Advanced Heightmap Compression Using Deep Learning in Dune: Awakening

ML modelling and deployment

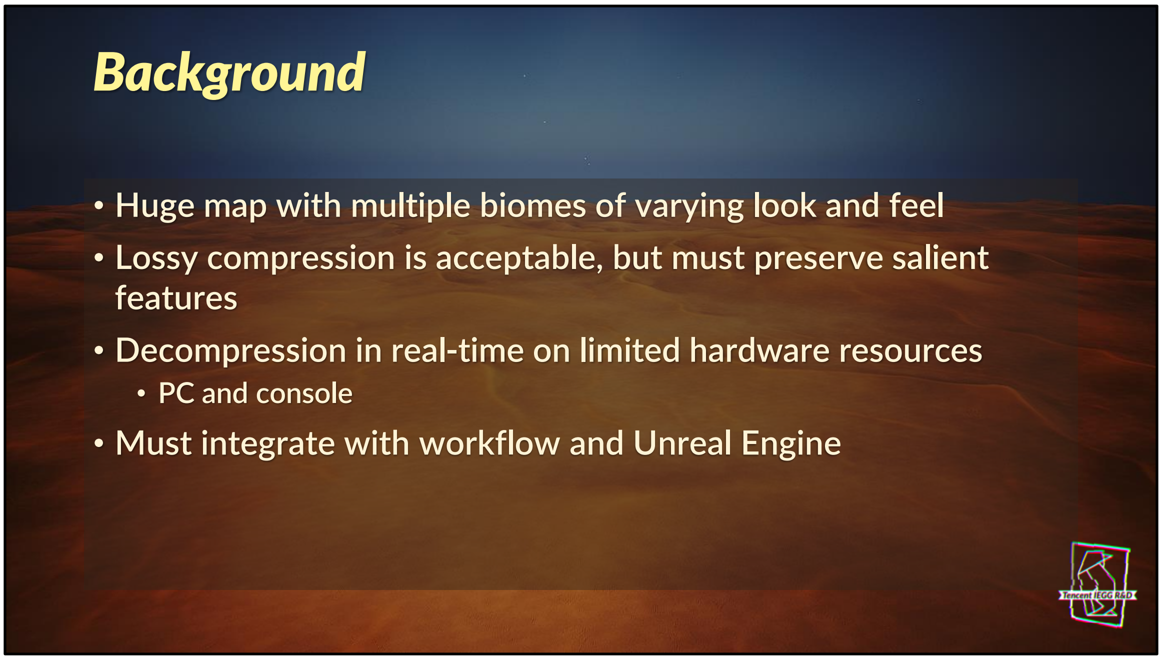
Calle Lejdfors
R&D Director

Presenting for Antonio Martini, ML expert

DUNE
AWAKENING
Tencent IEGG R&D

Important: no images from the upcoming game, from here on out it's "coder art" all the way

36x compression



Background

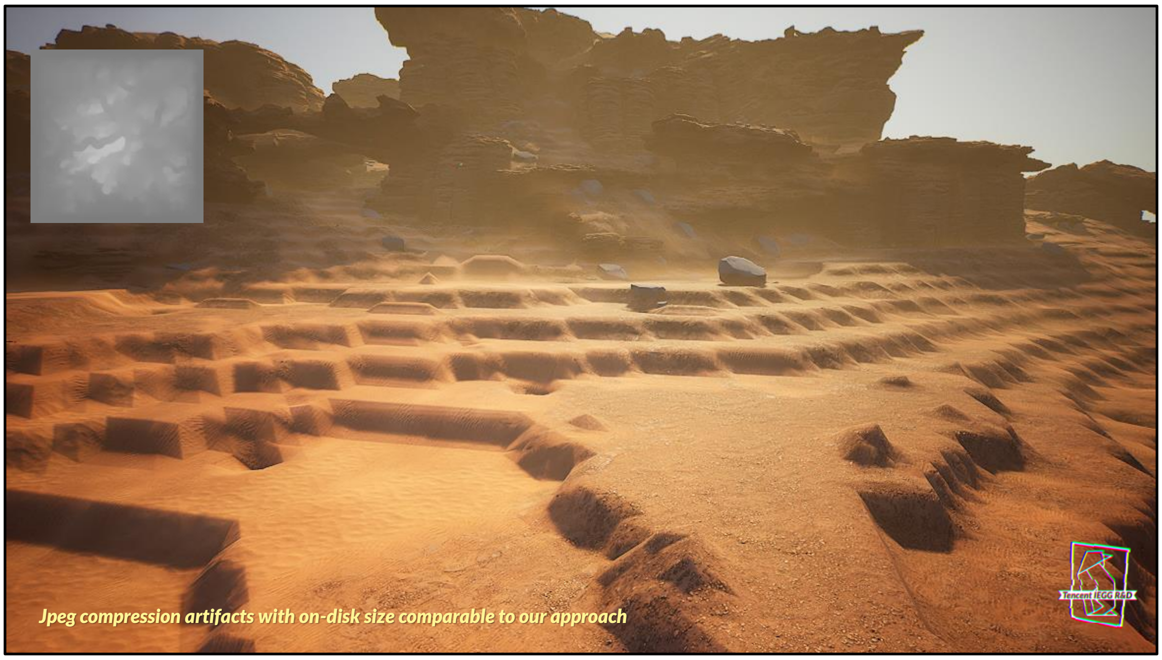
- Huge map with multiple biomes of varying look and feel
- Lossy compression is acceptable, but must preserve salient features
- Decompression in real-time on limited hardware resources
  - PC and console
- Must integrate with workflow and Unreal Engine

Tencent IEGG R&D

Solution not tied to Unreal, just integrated

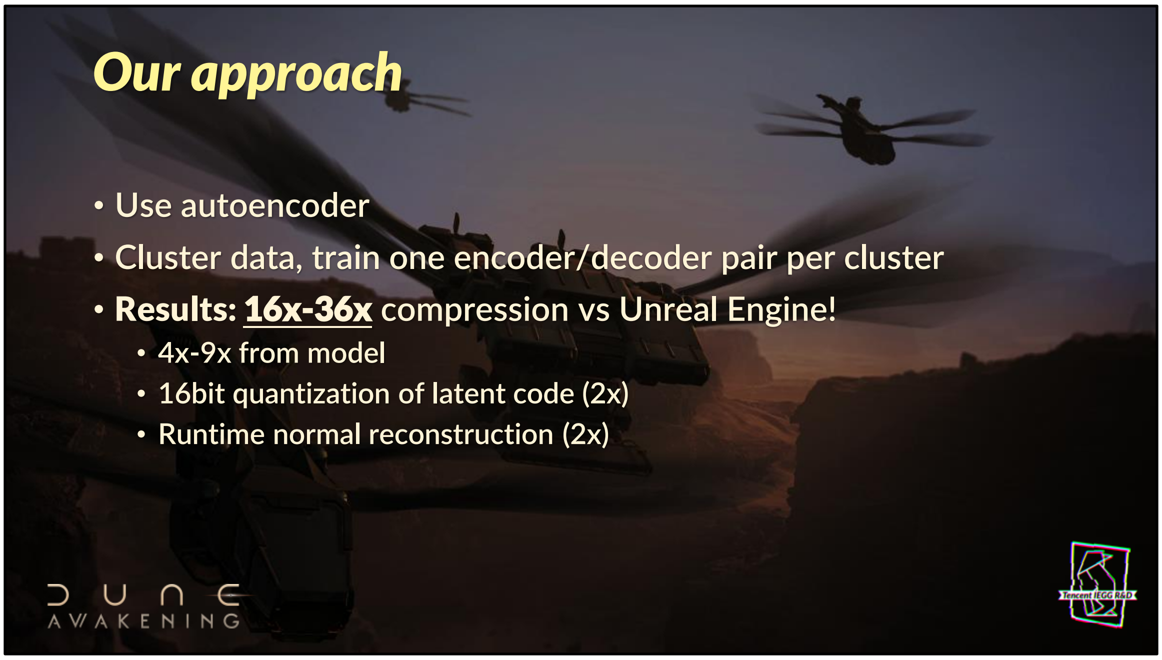
Why not use image compression?
Tencent IEGG R&D

- Image pixels have fixed size, color and point of view
- Heightfields can be observed from any viewpoint
- Materials, lighting model and light sources all contribute to visuals
- Contrast against background further amplifies artifacts
- Compression artifacts will be amplified by the above variables
- Perceptual metrics for image compression don’t transfer to heightfield compression


Jpeg compression artifacts with on-disk size comparable to our approach
Tencent IEGG R&D

This is using jpeg export from Gimp at 90% quality resulting in ~33x compression rate in on-disk size.

- Artifacts are largely related to quantization
- Also non-linear edge response due to perceptual model



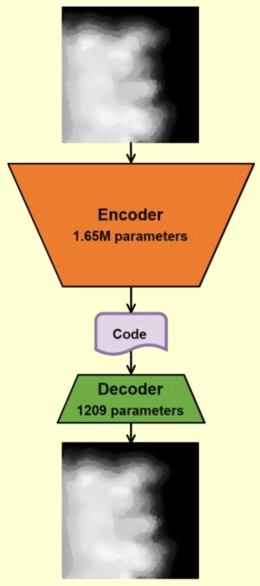
Our approach

- Use autoencoder
- Cluster data, train one encoder/decoder pair per cluster
- **Results: <u>16x-36x</u>** compression vs Unreal Engine!
  - 4x-9x from model
  - 16bit quantization of latent code (2x)
  - Runtime normal reconstruction (2x)

DUNE
AWAKENING
Tencent IEGG R&D

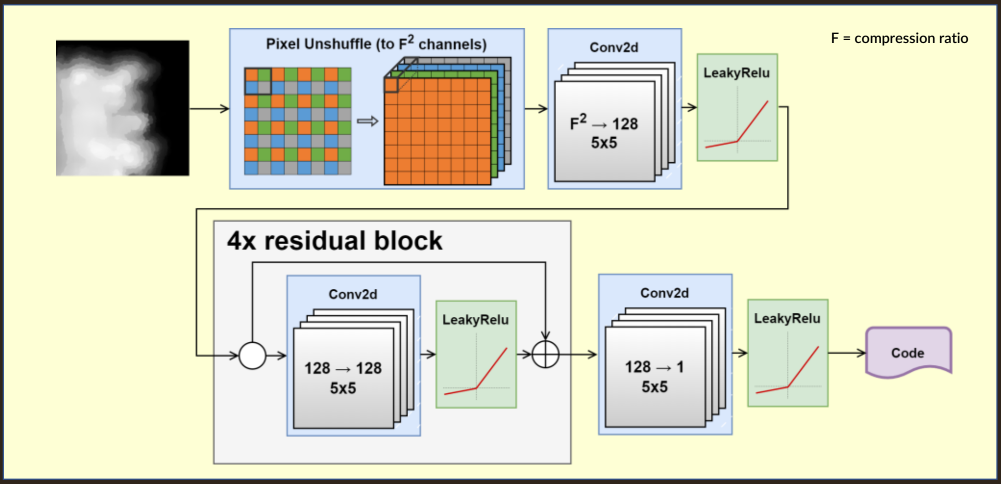
Asymmetric autoencoder

- Convolutional autoencoder with fixed encoder/decoder sizes
  - Encoder is large (1.65M parameters)
  - Decoder tiny and fast (1209 parameters)
- Selectable compression ratio
  - 2x or 3x (to a side $\rightarrow$ 4x to 9x compression)
- All data available at the time of compression
  - No out-of-sample generalization or validation data needed



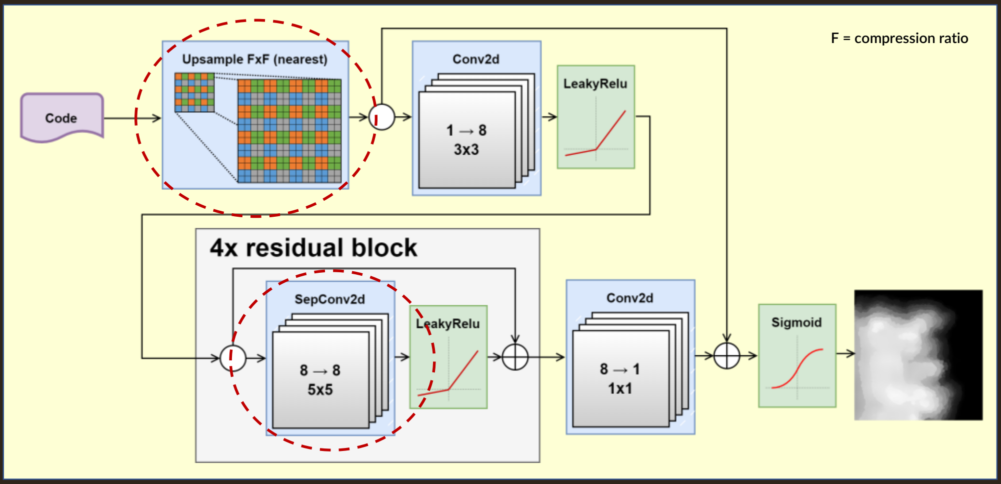
Warning, encoder size not to scale

Encoder



We use LeakyRelu to avoid “dead neuron problem.” See reference section for paper links

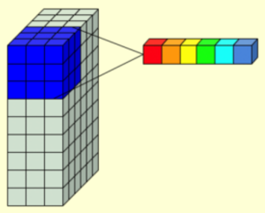
Decoder


F = compression ratio
Code
Upsample FxF (nearest)
Conv2d
1 → 8
3x3
LeakyRelu
4x residual block
SepConv2d
8 → 8
5x5
LeakyRelu
Conv2d
8 → 1
1x1
Sigmoid

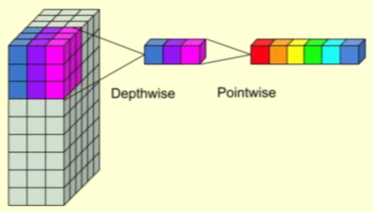
Decompressor details

- Uses depthwise separable convolutions (SepConv2d) instead of Conv2d
  - ~100x fewer parameters: 103393 vs 1209
  - Minimal loss of quality
  - Key to real-time
- Uses upscaling rather than pixel shuffle
  - Shuffling gives blocking artifacts [1,2,3]
- Use LeakyRelu to avoid "dead neuron problem" that occurs with standard Relu [4]

Depthwise convolution



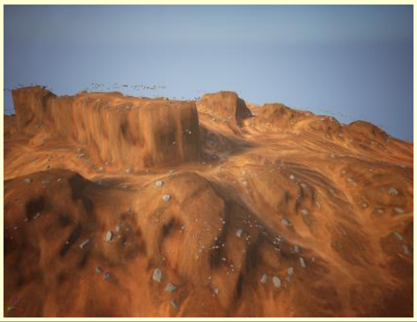
Depthwise separable convolution

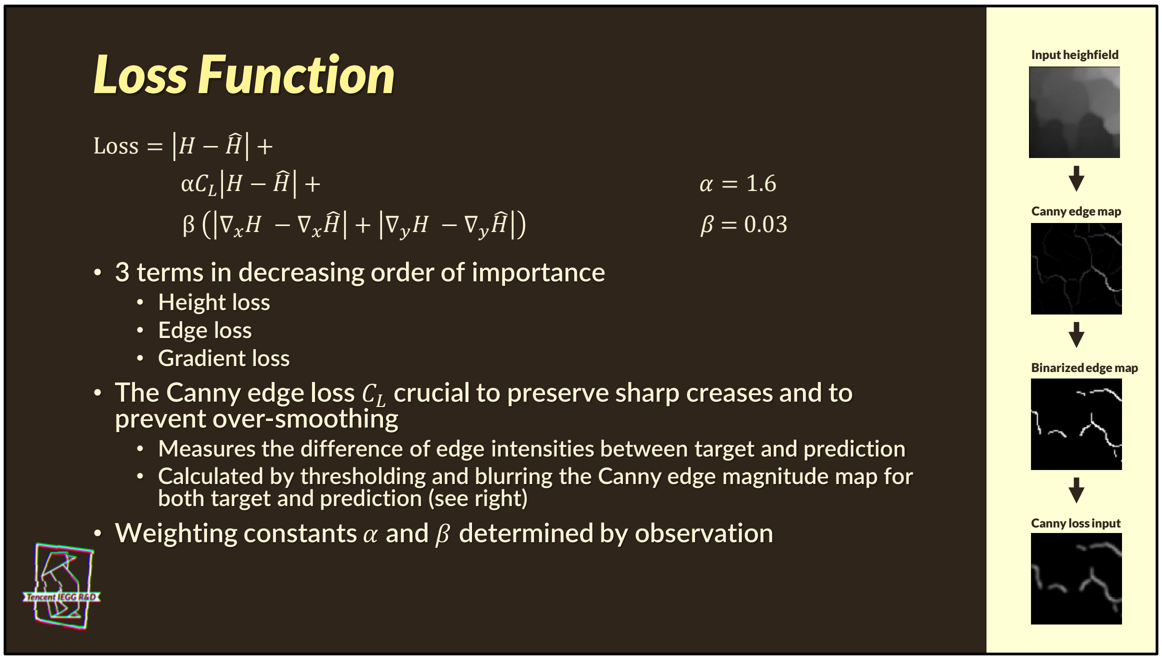


Images © https://www.paepper.com/blog/posts/depthwise-separable-convolutions-in-pytorch/ Used with permission.

Data pre-processing and clustering

- Data partitioned by biome name
- 2 possible clusters per biome
  - Smooth or step-like
  - Enable us to deal with large step-like gradients
  - Greatly reduces ringing artifacts for steep terrain
- Normalization: height data is converted in [0-1] range
- Training data use random sampling of 64x64 patches





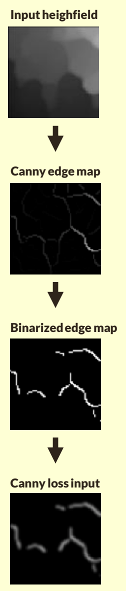
Loss Function

$$\text{Loss} = |H - \hat{H}| + \alpha C_L |H - \hat{H}| + \beta \left(|\nabla_x H - \nabla_x \hat{H}| + |\nabla_y H - \nabla_y \hat{H}| \right)$$

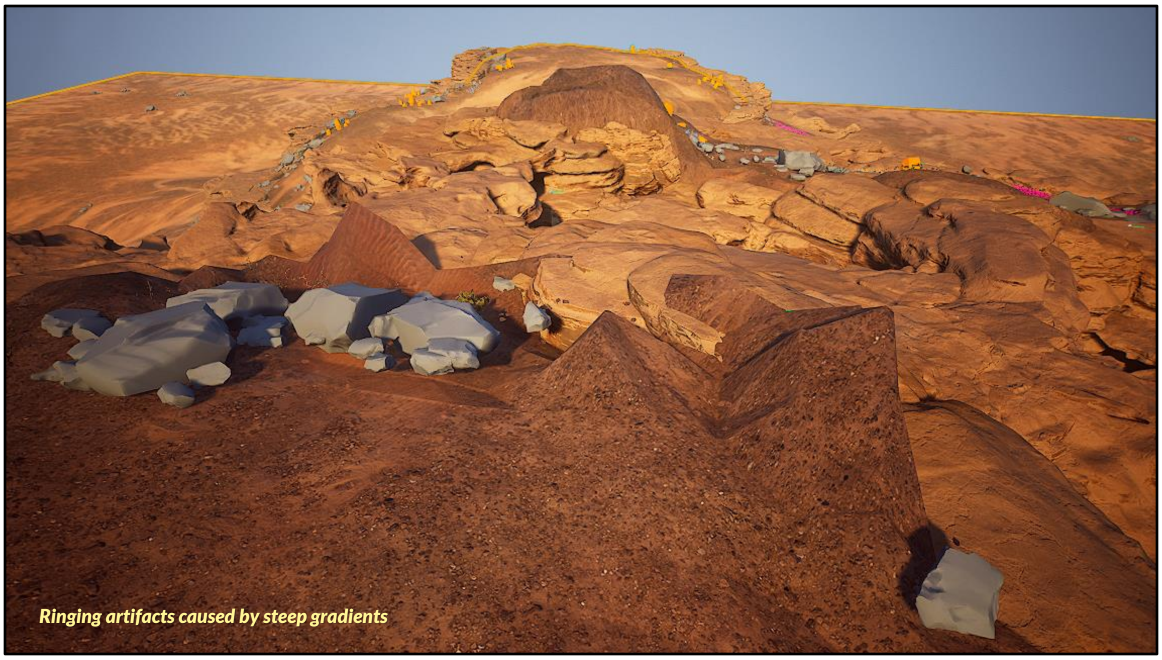
$\alpha = 1.6$

$\beta = 0.03$

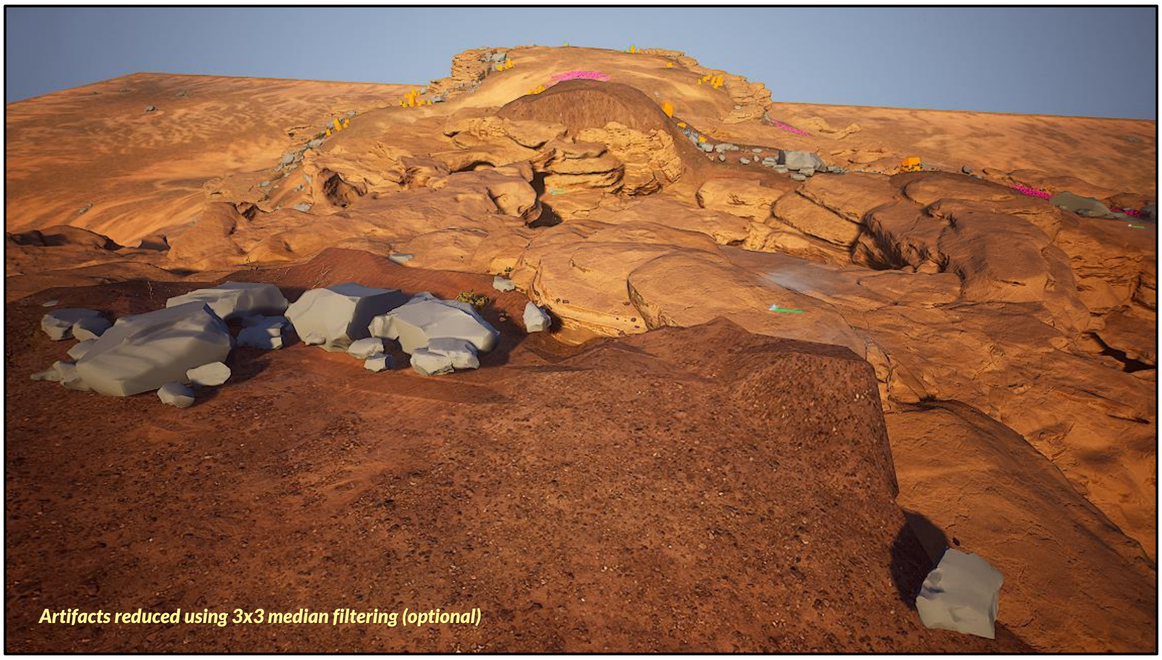
- 3 terms in decreasing order of importance
  - Height loss
  - Edge loss
  - Gradient loss
- The Canny edge loss C_L crucial to preserve sharp creases and to prevent over-smoothing
  - Measures the difference of edge intensities between target and prediction
  - Calculated by thresholding and blurring the Canny edge magnitude map for both target and prediction (see right)
- Weighting constants α and β determined by observation

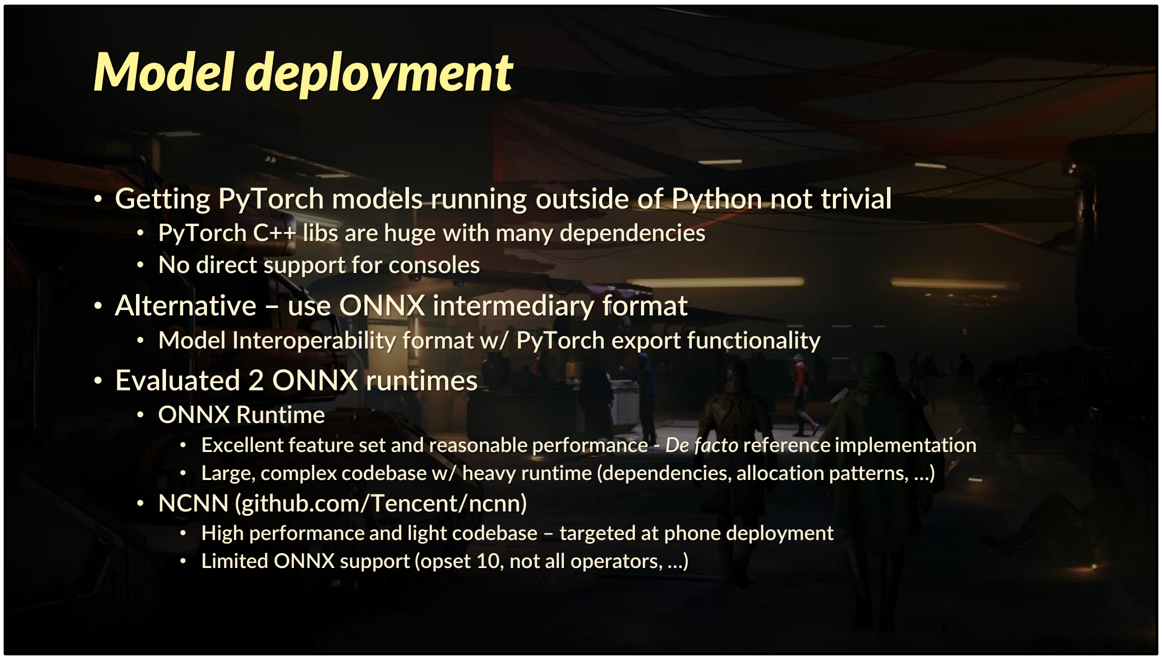


We use Kornia for all image operations. See kornia.filters.canny



Ringing artifacts coming in from emphasizing edge preservation in the loss function when we have step-like landscapes.


Artifacts reduced using 3x3 median filtering (optional)



Model deployment

- Getting PyTorch models running outside of Python not trivial
  - PyTorch C++ libs are huge with many dependencies
  - No direct support for consoles
- Alternative – use ONNX intermediary format
  - Model Interoperability format w/ PyTorch export functionality
- Evaluated 2 ONNX runtimes
  - ONNX Runtime
    - Excellent feature set and reasonable performance - *De facto* reference implementation
    - Large, complex codebase w/ heavy runtime (dependencies, allocation patterns, …)
  - NCNN (github.com/Tencent/ncnn)
    - High performance and light codebase – targeted at phone deployment
    - Limited ONNX support (opset 10, not all operators, …)

Our deployment solution - SeMLa

- Internally developed by the Production R&D team
- ONNX to C++ compiler
  - Generates a single function per model
  - Zero runtime allocations (uses scratch pad)
  - Supports most commonly used operators, easy to extend with new operators
- Very light runtime with no additional dependencies
  - Combination of Eigen and custom tensors and operators
- Supports reasonable compiler transformations
  - Tensor layout swizzling, constant/weight folding, pattern-based operator rewriting/fusion, dead code/data elimination, …

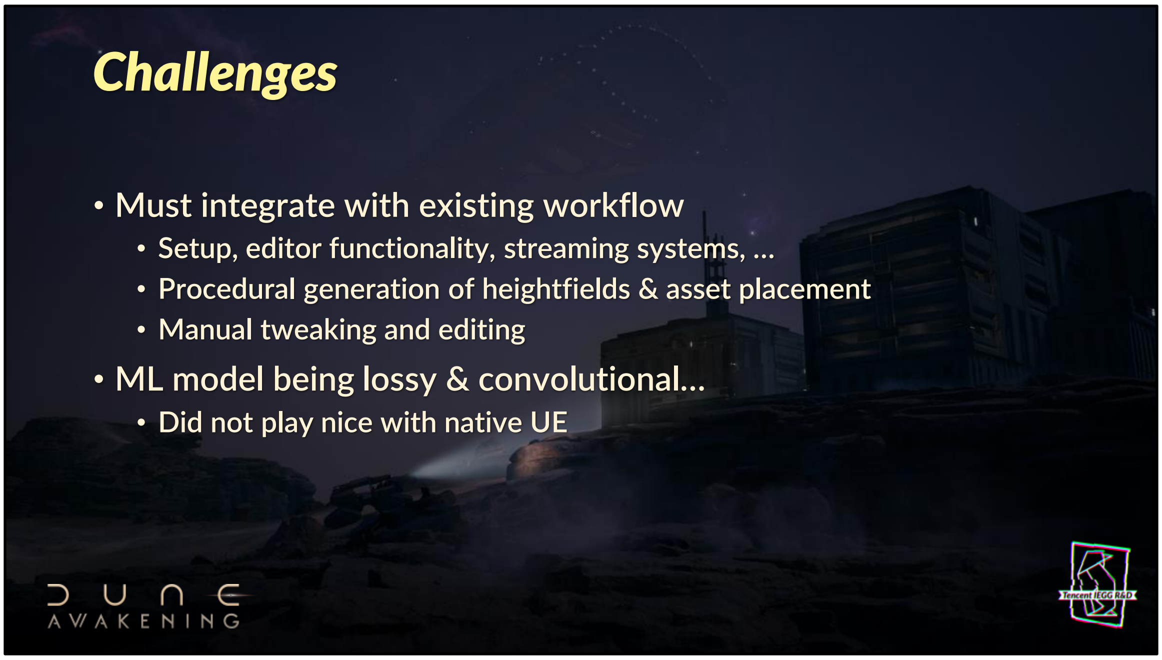
Photo © Visit Stockholm - https://www.visitstockholm.com/eat-drink/cafes/fat-tuesday/

	ONNX Runtime	SeMLa	Speed-up
Heightfield	9.20ms	5.54ms	**166%**
Model 1	28.44us	4.41us	**645%**
Model 2	215.8us	4.73us	**4560%**

Timings from PS5 devkit, single CPU, single thread

Engine Integration

Clemens Rögner
Senior R&D Programmer



Challenges

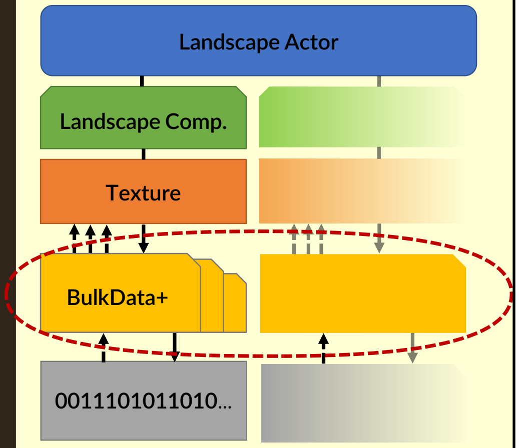
- Must integrate with existing workflow
  - Setup, editor functionality, streaming systems, …
  - Procedural generation of heightfields & asset placement
  - Manual tweaking and editing
- ML model being lossy & convolutional…
  - Did not play nice with native UE

DUNE AWAKENING
Tencent IEGG R&D

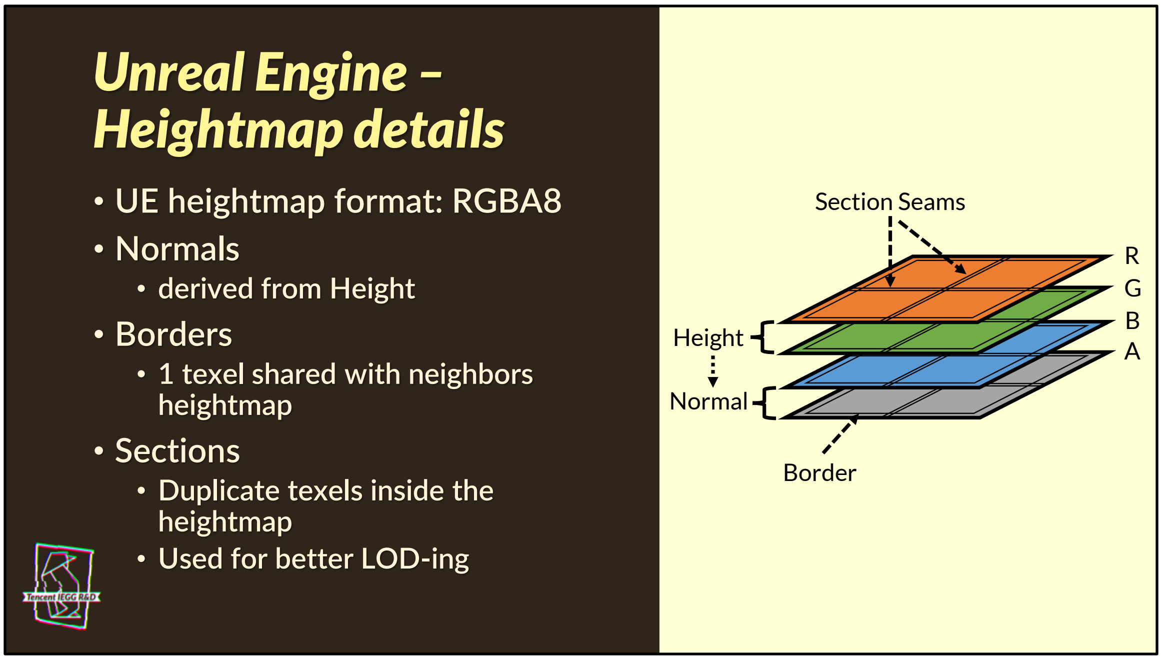
Reusing the existing functionality also means it is easier for people to pick up the system
It is important that our integration works with the existing in-engine and external tools, so work can be done efficiently
To enable non-Unreal support, we put our model execution code into a separate libs with weights embedded

Unreal Engine Landscape overview

- Encoding/Decoding is handled at the BulkData
- Using Custom Texture Class would lead to:
  - Difficult integration of new UE versions
  - Would need to reimplement support for editing, tools, etc.

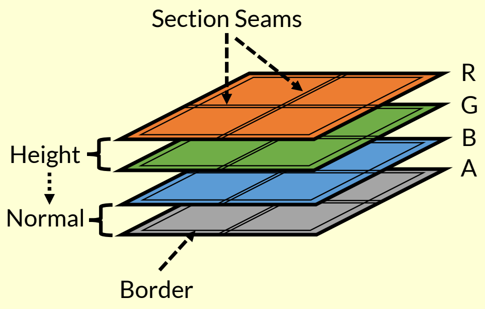


The illustration on the right is a simplified view of the system.
Implementing our system at the BulkData-Level all the other systems just work with the (de-)compressed data, EXCEPT the editing. It is also the path of least effort for us.

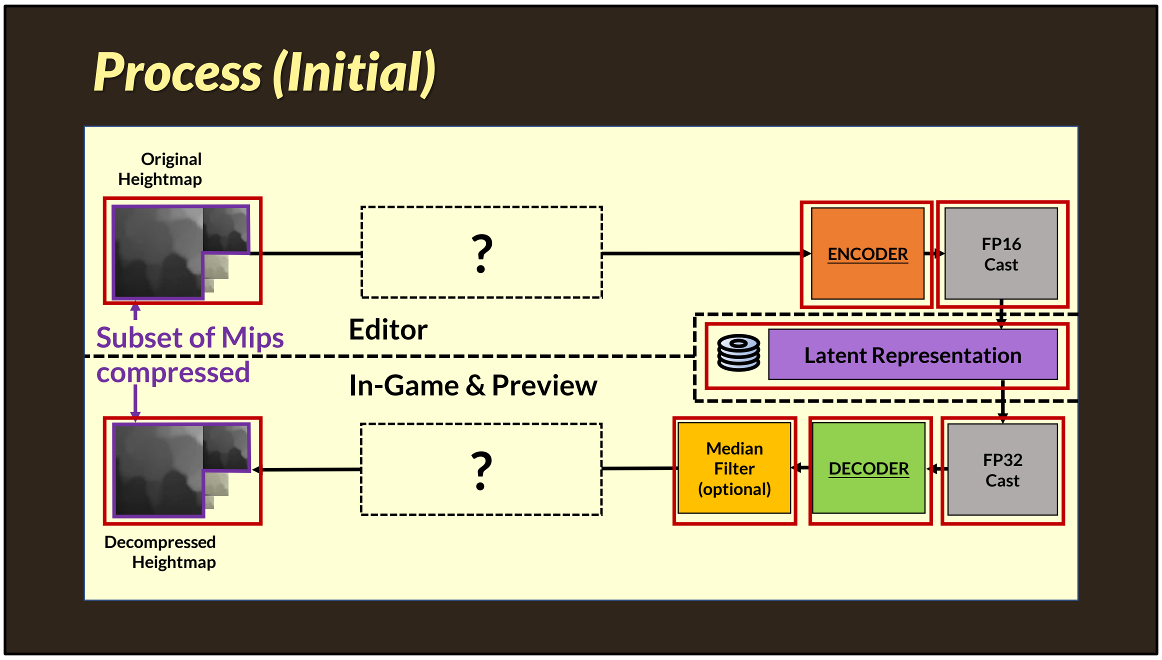


Unreal Engine – Heightmap details

- UE heightmap format: RGBA8
- Normals
  - derived from Height
- Borders
  - 1 texel shared with neighbors heightmap
- Sections
  - Duplicate texels inside the heightmap
  - Used for better LOD-ing

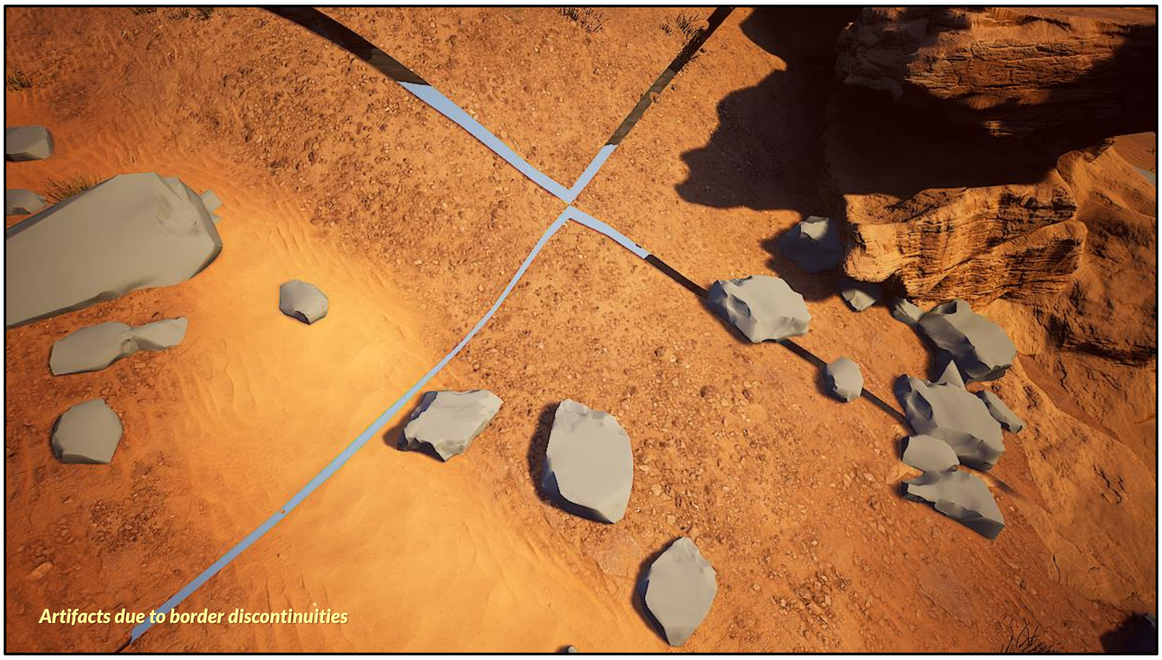


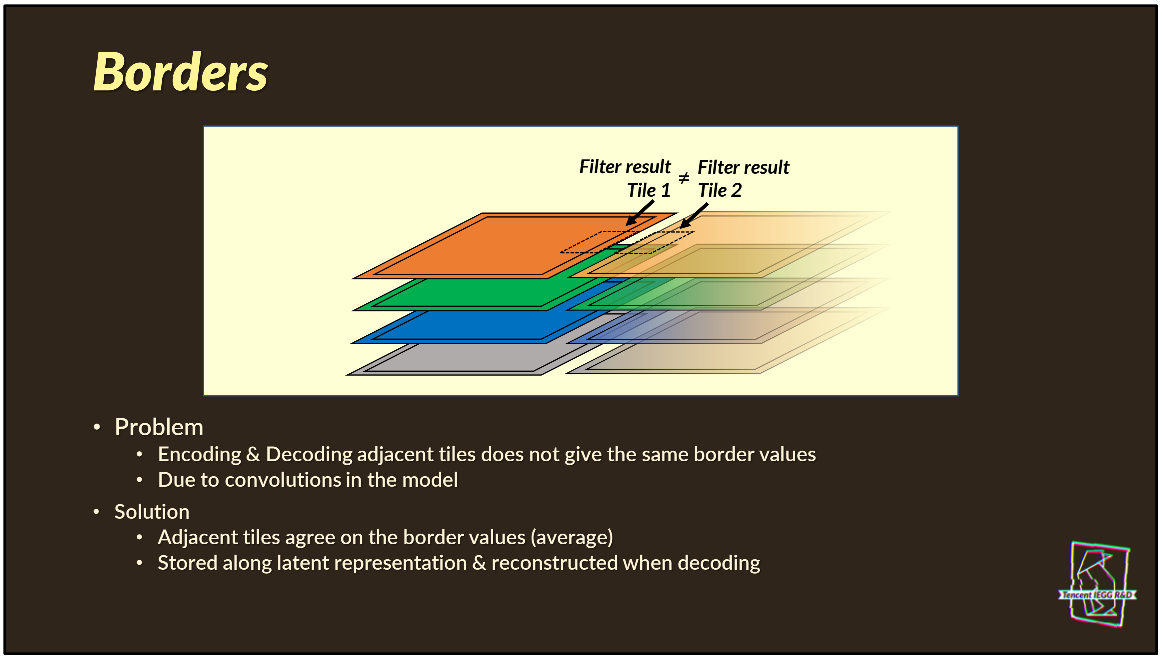
The last three "meta"-aspects of the heightmap texture all caused us difficulties during our integration (due to the interaction with the ML-model)
Unlike what the editor options for landscape suggest, there can be more than 2x2 section on one texture



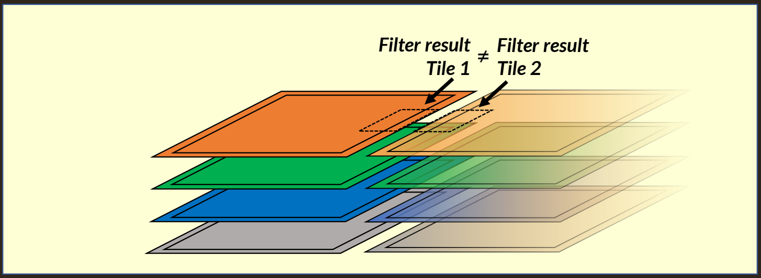
We have several options for compressing mips at different rates:

- 2x for all mips for areas where quality is more important than compression rate
- 3x for all mips for areas where compression rate is more important than quality
- 2x for the lowest mip and 3x for all following to give a good mixture of both


Artifacts due to border discontinuities



Borders

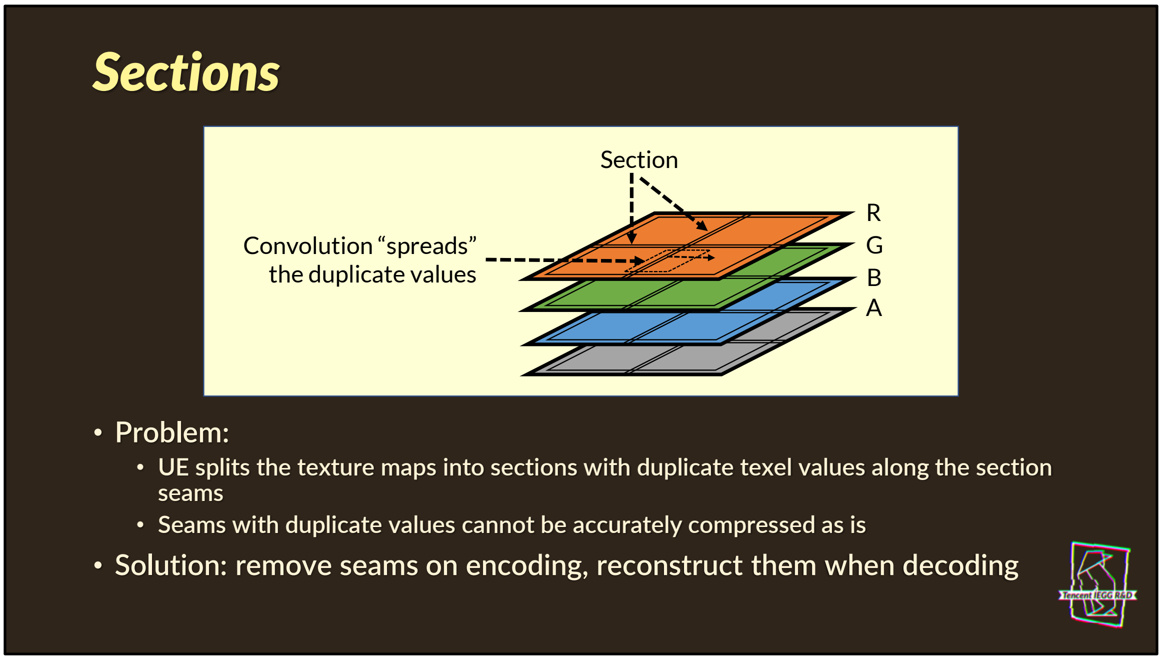


- Problem
  - Encoding & Decoding adjacent tiles does not give the same border values
  - Due to convolutions in the model
- Solution
  - Adjacent tiles agree on the border values (average)
  - Stored along latent representation & reconstructed when decoding

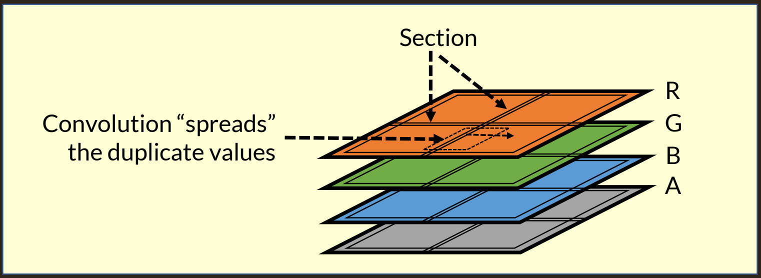
The problem can be summed up via: a filter with one input does not give the same results as a filter with another input

Reconstructed Border values

Seams due to compressing line of duplicate values



Sections

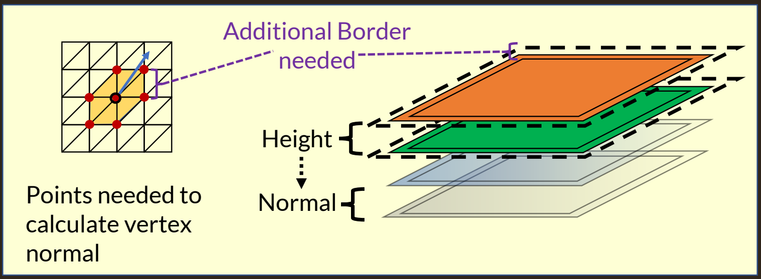


- Problem:
  - UE splits the texture maps into sections with duplicate texel values along the section seams
  - Seams with duplicate values cannot be accurately compressed as is
- Solution: remove seams on encoding, reconstruct them when decoding

Again, we have a problem where the design of the ML (specifically convolutions) affects the integration into an already existing system in curious ways

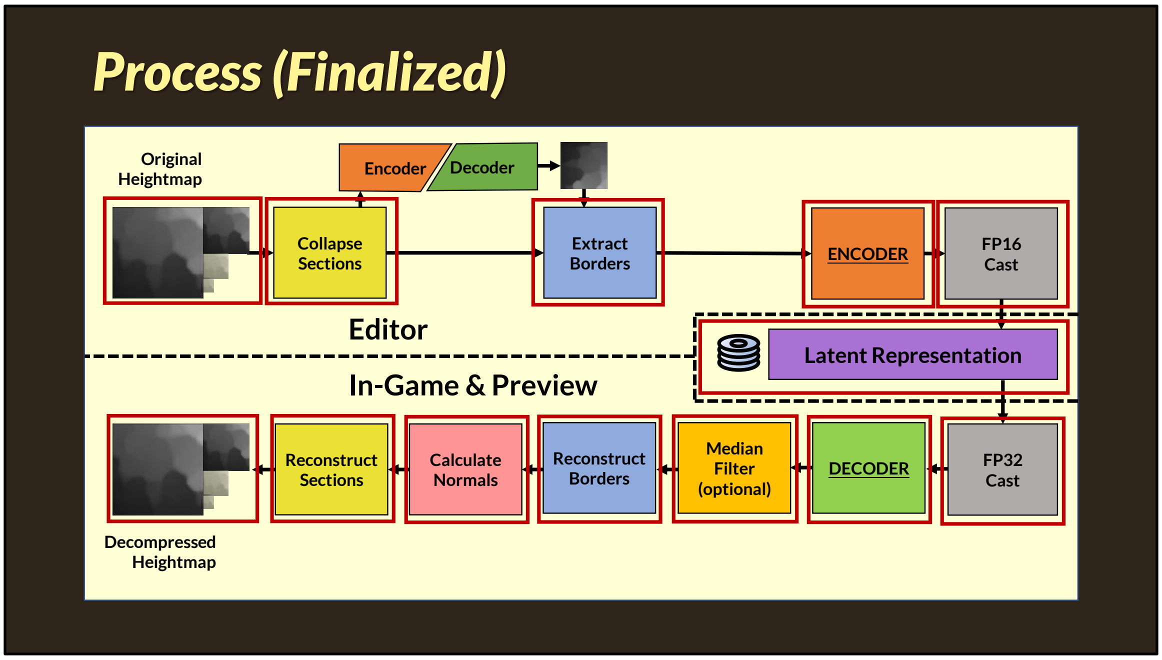
Removing and reconstructing section seams

Normal Vectors



- Problem: we don’t encode normal vectors, just heights
- UE calculates normal vectors by averaging face normals of the surrounding 3x3 neighborhood
- Solution : Store one more pixel outside the texture & calculate normal vectors at decompression -> additional 2x compression

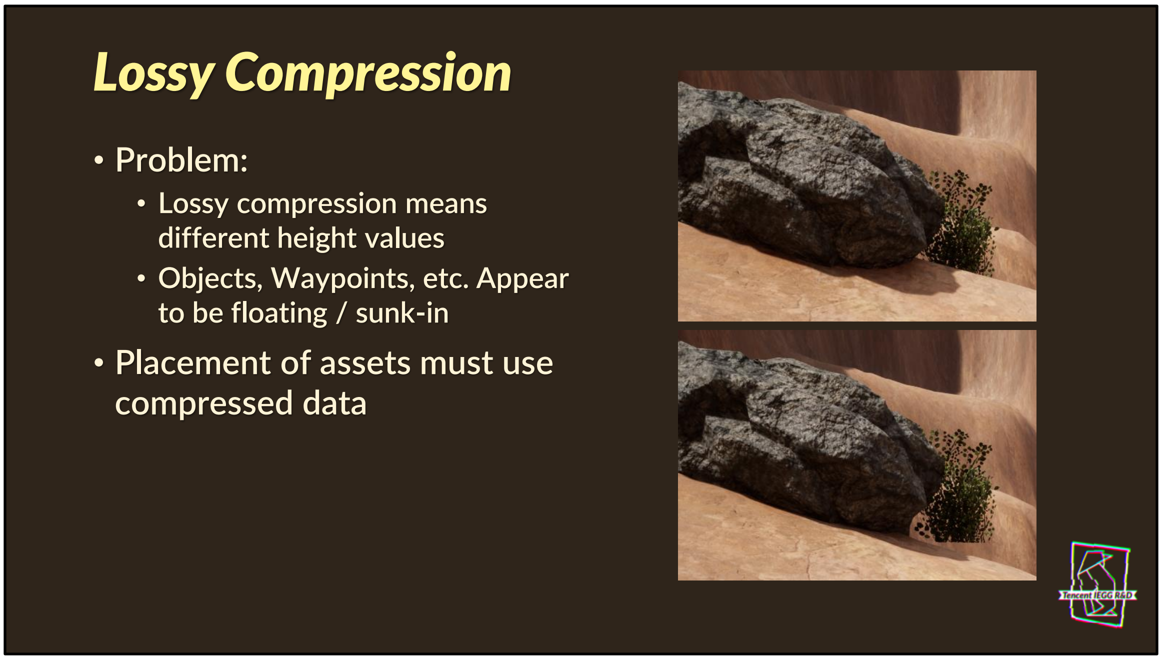
Compared to what unreal does, calculating the normal vectors during decoding gives us an additional 2x compression rate



Process (Finalized)

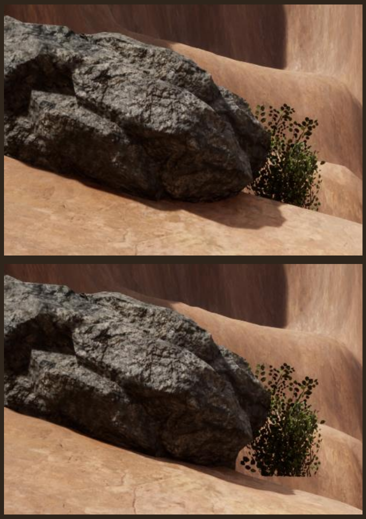
Those are the logical steps that happen, disregarding obvious optimizations

The resulting process ended up a bit more complex than initially anticipated

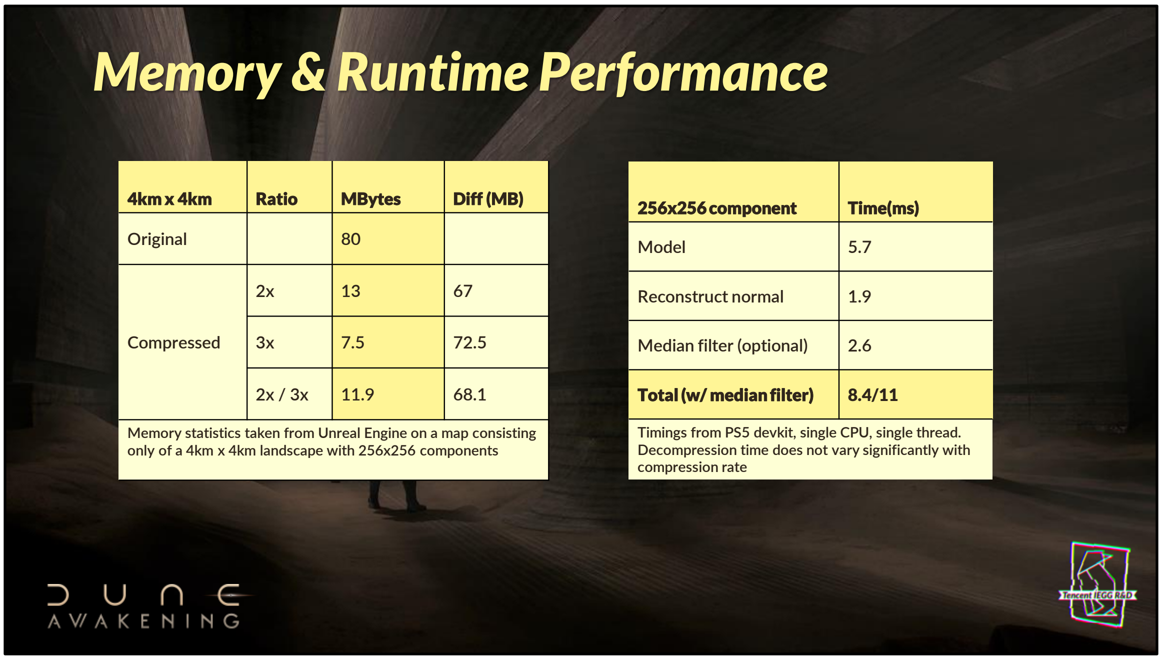


Lossy Compression

- Problem:
  - Lossy compression means different height values
  - Objects, Waypoints, etc. Appear to be floating / sunk-in
- Placement of assets must use compressed data



Everybody working on the levels needs to be aware of the lossy compression
Side note: Editing is only supported on original data

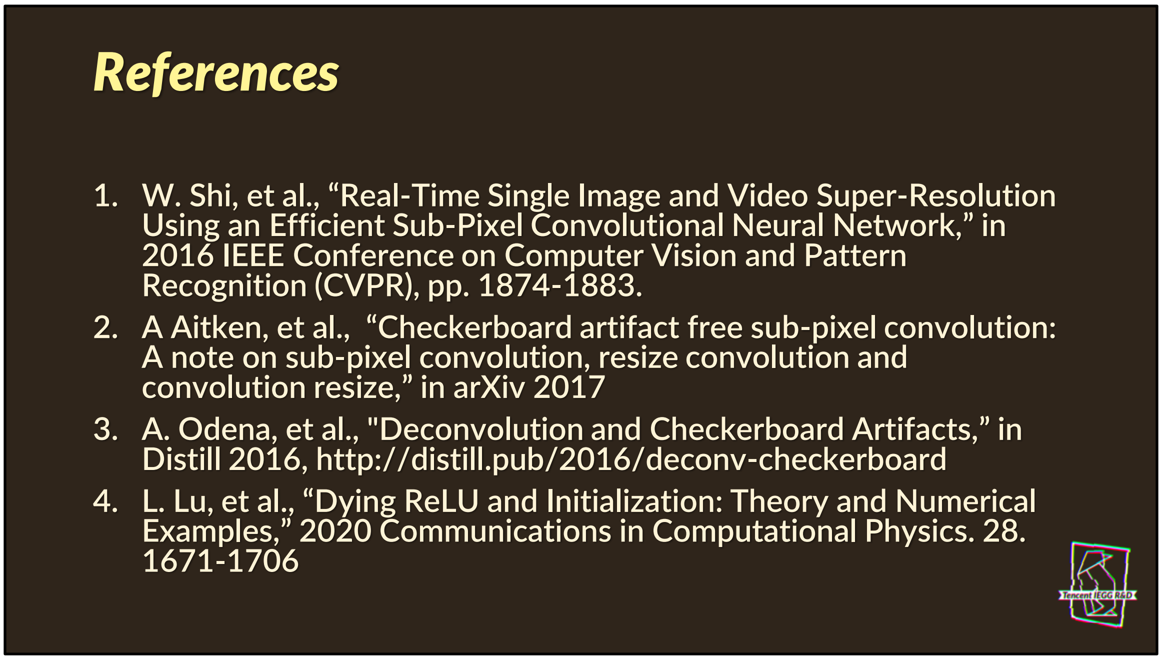


Memory & Runtime Performance

4km x 4km	Ratio	MBytes	Diff (MB)
Original		80	
Compressed	2x	13	67
	3x	7.5	72.5
	2x / 3x	11.9	68.1
Memory statistics taken from Unreal Engine on a map consisting only of a 4km x 4km landscape with 256x256 components			

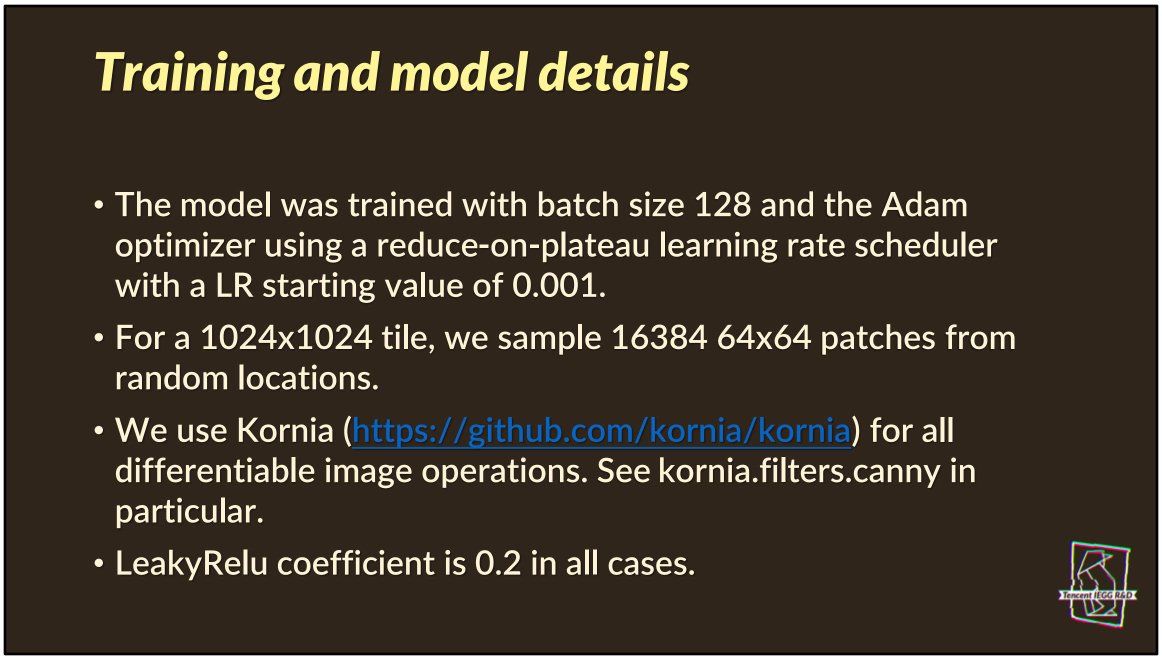
256x256 component	Time(ms)
Model	5.7
Reconstruct normal	1.9
Median filter (optional)	2.6
Total (w/ median filter)	**8.4/11**
Timings from PS5 devkit, single CPU, single thread. Decompression time does not vary significantly with compression rate	

Questions?



References

1. W. Shi, et al., “Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1874-1883.
2. A Aitken, et al., “Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize,” in arXiv 2017
3. A. Odena, et al., "Deconvolution and Checkerboard Artifacts,” in Distill 2016, http://distill.pub/2016/deconv-checkerboard
4. L. Lu, et al., “Dying ReLU and Initialization: Theory and Numerical Examples,” 2020 Communications in Computational Physics. 28. 1671-1706



Training and model details

- The model was trained with batch size 128 and the Adam optimizer using a reduce-on-plateau learning rate scheduler with a LR starting value of 0.001.
- For a 1024x1024 tile, we sample 16384 64x64 patches from random locations.
- We use Kornia (https://github.com/kornia/kornia) for all differentiable image operations. See kornia.filters.canny in particular.
- LeakyRelu coefficient is 0.2 in all cases.