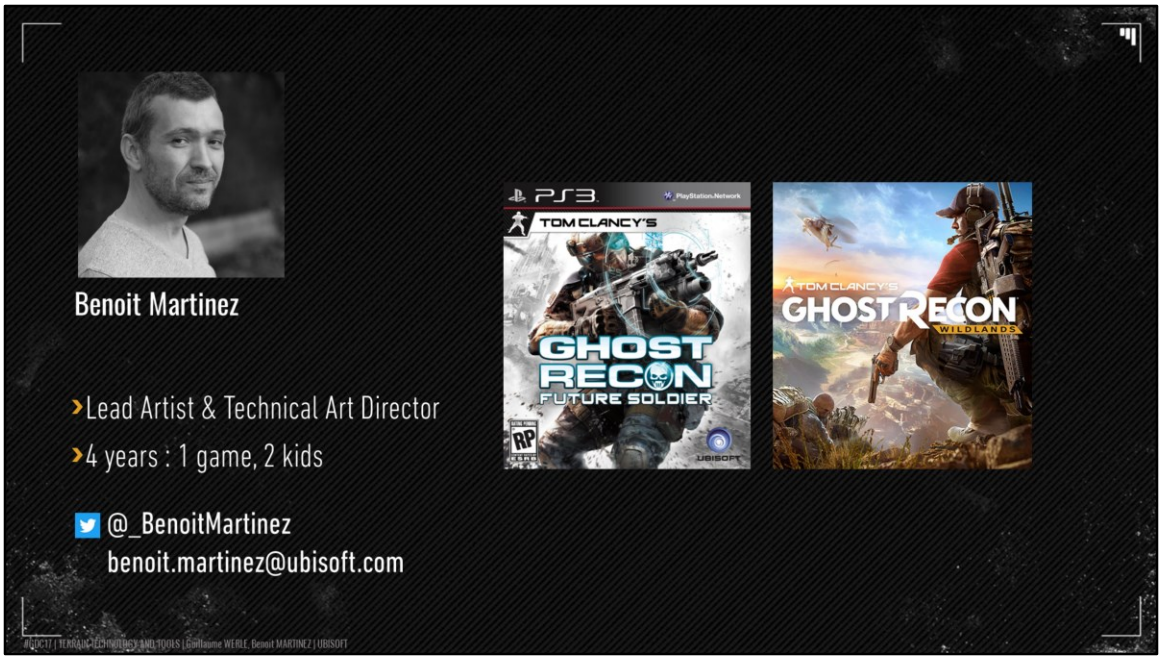

GDC
GHOST RECON WILDLANDS
TERRAIN TECHNOLOGY AND TOOLS
Guillaume WERLE
Expert Graphics Programmer - Ubisoft
Benoit MARTINEZ
Art Lead & Technical Art Director - Ubisoft
GAME DEVELOPERS CONFERENCE® | FEB 27–MAR 3, 2017 | EXPO: MAR 1-3, 2017 #GDC17
UBM

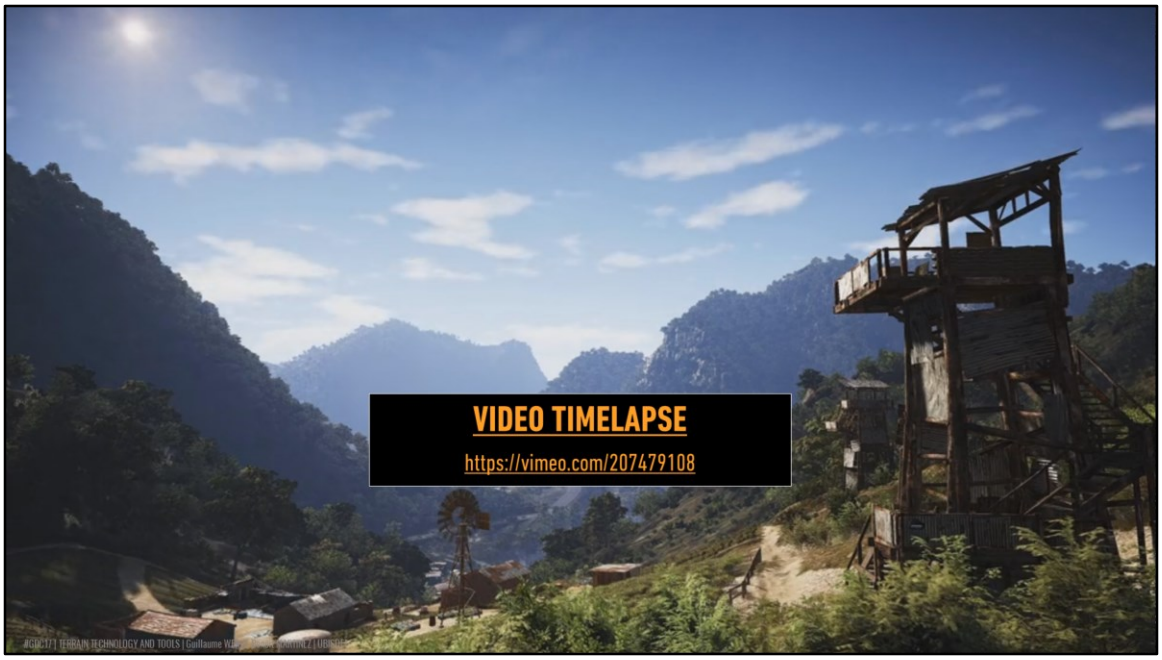


My name is Guillaume Werlé
I am a graphics programmer and I was in charge of the terrain technology on Ghost Recon Wildlands.
I enjoy creative programming and during my spare time I am an active member of the demoscene community.



My name is Benoit Martinez
Lead Artist and Technical Art Director. I was in charge of the level Art team and the Procedural content / Houdini team.
I joined Ubisoft in 2011 to work on GhostRecon: Future Soldier and then I was part of the Wildlands core team from the very beginning of the project.
Father of 2 boys born during Wildlands production, I've been very busy in the last 4 years !

Today we're going to talk about terrain, technology and tools
But first a short video…



https://vimeo.com/207479108

GhostRecon Wildlands is the largest action adventure open world ever made at Ubisoft

The game is set in Bolivia
We have a lot of diversity in the game, 11 completely different biomes

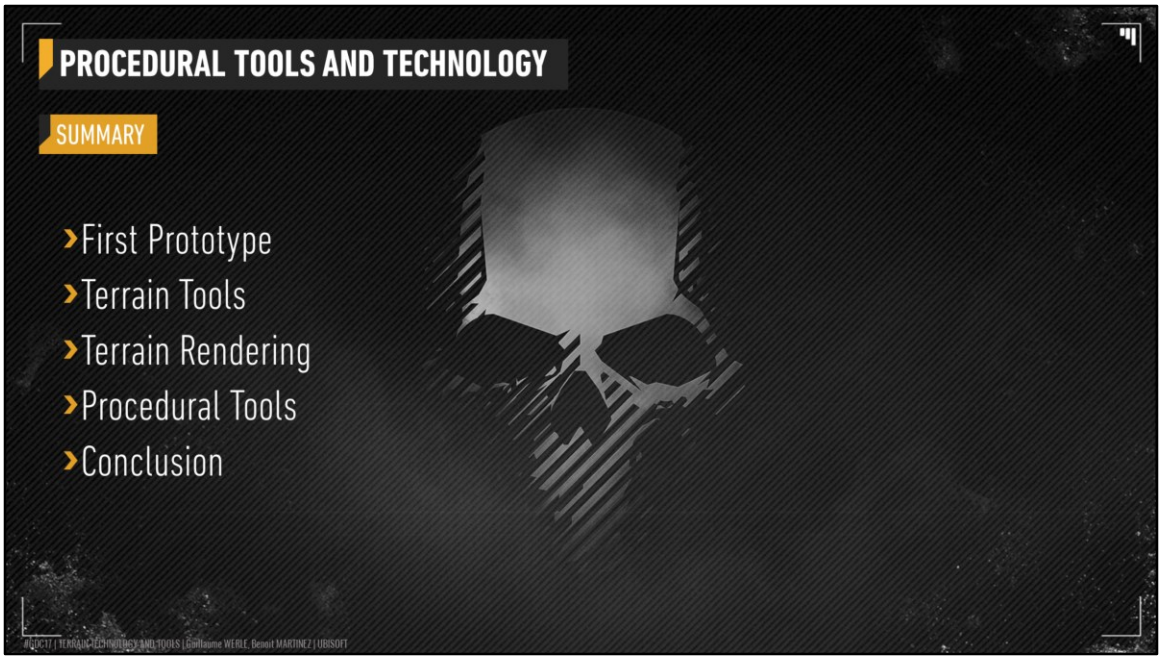
Lakes, rivers and streams over a surface of 16km2

It’s mostly a natural environment with many forests
Millions of trees, bushes and rocks

Roads : over 600km of roads and even more paths

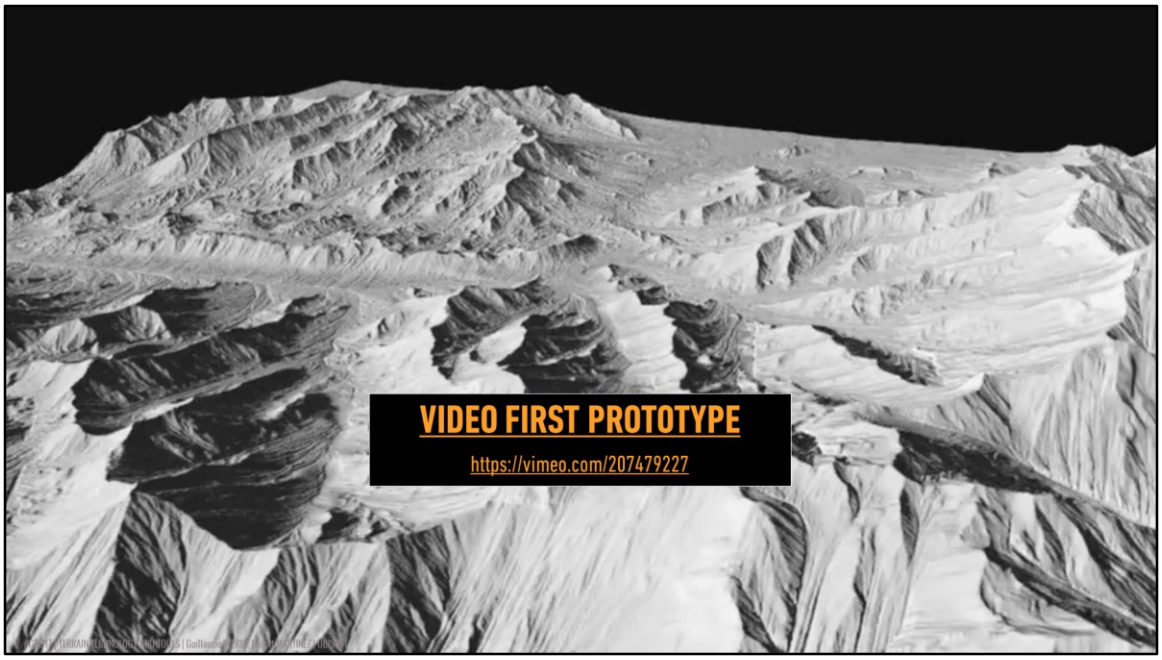
A full railway network going across the world

Over 200 specific locations in the game: camps, landmarks, outposts and 58 fully procedurally built villages



Summary

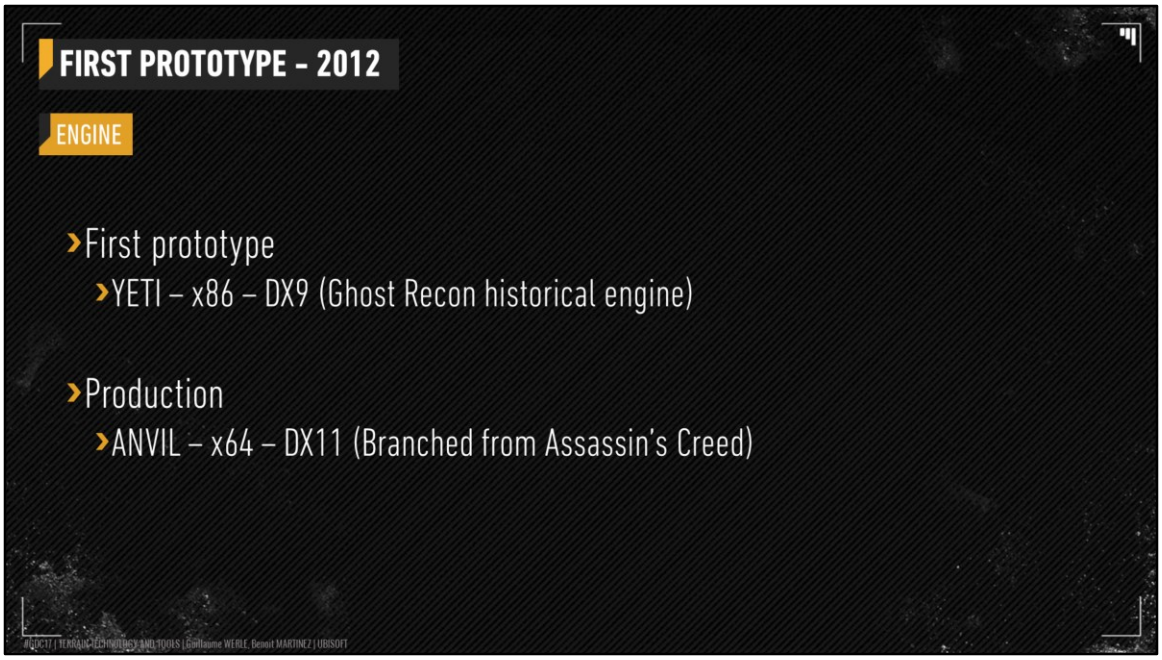
- **First prototype** – How we started back in 2012
- **Terrain Tools** – Guillaume will speak about the tools he has created
- Then he'll give all the details about the **Terrain Rendering**
- **Procedural Tool** – I (Benoit) will talk about the environment creation tools and pipeline
- We'll then conclude this talk and have some time for questions



https://vimeo.com/207479227
After GhostRecon Future soldier we started to experiment with bigger terrain.
Using real world data we refined DEM file with world machine.
Then with Houdini we created some tools to:

- Automate the creation of a tiled mesh with LOD.
- Define a splatting mask to distribute the materials according to some rules based on slope, altitude, roughness and other mask from worlMachine like flowmap,
- Scatter points to define vegetation, using rules: slope, altitude, material, density, distance between each trees, etc.

The results for a prototype build in a couple of months with only 4 artists and 1 graphic engineer was very promising.



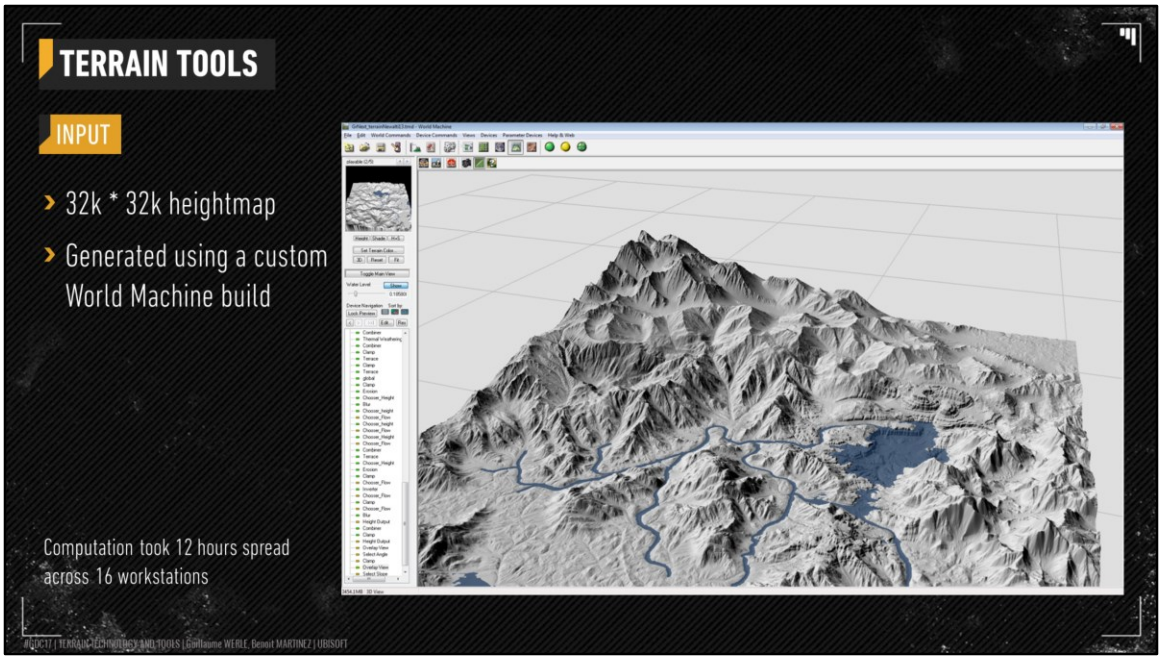
To build the prototype we used the GhostRecon legacy Engine : Yeti
Despite the fact we managed to achieve some good results it was not suited for our ambition and we decided to move to another engine,
We chose Anvil, branched from Assassins Creed

The prototype helped us to realize 2 things :

We had to improve our terrain workflow. Dealing with meshes, with all the back and forth from the editor to the DCC app was not practical.
We needed a terrain technology that would allow easy editing for the artists and fast performances at runtime.

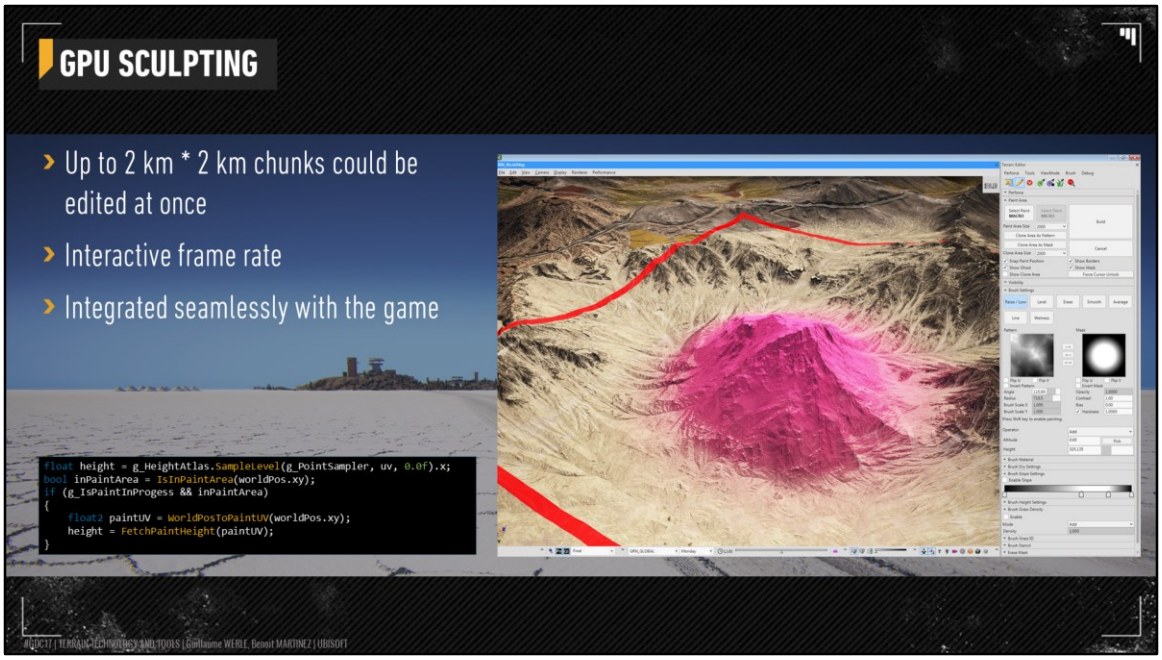
Houdini proved to be flexible and super efficient as a technical artist could create tools without a dedicated engineer.

(Guillaume Werlé)
I joined Ubisoft and the Ghost Recon team 4 years ago.
Just a few months after the development of this prototype and I started working on the terrain editor.



During those few months the art team had developed an heightmap generation pipeline based on WorldMachine.
The main goal of the terrain editor was to take this large resolution heightmap without any material information as an input and turn it into …

… a realistic looking world with a huge variety.
This screenshot was taken a few weeks before the submission and illustrate pretty much what we wanted to achieve.
We wanted to have an extreme amount of details when looking close and impressive vistas in the background with seamless transitions in between.
When cranking up the graphic options, the pixel ratio can go up to 10 texels per cm and 1 triangle every 2 cm.
The player can go everywhere in the game, it meant that this level of quality had to be consistent all across the world.



The first feature that we developed in the editor was the heightmap sculpting tool.
We wanted to do it on the GPU for several reasons.
First because it's fast.
We needed to be able to edit huge area of the map and stamp a whole mountain in a single click.
And also because it was the best way to have accurate visual feedbacks while editing.
The heightmap and the brushes are blended together in a floating point render target that can be used directly by the game shader.

Making a game is an iterative process.
We needed a way to try new ideas within the game with the possibility of reverting them without loosing too much time.
So we came up with the notion of layers.
The world machine output is kept into what we call the « base » layer and every changes done in the editor ends into the « macro » layers.
When the result of the editing wasn't satisfying, the « macro » layer content could easily be erased to go back to the original state of the heightmap.
Here's a short video demonstrating the heightmap sculpting tool that showcases the « copy/pasting » brush and the « erase » brush.

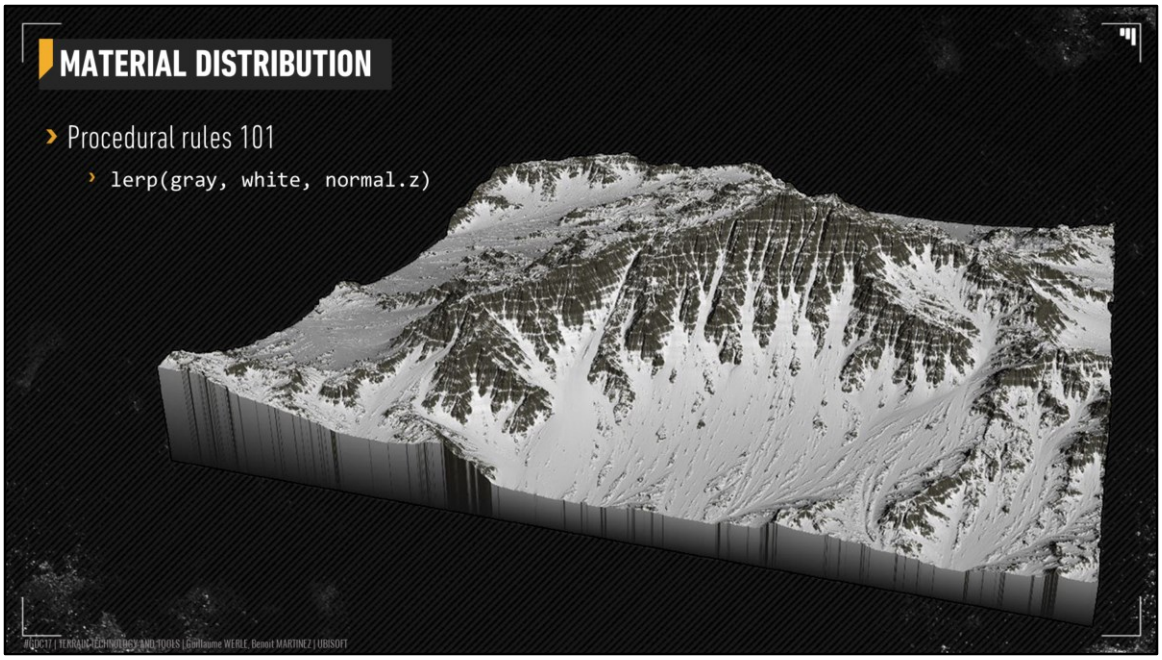
https://vimeo.com/207479248

We also created a dedicated layer for the houdini generated modifications.
This is an extremely important feature that will be described during the 2nd part of the talk by Benoit.
In short, using Houdini the technical artists had a complete read / write control over the terrain.

The last layers is the one that contains all the level design related changes who couldn't be done using Houdini.

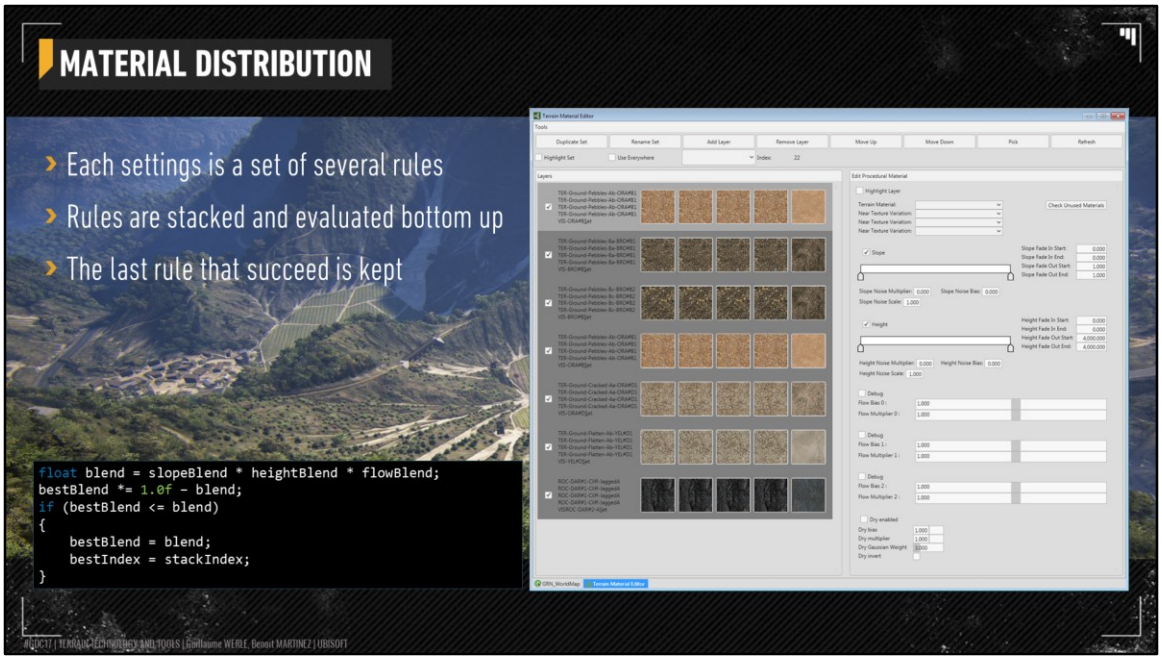
I’m now going to describe our material distribution process

As I've already said it before, we wanted a realistic looking world with a consistent level of quality.
By looking at the size of the map we quickly came to the conclusion that painting everything manually was a terrible idea.
We needed a semi automatic process.
So we decided to generate the material distribution procedurally.



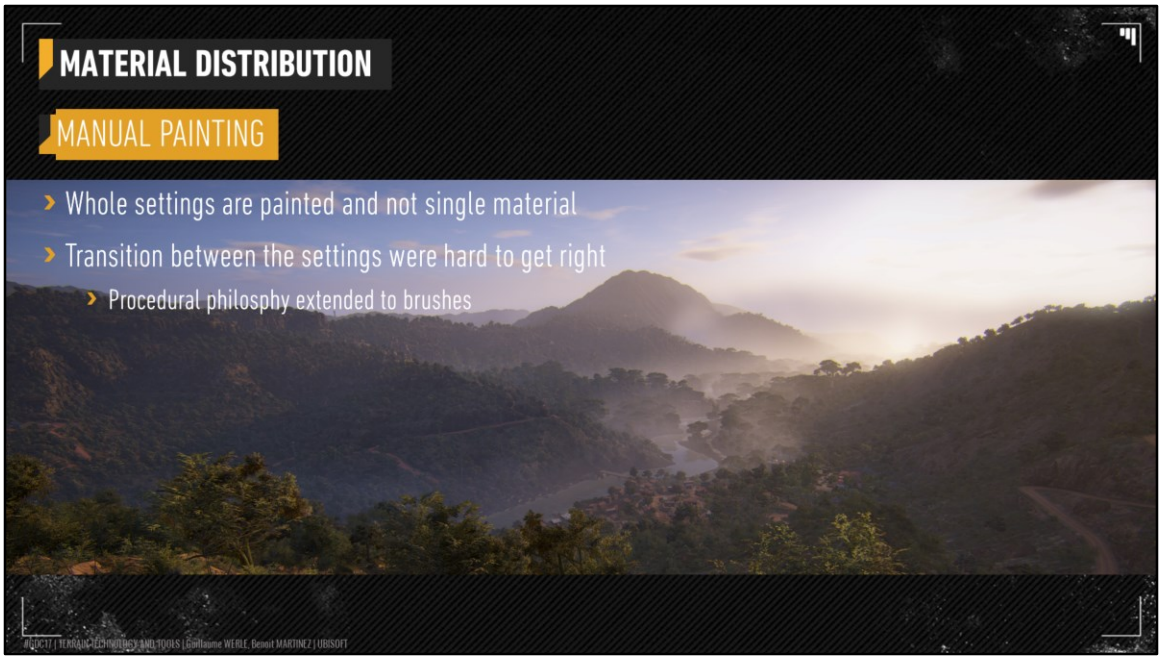
To give a better sense of the relief, the terrain editing tool was already tinting the heightmap depending on the normal direction.
When looking from far away, this simple rule is almost enough to generate a convincing mountain top.
We decided to extend this concept.

Instead of displaying a solid color we would display a full featured material when all conditions of a rule were satisfied.
We designed several conditions based on the topology, noise or simple kernels that could be evaluted in realtime in a pixel shader.



This is a screen capture of our settings editor.
As you can see each settings in Ghost Recon is made up of several of rules stacked one on top of each others and evaluated from the bottom to the top.
The material of the last rule to succeed during the evaluation will then be displayed.

https://vimeo.com/207479253



We were now able to paint a whole settings in a single click but the transitions still required a lot of work to look natural.
We decided to extend the « rules » philophy to drive the opacity of the brushes.
Here's another video

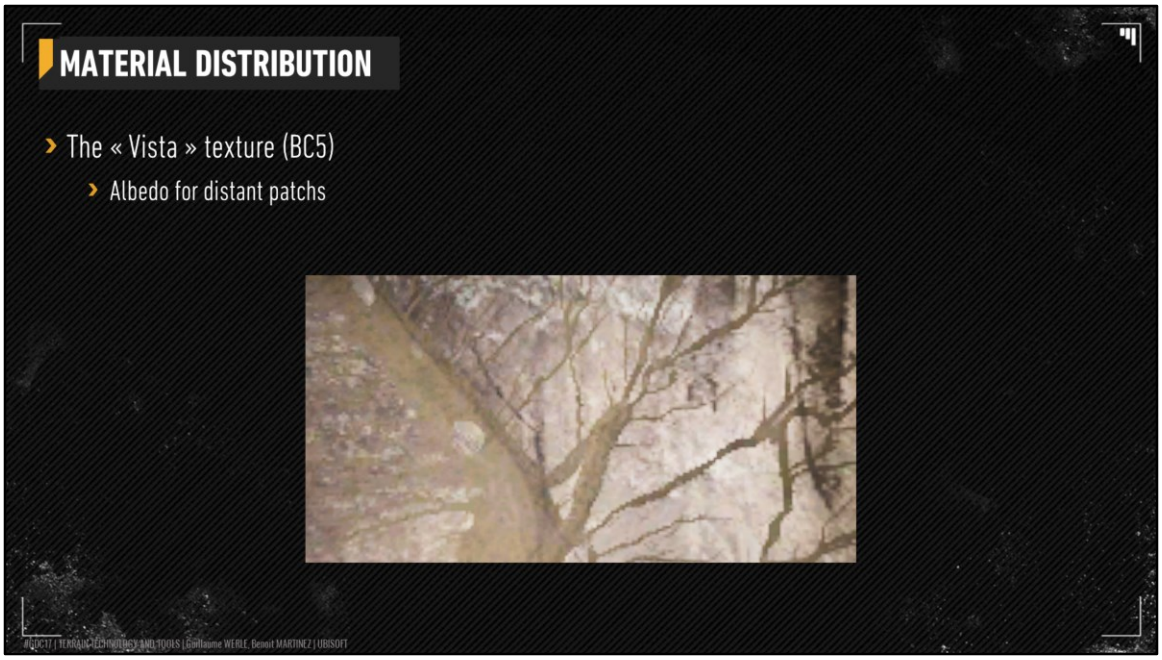
https://vimeo.com/207479279

MATERIAL DISTRIBUTION

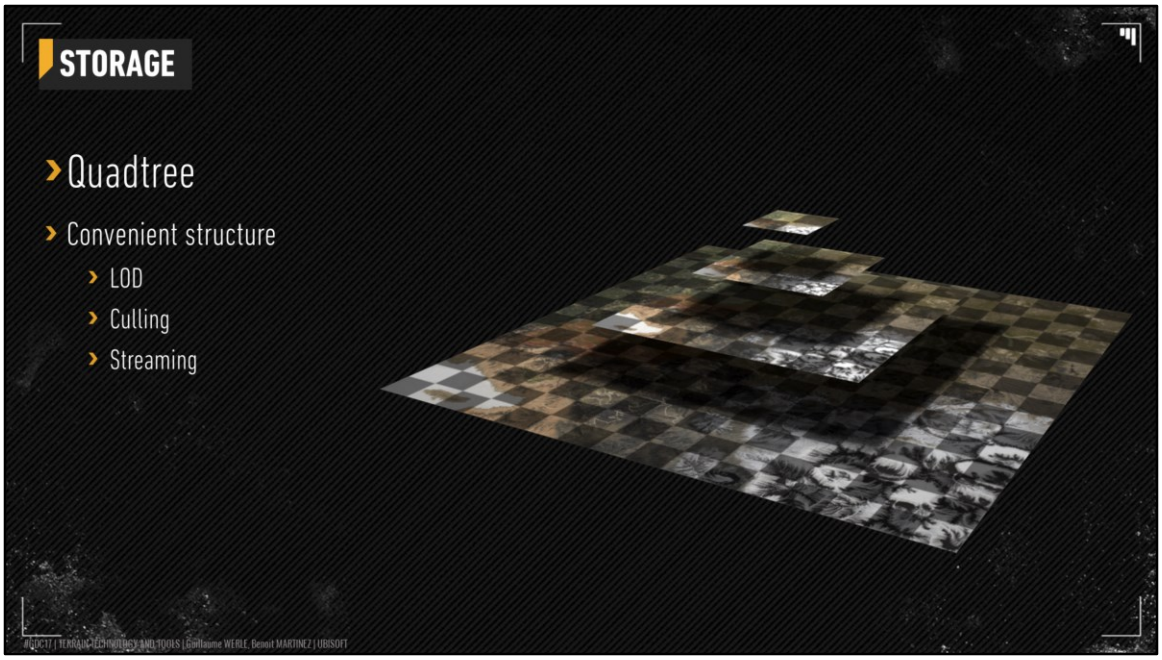
Result is saved into 2 textures with same resolution as the heightmap

- The « splatting » texture (R8)
  - Contains the material index

The result of the material distribution is then saved into 2 textures
We call the first one the « splatting ». It contains a material index for each entry of the heightmap

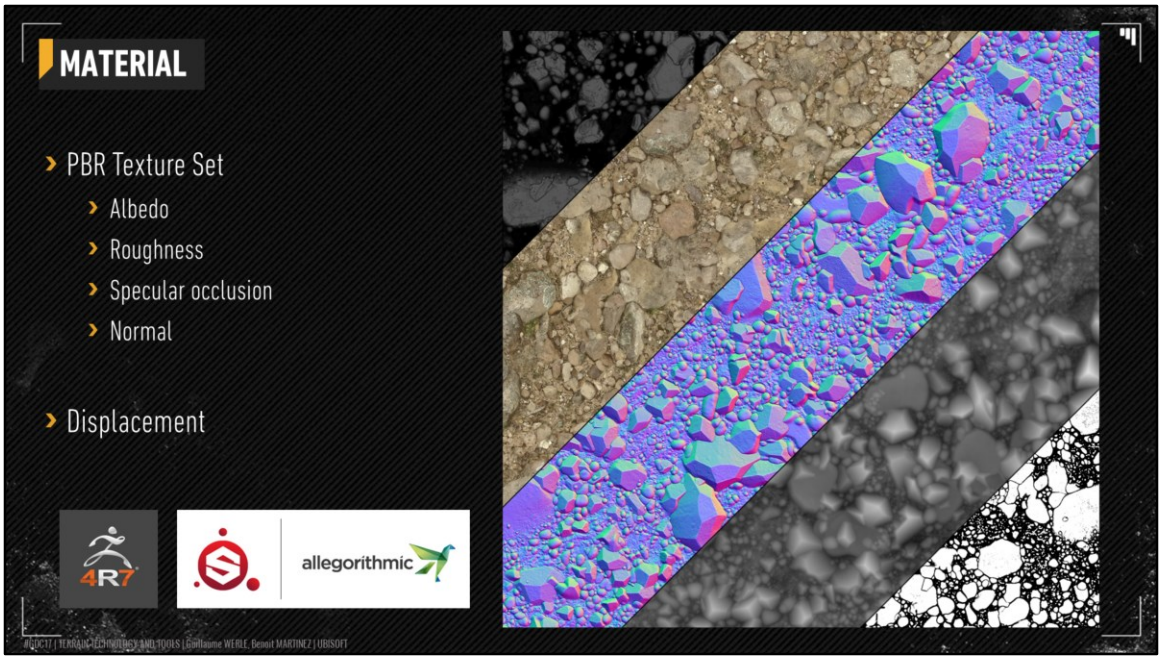


The second one is called the « Vista » texture and looks likes a google map screencapture.
It will be used as an albedo map when rendering the distant patches.

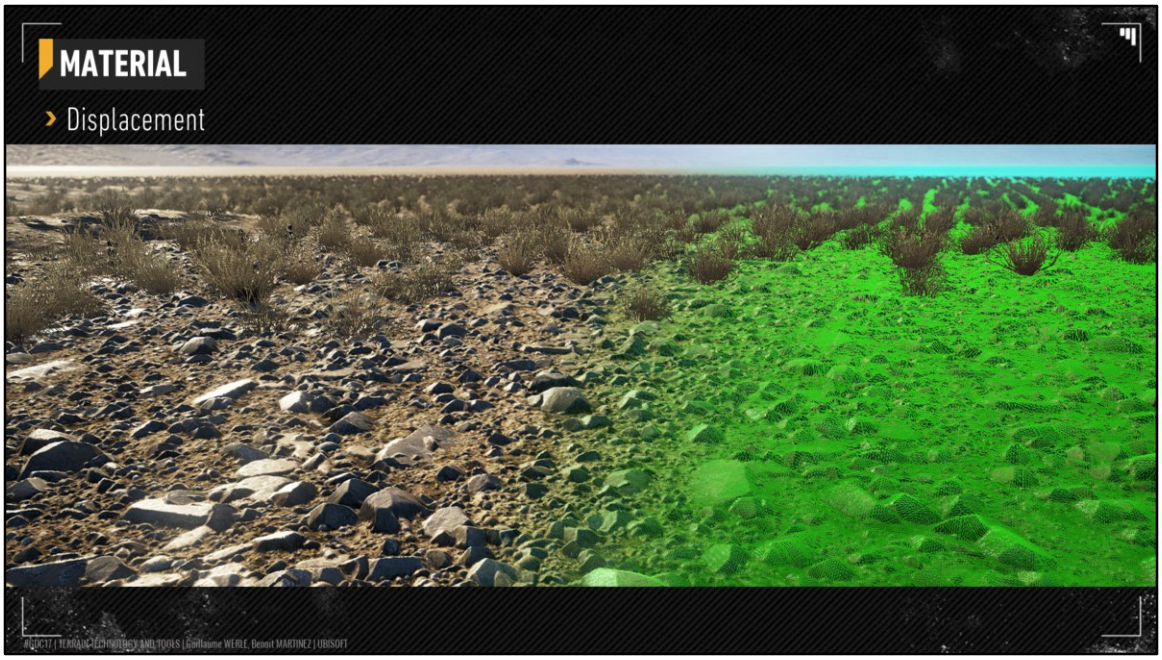


Everything is then cut into tiles, and compressed into a quadtree.
This simple structure was very convenient for LOD generations, and patch culling.
Each node of the quadtree has a small payload that can be streamed in or out by the loading thread depending on the culling result and the distance to the camera.

Before moving to the rendering I’m now going to quickly descibe our materials.

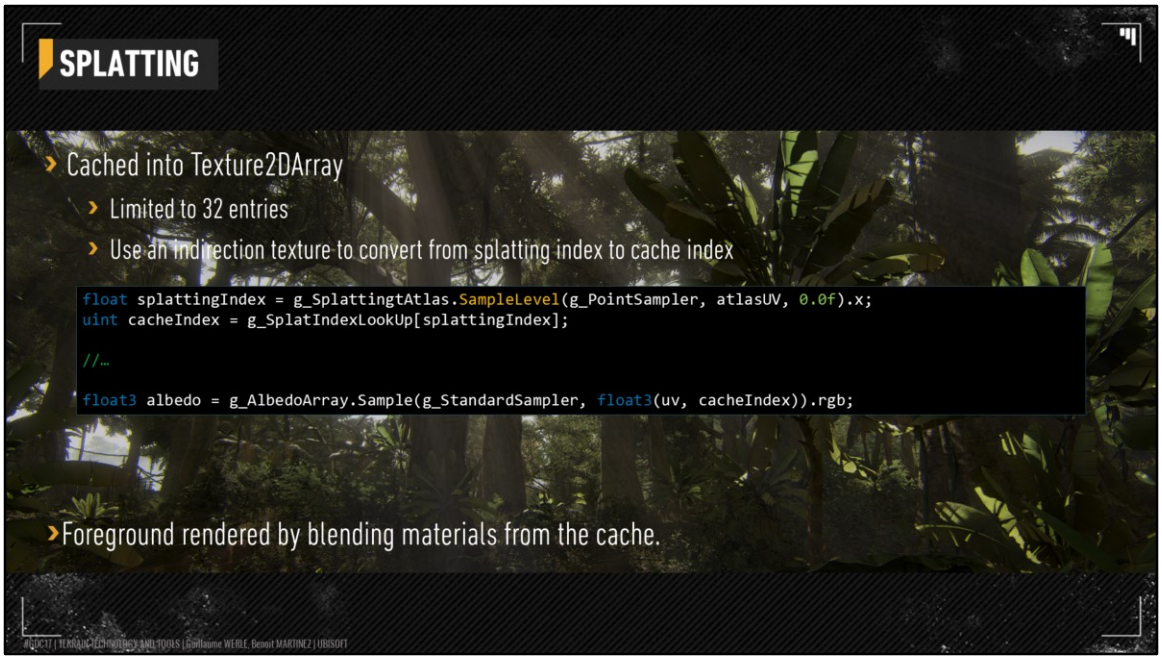


Each individul material in GR is PBR compliant and is made of several high resolution textures created with zbrush and substance.



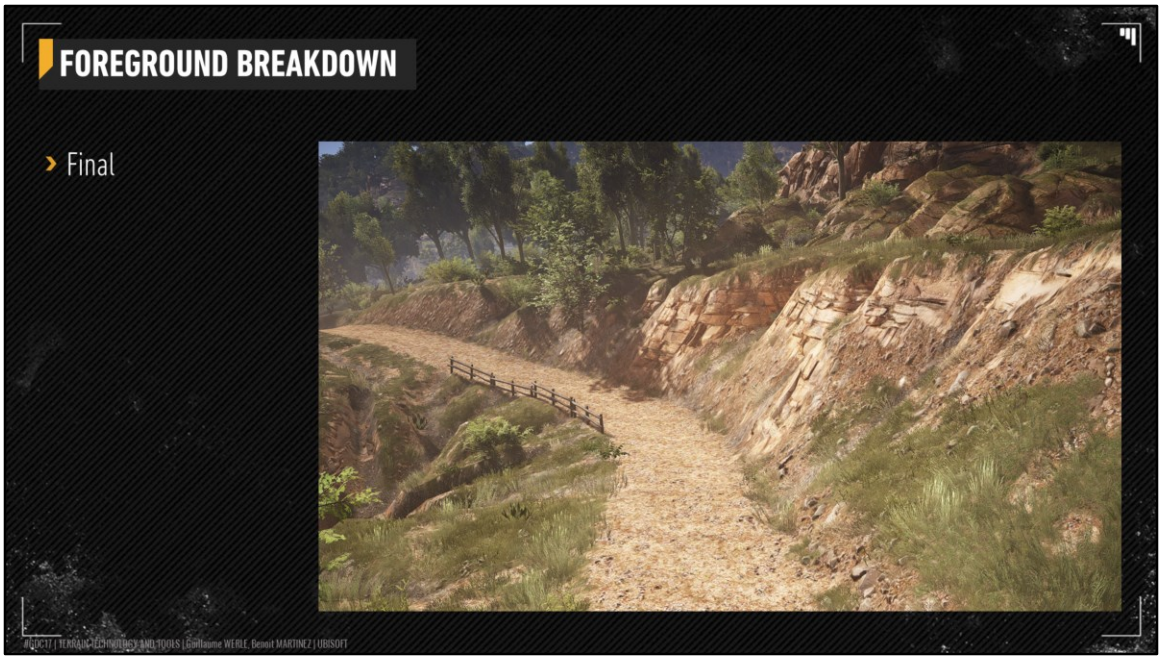
Each material also embed a displacement map.
Tesselation based displacement helped us to get the « next gen » look that we were looking for while developing the prototype.
Since we had very promising results we were willing to invest a lot of GPU time in it.

Most of the terrain renderers that I've seen use 4 materials at the same time and it's usually enough if you only have a single settings for your world.
Due to the diversity of our world, at the end of the production we had total of 143 unique materials on the terrain.
Loading all of them at start wasn't an option.

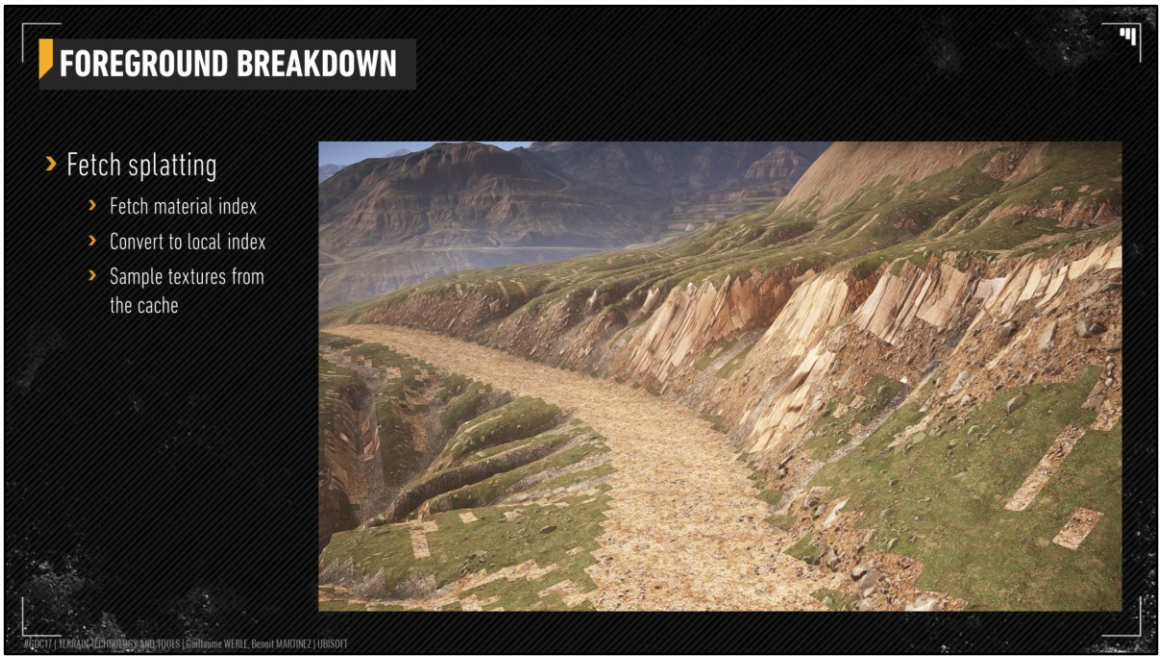


We cached the 32 nearest materials around the player in a texture array.
An indirection table is then used to convert from the global splatting index to the local cache index.
To render the foreground we then sample the textures directly from this cache.

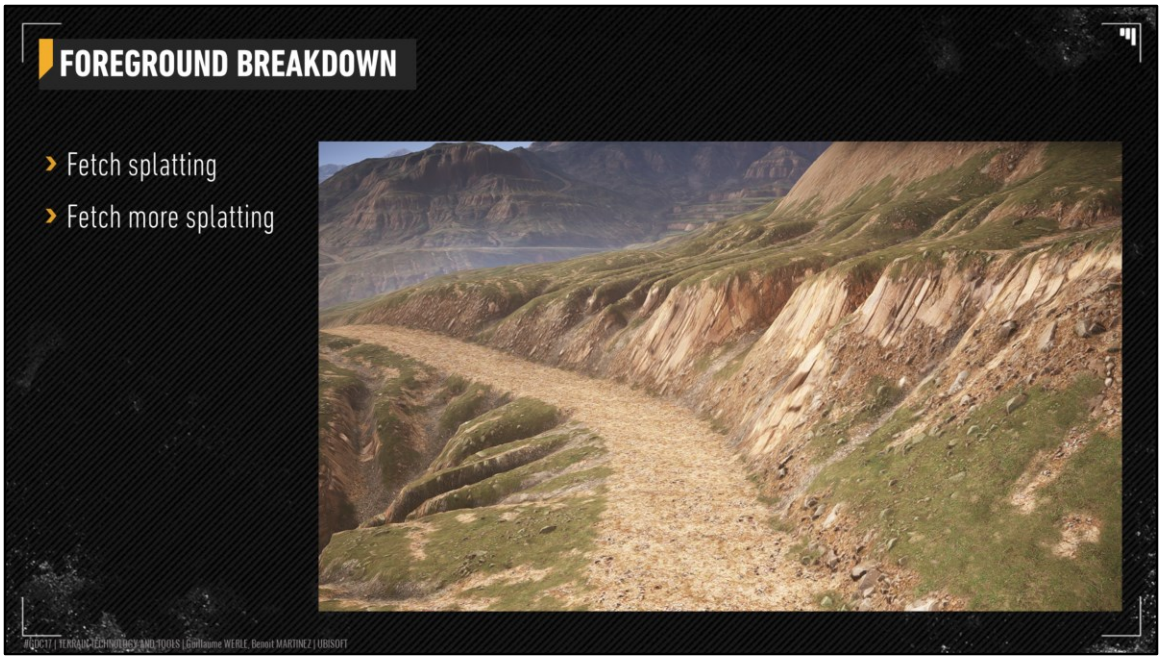
With this in mind I’m going to describe our rendering process.



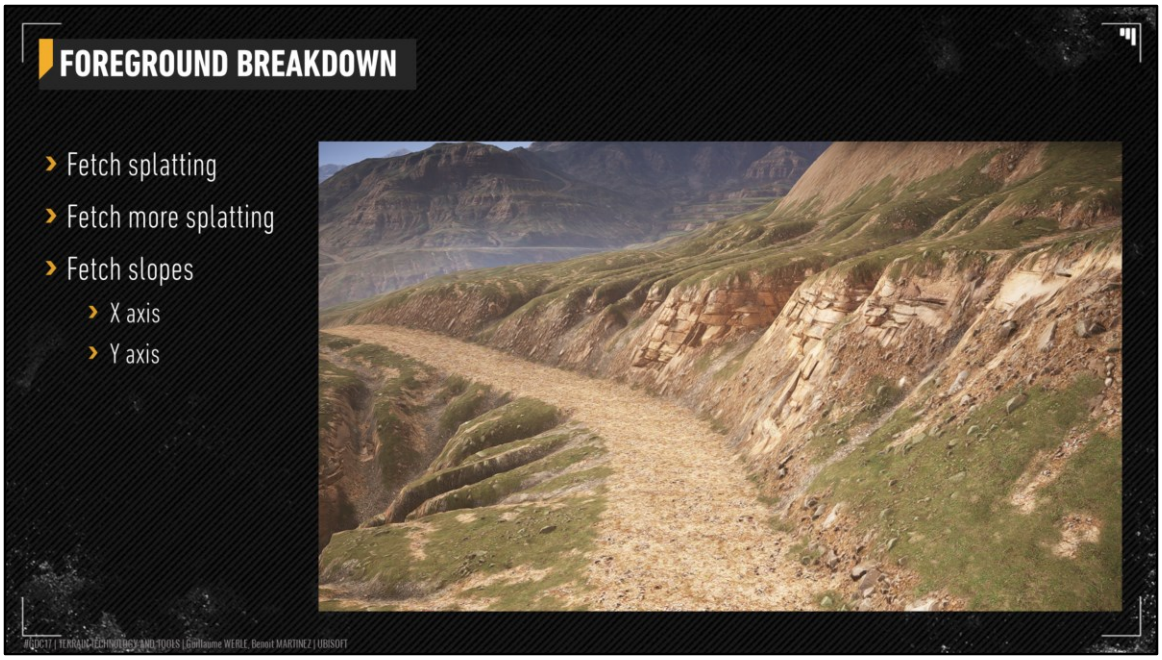
This is the final result of the foreground shader.
To give you a high level overview of what's going on inside of it I'll do a step by step breakdown



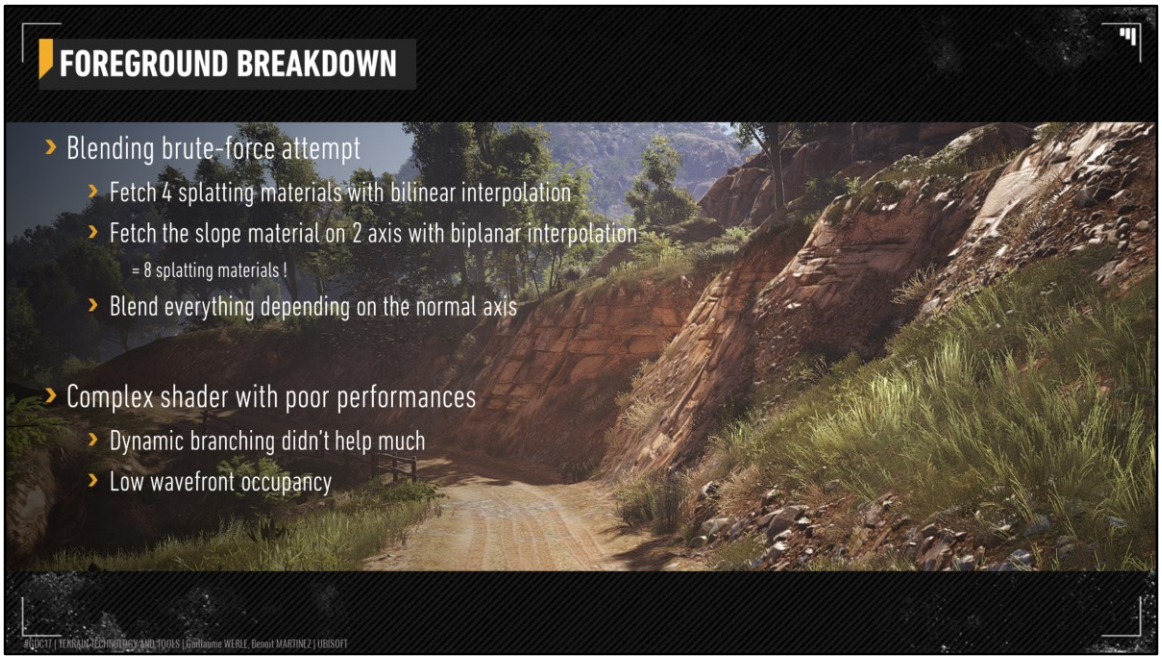
We start by fetching the splatting index
We translate it into a local cache index
Then we sample all the pbr textures from the cache



We do this again 3x with bilinear interpolation to have nice transitions between all the materials.



To avoid stretching we used biplanar projection for the slopes.
That meant we have to do all of this 2x
1x for each axis



The first shader version that was visually satisfying was also very costly in term of performance.
Too many textures were fetched and blended together both in the pixel shader and the domain shader.
The high register pressure caused by the complexity of the shader yielded a very low wavefront occupancy.
We really needed something simpler

While looking for optimization opportunity we noticed that the parts that were completly flat or completly slanted didn’t need to blend with so many different materials.
The idea was to make specific shaders for the following case

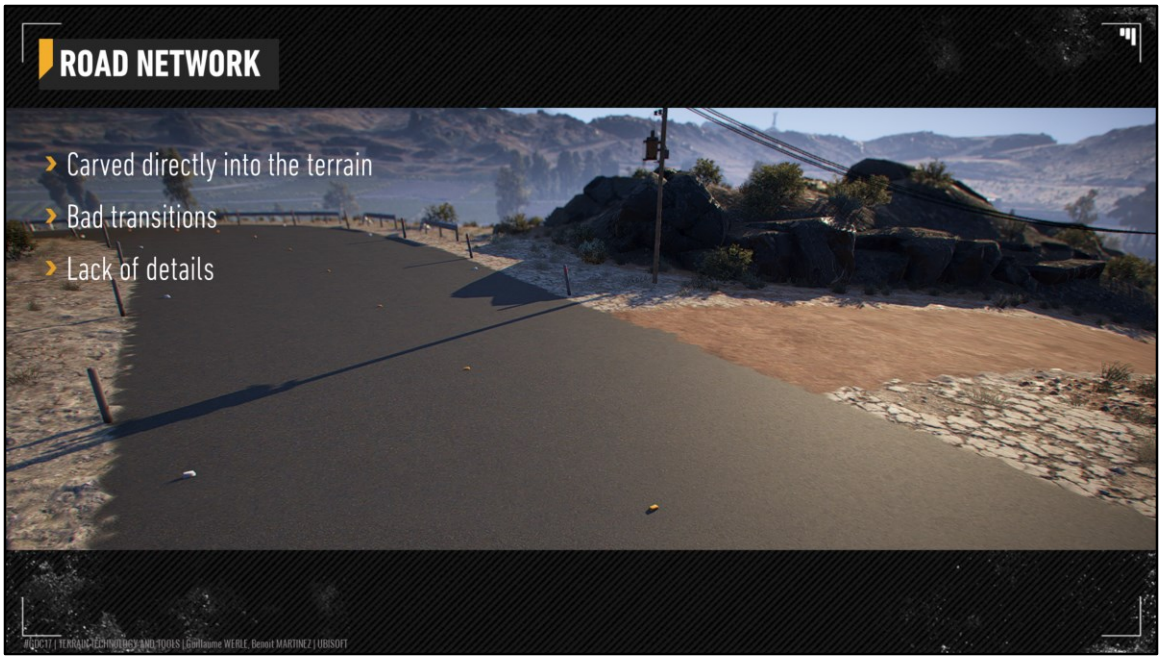
1st case
We have 4 neighbors with bilinear interpolation

The slopes with the biplanar projection.

Last case
The transition area where we still need to fetch everything.

Cutting the terrain in small unique patches to isolate such area would have been to costly in term of disk space.
For us, the solution was to generate them a runtime as soon as they enter the viewport using a compute shader.

This definitly helped on the performance side
Now approximatly 80% of the terrain fetch 4 materials instead of 12.

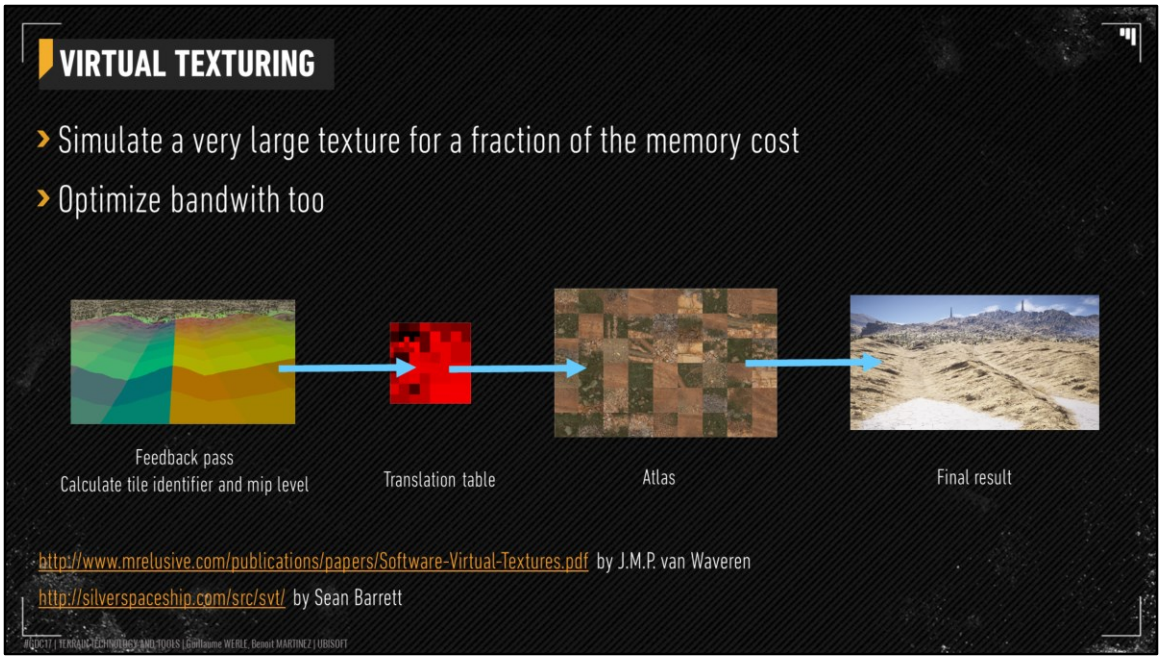


At this time we were very concerned by another challenge, the road network !
Generating it at runtime seemed like a technical nightmare and we were affraid to run out of disk space if we had to store it on the bluray.
We decided to carve the road direclty into the terrain instead of representing it with traditional geometry.
But this technique would raised a few issue like cheap material transition and an obvious lack of details.

Screen space decals were looking like a very promising solution.

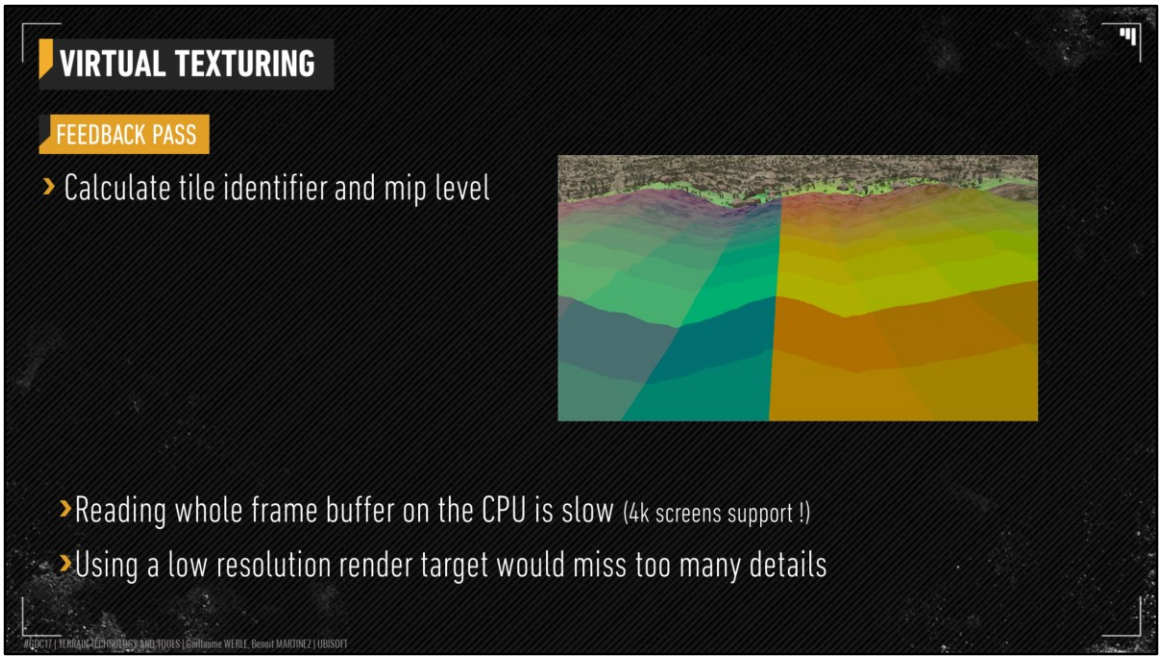
But their cost in terms of overdraw becam quicky prohibitive
Also they didn't affect the displacement topology

(Guillaume Werlé)
In our case, virtual texturing looked like a very good optimization opportunity.



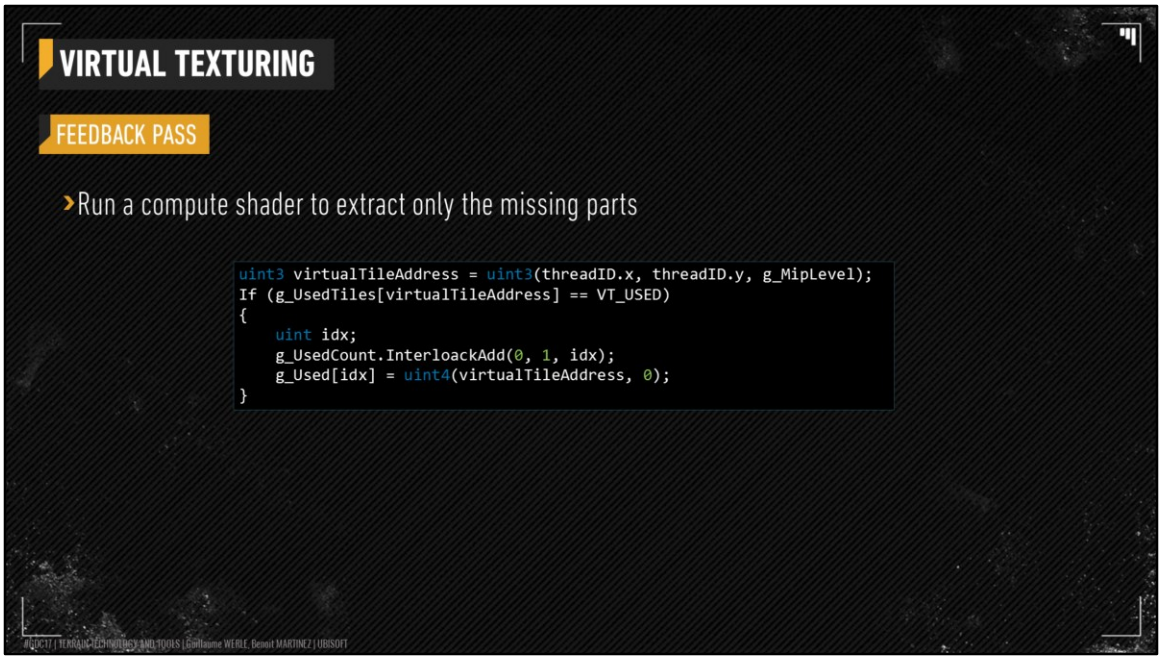
This would greatly reduce the required amount of texture fetch by precomputing the material blending and all the decals projections in an atlas.
Virtual texturing was already successfully used on other titles and I guess that most of you here at GDC are already familiar with the concept so I'm going to pass on the details.

Has with every rendering technique virtual texturing comes with some drawbacks.
We managed to improve over the traditional technique on the following points.



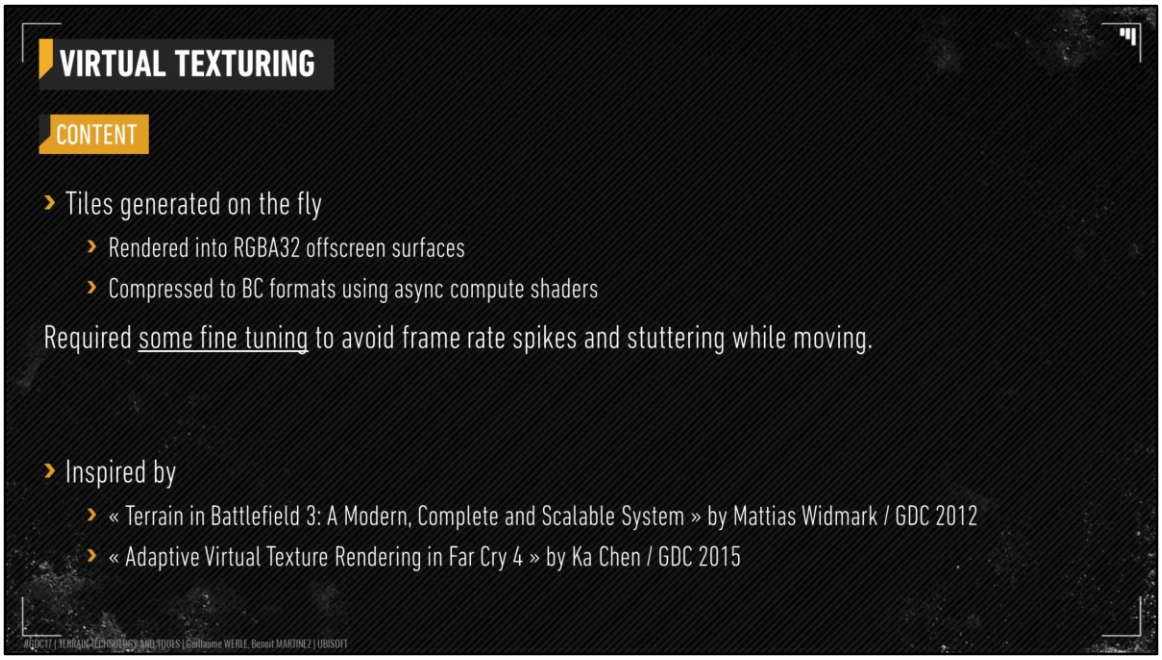
The easiest way to determine which part of the virtual texture is required is to rasterize the scene in an offscreen surface and to read the content on the CPU.
This is a costly operation espcially when your game is running on a 4k display.
Since our terrain renderer use tesselation and displays a lot of tiny triangles using a lower resolution render target would have missed too many details.

Instead of having a dedicated pass we bound an additionnal 3D texture as a render target during the gbuffer pass.
While we render the terrain, using the UV and the miplevel as output coordinates we directly tag the displayed part of the virtual texture.



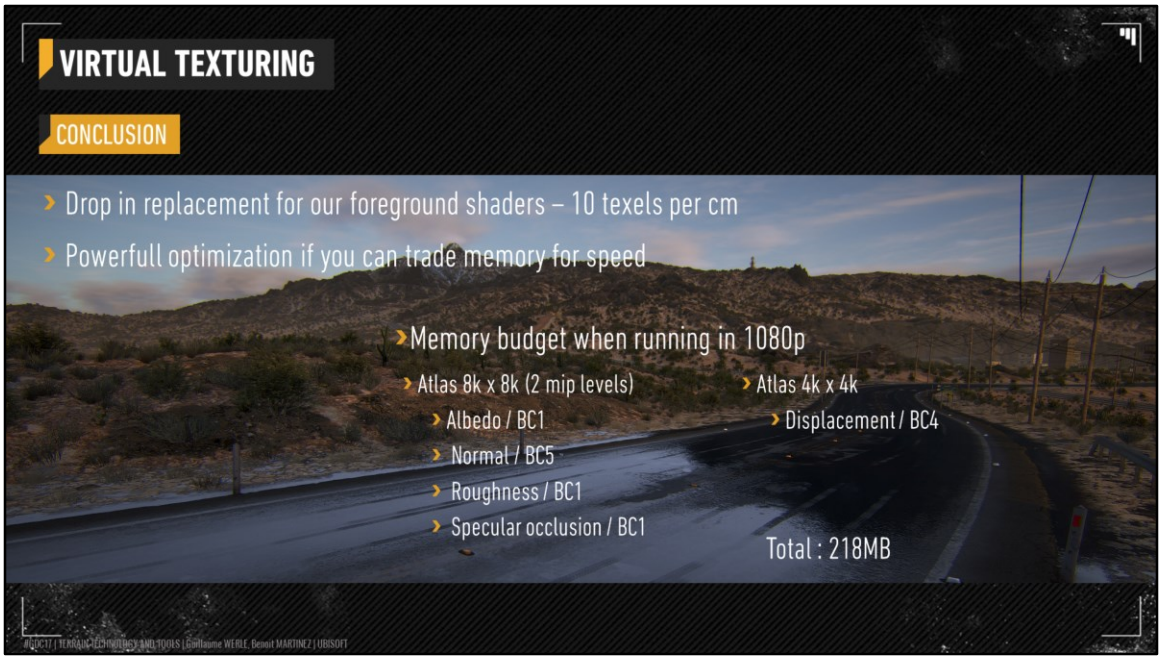
After that a compute shader parse the texture content and record the missing parts.
The results (just a few bytes) are then read later on the CPU.

Some words about the amount of content
When running with the highest graphic quality option the virtual texture displays a pixel ratio of 10 texels per cm.
If precomputed and compressed, this would require approximatly 2 petabits of storage.

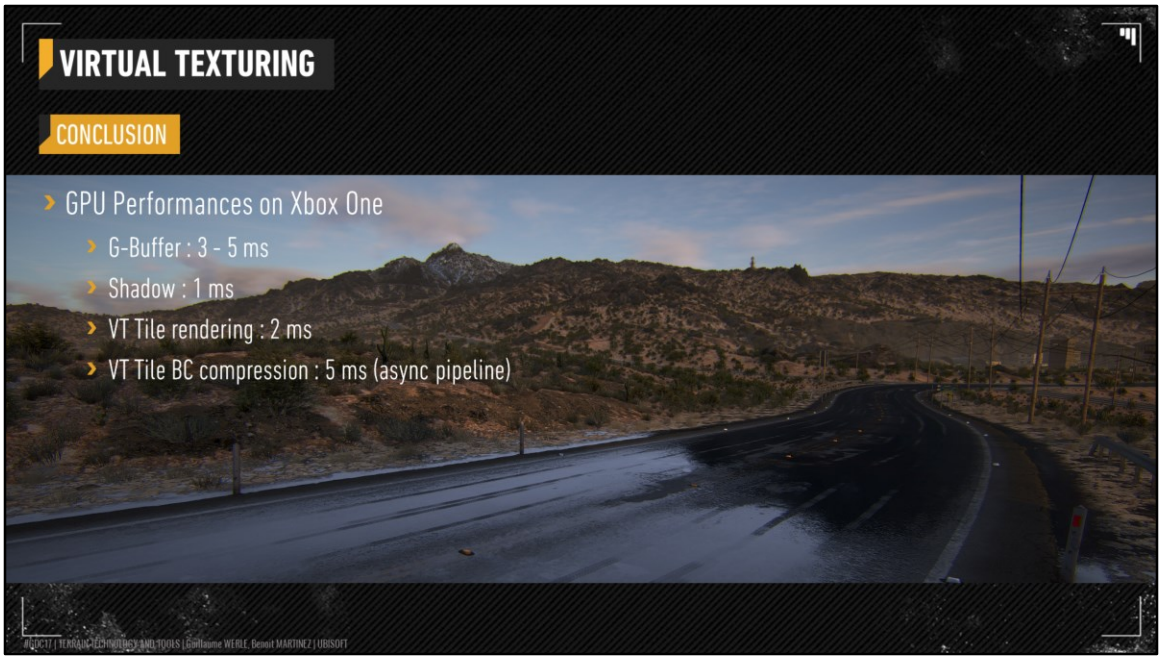


So we decided to generate the content at run time, the materials and the decals are blended together on off screens surfaces and compressed on the fly using the Async Compute pipeline.

The situations with low frame-to-frame coherency were tricky to handle since the required amount of tiles to update when driving a fast vehicle could quickly kill the frame rate.
The virtual texture update had to be time sliced across several frames and a lot of tweaking had to be done to find a right middle between keeping good frame rate and a low latency.



We managed to successfully replace our costly foreground shader with virtual texturing.
Luckily we took this decision early during the pre production and we adjusted the memory budget in consequence.
As you can see a bit more of 200MB are required to store the atlas when runing the game in 1080p.
This budget is increased to 1GB when running in native 4k.



I wanted to end this part of the talk by giving some performance stats on xbox one.
Those are averages and are heavily view depend but we can say that everything fits in less that 8ms on the GPU side most of the time.
Those 8 ms doesn't include the BC compression since it's running for free in parallel.

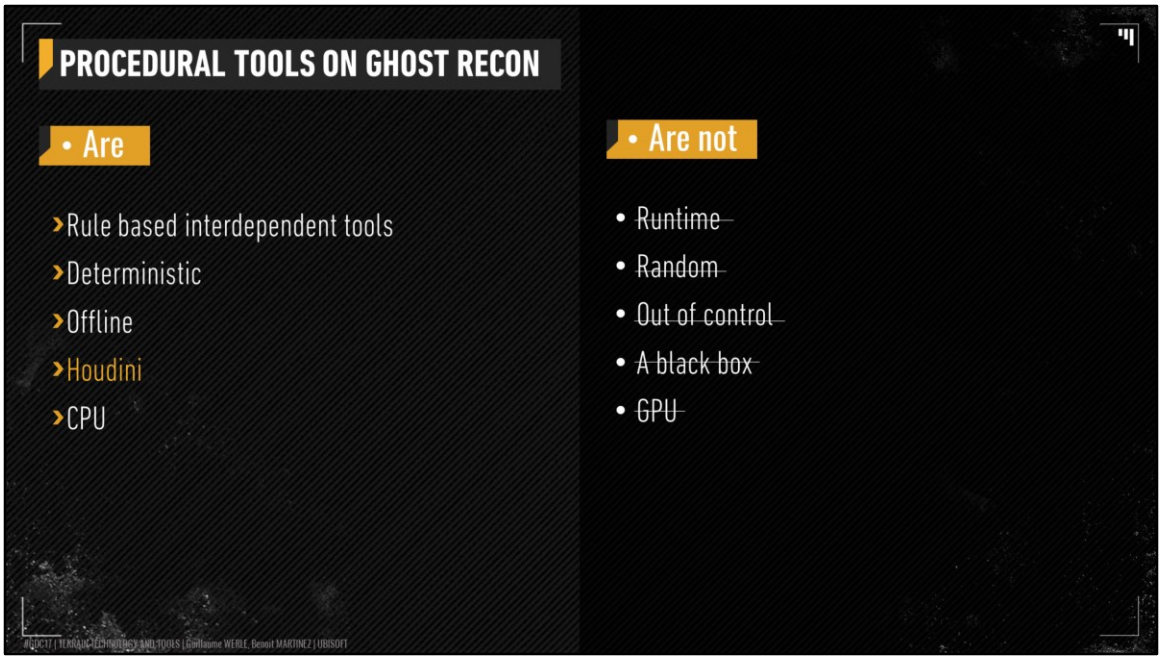
(Benoit Martinez)
Now Guillaume told you everything about the terrain I'm going talk about our procedural tools.
What are they, how they work and the philosophy behind them

Scale
- Tools have to work on a large scale but also locally.
- Keep the control from the smallest rock to the biggest montain was the production mantra when it came to tools.

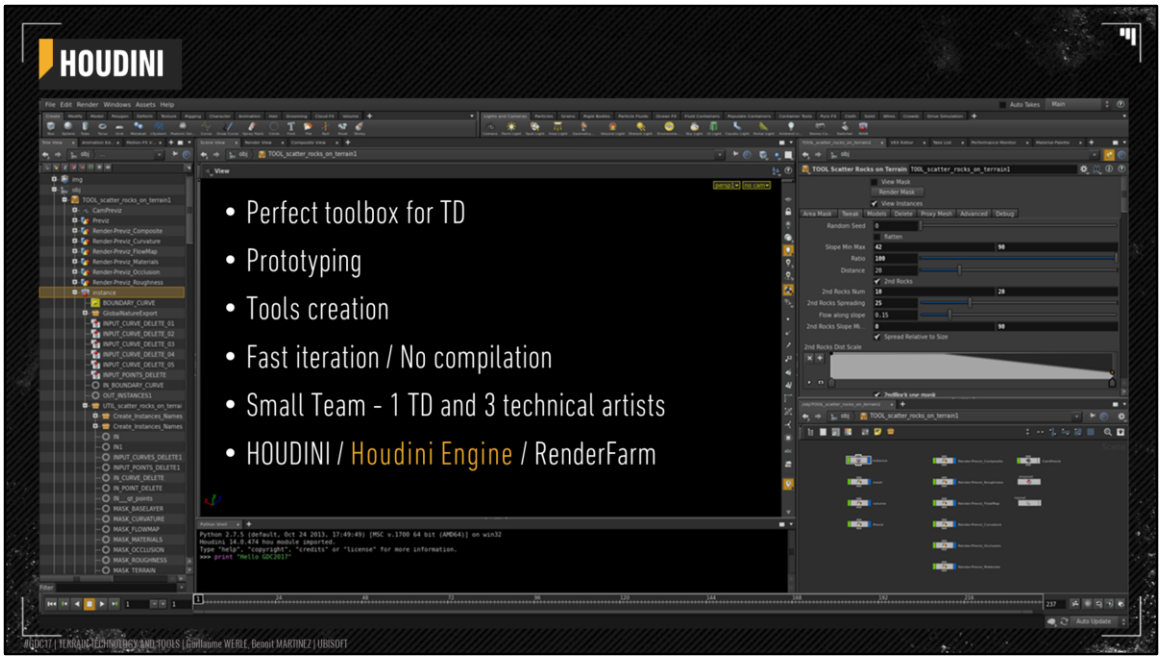
Collaborative
- One unique giant level
- 2 teams / locations for world building: Paris and Bucharest

Iterative
- We need to be able to change our mind
- Try new ideas
- Tool is a data: Tools are created and stored exactly like any other entities in the world. A tool is a meta entity which generates its own entities,

And most importantly: We had to maintain an always playable world on a daily basis.



- Rule based interdependent tools: Users edit parameters and tweak rules to produce content.
- Deterministic: Same inputs produce same results
- Offline: Nothing is generated at runtime. Everything is baked during the production.
- Houdini: The world creation pipeline and all the tools have been created using Houdini.
- CPU: Since we are using Houdini most of the tools are CPU based (latest version of Houdini can handle GPU better but in was not the case during the production). Relying on CPU is not an issue : CPUs are cheap and we can use more RAM and some of our tools require a lot of RAM!



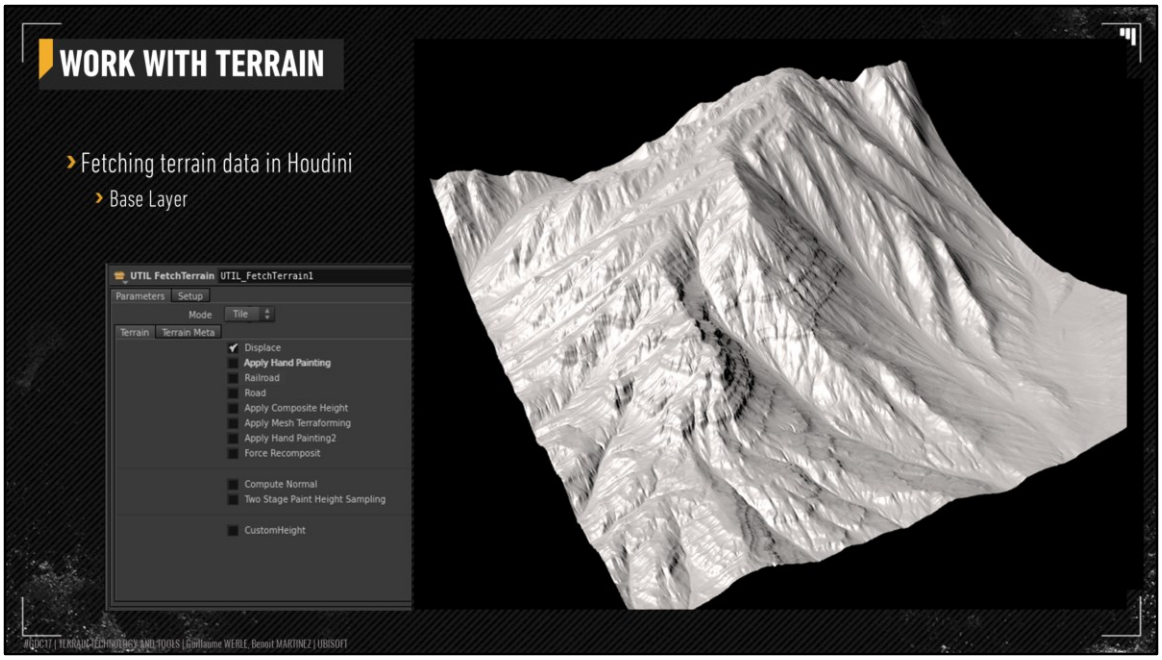
What's Houdini :
- DCC App from SideFX
- Widely used for SFX (not our case)
- We use it for environment creation

What do we use Houdini for ?
- Prototyping : Very good toolset to quickly sketch ideas
- Tools creation : Create content
- Previz data / get statistic / Crunch data / Optimize data

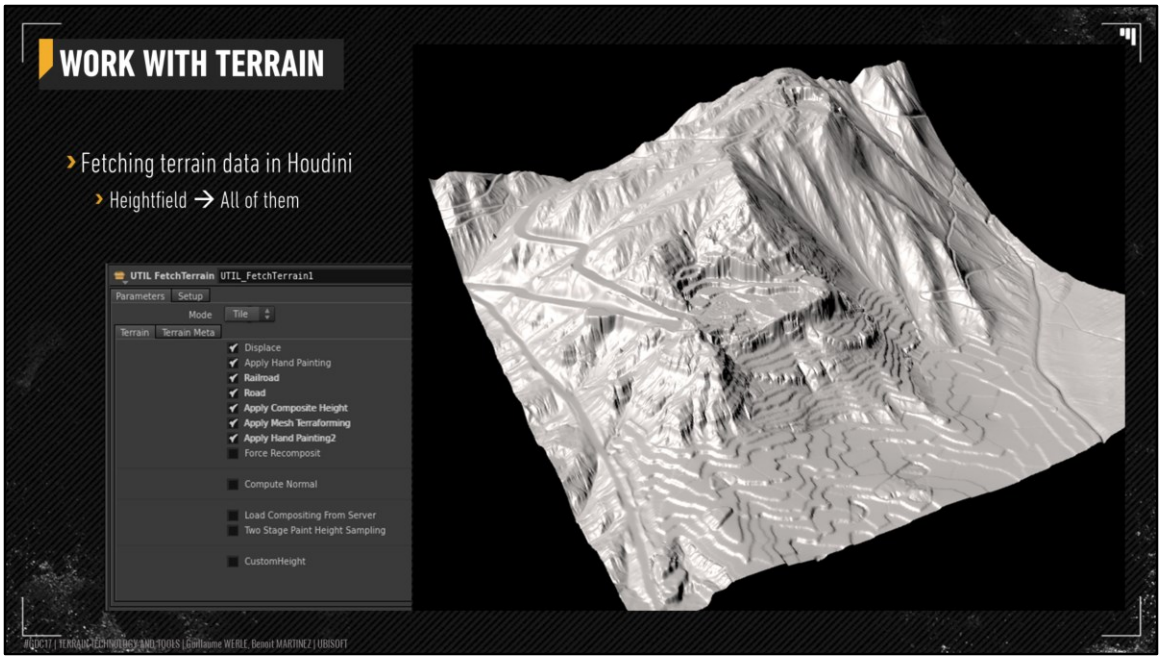
Why using Houdini and not C++ / GPU tools ?
- Fast iteration
- No compilation time: Tools are HDA (Houdini digital assets) - make changes, fix a bug in a HDA → deploy to the team → immediately available for everyone
- Performance? Of course dedicated tools would be faster but the flexibility of Houdini can't be beaten.

The Houdini Team :
- 1TD (myself) and 3 technical artists have created all the tools and setup the pipeline
- 30 world Builders (level artists and level designers) have been using the tools during Wildlands production

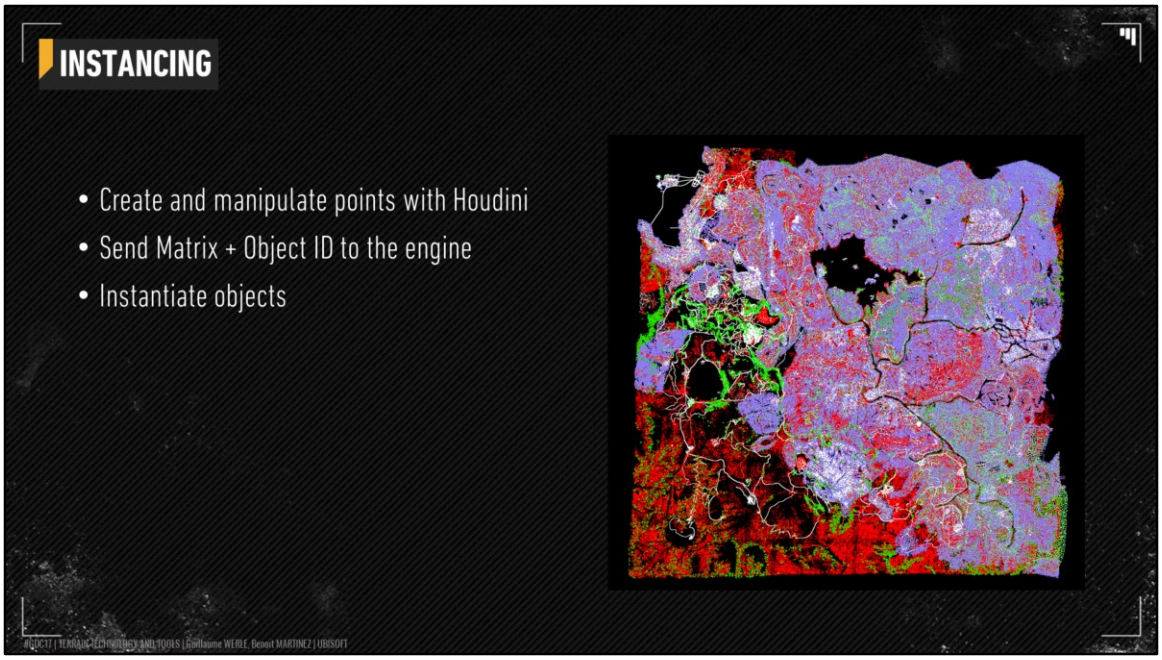
What SideFX product de we use ?
- Houdini (DCC app) → create HDA/Tools
- Houdini engine → Houdini Core API integrated in our game editor. Use HDA/Tools in the game editor
- Houdini Batch (render farm) → compute jobs on the farm



The terrain is literally the ground for everything else.
The first thing we had to do was to create a custom node to access terrain data,
Here is an example for a terrain patch (500m) applied to a grid in Houdini
As you can see we have several options available.
Here we are fetching only the base terrain layer

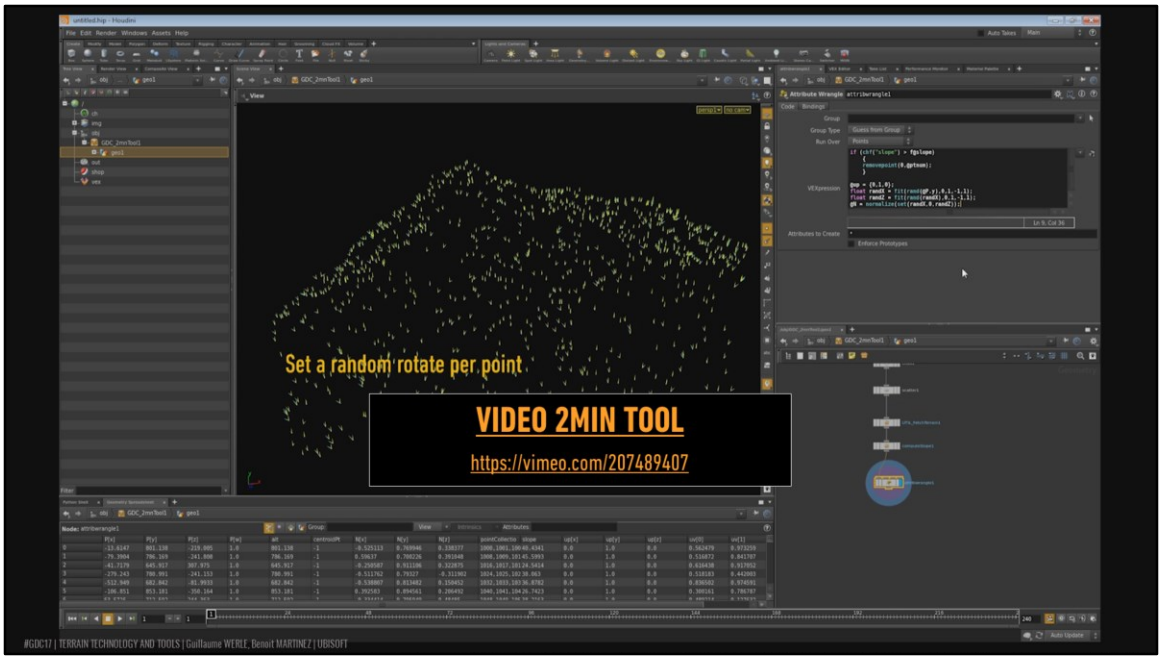


Same but with all layers applied.
Manually sculpted layers (macro and micro) but also generated layers like roads

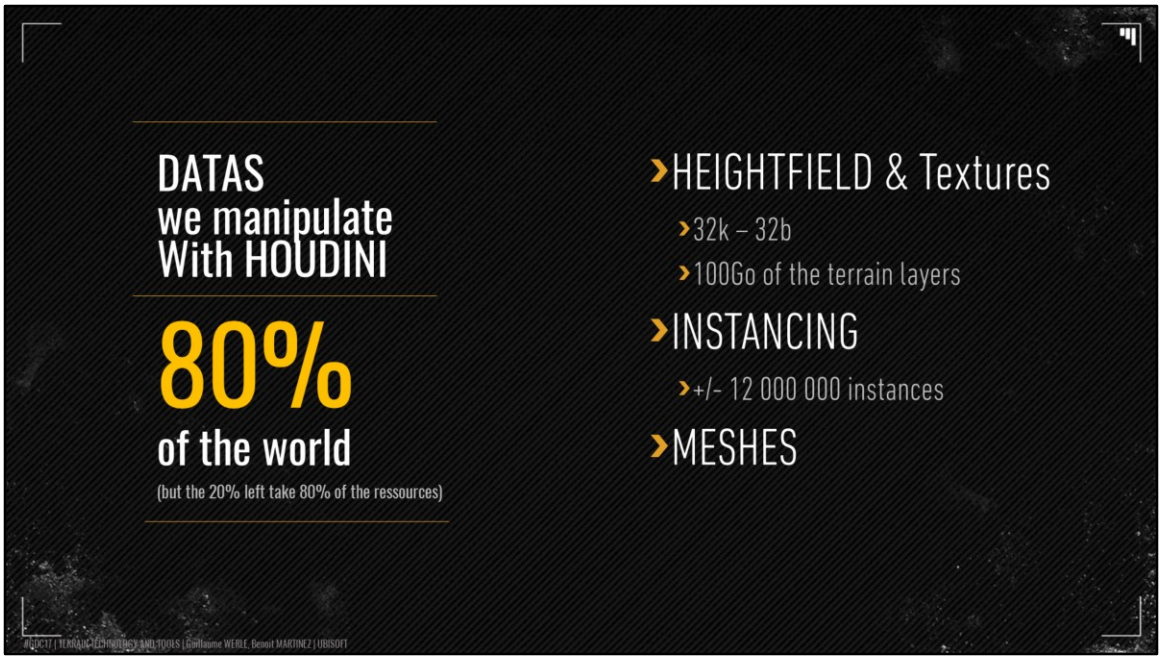


We have many tools to populate a lot of different assets in the world.
The tools do not directly created geometry but help the artist to define placement rules for existing assets.
Most of our Houdini tools deal with points and then just return:

- Matrix
- Existing Object ID



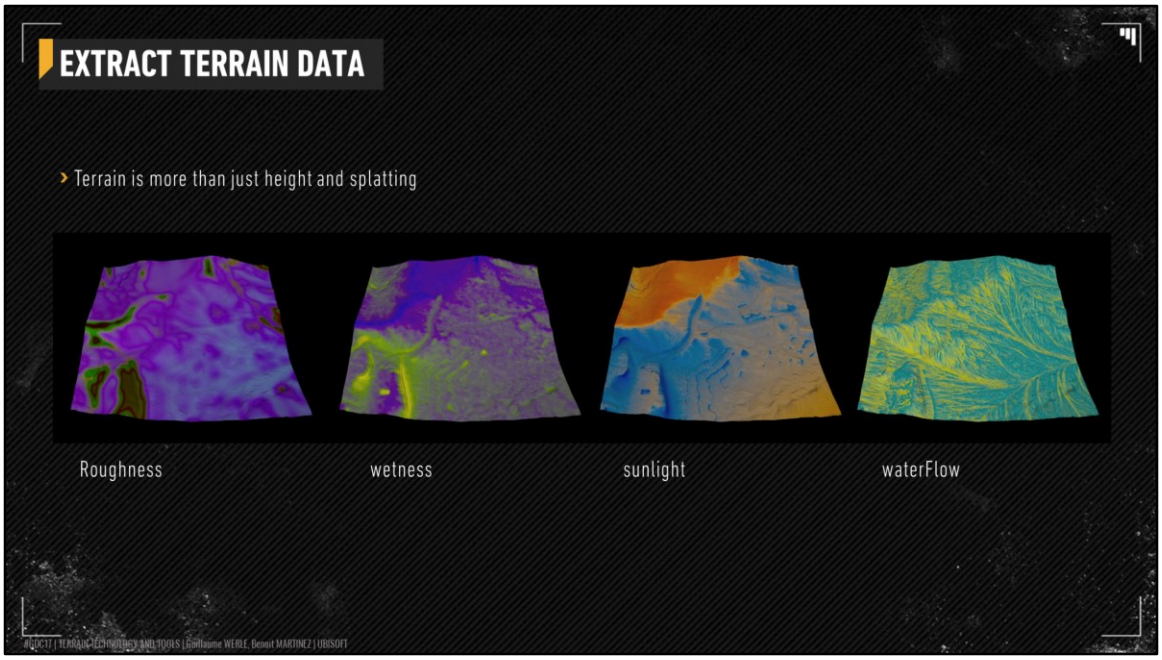
https://vimeo.com/207489407
To illustrate tool creation and how fast and flexible it is, here is a short video I called the 2 min tool



We estimated that 80% of the data has been created or manipulated using Houdini,
Since they are based on rules Tools are good at maintaining consistency,
Allowing to iterate on large sections of the world tools save time for the artist to focus on tasks which have greater value and couldn't be done using automated tools :

- Consistent big picture
- Give meaning to the world
- Tell story thought the environment

(Benoit Martinez)
Terrain is the very first layer, but lets go into detail of the other layers we build on top of it



Terrain is more than elevation and splatting (textures).
We extrapolate very useful information that we'll be able to reuse in many others tools.
A few examples :

- From the elevation we can extract roughness, detect crest, it will be useful to place rocks or vegetation,
- From the river, materials and some specific objects we can define the wetness and then use it in the vegetation rules or to affect the terrain specular.
- Averaging the sun we can get a useful mask to drive vegetation and ambient sounds.
- The initial water flow mask came from world machine and we use it to affect the terrain splatting. But since the terrain is sculpted and modified we want to maintain a waterflow mask up to date.

We automated process on the farm to update those terrain layers :
Every time a user submits some terrain topology change it triggers a recomputed on the Houdini farm.
The updated files stored on a server are accessible instantly by the Houdini tools.
Since it's an intermediate data and it can be updated any time, we don't have any version control for this.
No version control and no backup may sound scary but we never experienced any issues during the production.

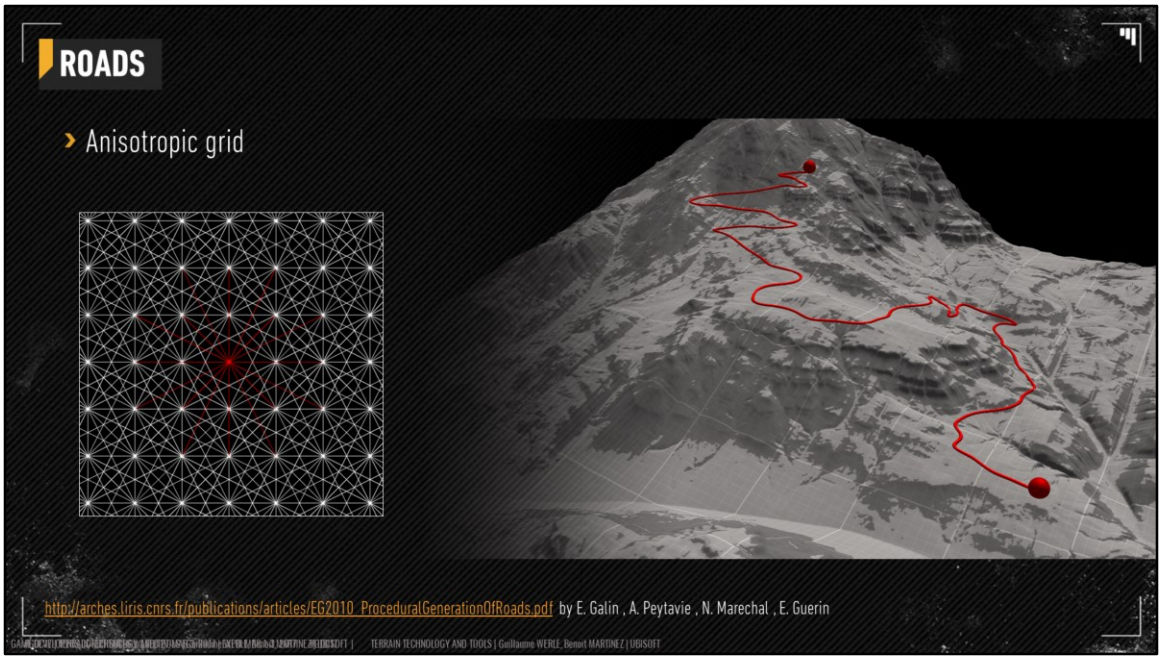
https://vimeo.com/207479343

Before getting into specifics details of our tool set a short video to demonstrate the procedurals layers.

- Roads generation and procedural proposing along roads
- Paths in the wild, vegetation, rocks. The 2 last big rocks are manually placed everything else is procedural, including the carving of the river, the water mesh and the vector flow.
- Even villages are generated, from the terraforming to the building's placement.
- Train tracks with tunnels and bridges
- And it works the same globally, connecting all the roads, computing the flow Map mask, placing vegetation and villages.


ROADS
› Define waypoints
› Pathfinding to connect
› Guarantees consistent rules
2013 – initial result
#GDC17 | TERRAIN TECHNOLOGY AND TOOLS | Guillaume WERLE, Benoit MARTINEZ | UBISOFT

First tests with pathfinding were promising but not good enough.

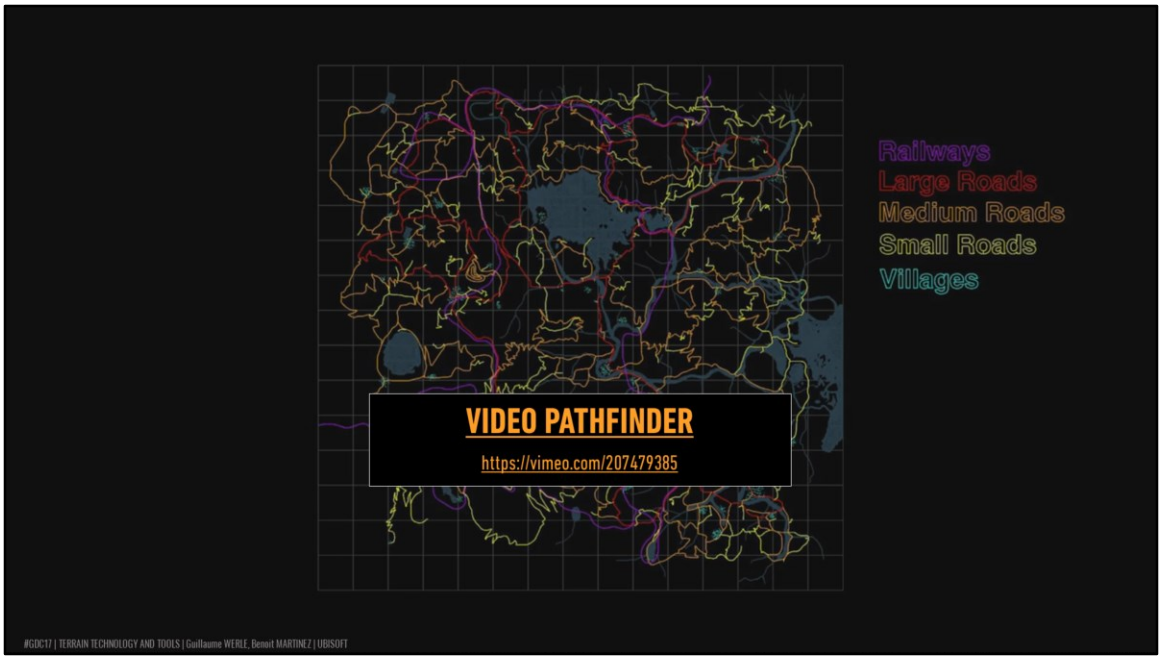


Following this paper :
http://arches.liris.cnrs.fr/publications/articles/EG2010_ProceduralGenerationOfRoads.pdf
We started to experiment with weighted anisotropic shortest path algorithm.
Instead of looking in 4 directions the key was to look in multiple directions and multiple distances for each direction.
It was just what we needed to get to the result we were looking for.

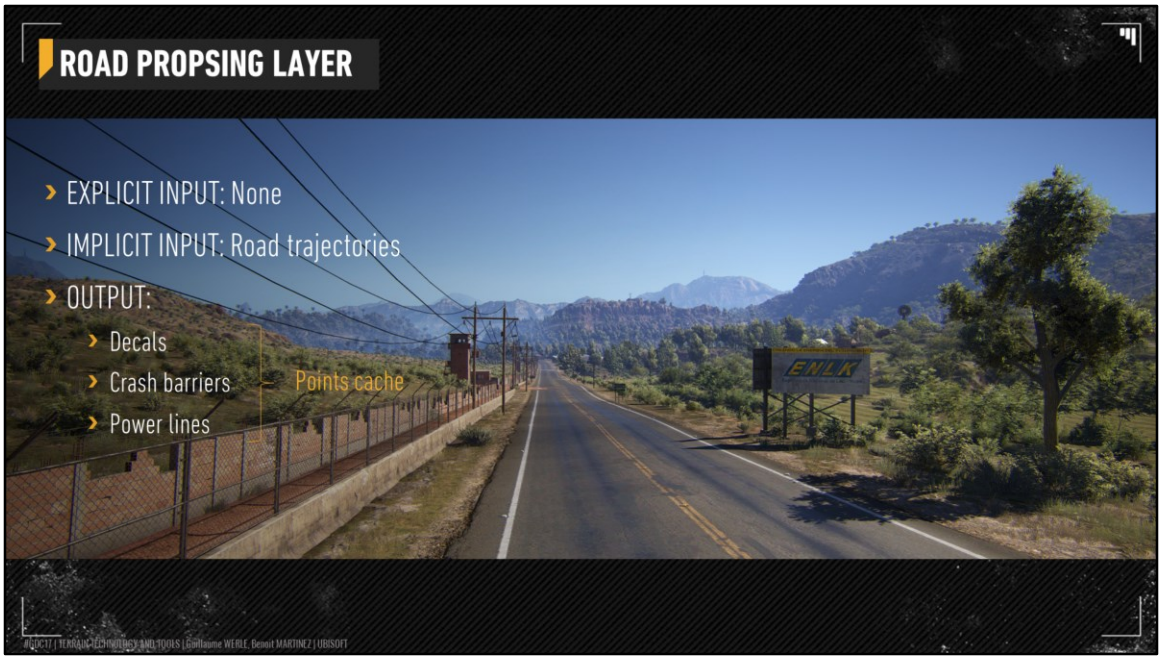
Placing a few way points on a terrain …

We obtain a consistent network with different types of roads and smallest paths.
Remember that it is all offline processed. Computing time is not (or not really) an issue (about 10min to compute a 2km*2km tile).

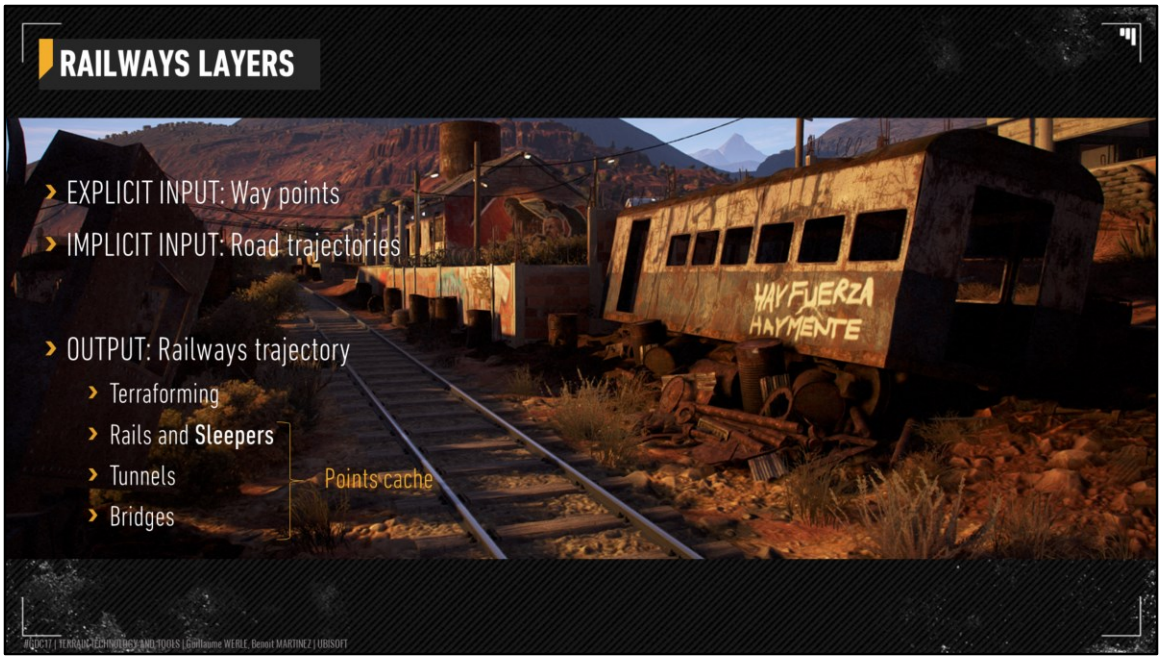
The road outputs are:
Road trajectories: we can reuse that for many purposes likes AI traffic
Terraforming: a heightfield texture to stamp the terrain
Splatting: a texture mask to update the terrain materials
Bridges: we allow the pathfinding algorithm to go over a river if it is the “best”/ less costly way.



https://vimeo.com/207479385
Video of the pathfinding over a tile of terrain and the full extent of road network (without small paths only main roads).



We had tools to manually create decals, power lines, fences and crash barriers then when we managed to create a full and convincing road network we realized that we could just input the road trajectories from the road network into those tools and get all the road propsing for “free”, with no manual placement.

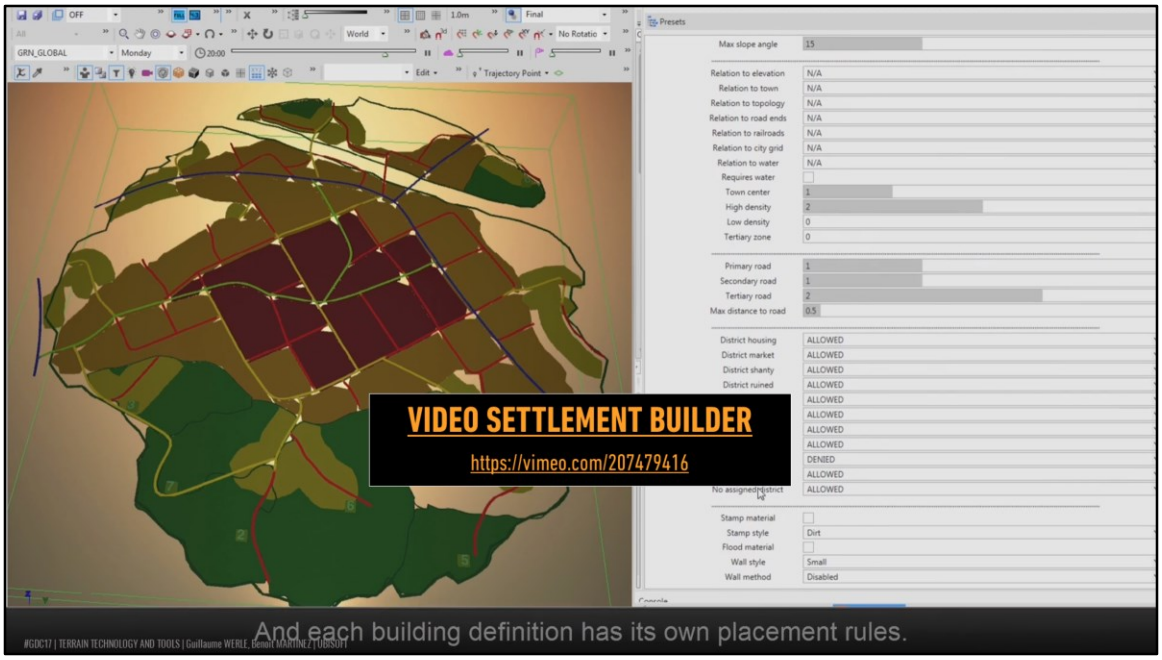


The railways work the same as the road tool, using pathing, but with slightly different rules. Lower slope, less and larger turns, and so more bridges and tunnel.
The road trajectory is an implicit input because the railways need to consider the road so as not overlap (stay parallel in some cases) and cross it always at a 90 degree angle to avoid dangerous crossing and potential traffic issues.

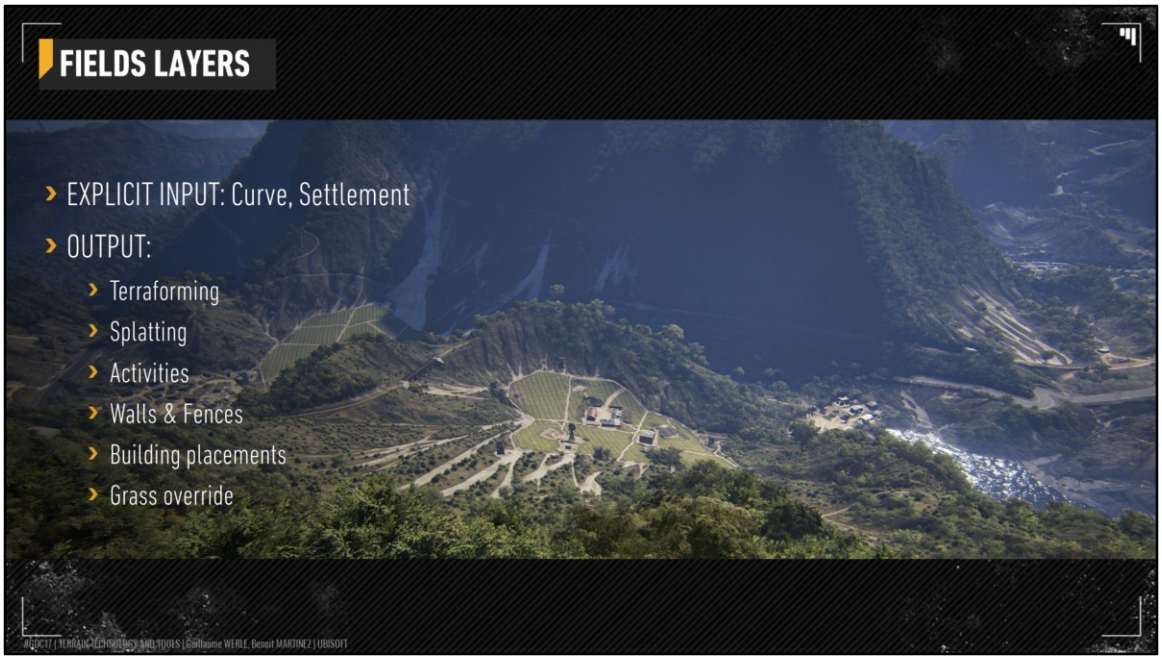
https://vimeo.com/207479400
As previously mentioned some tools, like the bridge tool here, can be used in standalone mode to manually create bridges or can be embedded in a bigger tool to automatically create content.
Note: The bridges and tunnels are made of modules. No new or specific geometry is created, its all just instancing.

16 km2 of the world is covered by water.
It's one unique continuous mesh split into tiles (per km2).
Water is about lakes, rivers and streams.
Rivers and lakes which are main features of the terrain can be sculpted using the terrain tool.
Streams on the other hand are numerous and are created using path finding to connect hilltops to rivers (then rivers connect to lakes).
For the terraforming we defined some layering order so the streams go under the roads, placing pipes but for the rivers we would favor bridges to keep river continuity,
Since we generated the trajectories for the streams (and rivers in some cases) we can reuse that information to define the vector flowmap. The flowmap is then stored in the vertex color.

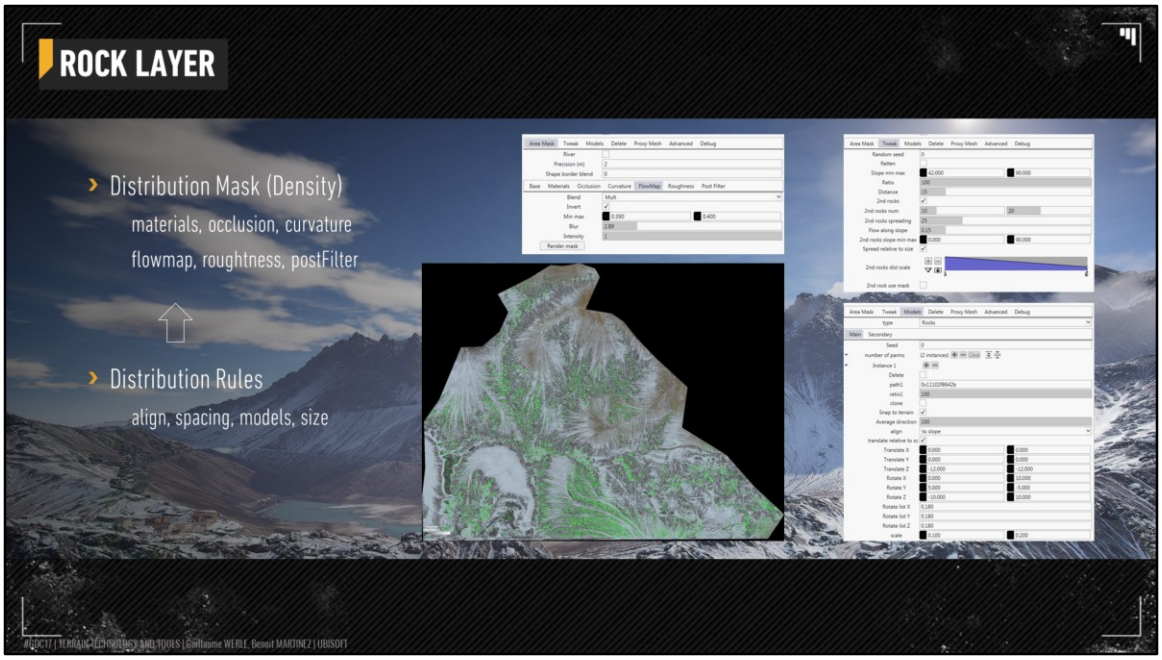
Settlement Builder is one of our most complex tools.
Given a center, a list of buildings and some parameters the goal was to fully generate villages for the game.
It took some time to get a stable, useable tool but we finally achieved the quality we were looking for.
We managed to lock the procedural villages early enough so artists were able to do a manual pass to add detail and specifics in each.



https://vimeo.com/207479416



Fields tool can be used as standalone to create isolated fields or it can used as settlement as an input to create fields in the surrounding areas.



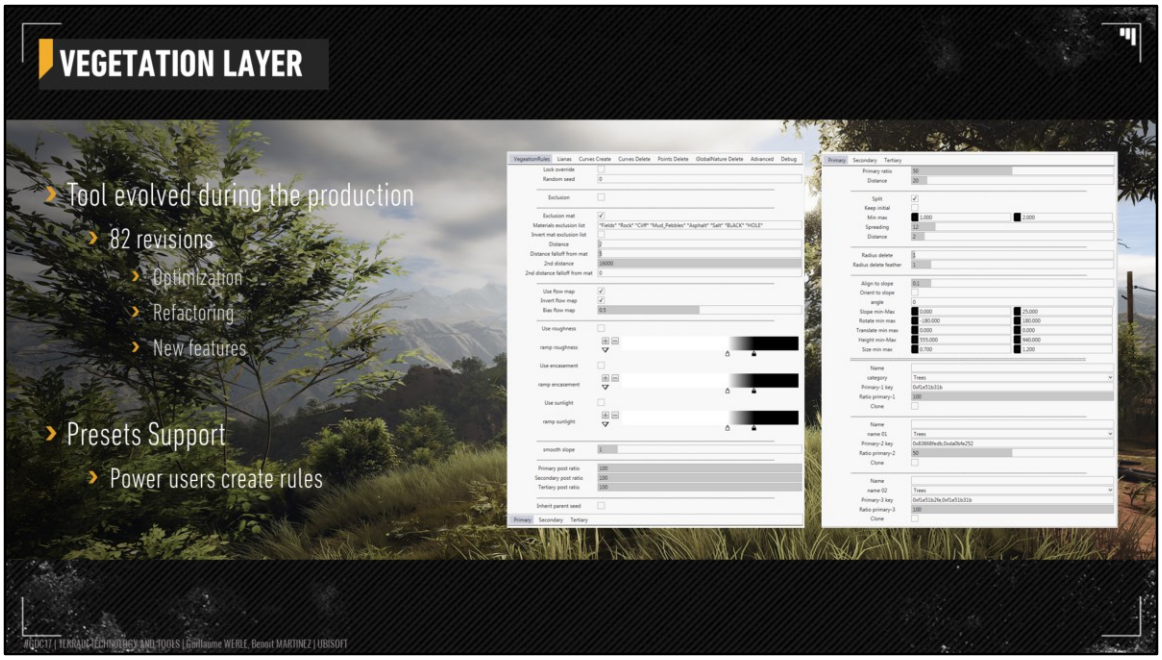
To distribute rock we use a conventional approach:

Define a distribution mask – the artist can previz the mask directly in the editor,
- Where the distribution rules apply
- Material – Occlusion – Curvature – Flowmap – Roughness – postFilter

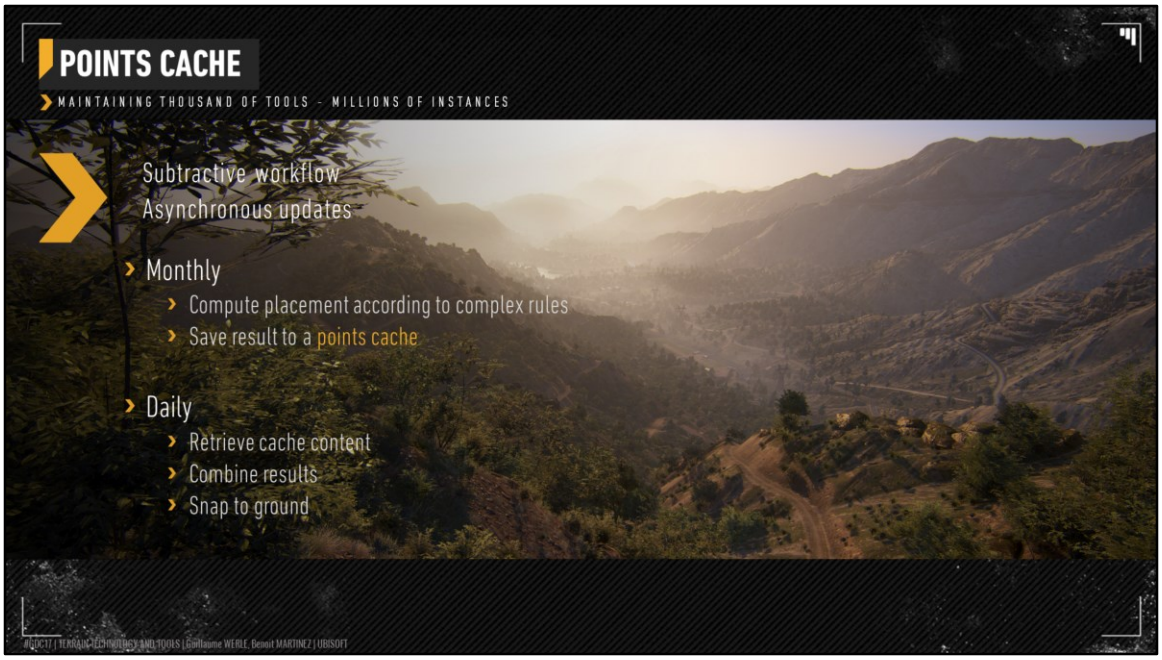
Create distribution rules that are going to apply inside the mask
- Models
- Slope relative alignment
- Scale factor
- Density / min distance

If needed the distribution will consider results from other tools to avoid rocks overlapping roads, buildings etc

Basically the vegetation tool works like the rocks tool, still it's a specific tool with it's own set of parameters.
Complex multipurpose tools have proven to be complex to maintain and less efficient. In all cases we rather like to have dedicated tools.



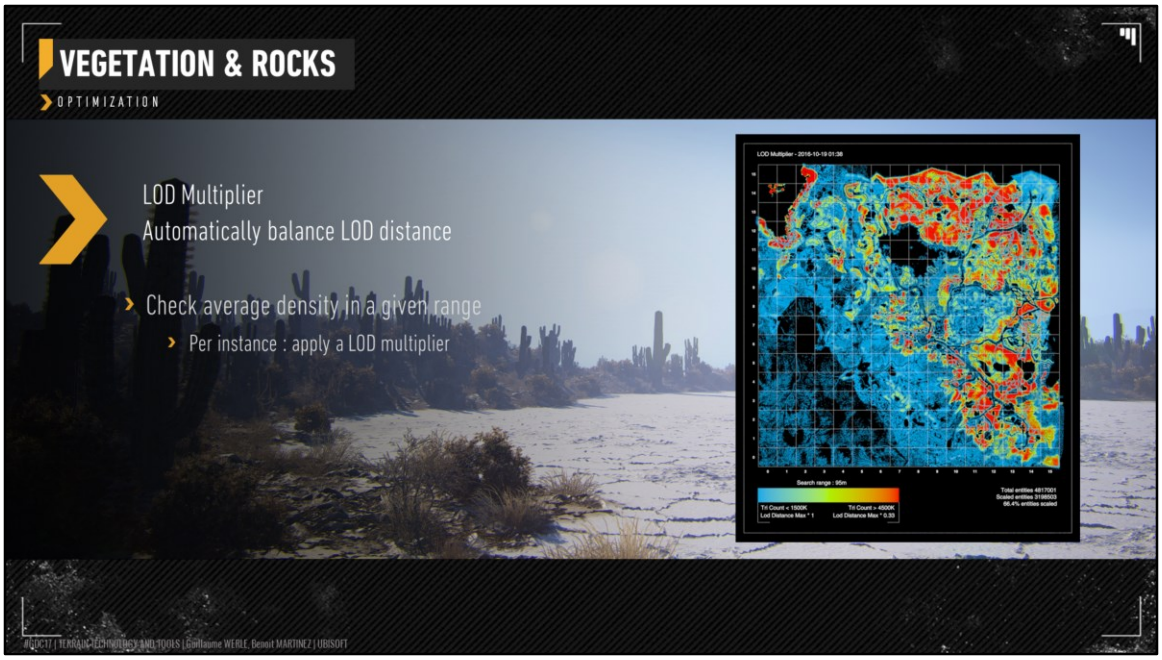
We were able to evolve the tools all along the production.
To avoid every possible user to have specific requests we decided to have an owner per tool. Usually an artist who's going to collaborate with the technical artist who's going to create the tool.
It's also the owner who's going to document the tool, to make a comprehensive user friendly doc rather than a technical documentation.
We also integrated a preset support so the owner can create sets of rules and most users would then just reuse existing presets without having to fully understand and use all the parameters,



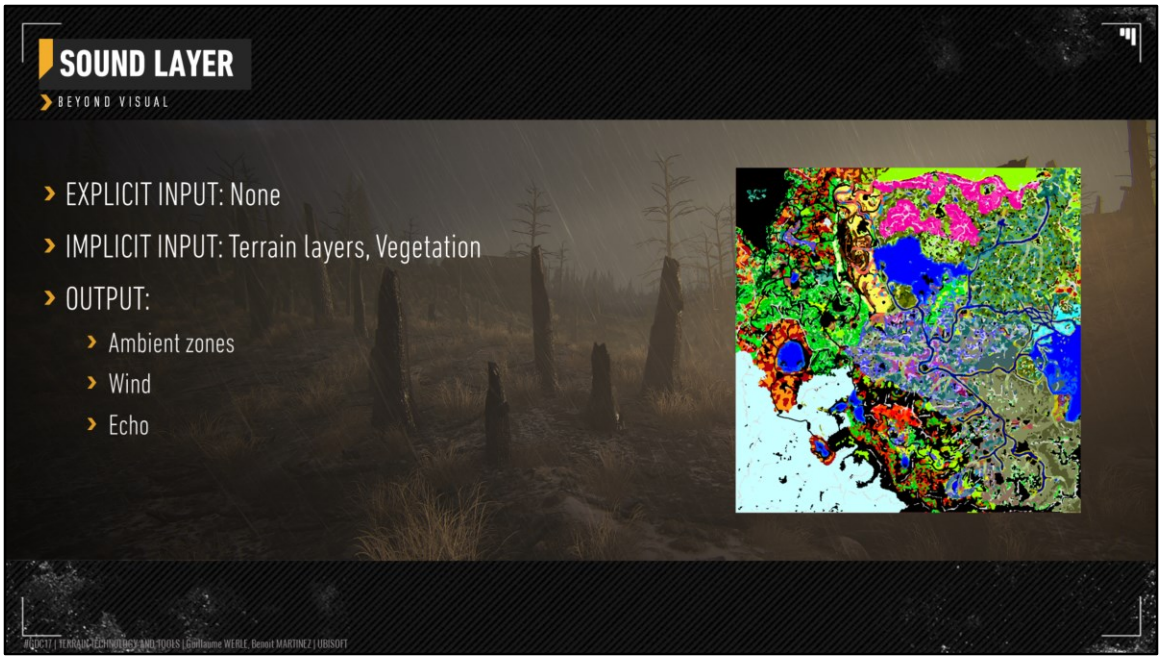
Subtractive workflow :
It's very complex to mix procedural content and manual control.
If someone want a fine manual control it should just be done by hand. All the tools support exclusion areas using manualy defined curve inputs, or implicit input (ex : exclusion vegetation from roads).
With this workflow artists in charge of vegetation or rocks placement don't have to consider villages, roads or anything. They can just freely create rules for the biomes, the necessary exclusion being done in a later integration process.

Asynchronous updates :
We need to guaranty an always playable world but it doesn't mean it has be accurate.
Then if the terrain topology change we don't necessary need to fully recompte the vegetation placement, most of the time we just need to be sure that nothing block the roads and everything snap to the terrain.
Remember that all those tools have been used independently without any constraint of other objects' placement and also they have been used on the raw base terrain.
Also we have a "point cache". All the points from each tool.

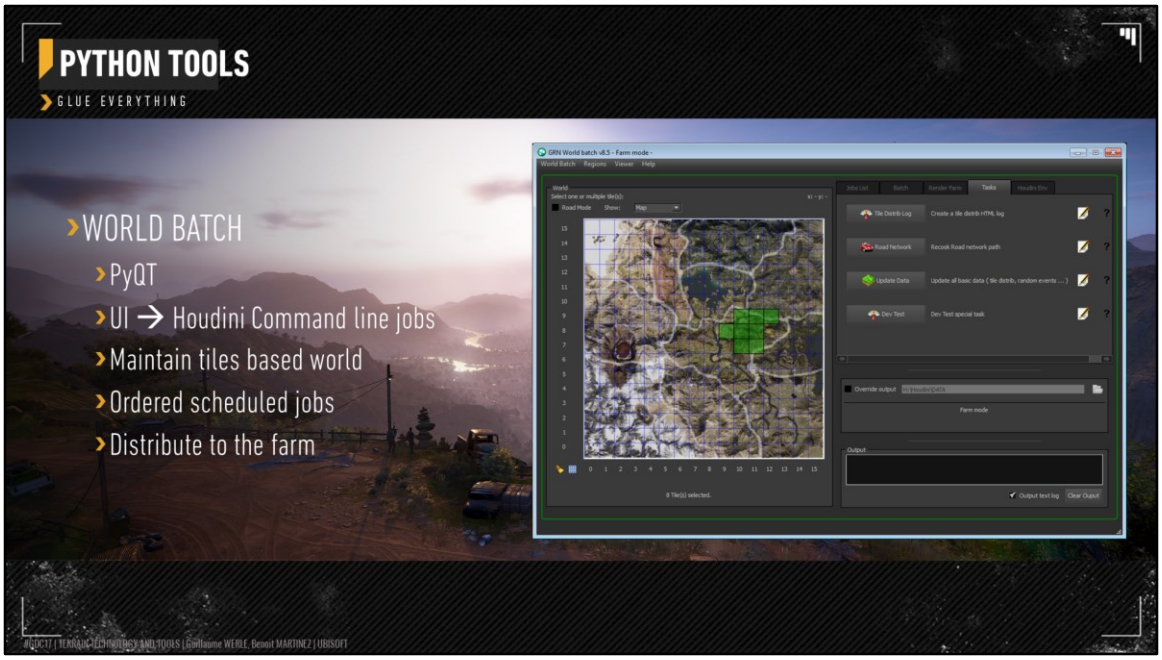
- First thing in the compiler is to check instances against terrain
    - Snap everything to terrain (preserving any relative terrain offset – think of rocks that are half buried)
    - Delete unwanted object for procedurally terraformed area
        - Remove vegetation from roads, rivers, settlement, etc
- Then we check all instances groups against each other.
    - Bridges and tunnels would delete rock
    - Rock will delete vegetation
    - Powelines can not go thought trees
    - And so on…



Since we have access to all the data in Houdini we've been able to go even further than placement of objects.
Here we check the object density (vegetation and rocks) and balance a per object LOD multiplier accordingly.



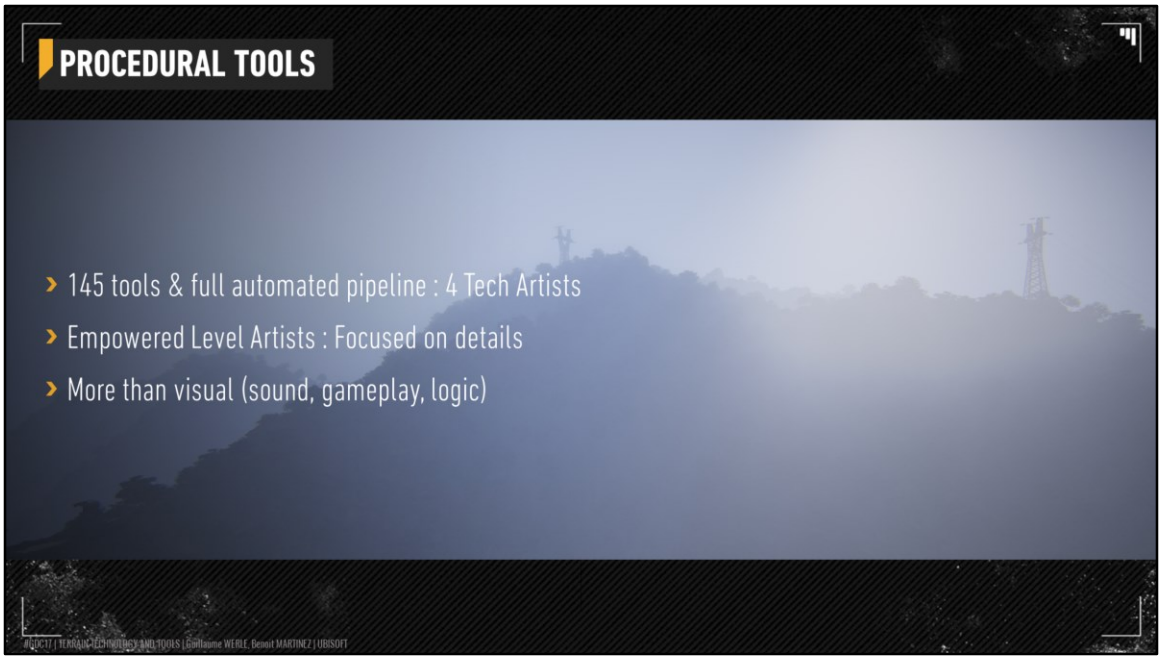
Beyond visuals we've been able to work with the sound designer.
We gave them a tool to define rules according to the type of objects and the terrain to generate a sound map.
Each color being an index for a specific ambient sound.



Worldbatch helps to do this task automatically or allows you to update any kind of data used by the world at any time.

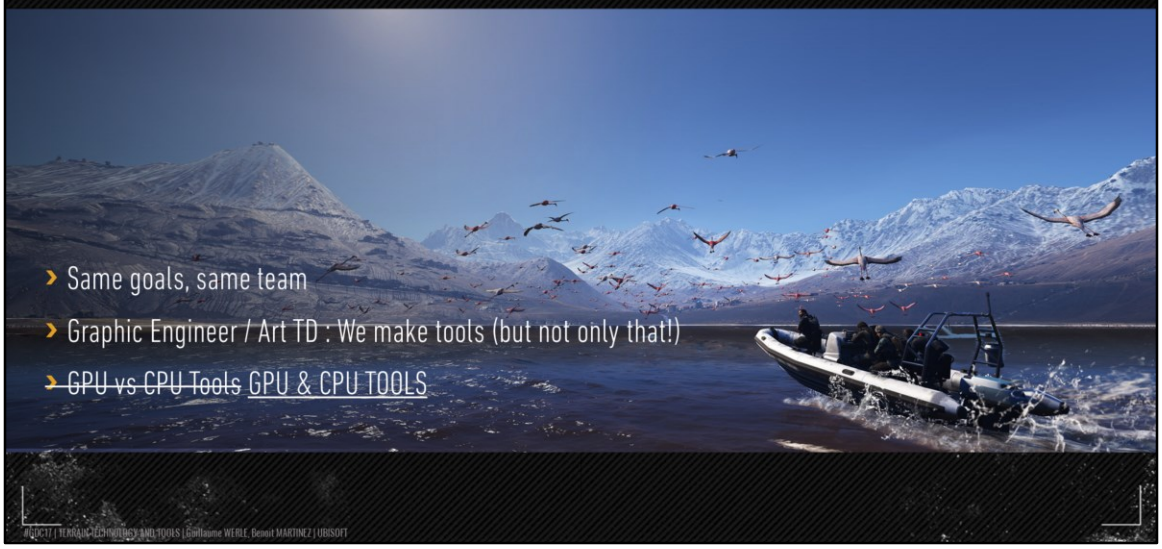
Worldbatch is based on Houdini python (hython) and uses "Hqueue" jobs distribution system (created by SideFX) to launch a data "job" on a "build render farm".
A job loads a Houdini asset (.otl) and sets parameters if needed and launches the data rendering process from that otl.
It can be used on its own, in UI mode, by a user who could choose what he wants to recompute directly from the world map, and send the job either on its own computer, or to the render farm.

I mentioned before about the Perforce addin, worldBatch will also be also launched automatically when a user submits a change list in Anvil involving the terrain, Worldbatch will be used by a script in command line and will send basic jobs to the farm in order to be sure data is always up to date even if the terrain changed.

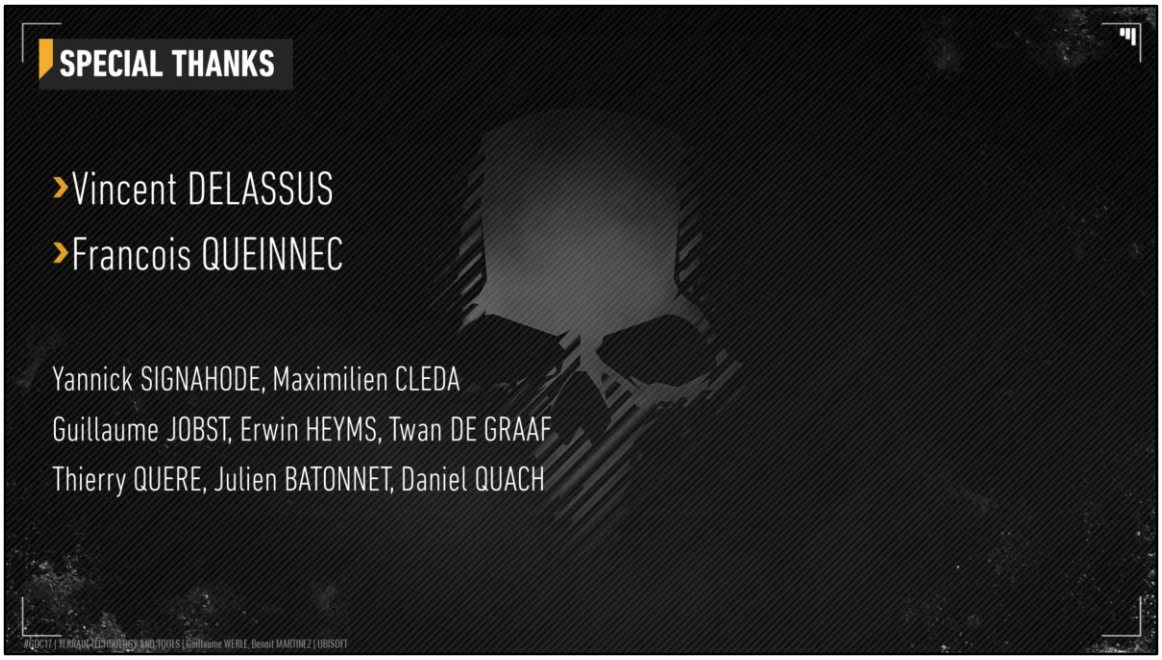

PROCEDURAL TOOLS
› 145 tools & full automated pipeline : 4 Tech Artists
› Empowered Level Artists : Focused on details
› More than visual (sound, gameplay, logic)
#GDC17 | TERRAIN TECHNOLOGY AND TOOLS | Guillaume WERLE, Benoit MARTINEZ | UBISOFT


CONCLUSION
TERRAIN & HOUDINI TOOLS
#GDC17 | TERRAIN TECHNOLOGY AND TOOLS | Guillaume WERLE, Benoit MARTINEZ | UBISOFT

CONCLUSION



- Same goals, same team
- Graphic Engineer / Art TD : We make tools (but not only that!)
- ~~GPU vs CPU Tools~~ GPU & CPU TOOLS



Special thanks to:
Vincent DELASSUS – Art Director and Art Technical Director
Francois QUEINNEC – Lead Graphics Programmer
and a few key people:
Yannick SIGNAHODE, Maximilien CLEDA
Guillaume JOBST, Erwin HEYMS, Twan DE GRAAF
Thierry QUERE, Julien BATONNET, Daniel QUACH,

Carreers.ParisStudio@Ubisoft.com

@666uille
guillaume.werle@ubisoft.com
Blog

@_BenoitMartinez
benoit.martinez@ubisoft.com