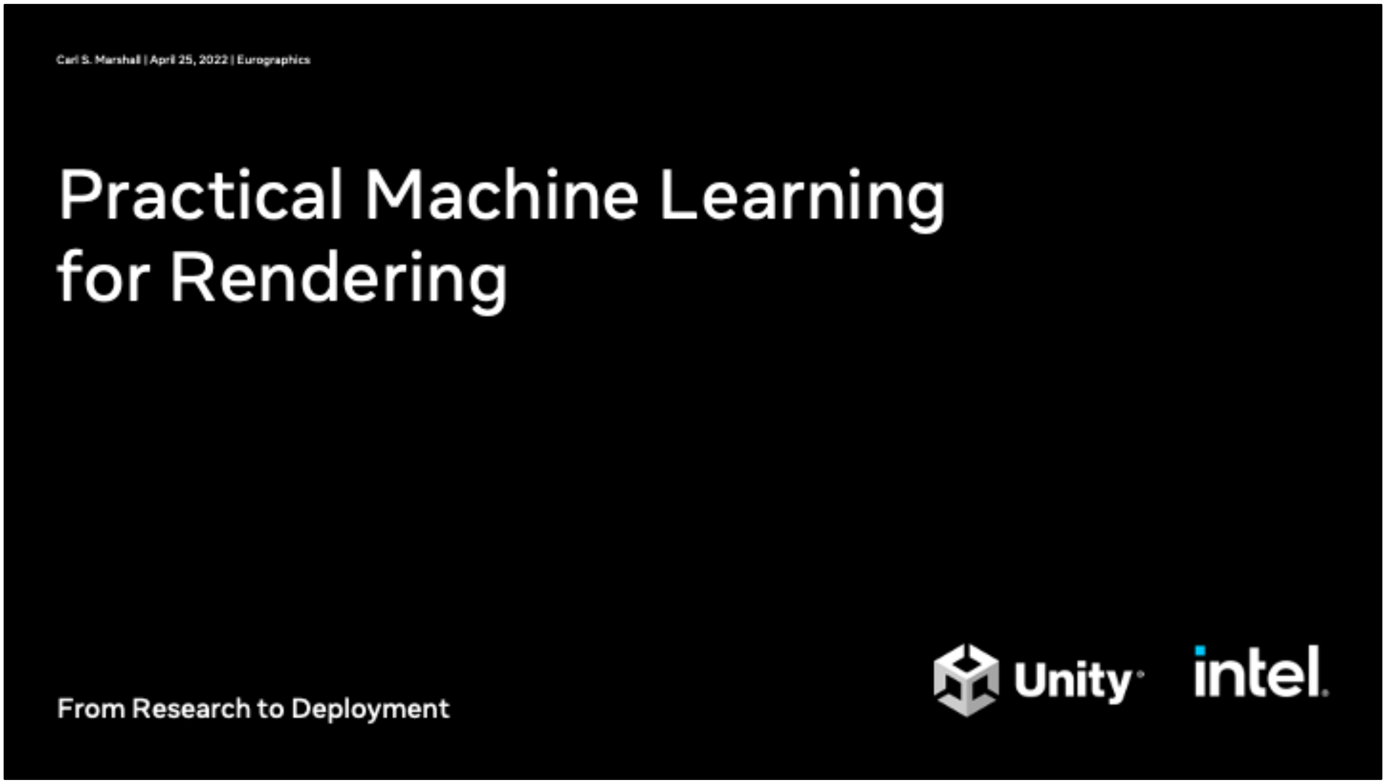


Carl S. Marshall | April 25, 2022 | Eurographics

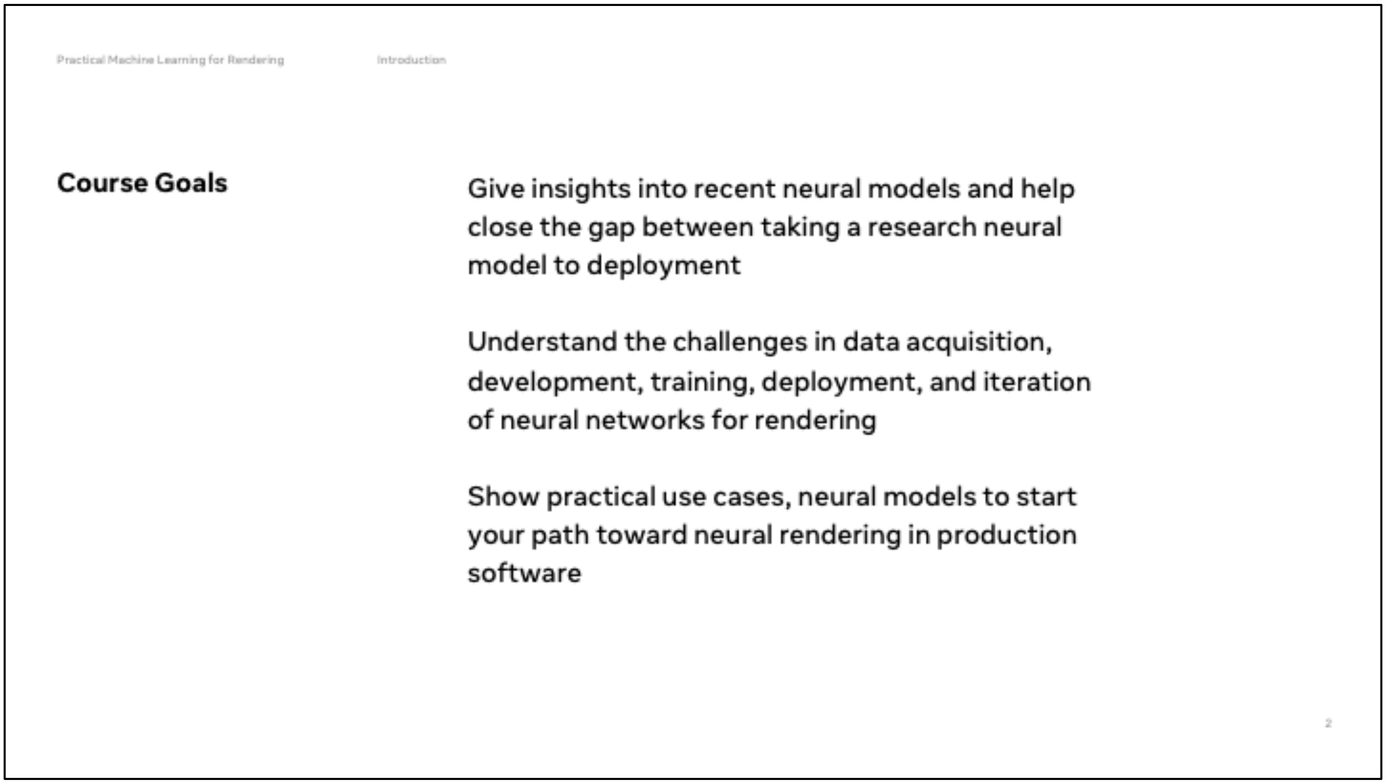
Practical Machine Learning for Rendering

From Research to Deployment

Unity
intel

- Hi. My name is Carl S. Marshall and I work in the Reality Labs Research at Meta. I transitioned to Reality Labs Research at Meta during this course submission from Intel Labs. The course slides were created during my employment at Intel Corporation and this content is accredited to Intel.

- This course is a collaboration of Intel Labs, Unity Labs, and Unity. I would like to thank all of the speakers for helping make this course possible. Please feel free to reach out to me or the team via email for any comments/questions: csmarshall@fb.com, deepak.s.vembar@intel.com, sujoy.ganguly@unity3d.com, florent@unity3d.com

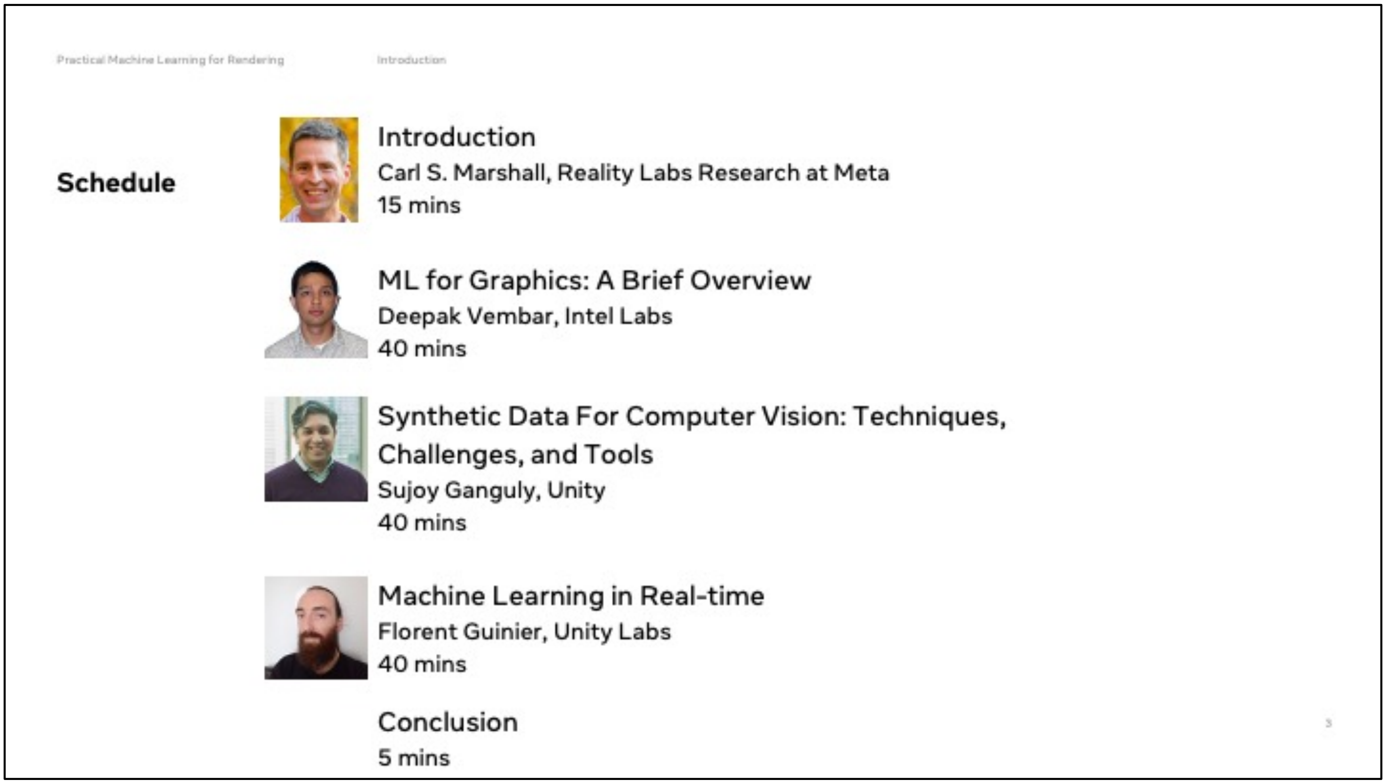


Course Goals

Give insights into recent neural models and help close the gap between taking a research neural model to deployment

Understand the challenges in data acquisition, development, training, deployment, and iteration of neural networks for rendering

Show practical use cases, neural models to start your path toward neural rendering in production software



Schedule

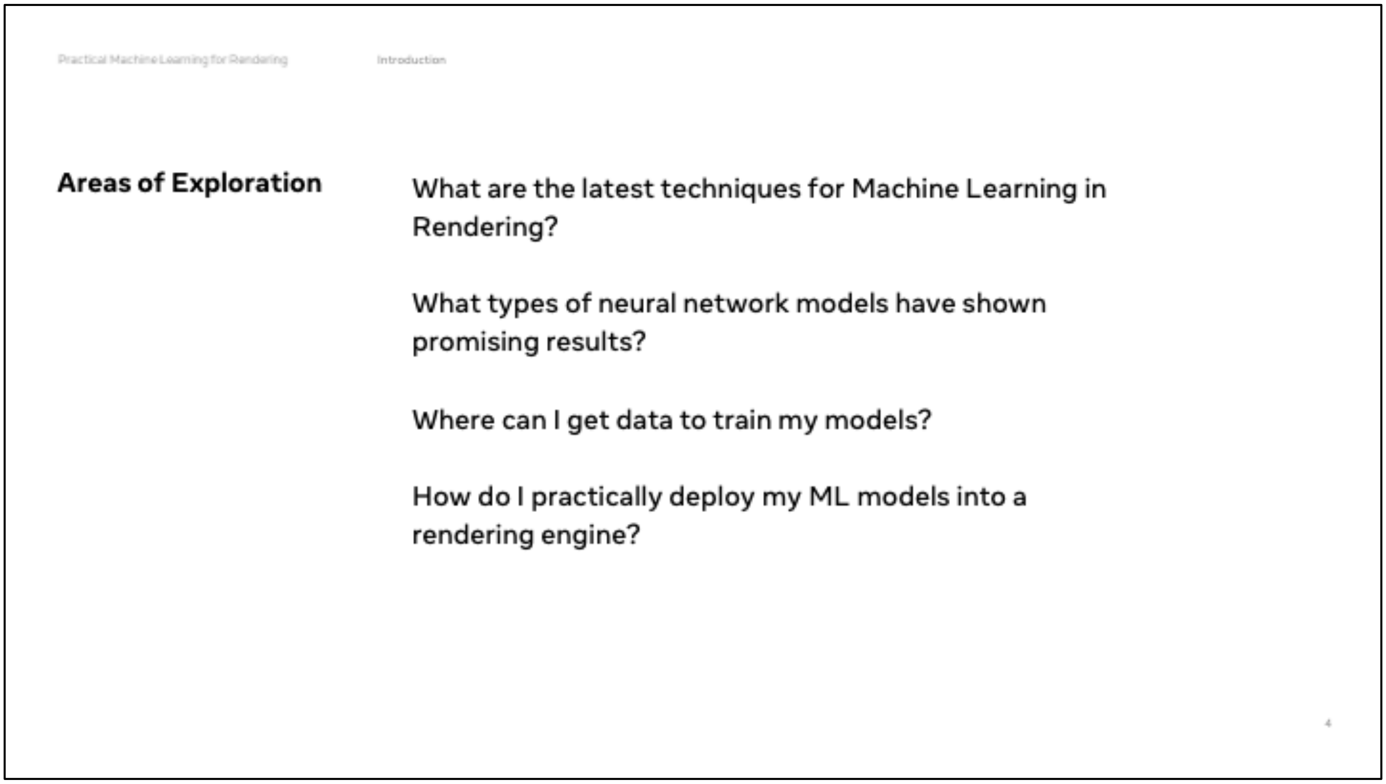
Introduction
Carl S. Marshall, Reality Labs Research at Meta
15 mins

ML for Graphics: A Brief Overview
Deepak Vembar, Intel Labs
40 mins

Synthetic Data For Computer Vision: Techniques, Challenges, and Tools
Sujoy Ganguly, Unity
40 mins

Machine Learning in Real-time
Florent Guinier, Unity Labs
40 mins

Conclusion
5 mins



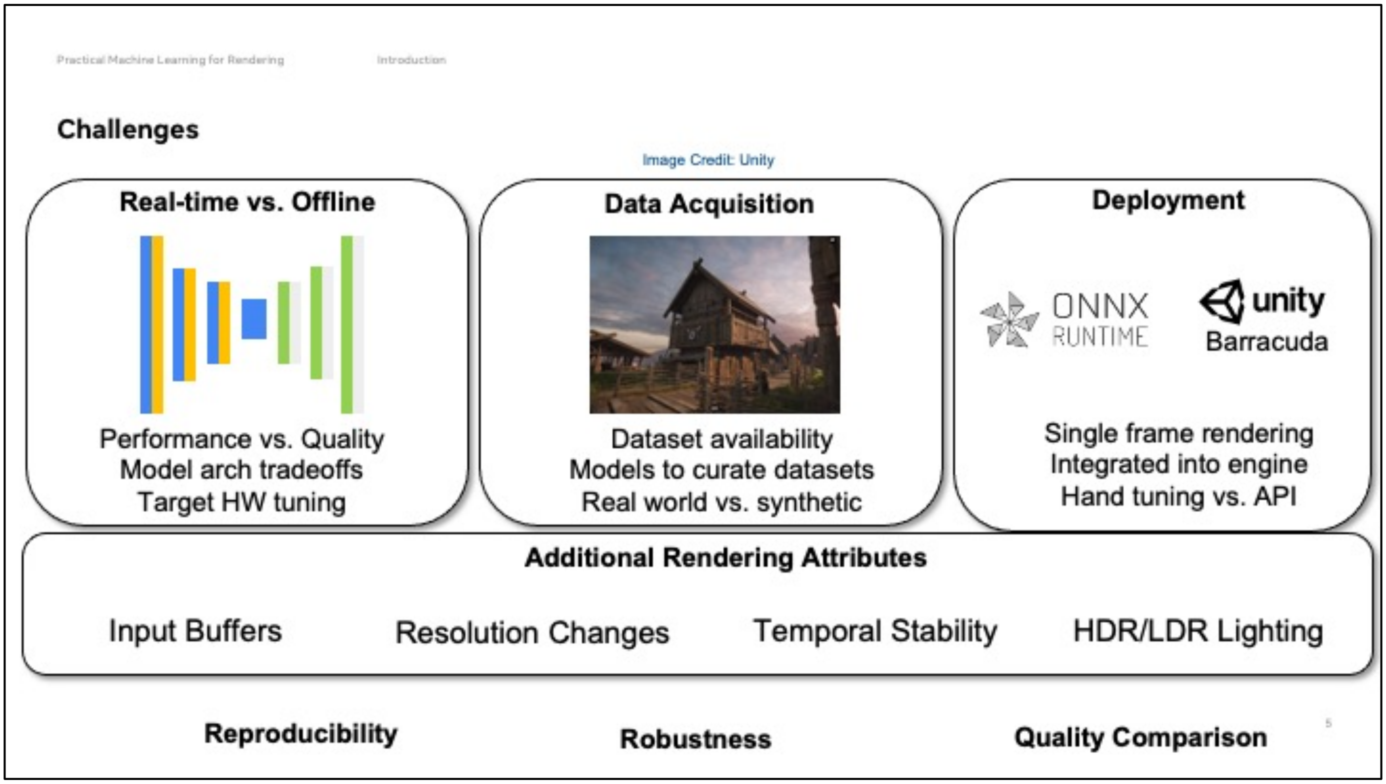
Areas of Exploration

What are the latest techniques for Machine Learning in Rendering?

What types of neural network models have shown promising results?

Where can I get data to train my models?

How do I practically deploy my ML models into a rendering engine?



Challenges

Image Credit: Unity

Real-time vs. Offline

Performance vs. Quality
Model arch tradeoffs
Target HW tuning

Data Acquisition

Dataset availability
Models to curate datasets
Real world vs. synthetic

Deployment

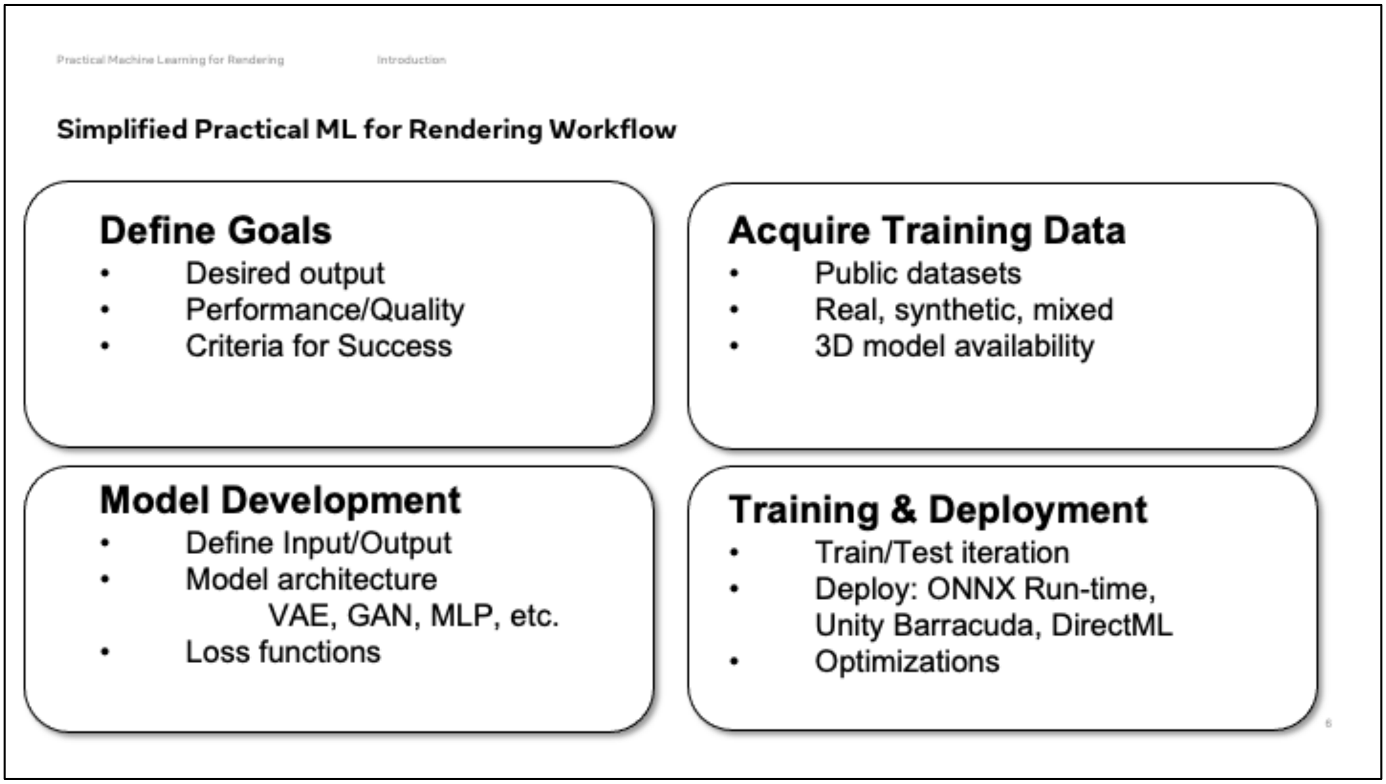
ONNX RUNTIME
unity
Barracuda

Single frame rendering
Integrated into engine
Hand tuning vs. API

Additional Rendering Attributes

Input Buffers Resolution Changes Temporal Stability HDR/LDR Lighting

Reproducibility **Robustness** **Quality Comparison**



Simplified Practical ML for Rendering Workflow

Define Goals

- Desired output
- Performance/Quality
- Criteria for Success

Acquire Training Data

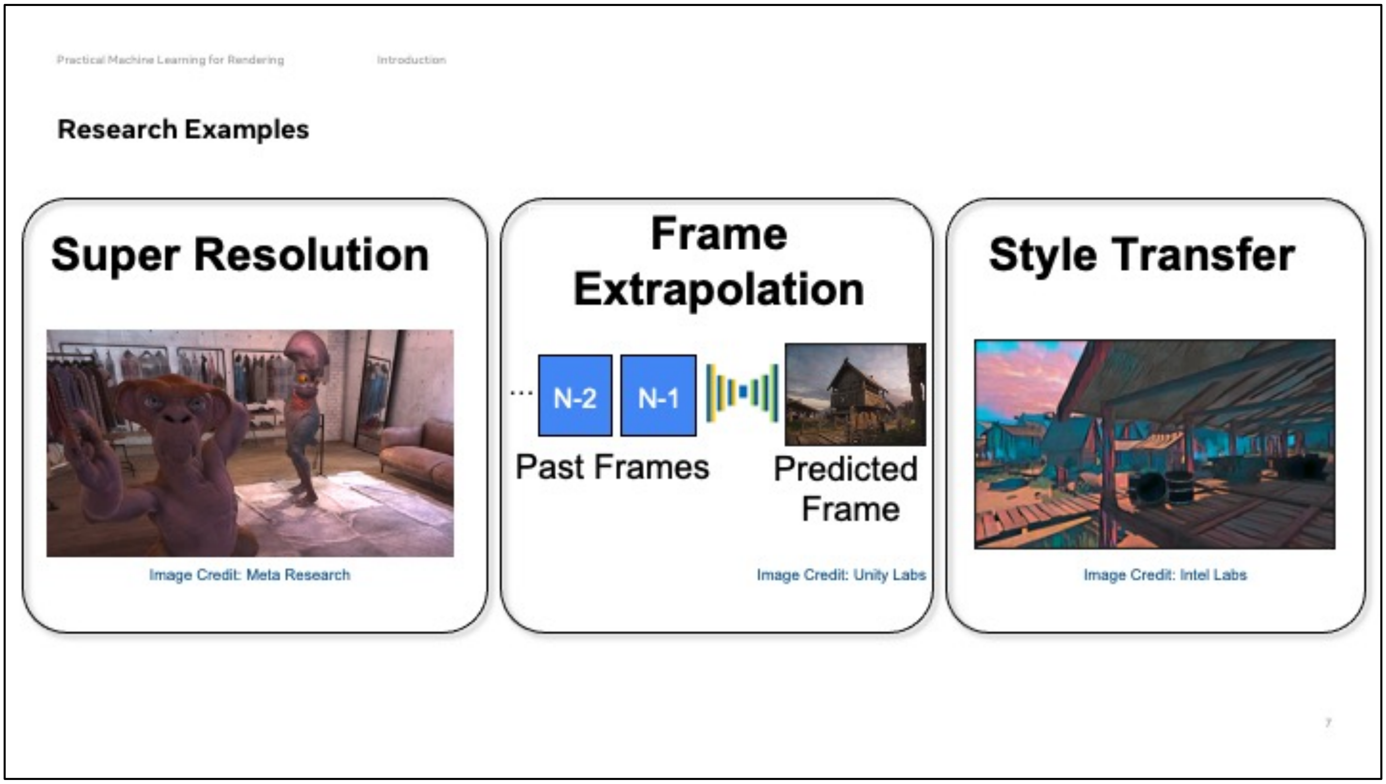
- Public datasets
- Real, synthetic, mixed
- 3D model availability

Model Development

- Define Input/Output
- Model architecture
  - VAE, GAN, MLP, etc.
- Loss functions

Training & Deployment

- Train/Test iteration
- Deploy: ONNX Run-time, Unity Barracuda, DirectML
- Optimizations



Research Examples

Super Resolution

Image Credit: Meta Research

Frame Extrapolation

... N-2 N-1

Past Frames

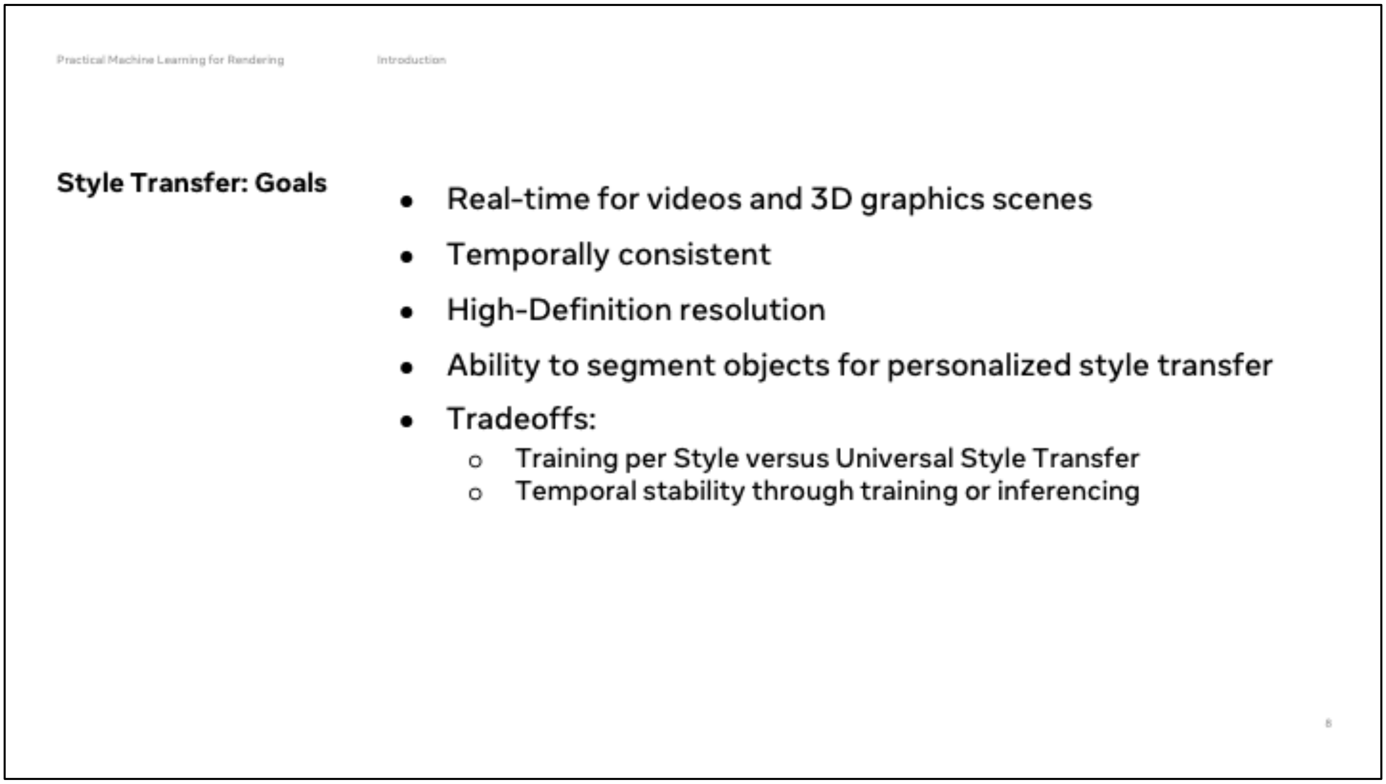
Predicted Frame

Image Credit: Unity Labs

Style Transfer

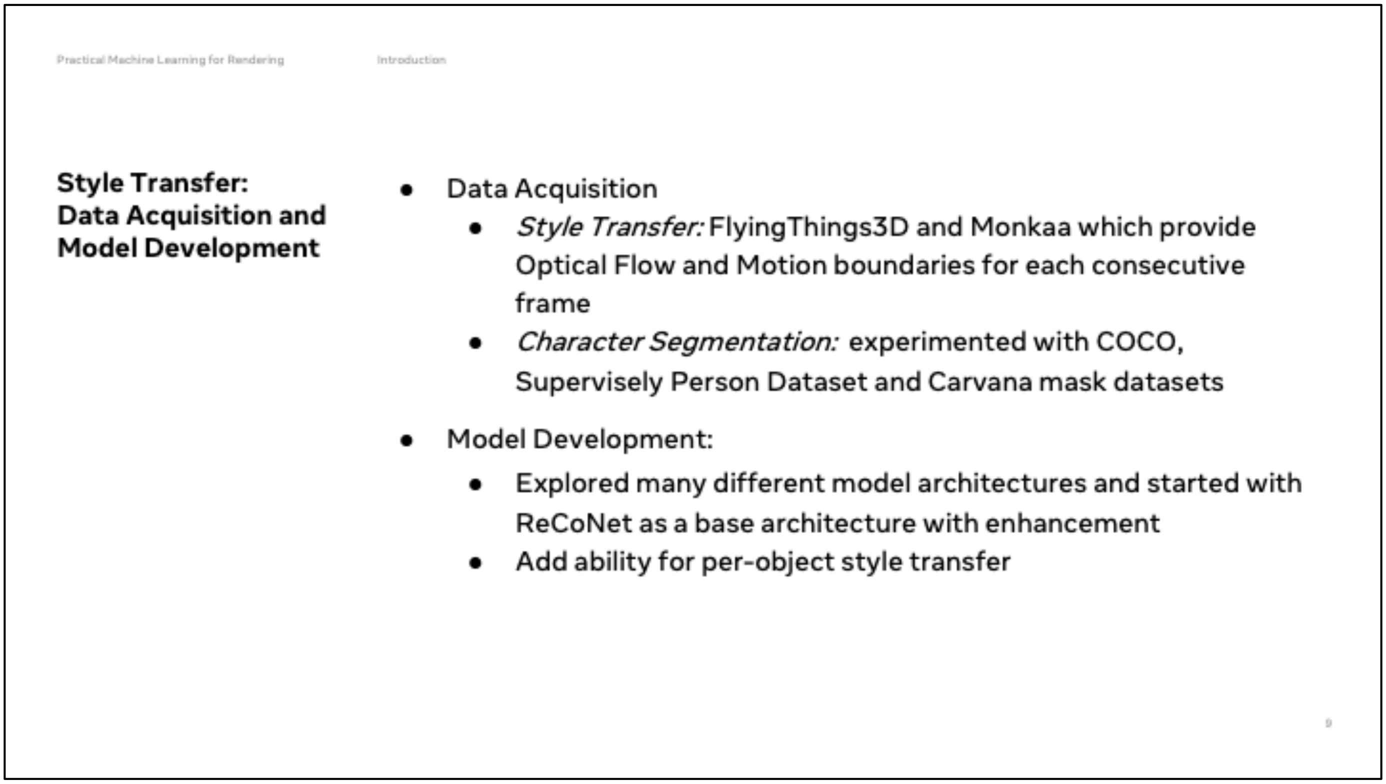
Image Credit: Intel Labs

Super Resolution Image - https://research.facebook.com/publications/neural-supersampling-for-real-time-rendering/
Viking Village images – Credit Unity Labs, Stylization – Credit Intel Labs



Style Transfer: Goals

- Real-time for videos and 3D graphics scenes
- Temporally consistent
- High-Definition resolution
- Ability to segment objects for personalized style transfer
- Tradeoffs:
  - Training per Style versus Universal Style Transfer
  - Temporal stability through training or inferencing

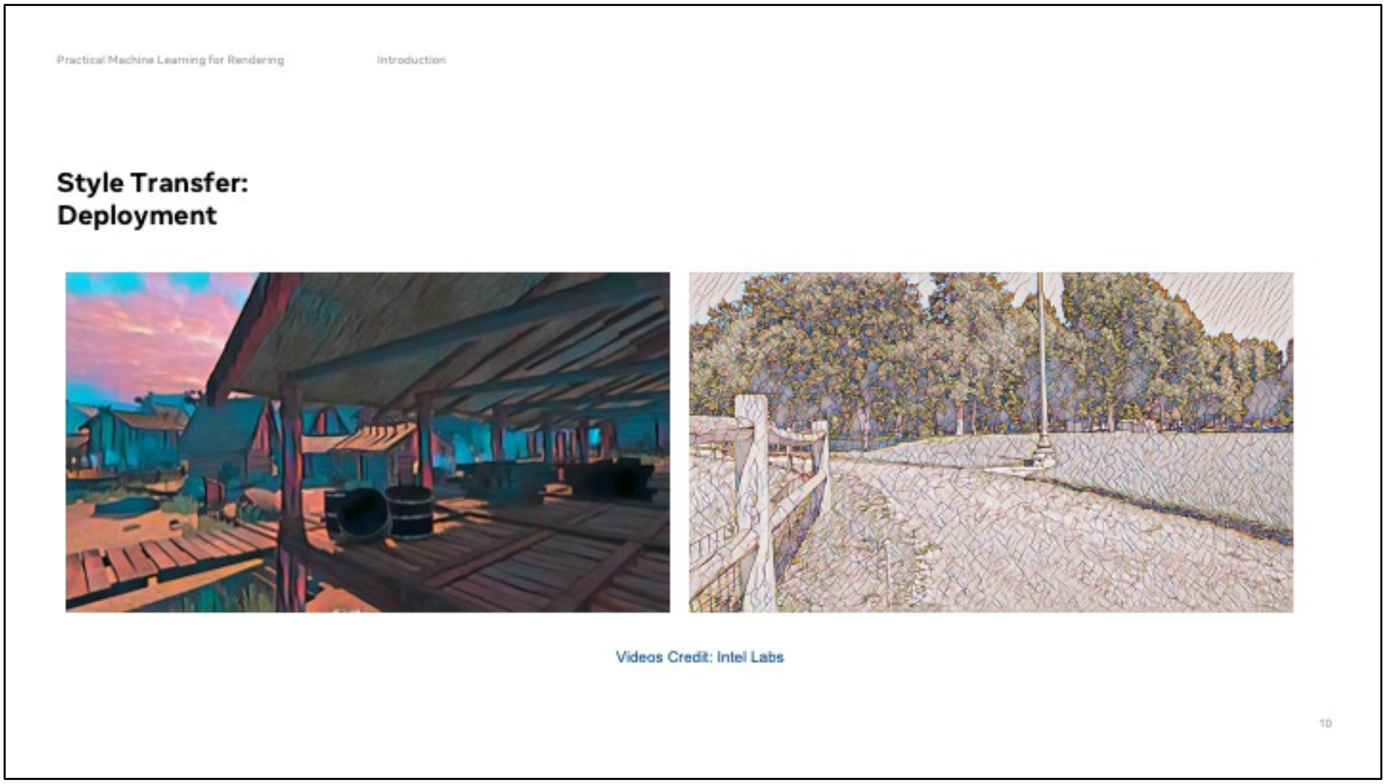


Style Transfer: Data Acquisition and Model Development

- Data Acquisition
  - *Style Transfer:* FlyingThings3D and Monkaa which provide Optical Flow and Motion boundaries for each consecutive frame
  - *Character Segmentation:* experimented with COCO, Supervisely Person Dataset and Carvana mask datasets
- Model Development:
  - Explored many different model architectures and started with ReCoNet as a base architecture with enhancement
  - Add ability for per-object style transfer

FlyingThings3D and Monkaa datasets: https://lmb.informatik.uni-freiburg.de/resources/datasets/SceneFlowDatasets.en.html
COCO dataset: https://cocodataset.org/#home
Supervisely Person dataset: https://supervise.ly/explore/projects/supervisely-person-dataset-23304/datasets
Carvana dataset: https://www.kaggle.com/c/carvana-image-masking-challenge/data
ReCoNet: Gao, Chang & Gu, Derun & Zhang, Fangjun & Yu, Yizhou. (2018). ReCoNet: Real-Time Coherent Video Style Transfer Network. 637-653. 10.1007/978-3-030-20876-9_40.

Slide data - Credit to Intel Labs



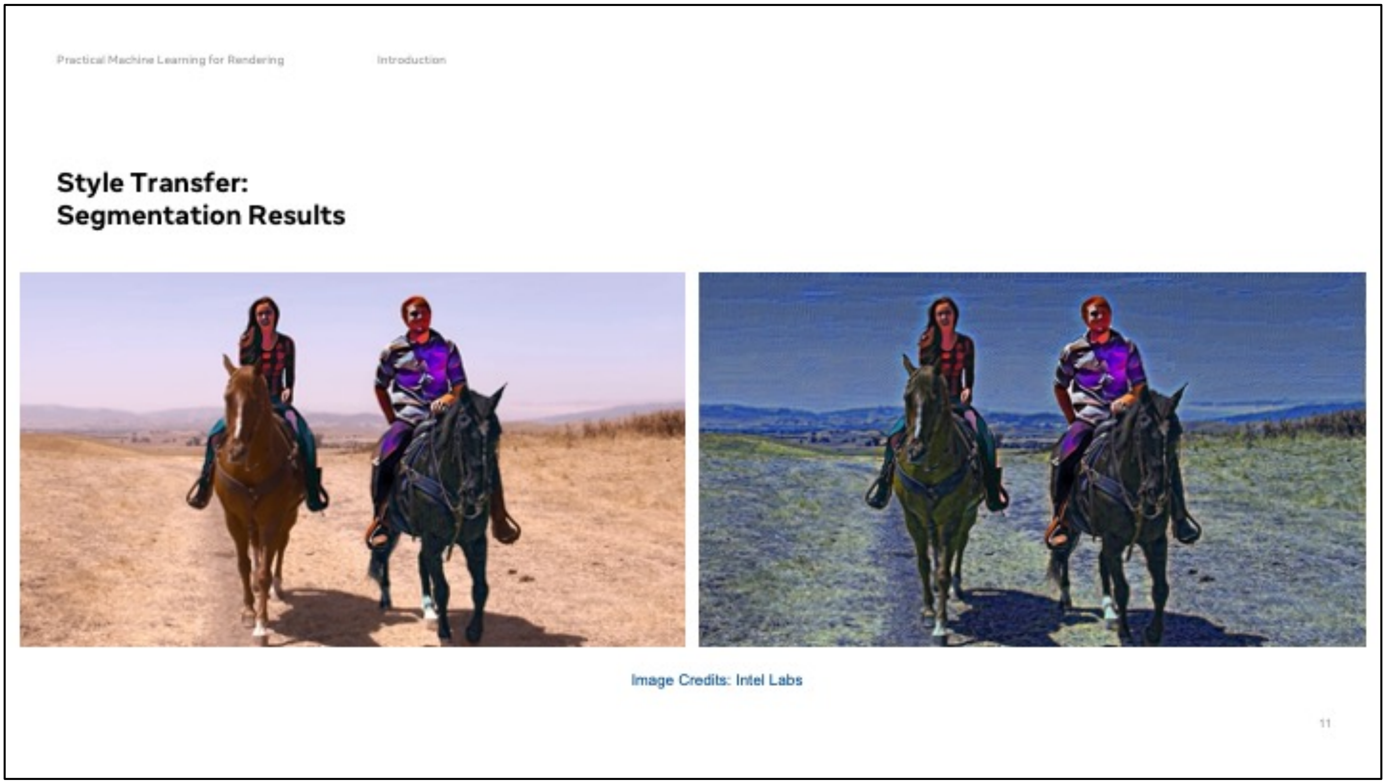
Style Transfer: Deployment

Videos Credit: Intel Labs

Viking Village Asset (Left video) – Credit Unity Technologies

- BikeQuick1_30REV (Right video)– Credit Intel
- Styles: Viking->Edtaonisl, Bike -> Mosaic
- Resolution: 1080p at 30 FPS

Stylization of videos - Credit to Intel Labs
Thanks to Honnesh Rohmetra, while interning at Intel, for creating these stylized videos and the segmented stylized images on the next slide.



Practical Machine Learning for Rendering   Introduction

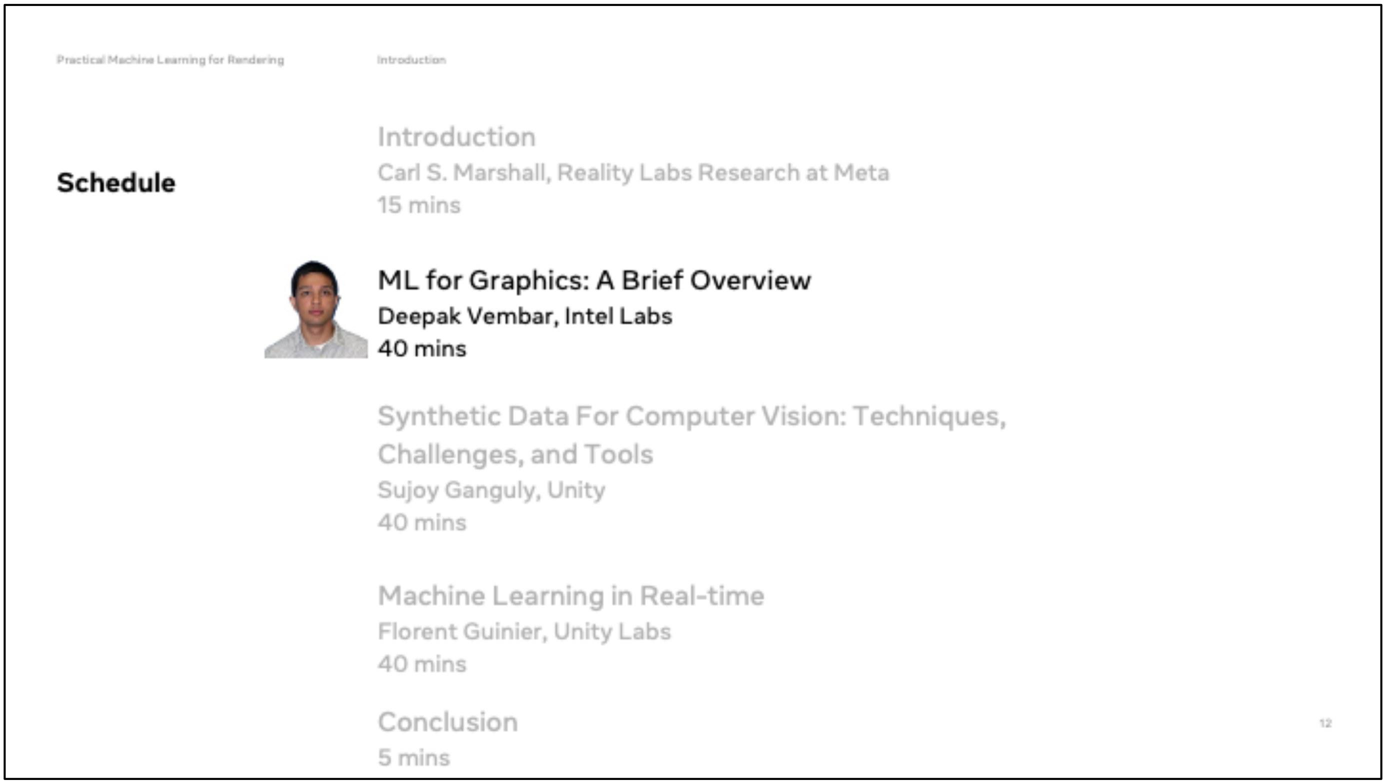
Style Transfer: Segmentation Results

Image Credits: Intel Labs

11

- Video: HorseScene1_60_REV (Intel)
- Styles: Person1 ->Edtaonisl, Person2 -> Composition, Background-> Starry night
- Resolution: 1080p at 30 FPS
- Instance Segmentation: CondInst trained on COCO dataset

Images Credit to Intel Labs



Schedule

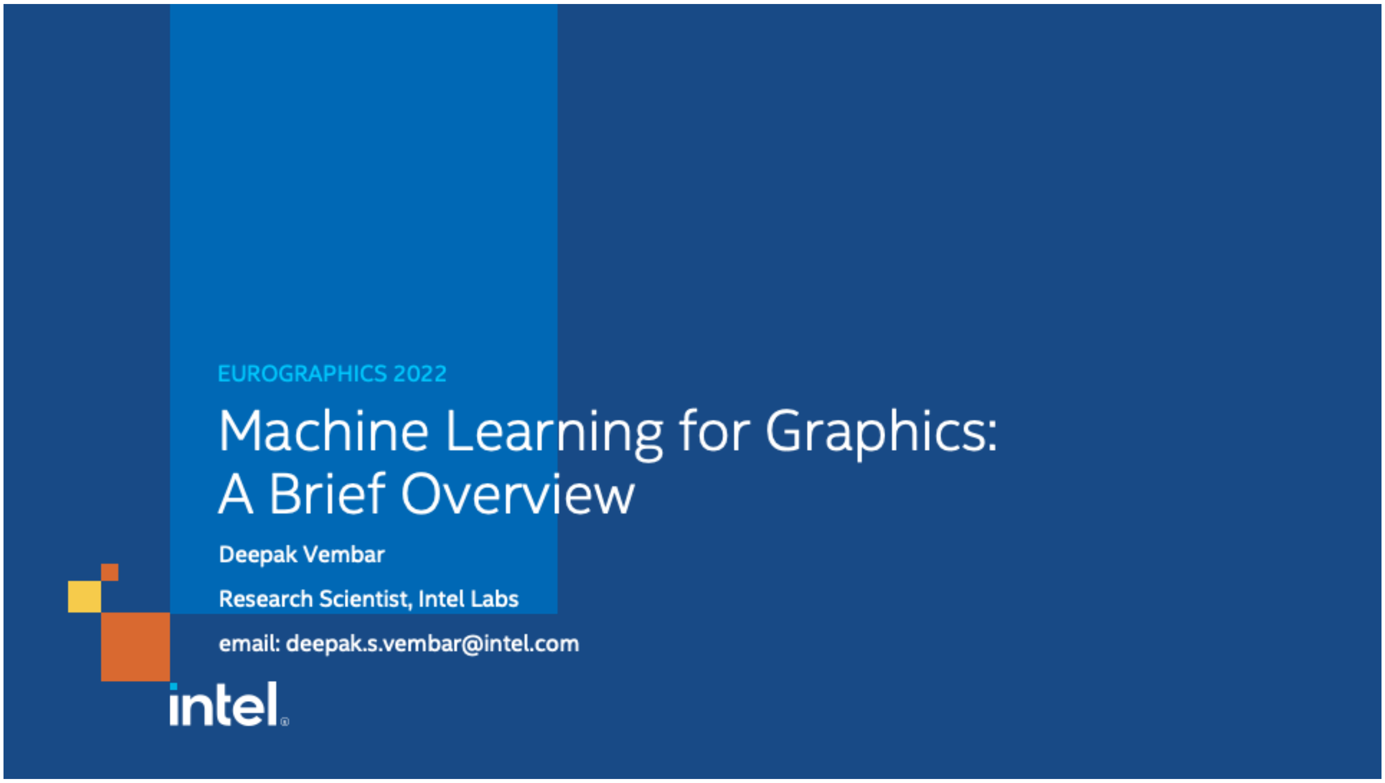
Introduction
Carl S. Marshall, Reality Labs Research at Meta
15 mins

ML for Graphics: A Brief Overview
Deepak Vembar, Intel Labs
40 mins

Synthetic Data For Computer Vision: Techniques, Challenges, and Tools
Sujoy Ganguly, Unity
40 mins

Machine Learning in Real-time
Florent Guinier, Unity Labs
40 mins

Conclusion
5 mins



EUROGRAPHICS 2022

Machine Learning for Graphics: A Brief Overview

Deepak Vembar

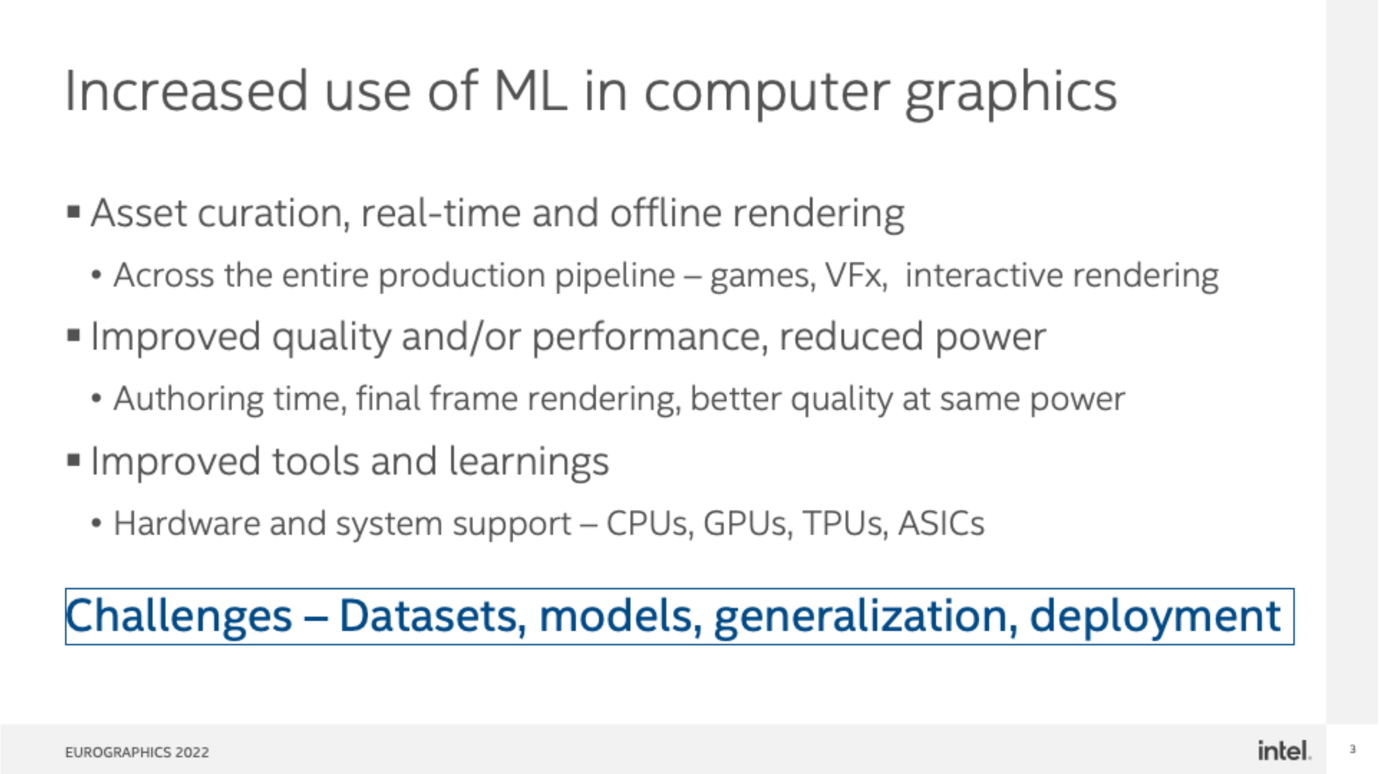
Research Scientist, Intel Labs

email: deepak.s.vembar@intel.com

Hello, my name is Deepak Vembar and I am a research scientist at Intel Labs working on generative graphics and neural rendering. In this section, I will present a brief overview of some relevant work, applications and deployment/ pitfalls on implementing AI techniques in the graphics pipeline, both offline and real-time.

Agenda

- Overview of machine learning in graphics (10 mins)
- ML in content generation pipelines (12 mins)
- ML to augment rendering (8 mins)
- Challenges and opportunities (10 mins)



Increased use of ML in computer graphics

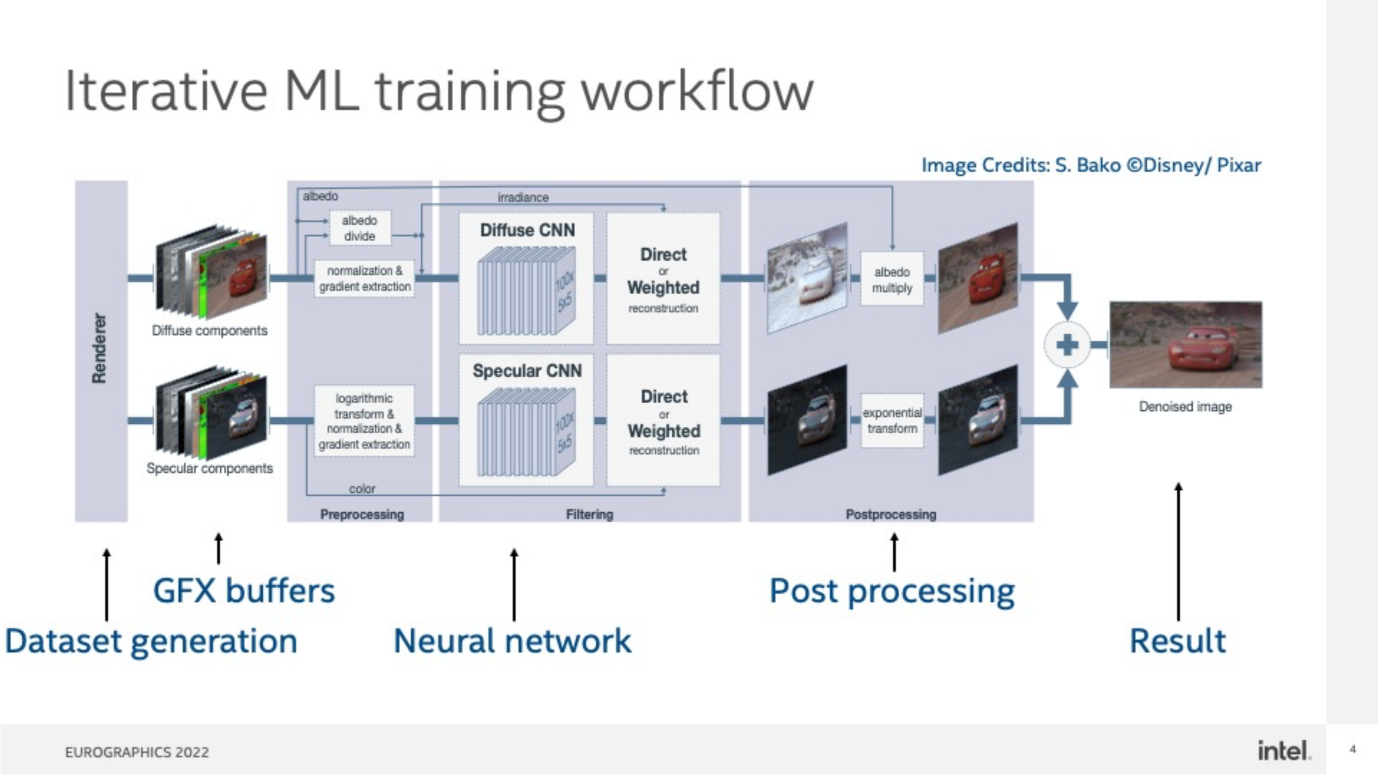
- Asset curation, real-time and offline rendering
  - Across the entire production pipeline – games, VFx, interactive rendering
- Improved quality and/or performance, reduced power
  - Authoring time, final frame rendering, better quality at same power
- Improved tools and learnings
  - Hardware and system support – CPUs, GPUs, TPUs, ASICs

Challenges – Datasets, models, generalization, deployment

The application of ML to rendering has increased – both in the actual applications and deployments to fundamental research into the new field as evidenced by past SGGRAPH papers.
some aspect of the content generation, editing pipeline are seeing increased use of ML for recommendation systems – anything to improve artist and animator productivity
Similarly, we are seeing increased examples of ML in production rendering both for cinematic and real-time content – denoisers and super resolution being tangible examples

This advance has resulted in improved visual quality at similar power envelopes and enabled new experiences and effects such as high-resolution high fps gaming. For artists, having a ML system that generates a rough draft has resulted in increased productivity.
These applications are supported by changes in the software and hardware ecosystems that enable high fidelity, low latency usages across the entire graphics and rendering spectrum.

Despite the advances, the field is still nascent. There are challenges in acquiring the datasets that could generalize to different types of content, variations in network and models that are deployed as well as challenges in meeting performance and power thresholds across different deployment systems.



Iterative ML training workflow

Image Credits: S. Bako ©Disney/ Pixar

GFX buffers

Dataset generation

Neural network

Post processing

Result

EUROGRAPHICS 2022

intel.

4

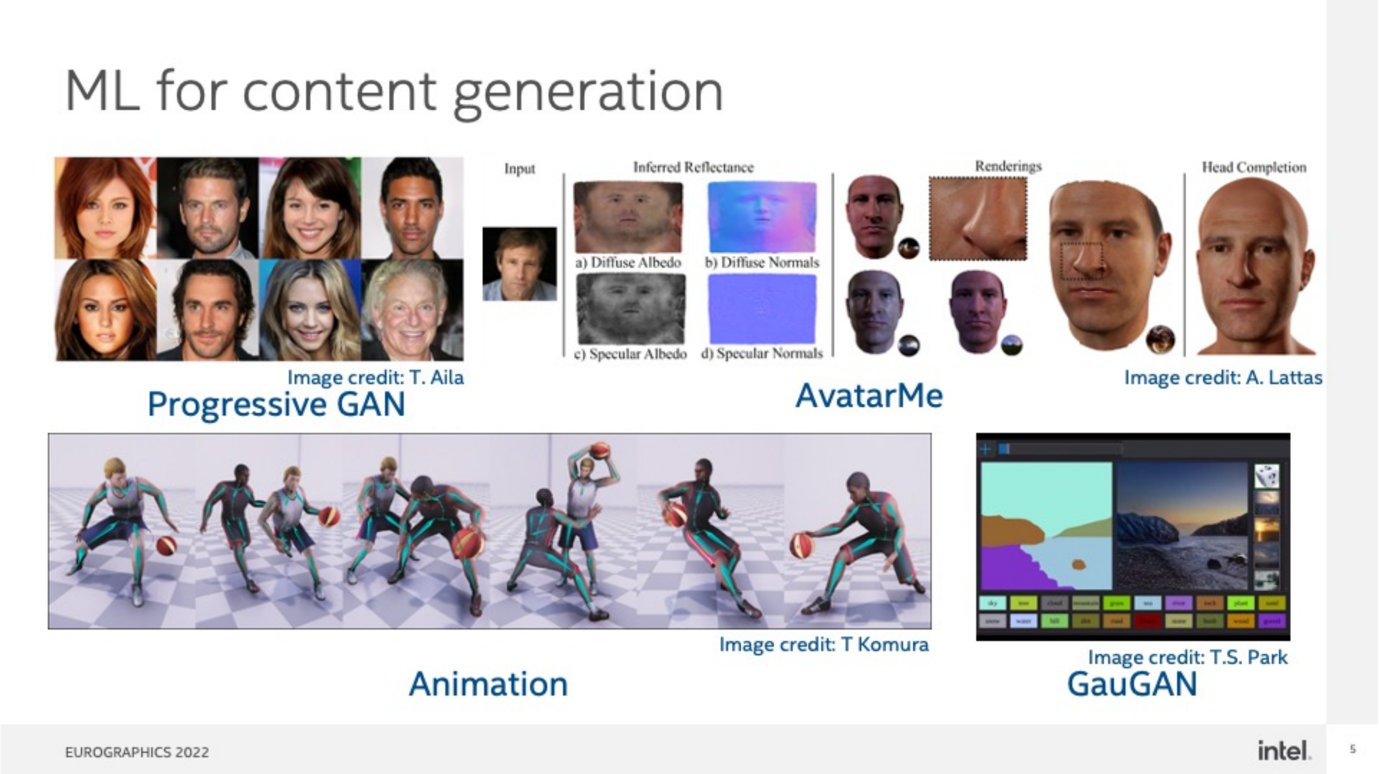
Let us look at an example of training a denoising network. Let us assume that we are looking to deployed published research into solving a problem that we have – so we have an architecture of the network and perhaps some trained weights. The datasets may or may not be released depending on licensing and rights.

Generally speaking, the training can be divided into

- Data collection – Unlike computer vision, we can use the renderer to generate the buffers and images that we need to train the network. These buffers can be directly generated through the rendering pipeline (depth/ normal/ albedo/ motion vectors), or can be obtained as a by product of it (temporal data, 3D meshes). Usualyy data has to be pre-processed (cropping, rotation etc) so as to have a good representation of all content that the network is expected to resolve.
- Network - this is a critical part of the process and determines what data needs to be collected to train the network as well as the performance and output. There are different types of networks , and depending on their performance, may or may not be suitable for the application. Accuracy and performance of the network are the defining factors in selecting it.
- Results – the output of performing inference with trained weights of an optimized network.

Note that this is only during training. The network has to be deployed as a part of the application and rendering (inference step) to do anything useful. Deployments could be done with the training framework (Pytorch/ Tensorflow) in case performance is not a bottleneck, or have to be included in the rendering pipeline using existing APIs and graphics frameworks (DirectX, Vulkan, Renderman etc)

Image courtesy: Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings
Steve Bako*, Thijs Vogels*, Brian McWilliams, Mark Meyer, Jan Novák,
Alex Harvill, Pradeep Sen, Tony DeRose, and Fabrice Rousselle
ACM Transactions on Graphics (Proceedings of SIGGRAPH 2017), vol. 36, no. 4



ML for content generation

Image credit: T. Aila

Progressive GAN

AvatarMe

Image credit: A. Lattas

Image credit: T Komura

Animation

Image credit: T.S. Park

GauGAN

Let us look at examples of using ML to augment the content creation process. Let us take the case of creating or customizing an avatar for integration into a game

We see an example of using generative networks (GANs) to generate high resolution images of people's faces.

This generated image can be used to generate a 3G textured mesh, using just the image generated as input.

We could use another network that learns and mimics the motion of humans so that we can integrate this with the generated 3D face mesh and make the character move.

Finally, we could generate different outdoor scenes for integration as backdrops into the game content.

Thse are just exemplar usages, and while it still takes effort to deploy this in a system, we expect that it will soon be possible to do so – enabling customization and democratization of content without having much knowledge of content creation pipelines to do so.

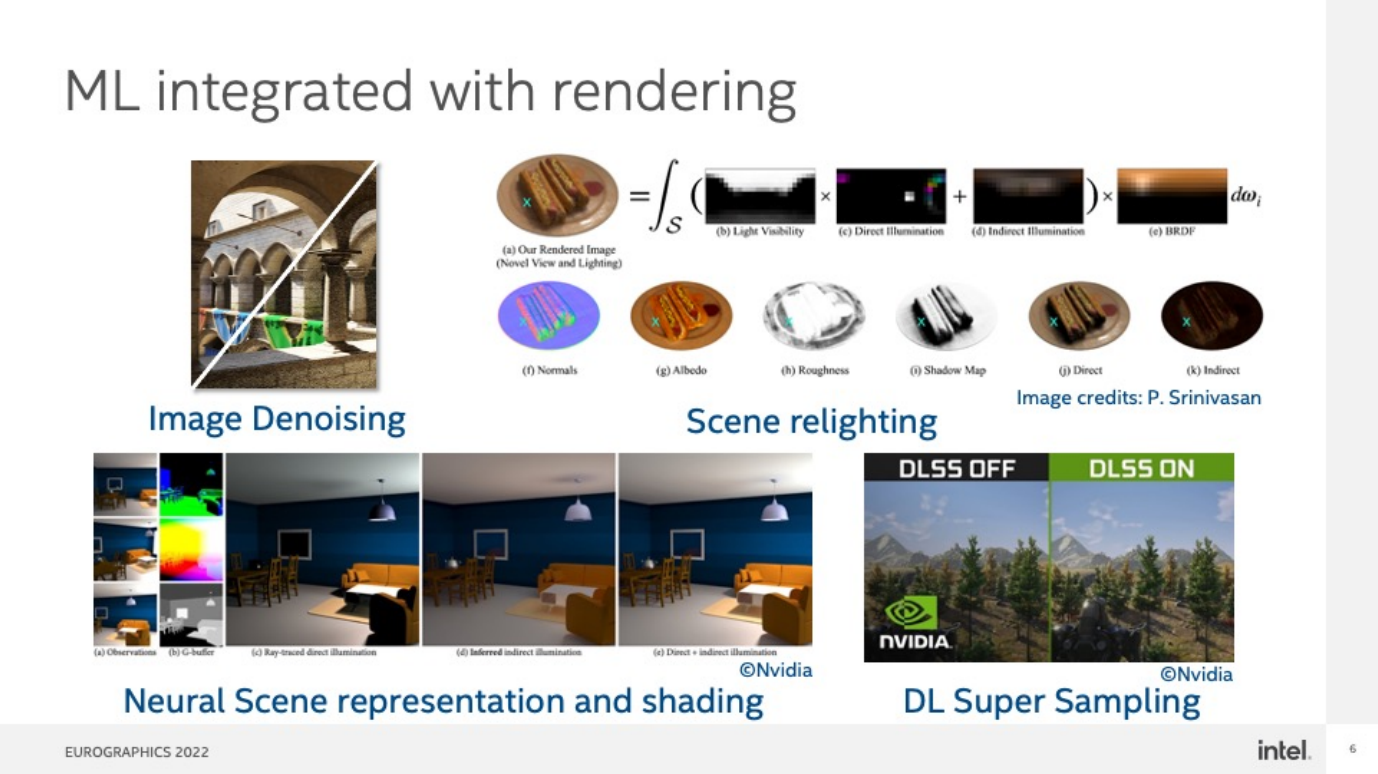
References:

ProgressiveGAN - https://arxiv.org/abs/1710.10196

AvatarMe - https://arxiv.org/abs/2003.13845

Animation - https://dl.acm.org/doi/abs/10.1145/3386569.3392450

GauGAN - https://arxiv.org/abs/1903.07291



ML integrated with rendering

Image Denoising

Scene relighting

Image credits: P. Srinivasan

©Nvidia

Neural Scene representation and shading

©Nvidia

DL Super Sampling

EUROGRAPHICS 2022

intel.

6

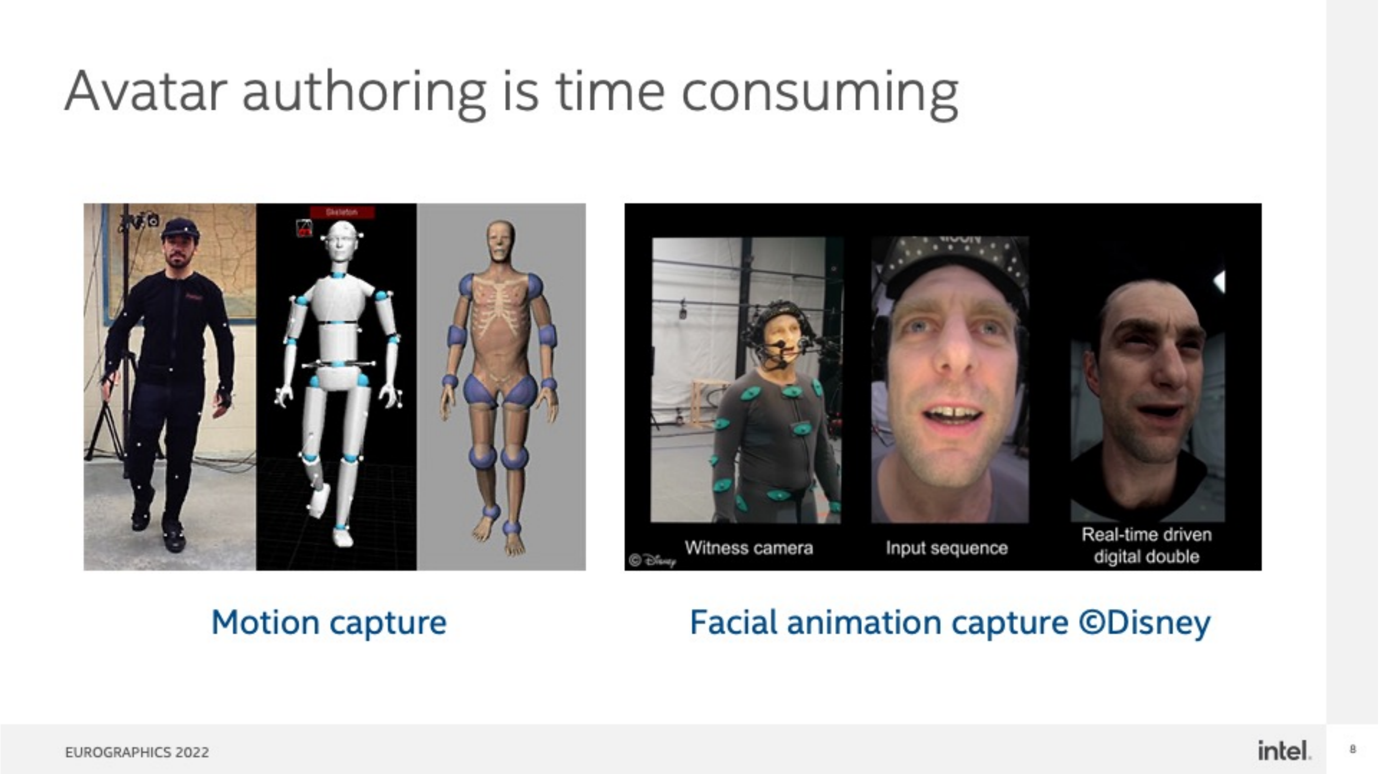
WE also see ML augmenting the rendering pipeline. By augmenting, we mean aiding the rendering engine to generate visuals – better quality at same power, reduced rendering times, or even new ways to generate the content from traditional 3D models/ pixels based ways.

Image denoising has been an interesting use case where past heuristic methods are being replaced by DL based methods, just because the quality is comparable to high spp rendering.
For real-tome rendering, we are seeing DLSS which applies AA and super resolution to a low resolution frame to upscale it to a high res frame.
IN addition to traditional rendering, we are seeing neural rendering – the idea of encoding a scene and its contents (lighting, materials, models) into a latent space representation that can be rendering using a neural renderer. The neural renderer could be used to generate the scene entirely, or only parts of the scene that are computationally expensive – e.g: GI, multi bounce lighting etc.

References:
Image Denoising - https://studios.disneyresearch.com/2018/07/30/denoising-with-kernel-prediction-and-asymmetric-loss-functions/
Scene relighting - ha
Compositional Neural Scene representation - https://jannovak.info/publications/CNSR/CNSR.pdf
DLSS - https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/

ML for content generation

Neural Animation, Codec avatars, Photorealistic backgrounds



Avatar authoring is time consuming

Motion capture

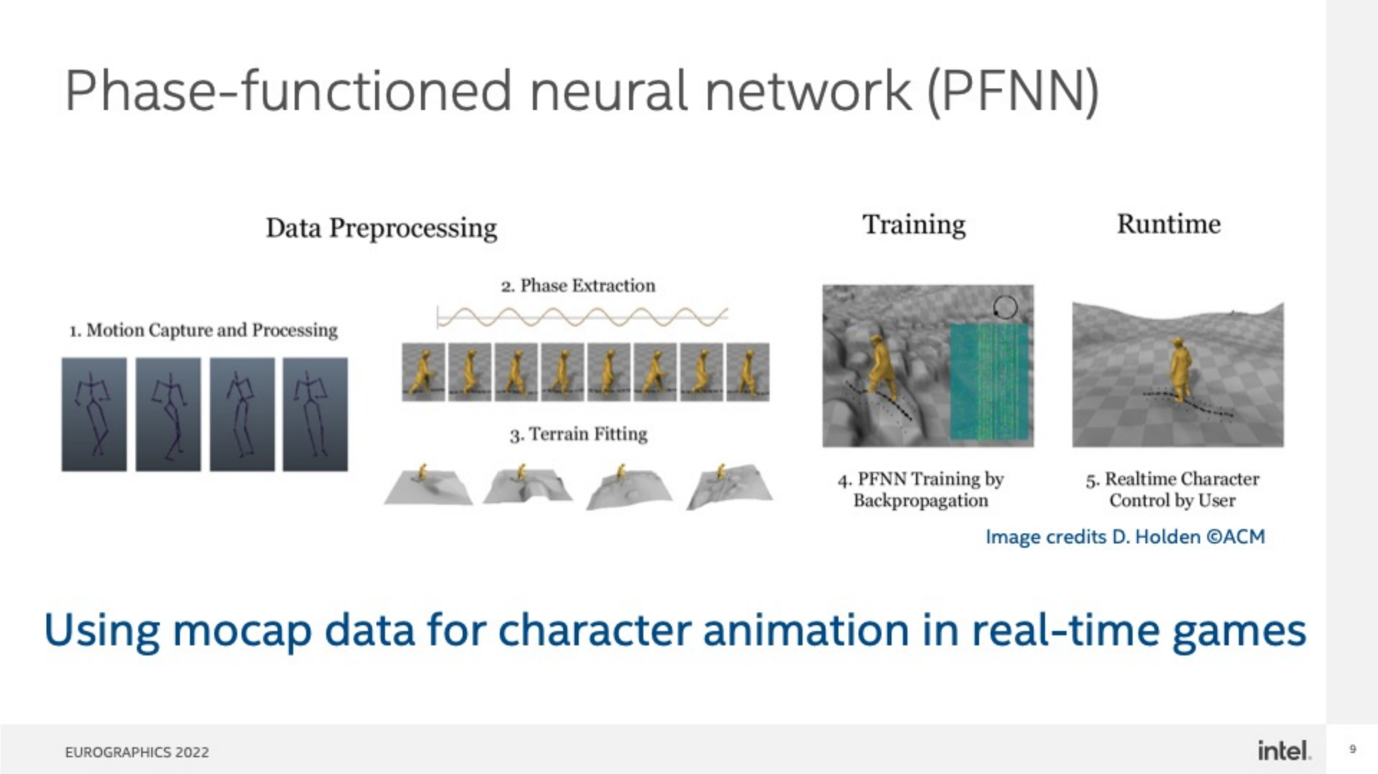
Facial animation capture ©Disney

Let us dive deeper into content generation example – here we look at animating a 3D person model and capturing the expressiveness of the facial features to be replicated in an avatar.

Past application of mocap required a highly trained animator to clean up, categorize and apply the motion captured data to a 3D skeleton that was rigged into the 3D body mesh.
Similarly, capturing and transmission of facial animations to the face was cumbersome. Facial expressions could be captured by marker based or markerless systems, often captured using cameras mounted close to the face.

Recent advances have made it possible to use NNs to ease both parts of this process, including for example, driving facial animations just through spoken audio from the person that is used to generate visually realistic facial animations in the avatar.

Motion capture suit image - https://neuronmocap.com/content/mocap-101-what-motion-capture
S. McDonagh, M. Klaudiny, D. Bradley, T. Beeler, I. Matthews and K. Mitchell, "Synthetic Prior Design for Real-Time Face Tracking," 2016 Fourth International Conference on 3D Vision (3DV), 2016, pp. 639-648, doi: 10.1109/3DV.2016.72.



Phase-functioned neural network (PFNN)

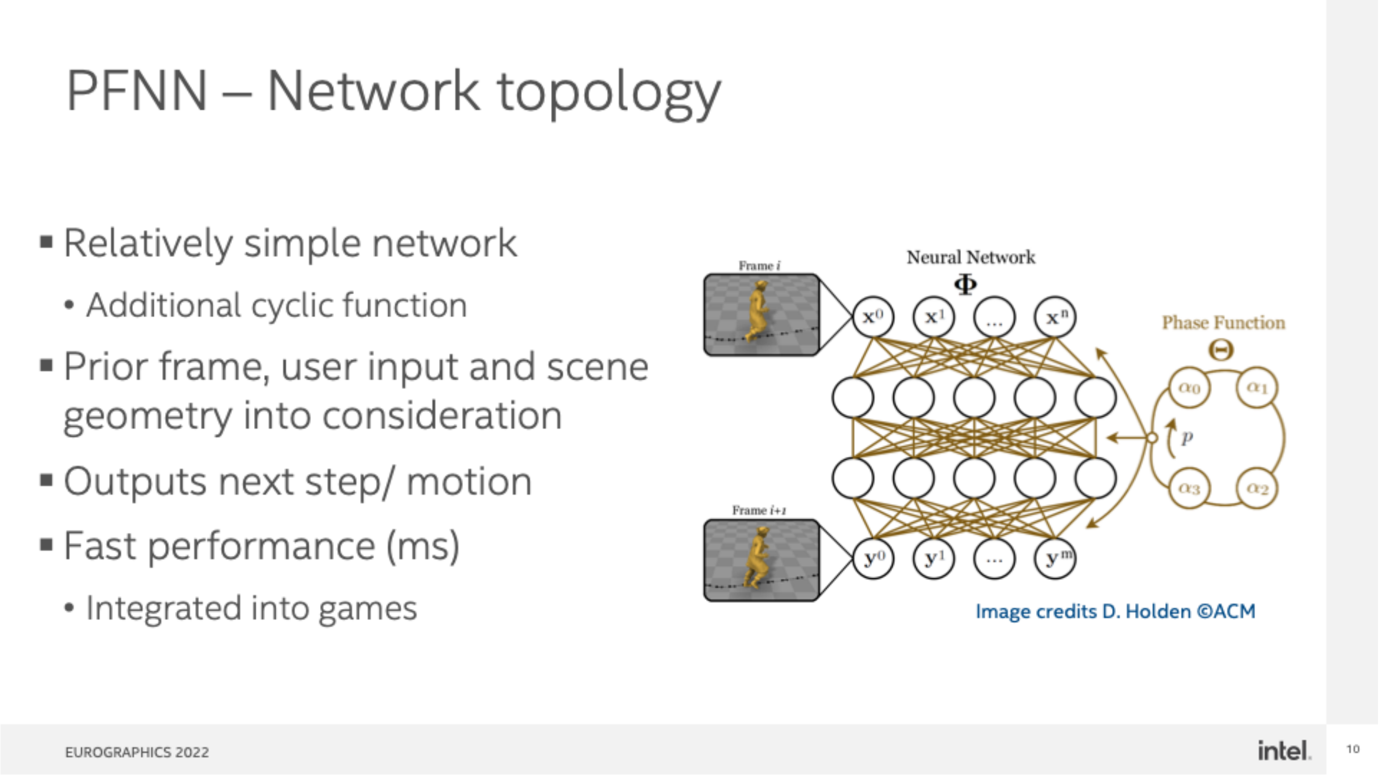
Data Preprocessing
Training
Runtime
1. Motion Capture and Processing
2. Phase Extraction
3. Terrain Fitting
4. PFNN Training by Backpropagation
5. Realtime Character Control by User

Image credits D. Holden ©ACM

Using mocap data for character animation in real-time games

PFNN provides a way to use mocap data to drive character animation in real-time games, that is conditioned on the locomotion, type of gait, terrain and user control of the character. Training dataset is captured and annotated mocap data of actors performing different actions in a controlled setting. This data is characterized to extract the phase and use to train a network that uses terrain model as input. Performance of the model during inference is pretty light and results in real-time driving of character animation.

Phase-functioned neural networks for character control : ACM Transactions on GraphicsVolume 36Issue 4July 2017 Article No.: 42pp 1–13https://doi.org/10.1145/3072959.3073663



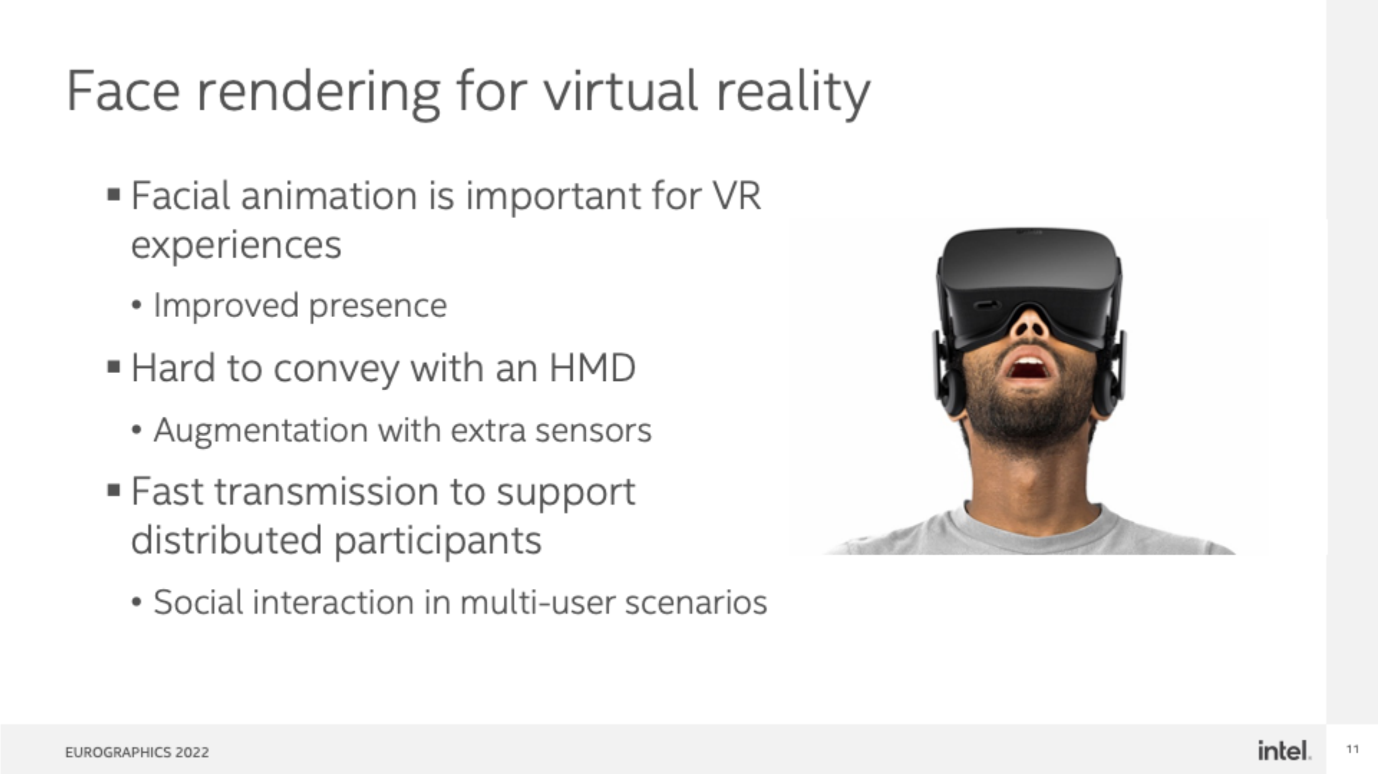
PFNN – Network topology

- Relatively simple network
  - Additional cyclic function
- Prior frame, user input and scene geometry into consideration
- Outputs next step/ motion
- Fast performance (ms)
  - Integrated into games

Image credits D. Holden ©ACM

The network is a simple 3 layer network, where the training weights are cyclically changed based on the phase of the motion (hence phase functioned).
Given the use case of integrating into games, the performance of the network is critical – often the computation of the next skeleton position is in ms.
While the initial training was limited to lower extremities of the person, recent work has resulted in capturing whole body motion as well as challenging use cases such as quadruped locomotion into game content.
Deployments include games such as Ubisoft's Assasins creed, as well as demos.

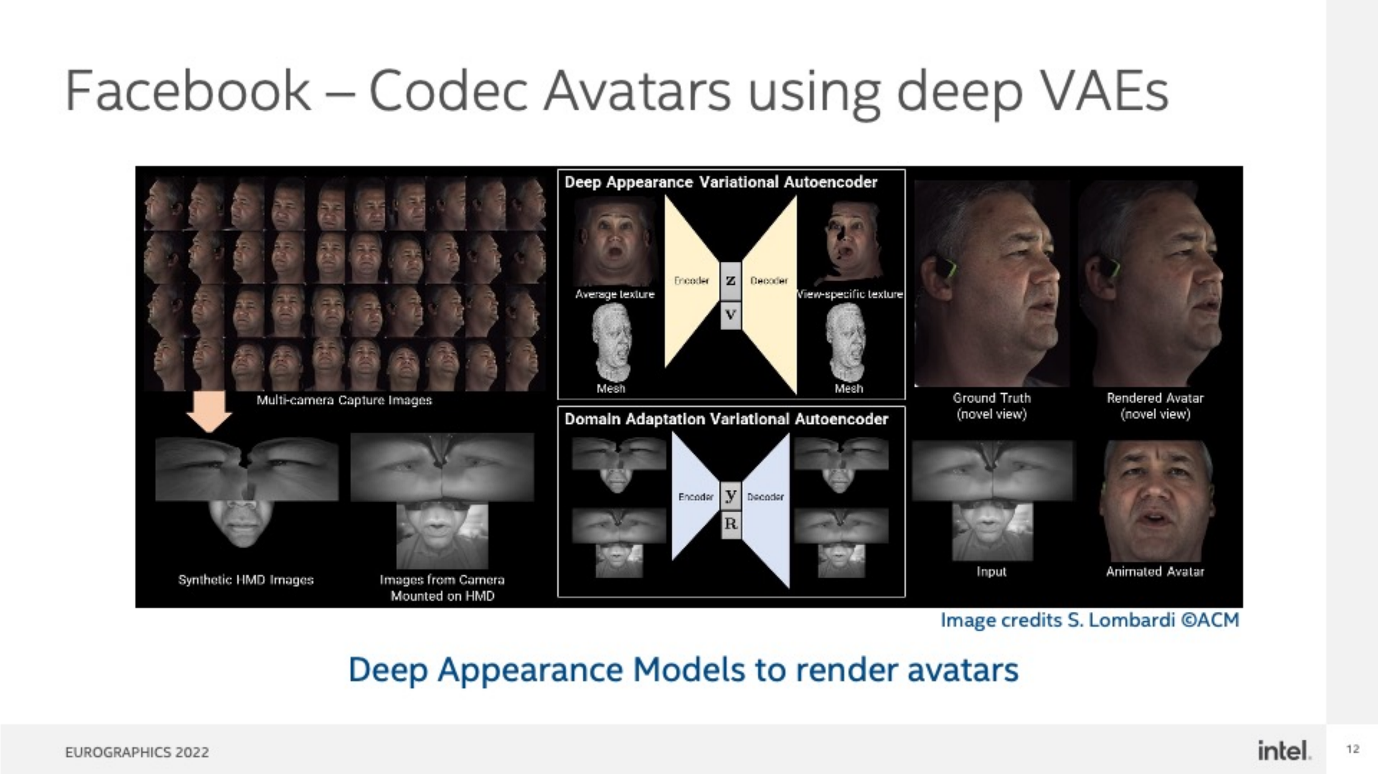
Phase-functioned neural networks for character control : ACM Transactions on GraphicsVolume 36Issue 4July 2017 Article No.: 42pp 1–13https://doi.org/10.1145/3072959.3073663



Face rendering for virtual reality

- Facial animation is important for VR experiences
  - Improved presence
- Hard to convey with an HMD
  - Augmentation with extra sensors
- Fast transmission to support distributed participants
  - Social interaction in multi-user scenarios

Facial expression is important when having virtual conversations – especially in physically distant environments
Zoom does provide video, but limited ability to change viewpoints independent of camera capture.
VR provides the ability for users to have shared presence in a virtual environment, but facial expressions are hard to convey with an HMD that blocks external augmentation of the space with cameras/ sensors.



Facebook – Codec Avatars using deep VAEs

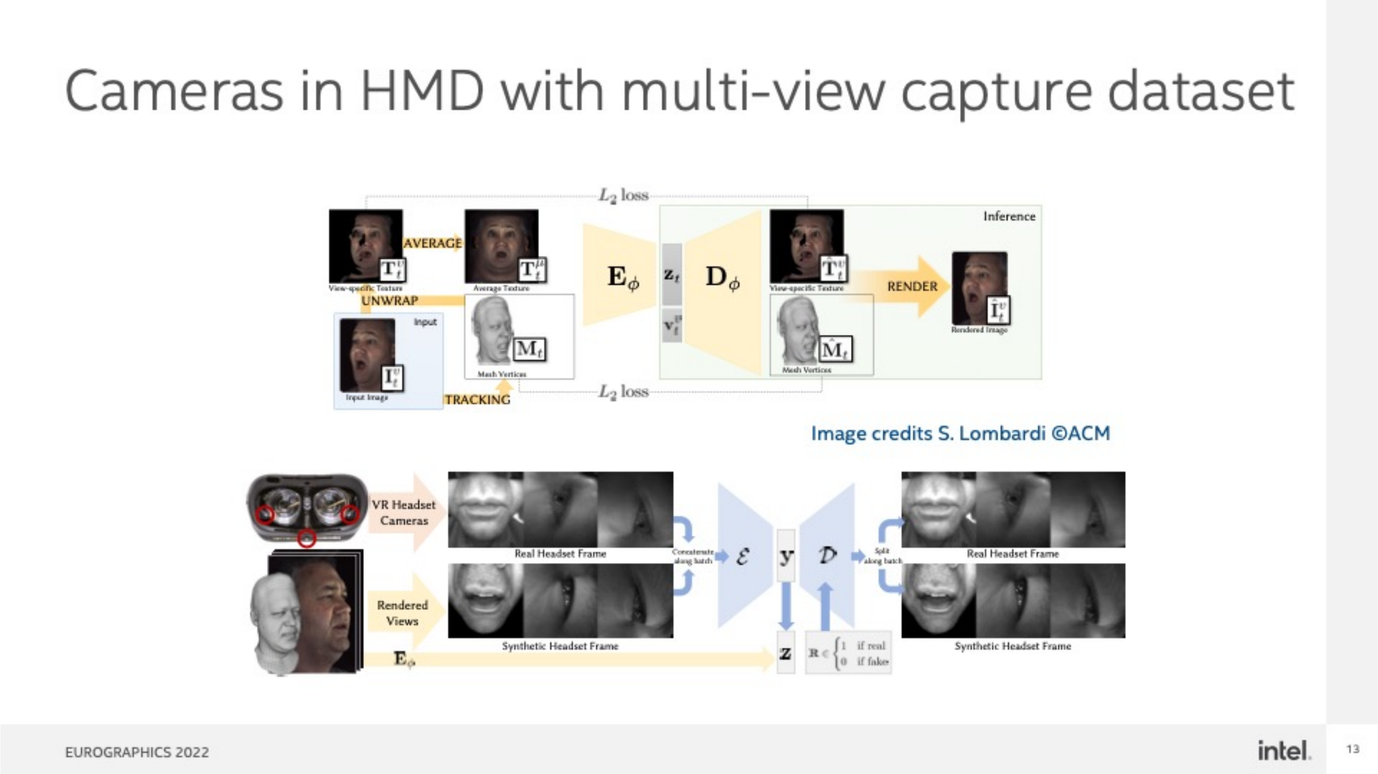
Multi-camera Capture Images
Synthetic HMD Images
Images from Camera Mounted on HMD
Deep Appearance Variational Autoencoder
Average texture
Mesh
Encoder
z
v
Decoder
View-specific texture
Mesh
Domain Adaptation Variational Autoencoder
Encoder
y
R
Decoder
Ground Truth (novel view)
Rendered Avatar (novel view)
Input
Animated Avatar

Image credits S. Lombardi ©ACM

Deep Appearance Models to render avatars

An example of using DL to solve this issue in an end to end system is the codec avatars work from Facebook.
Here there are 2 networks that need to be trained – one for view dependent rendering of face and animations, and another for translating limited camera input from HMD of the user's face to a 3D model.
Training dataset is captured using a multi camera capture system under fixed lighting conditions
A VAE is used for both networks.
The encoder translates the input texture and mesh into a latent representation for the view dependent rendering
The encoder correlated the captures view with the synthetic virtual camera to simulate the full face view – essentially translating from warped camera images to the animated avatar

Deep appearance models for face rendering'
Stephen Lombardi , Jason Saragih, Tomas Simon, Yaser Sheikh
ACM Transactions on GraphicsVolume 37Issue 4August 2018 Article No.: 68pp 1–13https://doi.org/10.1145/3197517.3201401

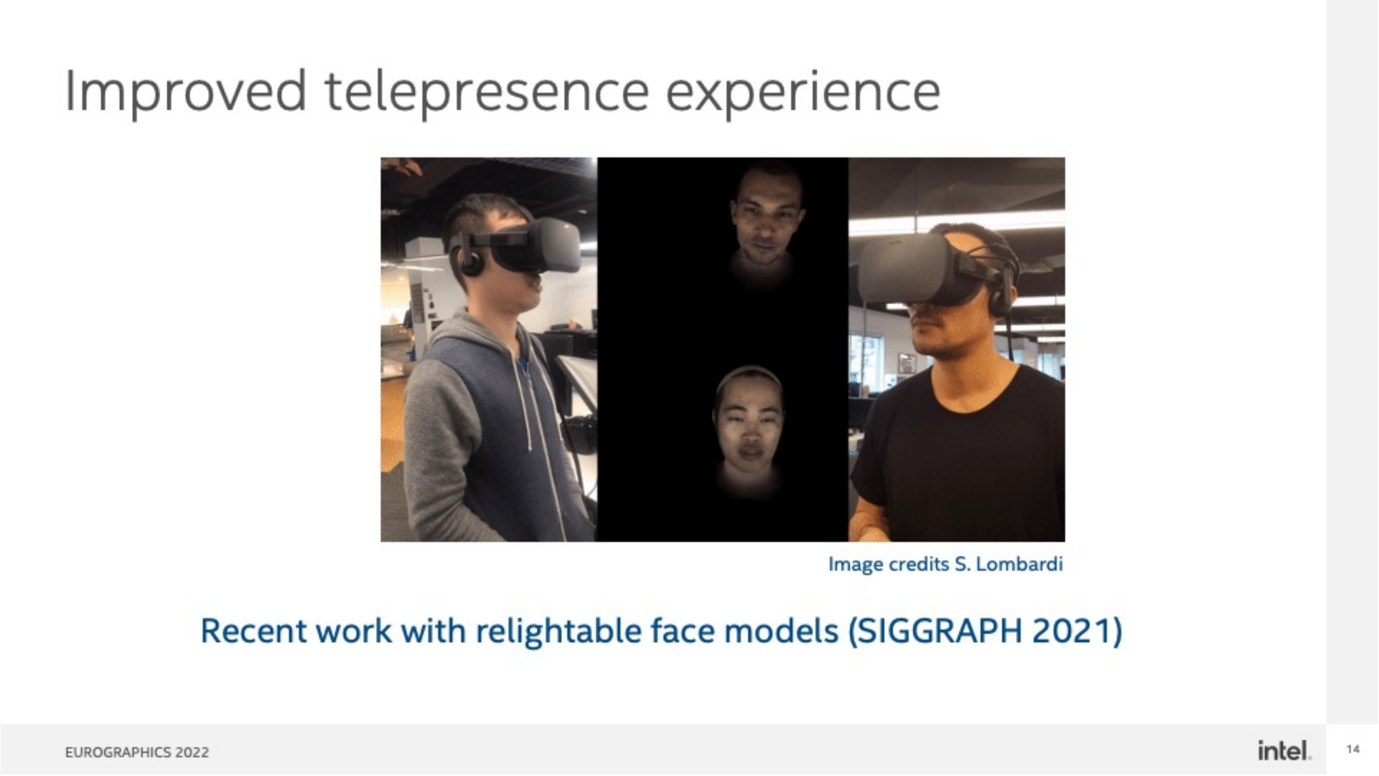


Cameras in HMD with multi-view capture dataset

Image credits S. Lombardi ©ACM

To capture the images from the HMD, cameras are placed and built into the HMD that capture the eye movements as well as the lower part of the face of the user.
Unlike the synthetic virtual camera images, the real camera also captures the background.
The decoder network at the other participant uses the latent space representation and the view dependent term to re-render the talking head of the user with real facial animations.

Deep appearance models for face rendering'
Stephen Lombardi , Jason Saragih, Tomas Simon, Yaser Sheikh
ACM Transactions on GraphicsVolume 37Issue 4August 2018 Article No.: 68pp 1–13https://doi.org/10.1145/3197517.3201401



Improved telepresence experience

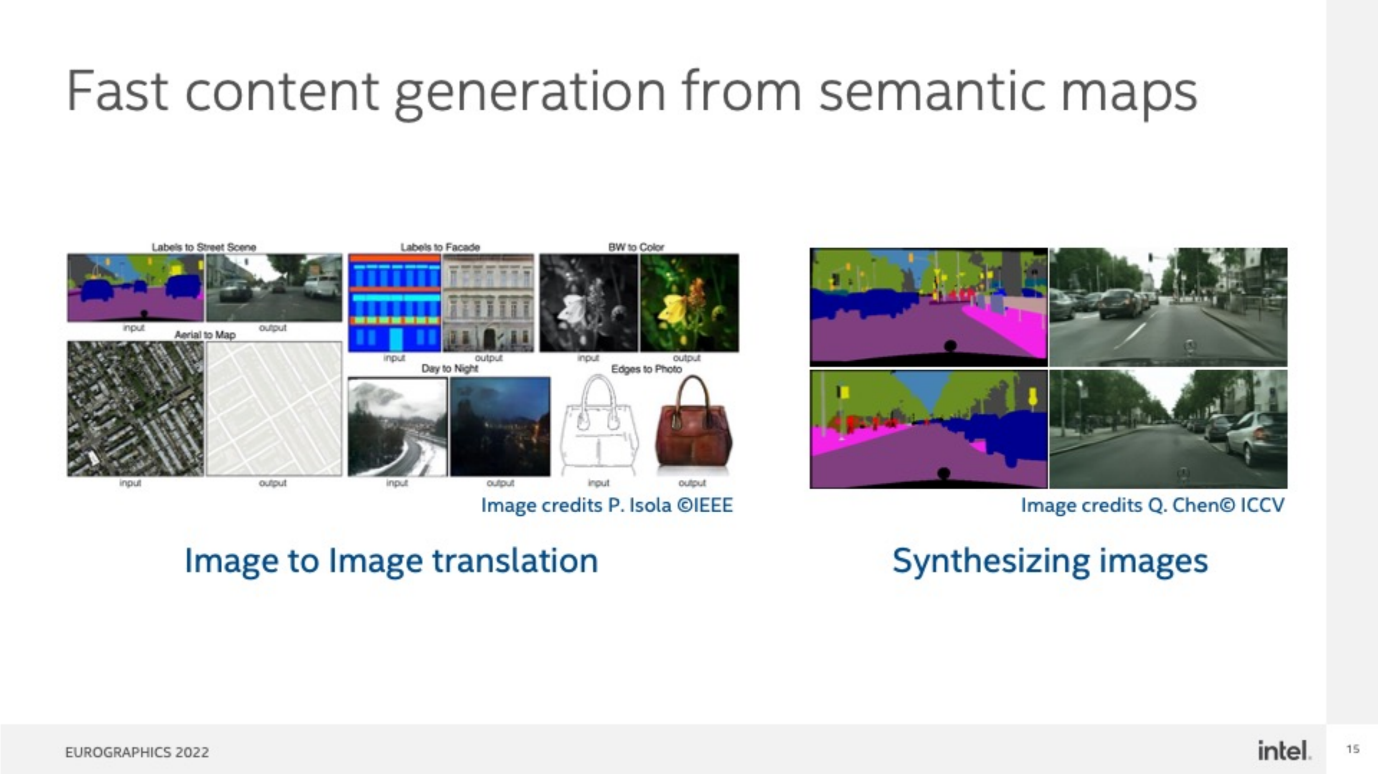
Image credits S. Lombardi

Recent work with relightable face models (SIGGRAPH 2021)

The final outcome is that the user in a shared virtual environment can see the head motions and facial expressions of each other, leading to better presense
Note that the talk head avatar is only lit by the same light source as what was captured. However newer work has looked at relighting the captured face mode, as well as driving this with just audio input (spoked words).

Deep appearance models for face rendering'
Stephen Lombardi , Jason Saragih, Tomas Simon, Yaser Sheikh
ACM Transactions on GraphicsVolume 37Issue 4August 2018 Article No.: 68pp 1–13https://doi.org/10.1145/3197517.3201401

https://stephenlombardi.github.io/projects/deepappearancemodels/



Fast content generation from semantic maps

Image credits P. Isola ©IEEE

Image to Image translation

Image credits Q. Chen© ICCV

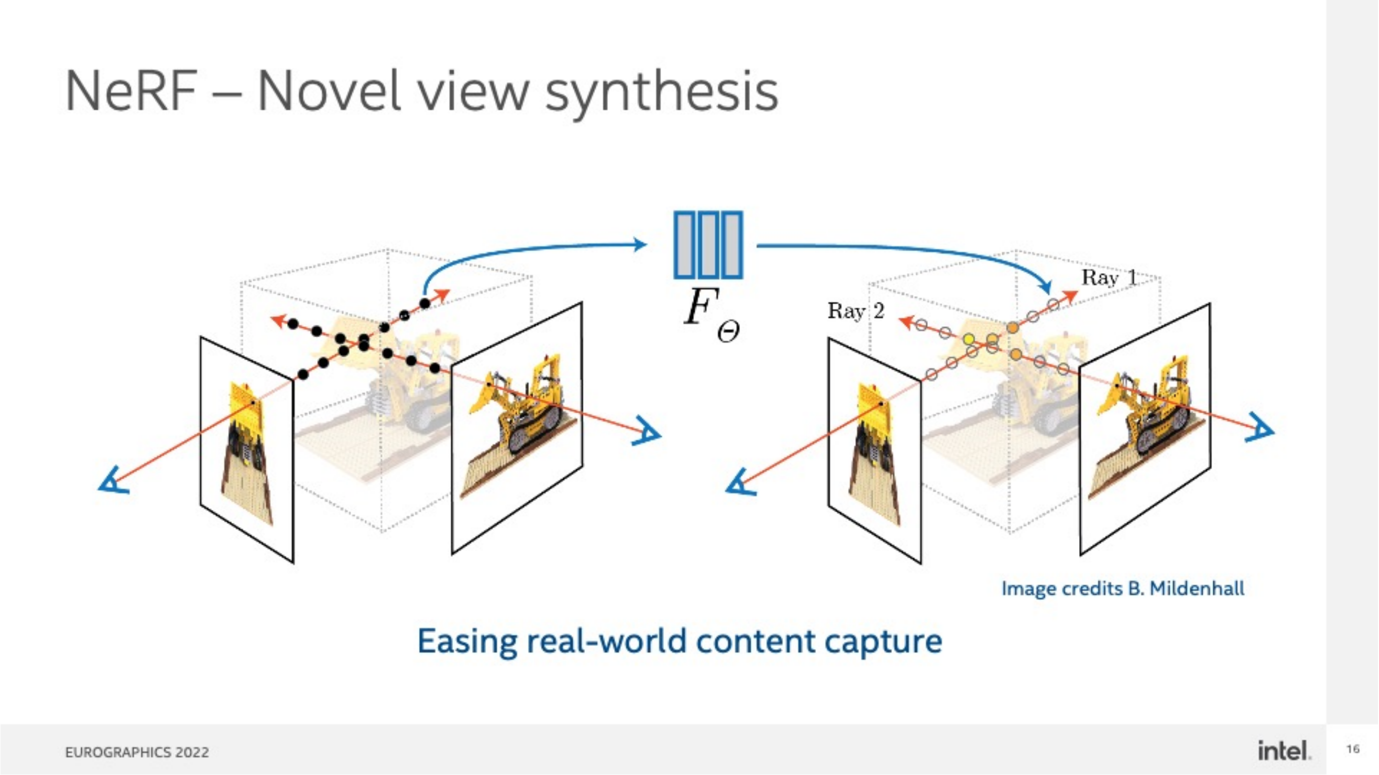
Synthesizing images

Image to image translation has been used across multiple field to stylize or change from one image content domain to another.]
Examples include sketches to images, recoloring b&W images as well as semantic maps to photorealistic images for autonomous driving.
Network used include adversarial networks as well as traditional CNNs – all with the intent of generating high resolution images as output
Temporal consistency is also important for videos, so that this are not artifacts during the rendering process.

Image-to-Image Translation with Conditional Adversarial Networks
Isola, Phillip and Zhu, Jun-Yan and Zhou, Tinghui and Efros, Alexei A
CVPR, 2017

Photographic Image Synthesis with Cascaded Refinement Networks
Qifeng Chen and Vladlen Koltun
International Conference on Computer Vision (ICCV), 2017 (Selected for full oral presentation)

https://openaccess.thecvf.com/content_CVPR_2019/papers/Park_Semantic_Image_Synthesis_With_Spatially-Adaptive_Normalization_CVPR_2019_paper.pdf
https://openaccess.thecvf.com/content_CVPR_2019/html/Park_Semantic_Image_Synthesis_With_Spatially-Adaptive_Normalization_CVPR_2019_paper.html



NeRF – Novel view synthesis

Image credits B. Mildenhall

Easing real-world content capture

An example of democratizing content generation is GauGAN which is used to generate high resolution photorealistic images using a GAN
Similar to image to image networks, the output is conditioned on the various categories in the semantic map'
In addition, the output is also conditioned by the style type of the reference image – e.g: you could generate all outputs to simulate images captured at sunset with the orange tinge to the generated images, without changing the content of the images itself.

https://openaccess.thecvf.com/content_CVPR_2019/papers/Park_Semantic_Image_Synthesis_With_Spatially-Adaptive_Normalization_CVPR_2019_paper.pdf
https://openaccess.thecvf.com/content_CVPR_2019/html/Park_Semantic_Image_Synthesis_With_Spatially-Adaptive_Normalization_CVPR_2019_paper.html

Instant NGP - ~real-time training

This is achieved by introduction od spatially adaptive denormalization blocks. The network is similar to prior work on pix2pixHD

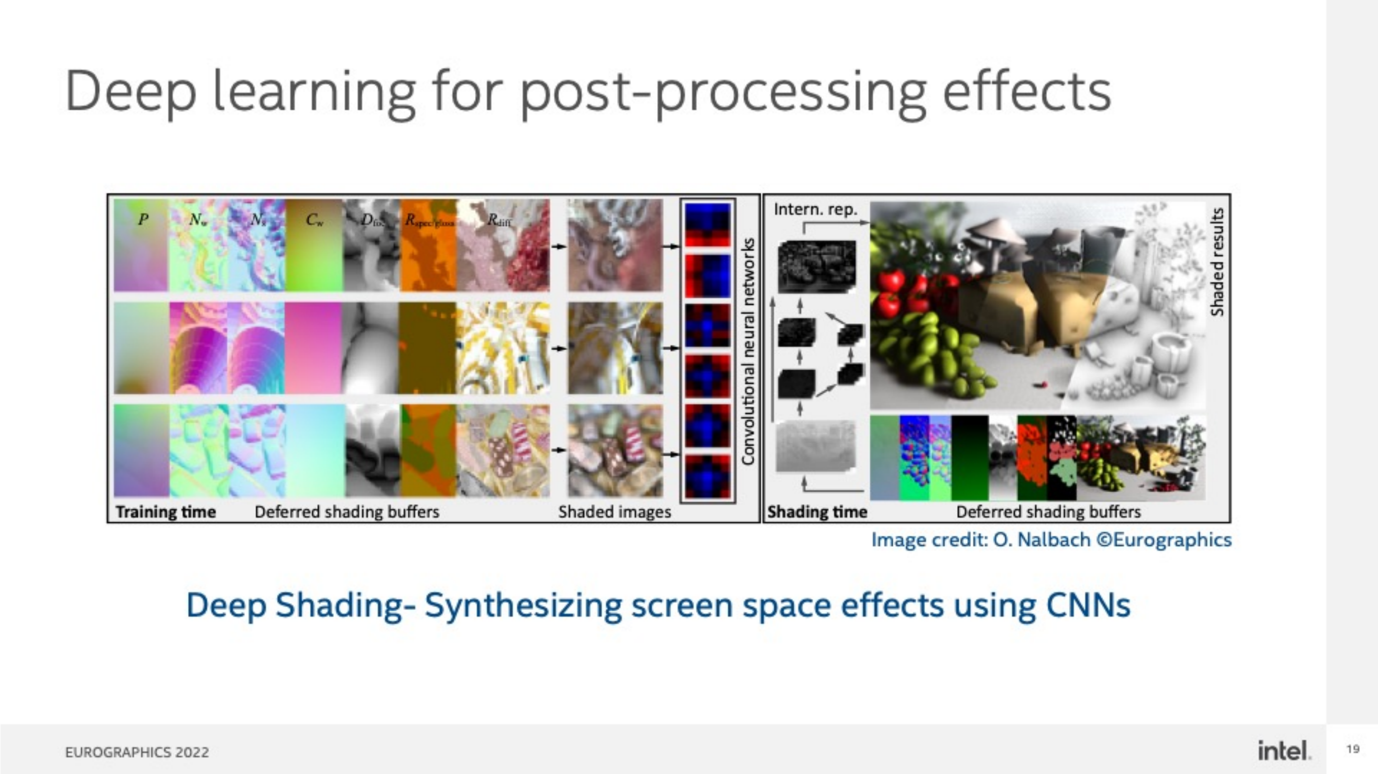
Training dataset uses a variety of open source images of diverse scenes – hence it is able to generate across multiple different indoor and outdoor environments

The output is only limited by the semantic categories that can be generated via the input.

https://openaccess.thecvf.com/content_CVPR_2019/html/Park_Semantic_Image_Synthesis_With_Spatially-Adaptive_Normalization_CVPR_2019_paper.html

ML for Rendering

Post processing, super sampling, denoising



Deep learning for post-processing effects

Image credit: O. Nalbach ©Eurographics

Deep Shading- Synthesizing screen space effects using CNNs

Post processing effects are used in both rasterization and ray tracing pipelines to simulate effects such as anti aliasing, motion blur, depth of field etc.
WE have examples of deep CNNs being used to replace the traditional methods with DL based techniques
Inputs to the network are the buffers generated as a part of the rendering – albedo, normal, depth, motion vectors

Deep Shading: Convolutional Neural Networks for Screen Space Shading
O Nalbach, E Arabadzhiyska, D Mehta, HP Seidel, T Ritschel
Computer Graphics Forum 36 (4), 65-78

Deep Shading network architecture

- U-shaped CNN
- Input buffers depend on post processing effect desired
  - Usually Normals, albedo, motion vectors
- Combined effects using same network
- Fast inference performance

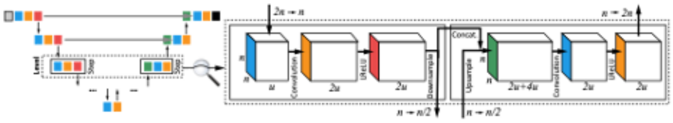
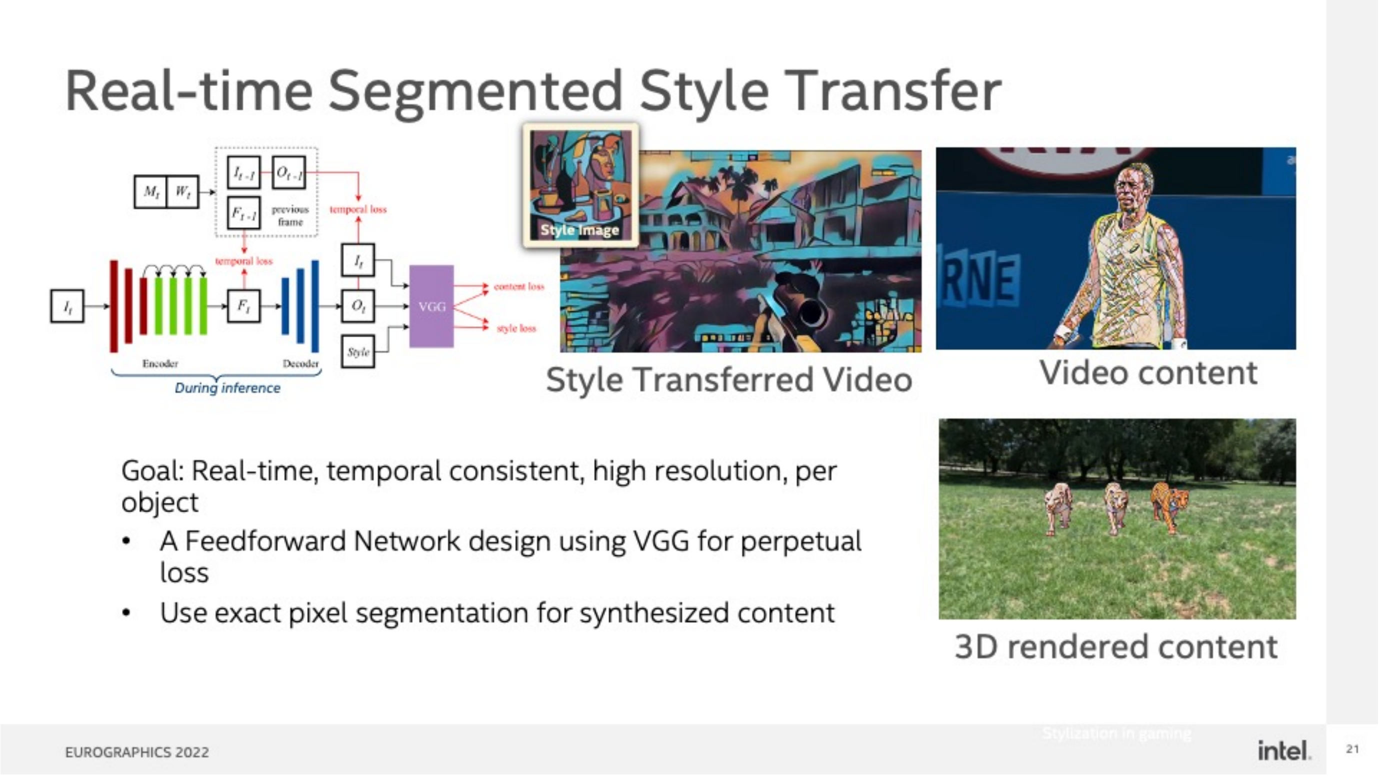


Image credit: O. Nalbach ©Eurographics

In Deep Shading example, the network is au shaped CNN with the network being retrained to perform different effects.
The network could also be used to perform combined effects - eg: depth of field with motion blur using the same network architecture
Performance in the past was in the tens of ms,

Deep Shading: Convolutional Neural Networks for Screen Space Shading
O Nalbach, E Arabadzhiyska, D Mehta, HP Seidel, T Ritschel
Computer Graphics Forum 36 (4), 65-78



Real-time Segmented Style Transfer

Style Transferred Video

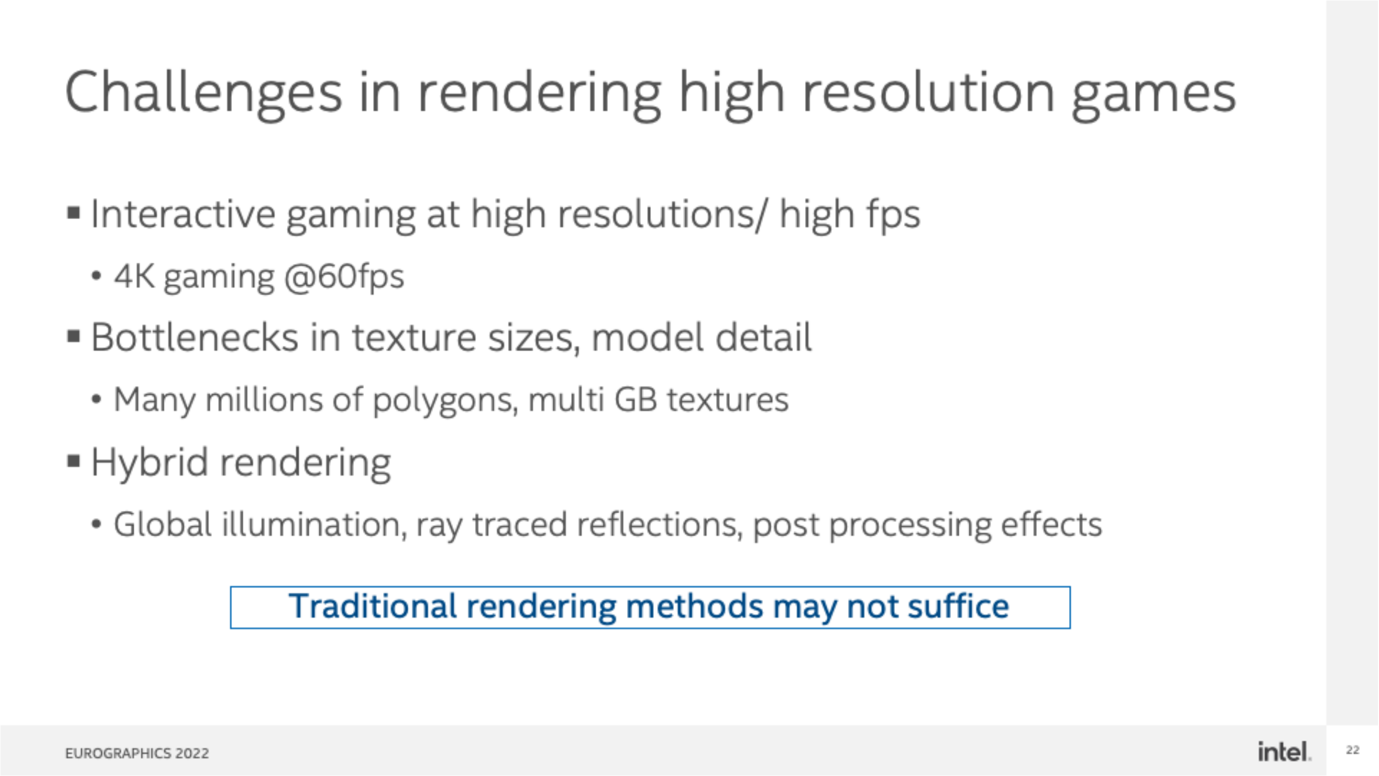
Video content

Goal: Real-time, temporal consistent, high resolution, per object

- A Feedforward Network design using VGG for perpetual loss
- Use exact pixel segmentation for synthesized content

3D rendered content

Here are examples of style transferred videos. Both full frame, segmented and for 3D content. Compared to video segmentation, given we can get pixel precise segmentation and depth maps for rendered content, it is far easier to integrate this directly as a post processing effect in the rendering of a frame.



Challenges in rendering high resolution games

- Interactive gaming at high resolutions/ high fps
  - 4K gaming @60fps
- Bottlenecks in texture sizes, model detail
  - Many millions of polygons, multi GB textures
- Hybrid rendering
  - Global illumination, ray traced reflections, post processing effects

Traditional rendering methods may not suffice

Gaming is moving to high resolution high fps experiences – 4K @60fps
This is hard to render and meet performance limitations across all systems
IN addition to traditional rasterization, we are seeing use of ray tracing for GI, reflections, caustics. Doing this every frame is computationally intensive
Along with known methods, we need to explore other techniques to bridge the performance gap.

Deep learning super sampling (DLSS)

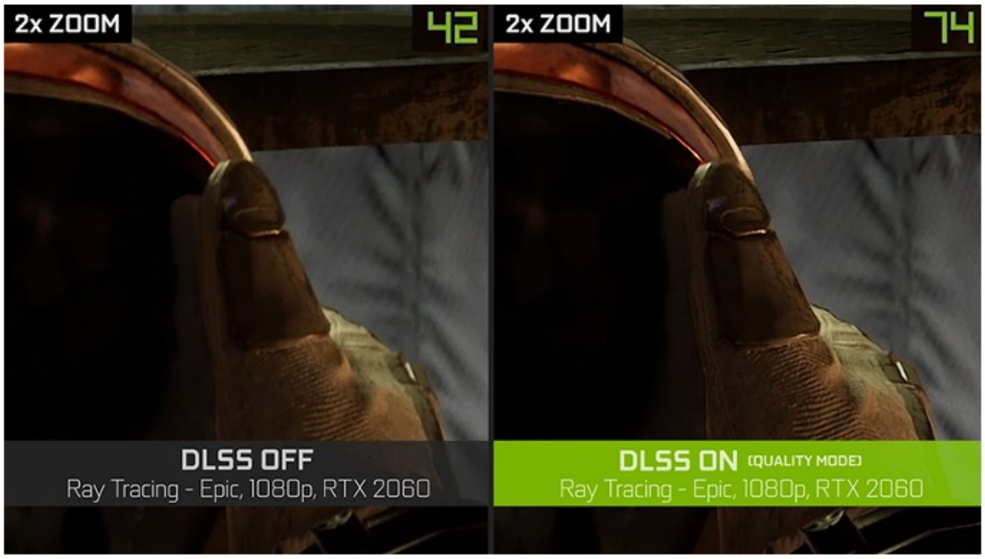

2x ZOOM
42
2x ZOOM
74
DLSS OFF
Ray Tracing - Epic, 1080p, RTX 2060
DLSS ON (QUALITY MODE)
Ray Tracing - Epic, 1080p, RTX 2060

Image credit ©Nvidia

An example of using DL techniques if DLSS – applying AA and super resolution using a DL network to upsample low res images to high res images
The renderer has to input a low res rendered image to the network. The output is a high res image with same or better quality at a fraction of the cost and time needed to render the hish res image
Note the higher fps from the upsampled content.

https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/

DLSS 2.0 – Auto encoder with motion vectors

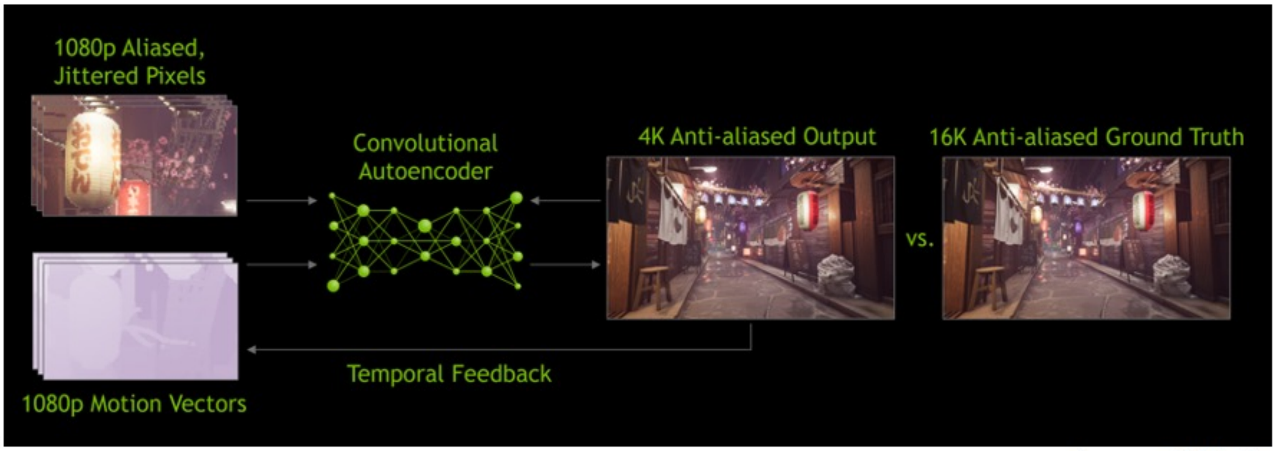

1080p Aliased, Jittered Pixels
Convolutional Autoencoder
4K Anti-aliased Output
16K Anti-aliased Ground Truth
vs.
Temporal Feedback
1080p Motion Vectors

Image credit ©Nvidia

The latest iteration uses an auto encoder with past frames as well as motion vectors as input to maintain temporal consistency.
Ground truth during training is highly sampled anti aliased images
Input is 1080p images (color and MV) and output is 4K image

https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/

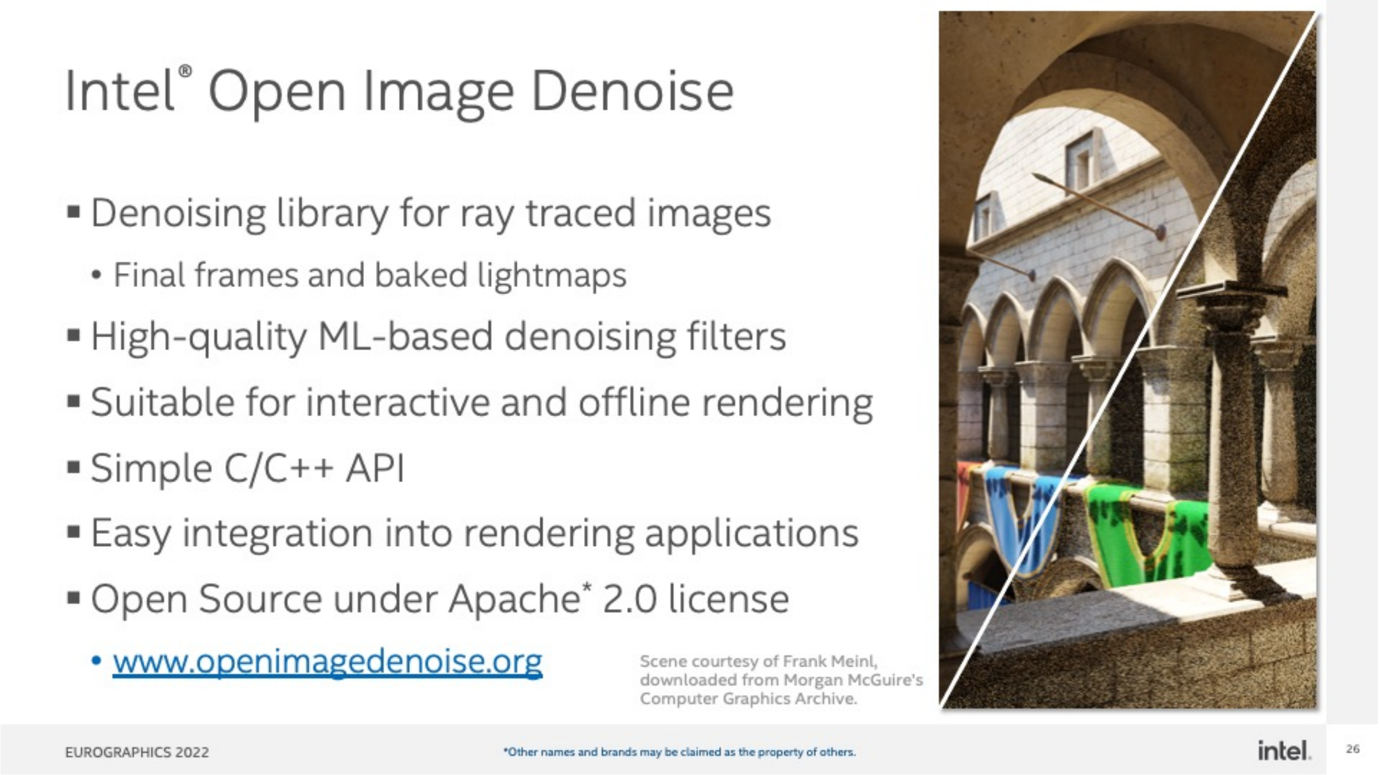
Improved performance in games

- Render low resolution image
  - Image upscale using DL
- Async compute using Tensorcores
  - Significant performance improvement v/s rendering at higher resolution
- Wide adoption
  - Unity/ Unreal

DLSS OFF
DLSS ON

Image credit ©Nvidia

The performance is obtained through async compute using Tensorcores – rendering and DL happen independently
Quality comparisons show almost indistinguishable differences – at improved frame rates and performance
In the past the network had to be retrained for each game content, but now a single network can be deployed across multiple games.
While the technique itself is promising, the adoption within game engines as a plugin deployment mechanism enables it to reach across different systems with minimal overhead.



Intel® Open Image Denoise

- Denoising library for ray traced images
  - Final frames and baked lightmaps
- High-quality ML-based denoising filters
- Suitable for interactive and offline rendering
- Simple C/C++ API
- Easy integration into rendering applications
- Open Source under Apache* 2.0 license
  - www.openimagedenoise.org

Scene courtesy of Frank Meinl, downloaded from Morgan McGuire's Computer Graphics Archive.

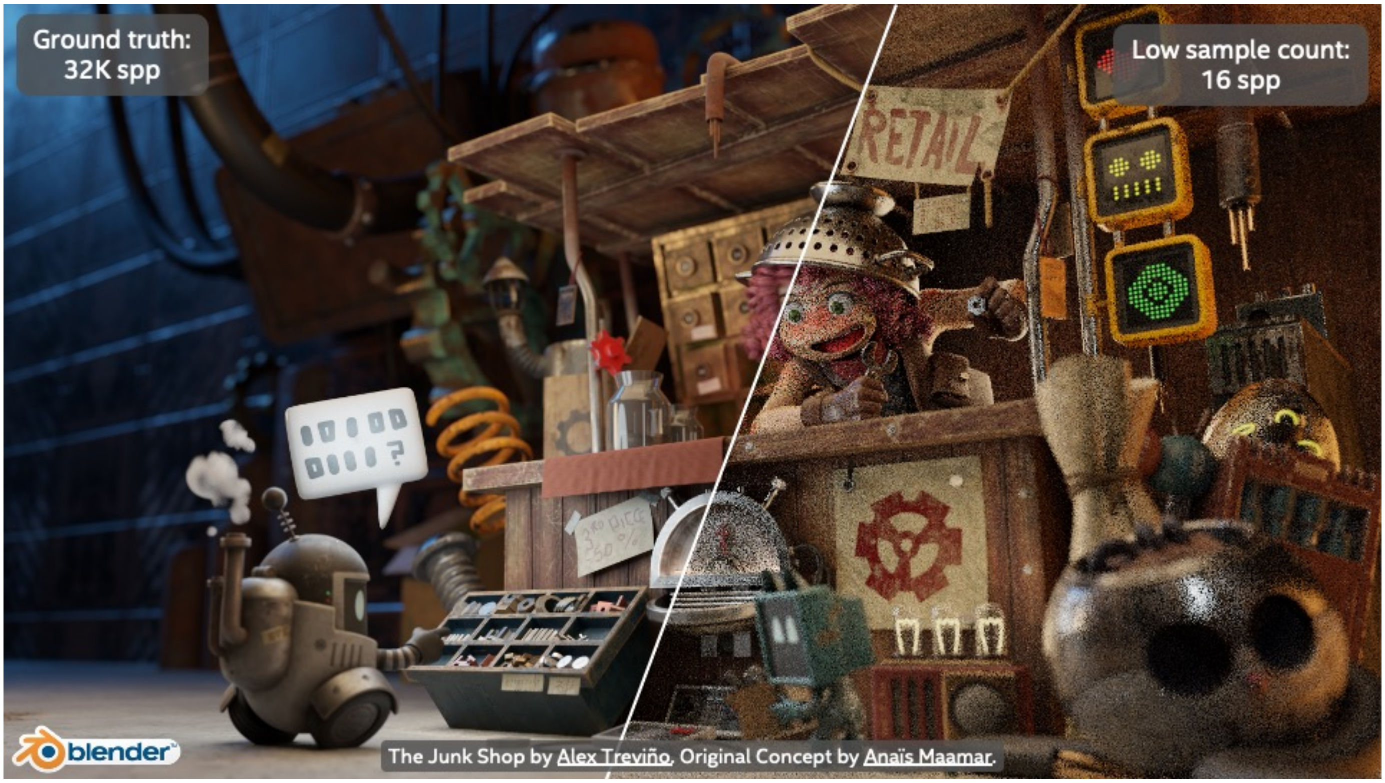
*Other names and brands may be claimed as the property of others.

So what is Intel Open Image Denoise? It is a library for denoising ray traced images, which are most commonly rendered with Monte Carlo path tracing. It can denoise final frames and baked lightmaps as well. To achieve this, it uses a collection of high-quality machine learning based denoising filters, which are suitable for both preview and offline rendering. One of the key features that makes Open Image Denoise quite powerful is its very simple C/C++ API, which makes integrating the library into rendering applications suprisingly easy. And last but not least, the library is completely open source and is available under the permissive Apache 2.0 license, which makes it even easier to adopt and customize.

Ground truth:
32K spp
RETAIL
blender

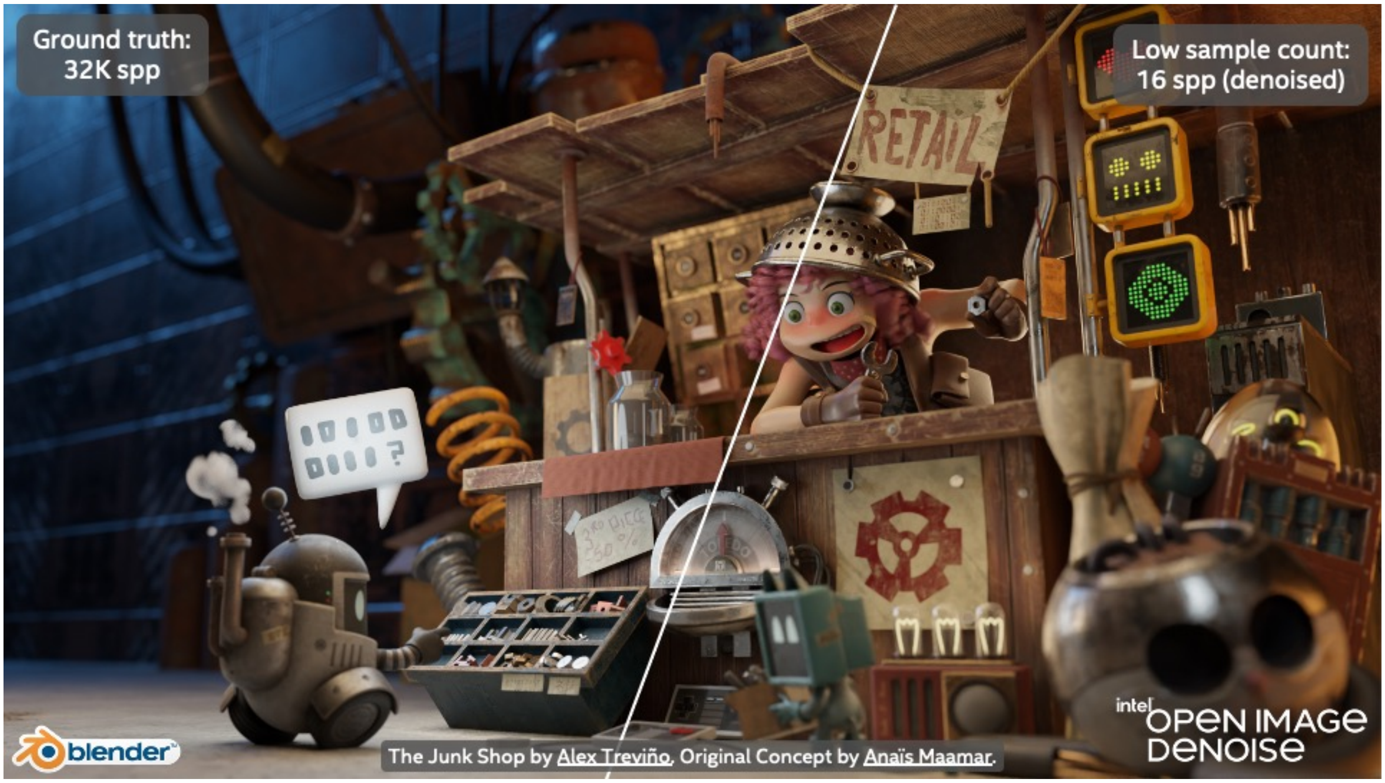
The Junk Shop by Alex Treviño, Original Concept by Anaïs Maamar.

So why should we use Monte Carlo path tracing for rendering images? The answer is that it can capture the wide variety of effects of light transport that other rendering algorithms typically end up becoming overburdened by. Each pixel is taking a random sample of the scene and over the course of many samples per pixel it converges to an accurate image. But, unfortunately, full convergence can be quite slow, resulting in noisy images. Fully converged final frame production quality renders can often take hours or even days on today's hardware, which is detrimental to the artistic process. A potential solution to this problem is using denoising algorithms. The question then becomes: can we get something as good as this ground truth image here, done with 32 thousand samples per pixel, but on a much smaller budget?

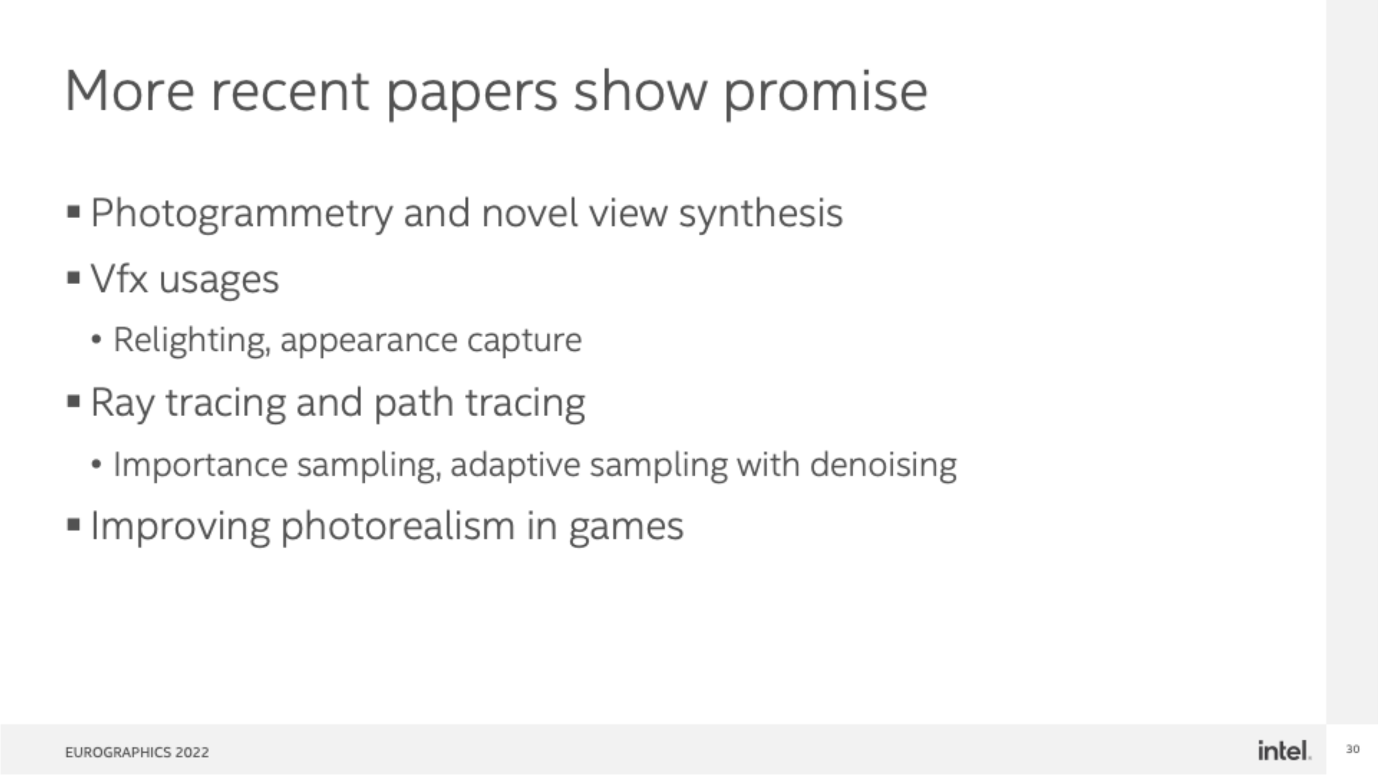


The Junk Shop by Alex Treviño, Original Concept by Anaïs Maamar.

That is, can we do it with orders of magnitude less samples? Here is a raw render with only 16 samples per pixel next to ground truth.



And the answer is yes, with Intel Open Image Denoise we can efficiently get much better estimates of the final resulting image in a fraction of the rendering time needed for a fully converged image. And when used on higher sample counts, say around 1,000 samples per pixel, we can get results virtually indistinguishable from the ground truth, but much faster.



More recent papers show promise

- Photogrammetry and novel view synthesis
- Vfx usages
  - Relighting, appearance capture
- Ray tracing and path tracing
  - Importance sampling, adaptive sampling with denoising
- Improving photorealism in games

More recent usages for content generation include

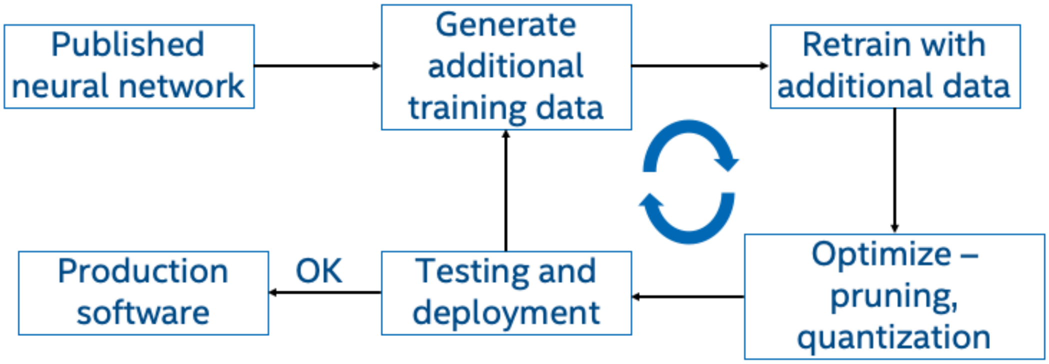
- photo-realistic mesh and texture generation using Multiview images
- Cinematic relighting and appearance capture for movies and animations
- Speeding up ray tracing using DL for adaptive denoising and importance sampling
- As well as improving the photo realism of games with a network applies to the full frame buffer

These are just research applications, but the promise to improve output is evident

Challenges

Datasets, neural networks, deployment

Testing and deploying a published ML model



Published neural network

Generate additional training data

Retrain with additional data

Optimize – pruning, quantization

Testing and deployment

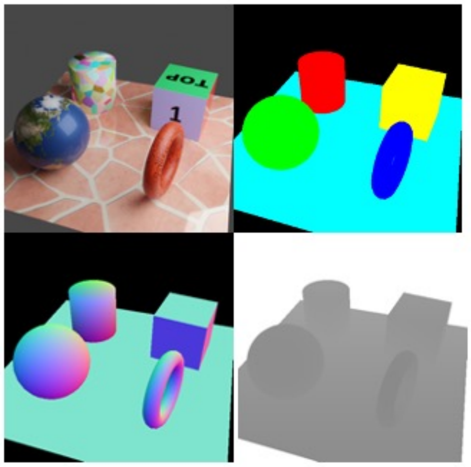
OK

Production software

Iterative process and requires a lot of additional steps

Dataset curation and augmentation

- Common buffers used – normal, albedo, color, position, depth, specular, motion vectors
- Most data can be directly obtained from renderers
- Input resolution
- Rendering time for dataset



Some considerations for dataset generation

- Size of dataset to be collected – small v/s large
  - Training time v/s quality
- Licensing of datasets – open v/s closed
- Generalizability across different scenes
  - Rendering time implications
- Compressed v/s uncompressed data
  - Memory costs and training time
- Data format – color space, dynamic range

Neural network architecture and optimizations

- Takes some effort to deploy published work
  - Understand performance targets and deployment system
- Common network optimizations
  - Pruning, quantization, sparsity
- Use tools such as TensorRT, OpenVINO for auto optimization
  - Most take an ONNX file as input
- Considerations for extending from images to videos
  - Minimize flicker, include temporal loss terms

Deployment considerations

- Training generally uses Pytorch/ Tensorflow
  - Impractical for deployment in real-time usages
- Hardware compatibility and driver support
  - Fallback software path may not be as performant as hardware supported path
- Standards and APIs evolving to support ML
  - DirectML with DirectX, ONNX as model interchange format
- Third party and ecosystem support
  - E.g: Unity Barracuda for inference deployments

Exciting time to be in graphics

- Increased use of ML in graphics
- Potential to improve quality, reduce rendering times and democratize content generation costs
- Improved hardware and systems support,

But..

- Challenges – datasets, networks, deployments

We have just scratched the surface

References

- Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings, S. Bako*, T. Vogels*, B. McWilliams, M. Meyer, J. Novák, A. Harvill, P. Sen, T. DeRose, and F. Rousselle, "Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings," ACM Transactions on Graphics, Vol. 36, No. 4, Article 97, July 2017, (Proceedings of ACM SIGGRAPH 2017)
- ProgressiveGAN - https://arxiv.org/abs/1710.10196
- AvatarMe: Realistically Renderable 3D Facial Reconstruction "In-the-Wild", Lattas, Alexandros and Moschoglou, Stylianos and Gecer, Baris and Ploumpis, Stylianos and Triantafyllou, Vasileios and Ghosh, Abhijeet and Zafeiriou, Stefanos, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2020, Animation - https://dl.acm.org/doi/abs/10.1145/3386569.3392450
- Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu., "Semantic Image Synthesis with Spatially-Adaptive Normalization", in CVPR, 2019.
- Denoising with kernel prediction and asymmetric loss functions, Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, Jan Novák, ACM Transactions on Graphics, Volume 37, Issue 4, August 2018 Article No.: 124, pp 1–15, https://doi.org/10.1145/3197517.3201388
- Srinivasan, Pratul & Deng, Boyang & Zhang, Xiuming & Tancik, Matthew & Mildenhall, Ben & Barron, Jonathan. (2020). NeRV: Neural Reflectance and Visibility Fields for Relighting and View Synthesis.
- Compositional Neural Scene Representations for Shading Inference, Jonathan Granskog, Fabrice Rousselle, Marios Papas, Jan Novák, Transaction on Graphics (Proceedings of SIGGRAPH 2020), vol. 39, no. 4
- DLSS 2.0, retrieved June 2021, https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/
- S. McDonagh, M. Klaudiny, D. Bradley, T. Beeler, I. Matthews and K. Mitchell, "Synthetic Prior Design for Real-Time Face Tracking," 2016 Fourth International Conference on 3D Vision (3DV), 2016, pp. 639-648, doi: 10.1109/3DV.2016.72.
- Phase-functioned neural networks for character control : ACM Transactions on GraphicsVolume 36Issue 4July 2017 Article No.: 42pp 1–13, https://doi.org/10.1145/3072959.3073663
- Deep appearance models for face rendering, Stephen Lombardi , Jason Saragih, Tomas Simon, Yaser Sheikh, ACM Transactions on GraphicsVolume 37Issue 4August 2018 Article No.: 68pp 1–13https://doi.org/10.1145/3197517.3201401
- Image-to-Image Translation with Conditional Adversarial Networks, Isola, Phillip and Zhu, Jun-Yan and Zhou, Tinghui and Efros, Alexei A, CVPR, 2017
- Photographic Image Synthesis with Cascaded Refinement Networks, Qifeng Chen and Vladlen Koltun, International Conference on Computer Vision (ICCV), 2017
- Deep Shading: Convolutional Neural Networks for Screen Space Shading, O Nalbach, E Arabadzhiyska, D Mehta, HP Seidel, T Ritschel, Computer Graphics Forum 36 (4), 65-78
- Thomas, M. M. and Forbes, A. G., "Deep Illumination: Approximating Dynamic Global Illumination with Generative Adversarial Network", arXiv e-prints, 2017

Schedule

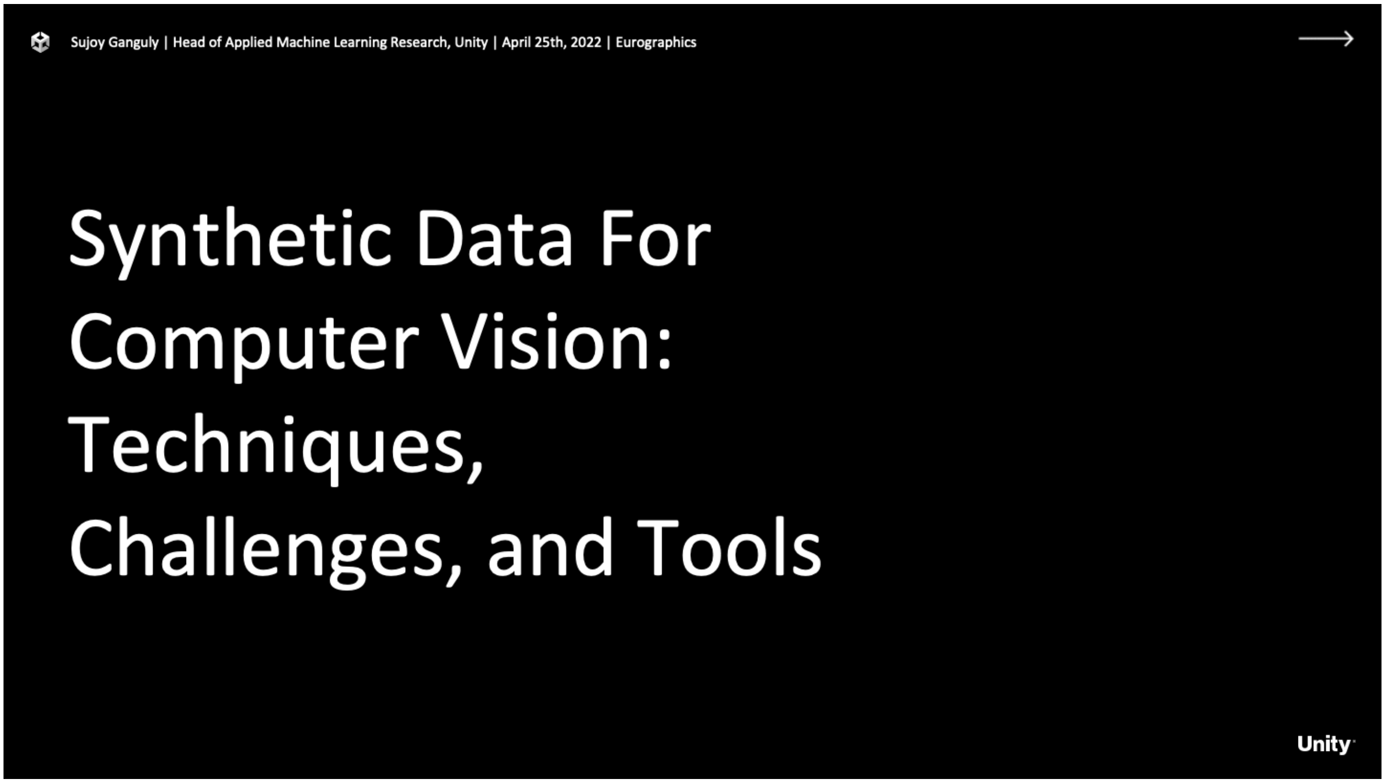
Introduction
Carl S. Marshall, Reality Labs Research at Meta
15 mins

ML for Graphics: A Brief Overview
Deepak Vembar, Intel Labs
40 mins

Synthetic Data For Computer Vision: Techniques, Challenges, and Tools
Sujoy Ganguly, Unity
40 mins

Machine Learning in Real-time
Florent Guinier, Unity Labs
40 mins

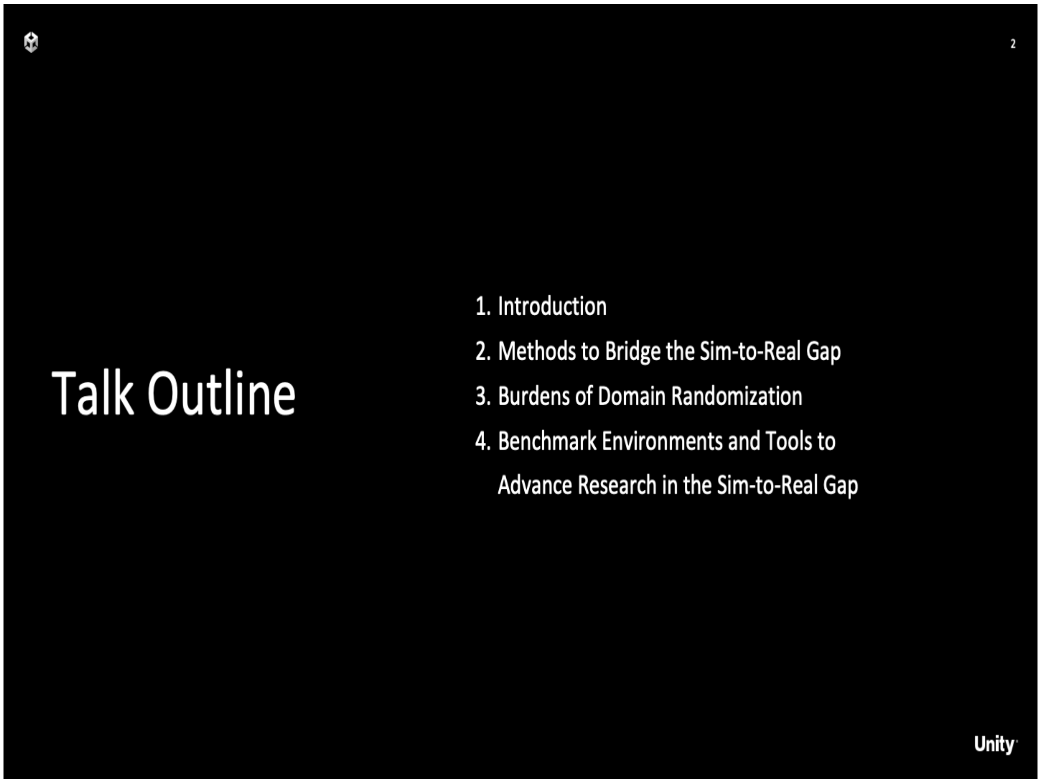
Conclusion
5 mins



Sujoy Ganguly | Head of Applied Machine Learning Research, Unity | April 25th, 2022 | Eurographics

Synthetic Data For Computer Vision: Techniques, Challenges, and Tools

Unity

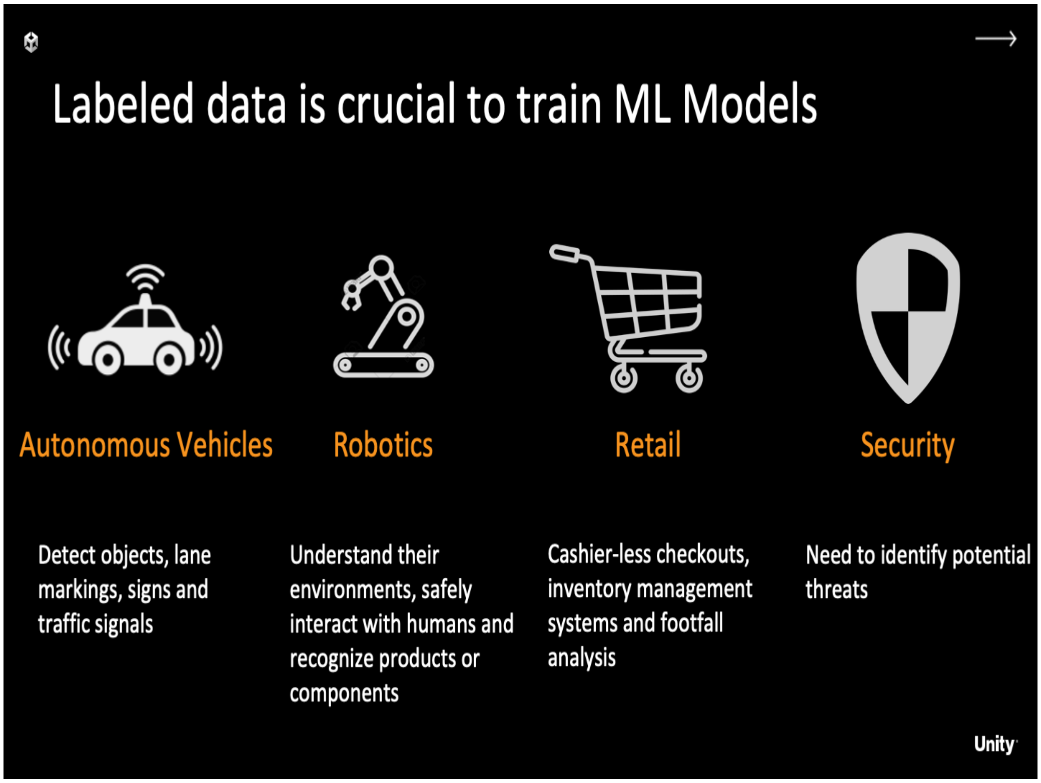


Talk Outline

1. Introduction
2. Methods to Bridge the Sim-to-Real Gap
3. Burdens of Domain Randomization
4. Benchmark Environments and Tools to Advance Research in the Sim-to-Real Gap

Unity

Introduction to challenges of train on real world data and the way synthetic data can help.
Methods that bridge the gap between models train on synthetic data and their real world performance.
Burdens those methods place on the simulation environment and content.
The tools we have developed to aid in create synthetic data.
Finally I will present a set of environments and tools for researchers to use to study and close the sim to real gap.



Labeled data is crucial to train ML Models

Autonomous Vehicles

Detect objects, lane markings, signs and traffic signals

Robotics

Understand their environments, safely interact with humans and recognize products or components

Retail

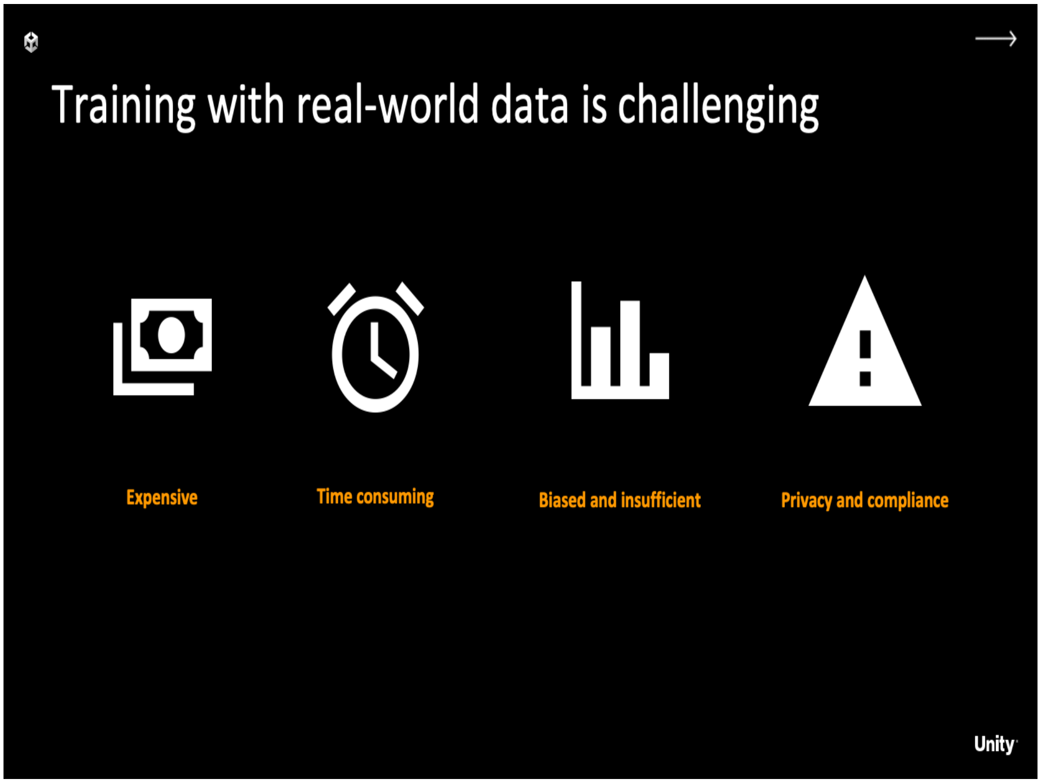
Cashier-less checkouts, inventory management systems and footfall analysis

Security

Need to identify potential threats

Unity

In order to build these systems we need to train them on massive amounts of labelled data. That is examples from the paired with labels on what you information want to extract from those examples.
For vision based systems the current method of getting data is to capture images from real world and then label it.
The entire process is manual and labor intensive, making it expensive and time consuming.
As a result there is not always sufficient data and the sample set that we may have can be biased.
Finally there are situations where we can get the real world data that we need due to privacy and compliance issues.



Training with real-world data is challenging

Expensive

Time consuming

Biased and insufficient

Privacy and compliance

Unity

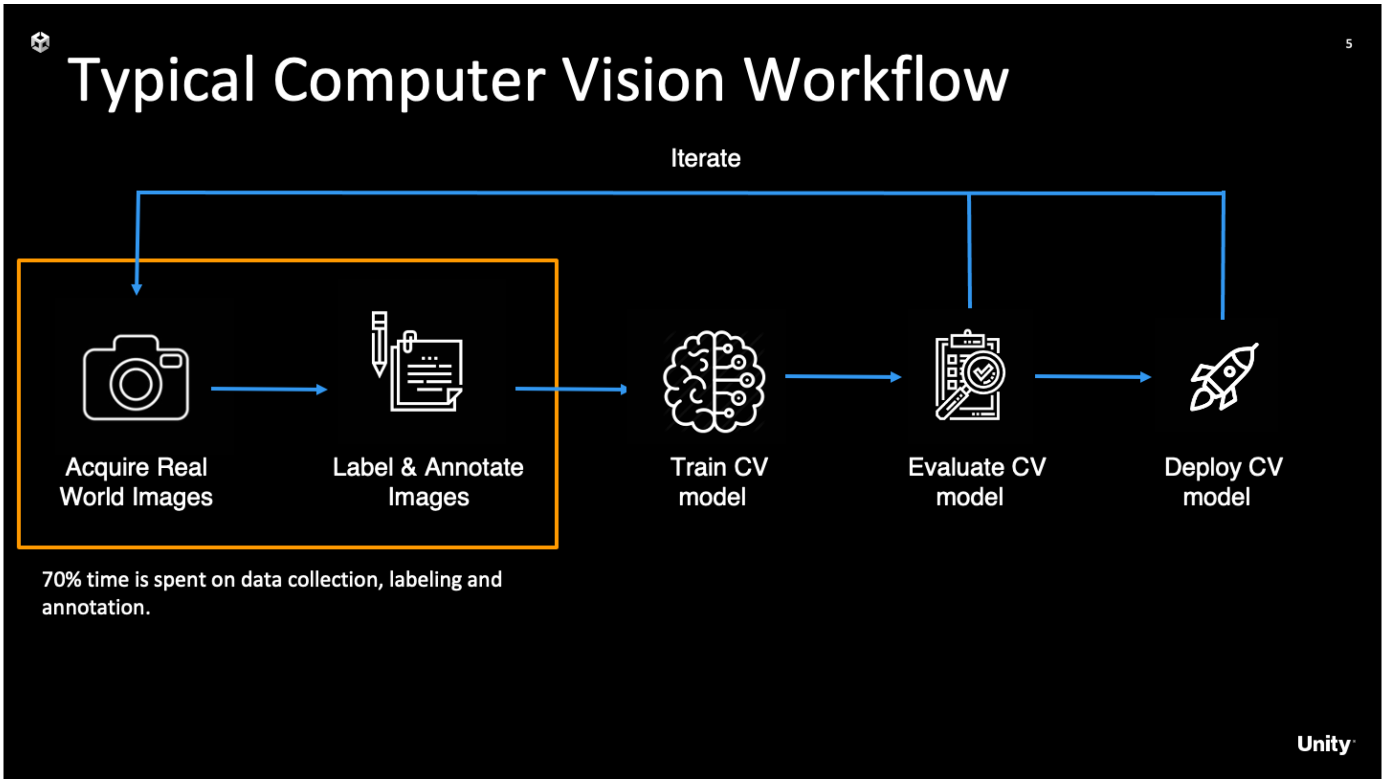
In order to build these systems we need to train them on massive amounts of labelled data. That is examples from the paired with labels on what you information want to extract from those examples.

For vision based systems the current method of getting data is to capture images from real world and then label it.

The entire process is manual and labor intensive, making it expensive and time consuming.

As a result there is not always sufficient data and the sample set that we may have can be biased.

Finally there are situations where we can get the real world data that we need due to privacy and compliance issues.



Typical Computer Vision Workflow

Iterate

Acquire Real World Images

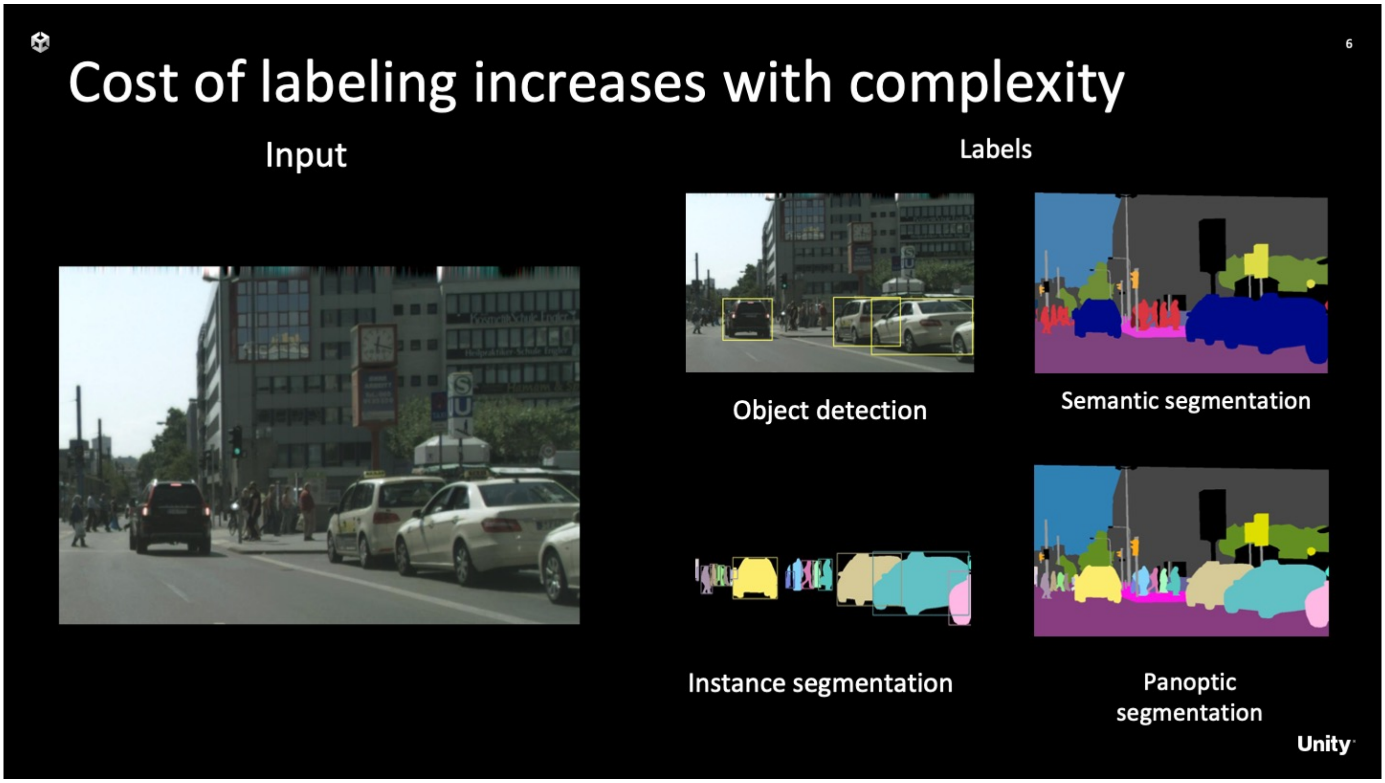
Label & Annotate Images

Train CV model

Evaluate CV model

Deploy CV model

70% time is spent on data collection, labeling and annotation.



Cost of labeling increases with complexity

Input

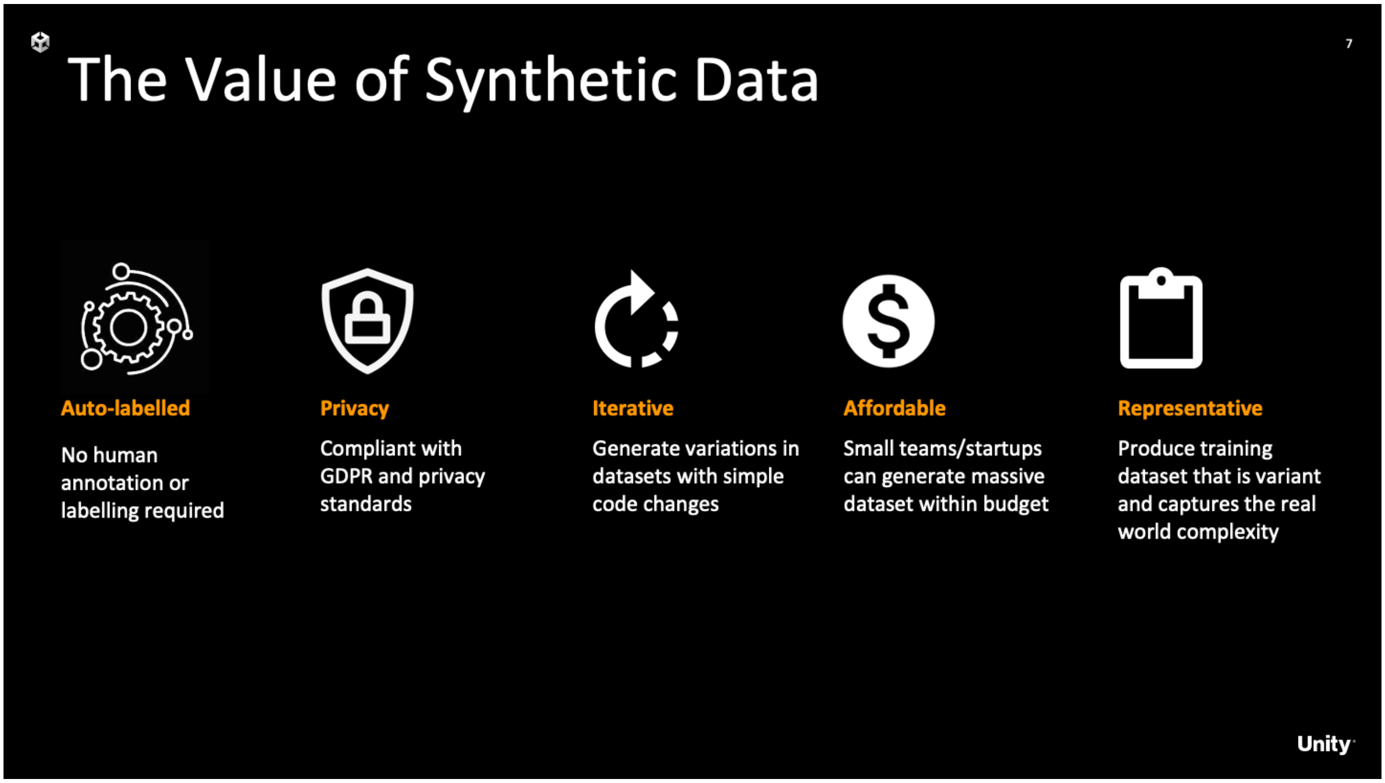
Labels

Object detection

Semantic segmentation

Instance segmentation

Panoptic segmentation



The Value of Synthetic Data

Auto-labelled

No human annotation or labelling required

Privacy

Compliant with GDPR and privacy standards

Iterative

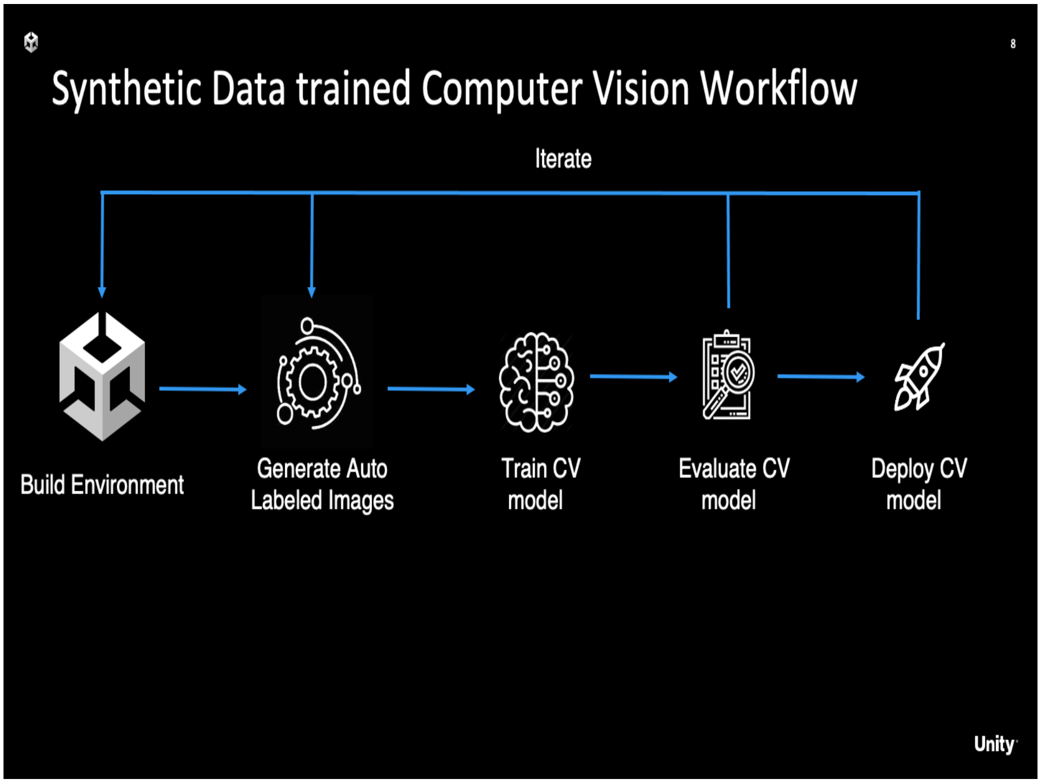
Generate variations in datasets with simple code changes

Affordable

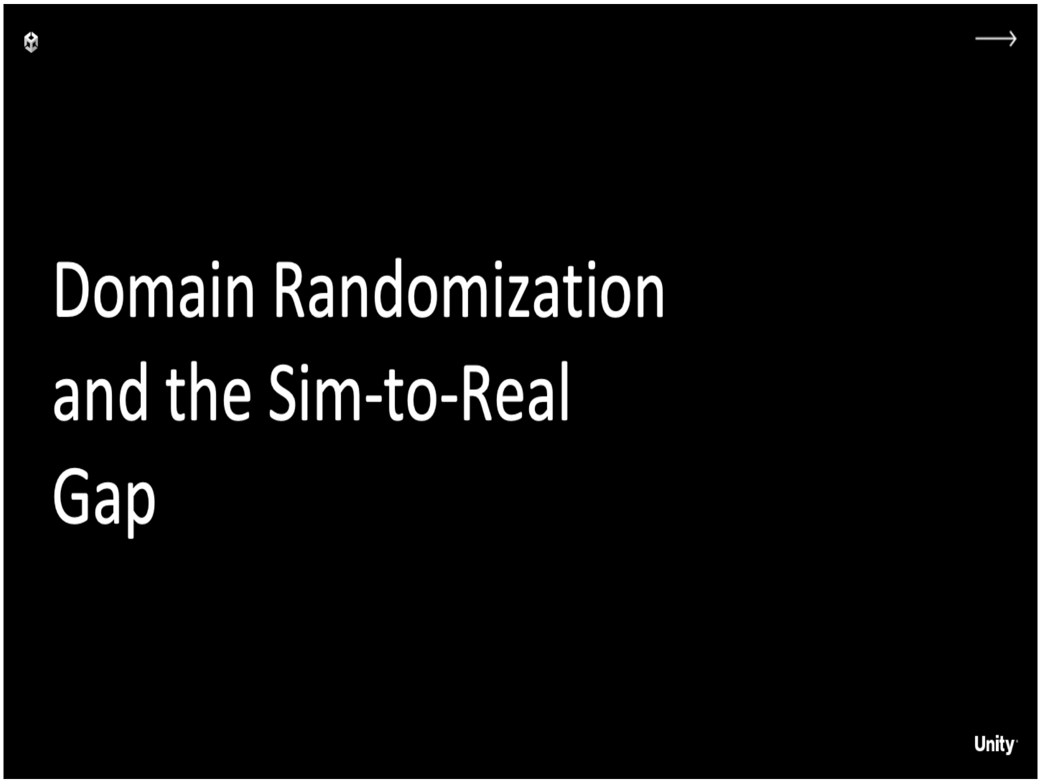
Small teams/startups can generate massive dataset within budget

Representative

Produce training dataset that is variant and captures the real world complexity



- Eliminating the annotation iteration cycles - you can always trust the annotations
- Eliminating the long wait for more data. It allows you to test many hypothesis quickly
- Eliminating edge cases you only find after deployment since you can just retrain with more data that includes those edge cases.



Domain Randomization and the Sim-to-Real Gap

Unity

- Though synthetic data has many advantages it presents one key challenge. How to ensure the model that is train in the simulated domain performs well on real world data
- Two related methods have been proposed in the literature around bridging this gap, called Domain Randomization and Meta-Learning

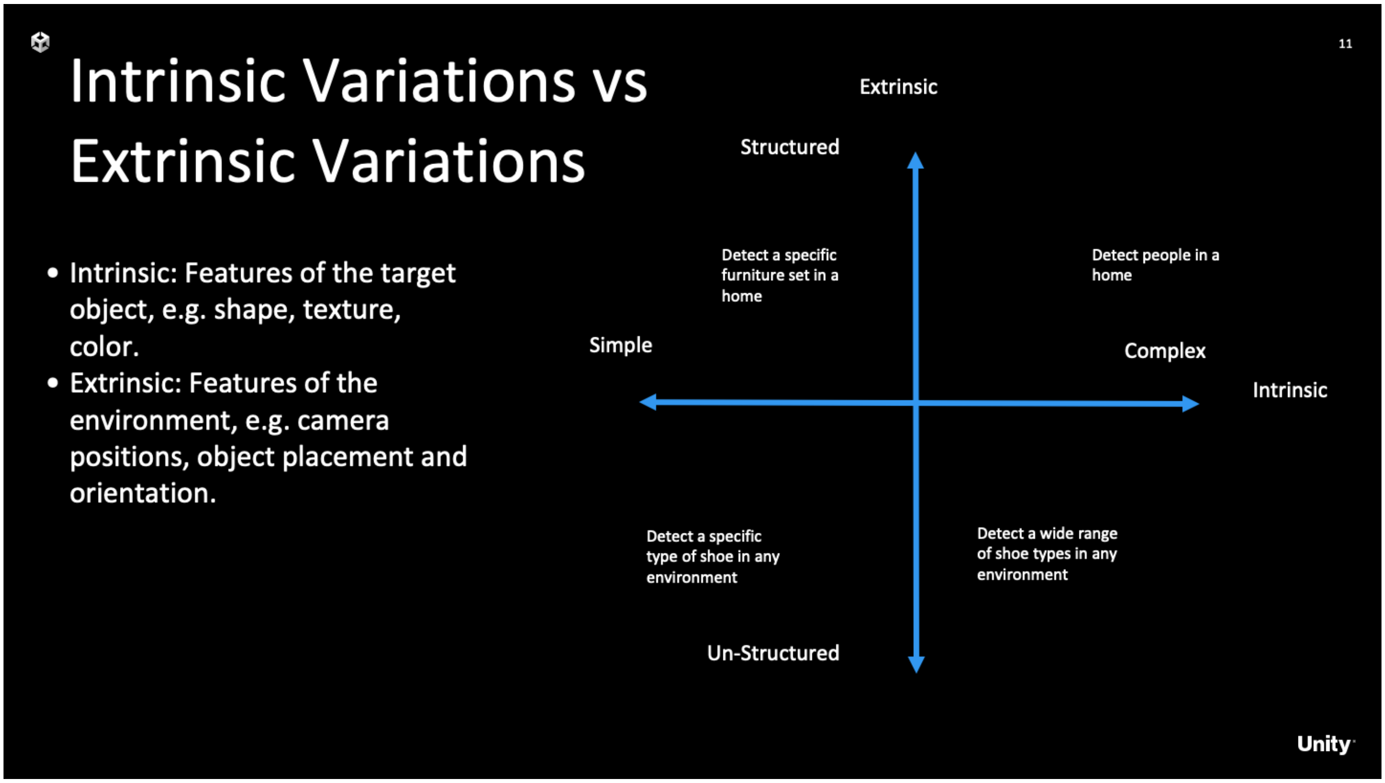
Domain Randomization

- Create the most diverse data set that the model can learn by varying properties of the simulation[1,2].
- For Example:
  - Spatial Location and Orientations
  - Color and texture of the background
  - Lighting
  - Optical Occlusions
  - Camera position, orientation, and field of view

Domain Randomized images with bounding box labels

[1] Tobin et al. "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," 2017 IROS

[2] Hinterstoisser et al. "An Annotation Saved is an Annotation Earned: Using Fully Synthetic Training for Object Instance Detection, " 2019 ICCVW



Intrinsic Variations vs Extrinsic Variations

- Intrinsic: Features of the target object, e.g. shape, texture, color.
- Extrinsic: Features of the environment, e.g. camera positions, object placement and orientation.

Extrinsic

Structured

Detect a specific furniture set in a home

Detect people in a home

Simple

Complex

Intrinsic

Detect a specific type of shoe in any environment

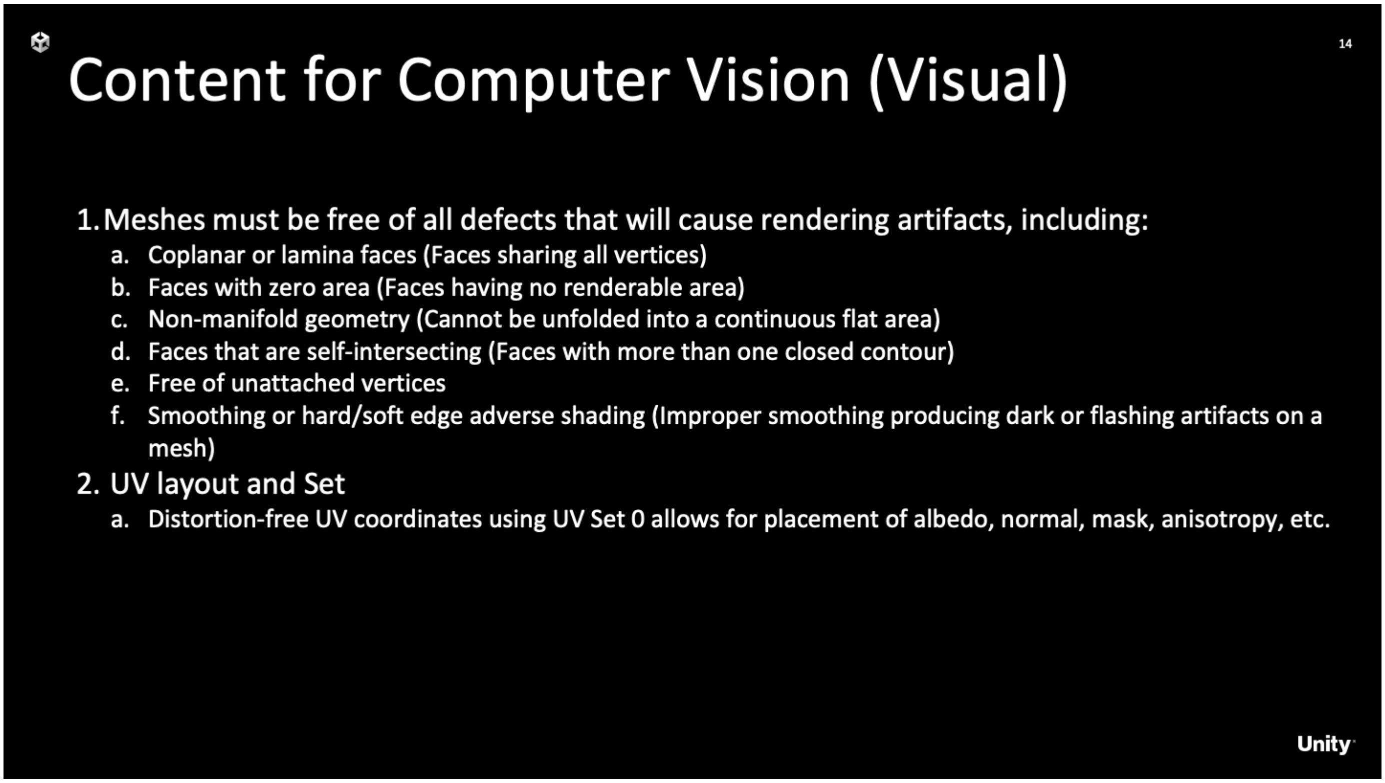
Detect a wide range of shoe types in any environment

Un-Structured

Burdens of Domain Randomization

Levels of Content for Machine Learning

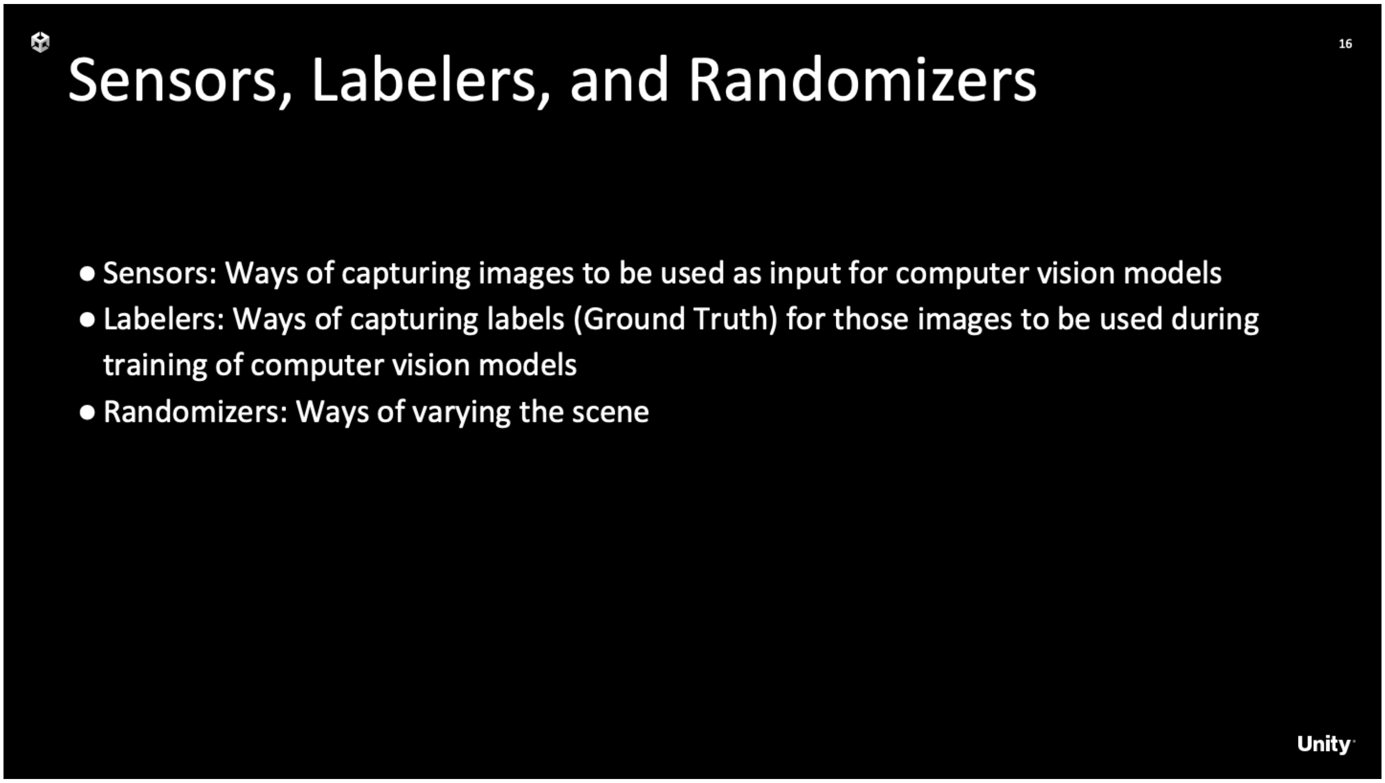
- Visual
  - Renders correctly under a wide range of conditions, e.g., assets should be free of smoothing or hard/soft edge adverse shading (Improper smoothing producing dark or flashing artifacts on a mesh)
  - Appearance can be varied, e.g., textures and materials can be changed programmatically
- Physical
  - Assets should have accurate colliders
  - Mass and density can be varied
  - Friction, etc. can be varied
- Kinematic
  - Objects can be rigged and animated
  - Objects in the same class can share rigs and animations
  - Animations can be varied
- Fully embodied content
  - Content reacts to the action of agents



Content for Computer Vision (Visual)

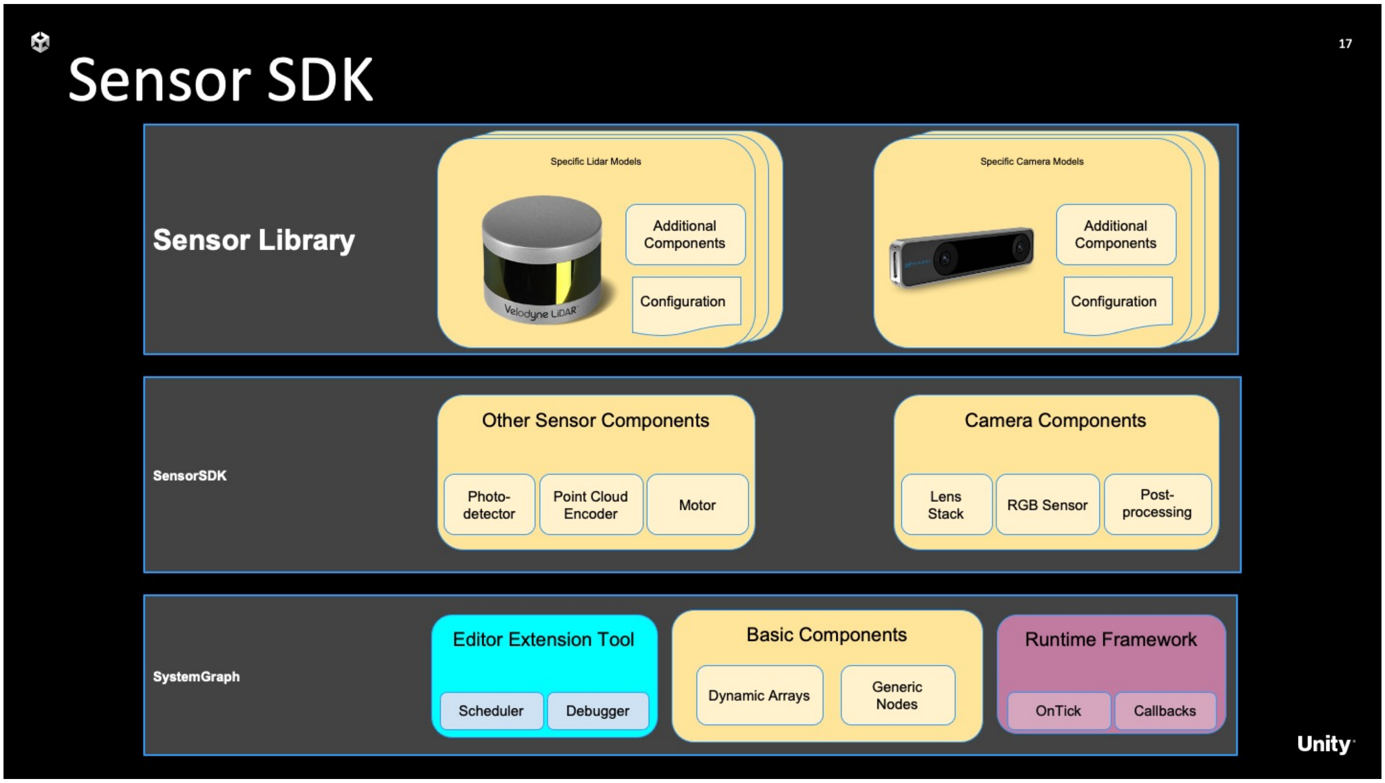
1. Meshes must be free of all defects that will cause rendering artifacts, including:
   a. Coplanar or lamina faces (Faces sharing all vertices)
   b. Faces with zero area (Faces having no renderable area)
   c. Non-manifold geometry (Cannot be unfolded into a continuous flat area)
   d. Faces that are self-intersecting (Faces with more than one closed contour)
   e. Free of unattached vertices
   f. Smoothing or hard/soft edge adverse shading (Improper smoothing producing dark or flashing artifacts on a mesh)
2. UV layout and Set
   a. Distortion-free UV coordinates using UV Set 0 allows for placement of albedo, normal, mask, anisotropy, etc.

Tools for Synthetic Data Generation



Sensors, Labelers, and Randomizers

- Sensors: Ways of capturing images to be used as input for computer vision models
- Labelers: Ways of capturing labels (Ground Truth) for those images to be used during training of computer vision models
- Randomizers: Ways of varying the scene



Sensor SDK

Sensor Library

Specific Lidar Models

Additional Components

Configuration

Specific Camera Models

Additional Components

Configuration

SensorSDK

Other Sensor Components

Photo-detector

Point Cloud Encoder

Motor

Camera Components

Lens Stack

RGB Sensor

Post-processing

SystemGraph

Editor Extension Tool

Scheduler

Debugger

Basic Components

Dynamic Arrays

Generic Nodes

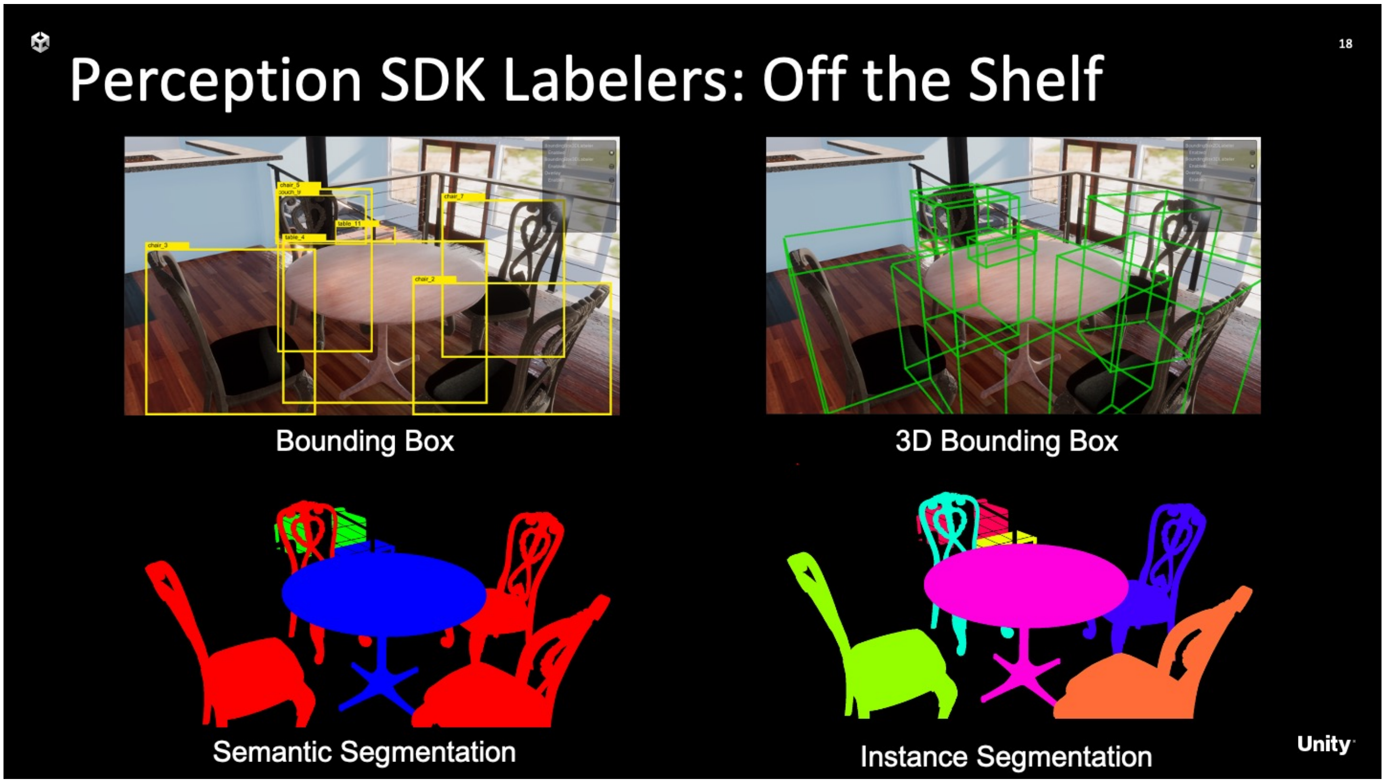
Runtime Framework

OnTick

Callbacks

Unity

Perception SDK Labelers: Off the Shelf



Bounding Box

3D Bounding Box

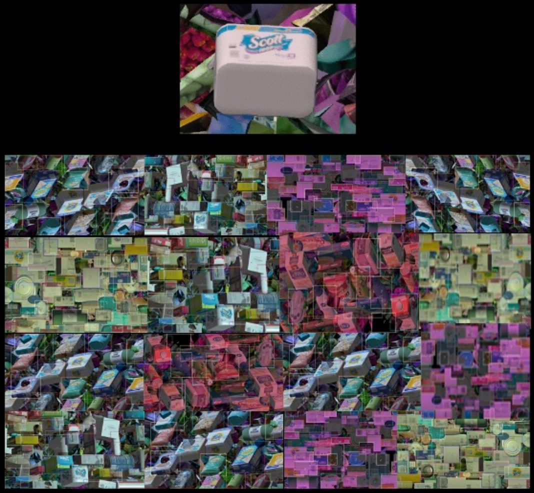
Semantic Segmentation

Instance Segmentation

Unity

Perception SDK: Extrinsic Randomizations

Unstructured / Semi-Structured

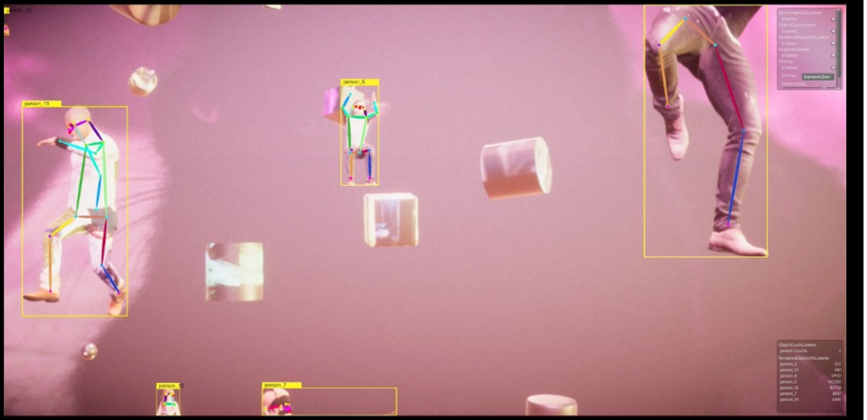


Structured

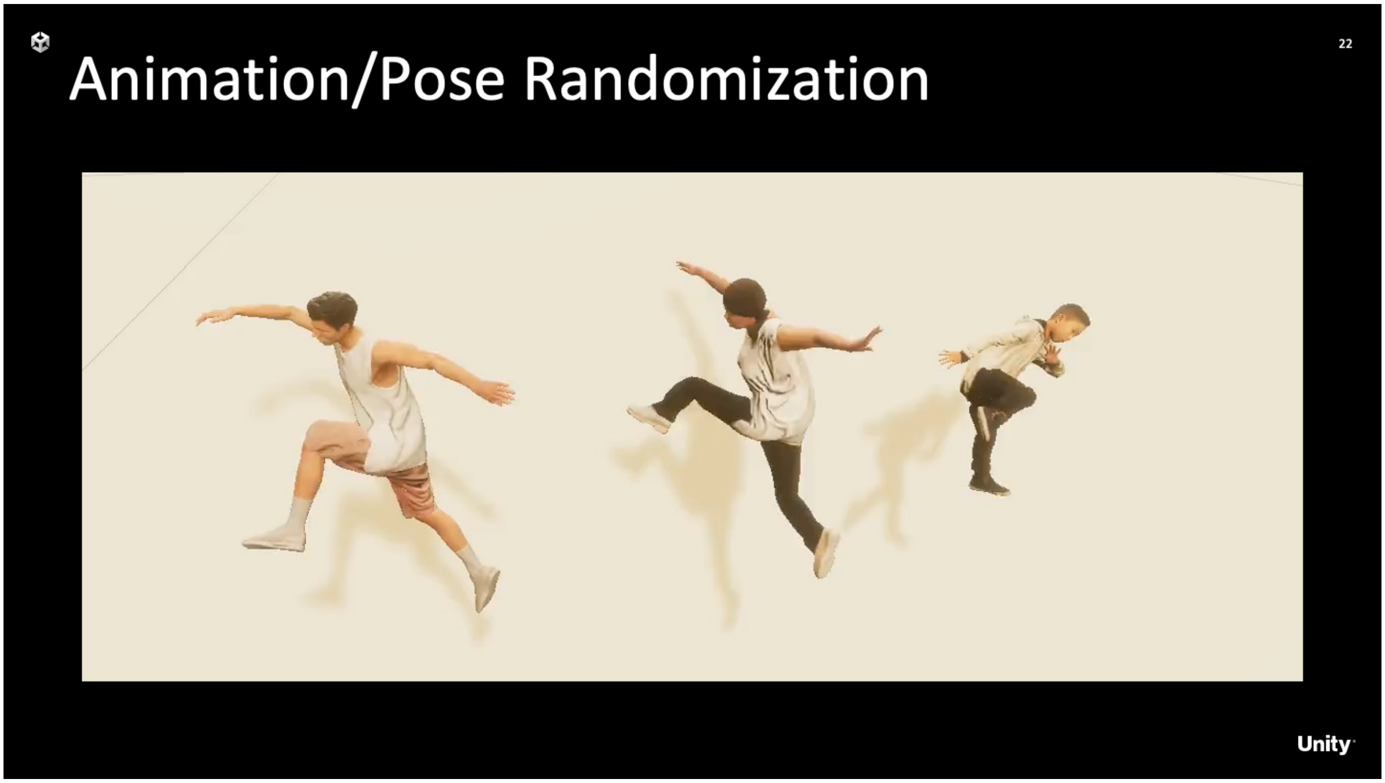
Benchmark Environment of Human Centric Computer Vision

PeopleSansPeople

- 28 Human Assets
- 39 diverse Animations sequences
- 21,952 clothing textures
- Parameterized Placement randomizer
- Parameterized Lighting and Camera System
- Occluders/Distractor objects
- RGB image capture with High Definition Render Pipeline
- Labelers:
  - Bounding Box
  - Semantic Segmentation
  - Instance Segmentation
  - Pose Labeler
    - COCO keypoints
- Packaged macOS and Linux binaries
- CLI + configs to update all parameters



Unity



Animation/Pose Randomization

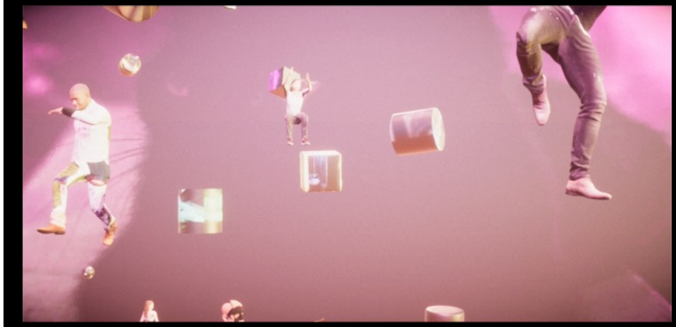
Clothing Texture (Shader-Graph) Randomizer

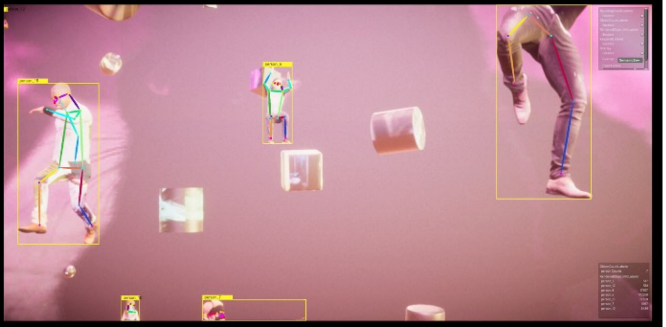
Shader Graph randomizer
https://youtu.be/qwfZ9gh_BUc

Wall randomizer
https://youtu.be/kXs_vpquJCg

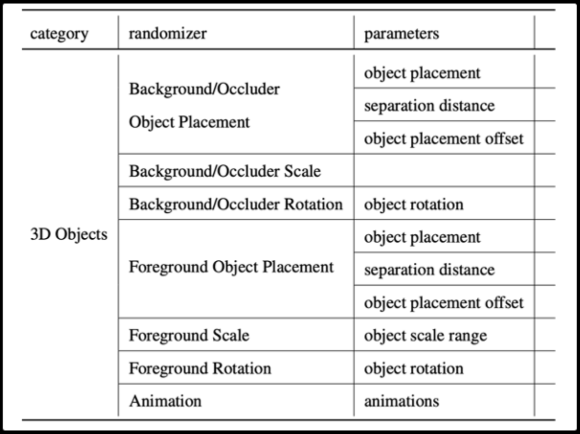
Full playlist: https://youtube.com/playlist?list=PL-0JKmA4rKK59eQ8XPtsh2YAzrYX-HgLp

PeopleSansPeople

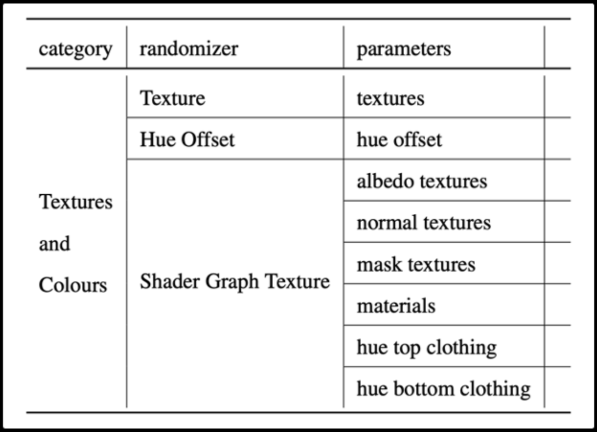




PeopleSansPeople - Exposed Parameters, Objects



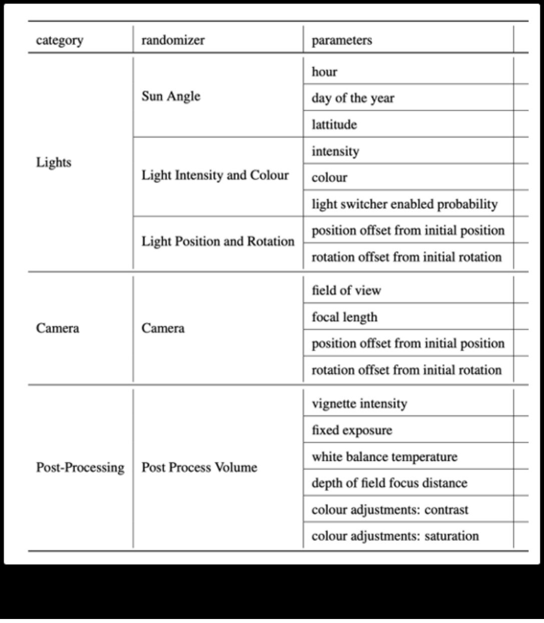
category	randomizer	parameters
3D Objects	Background/Occluder Object Placement	object placement
		separation distance
		object placement offset
	Background/Occluder Scale	
	Background/Occluder Rotation	object rotation
	Foreground Object Placement	object placement
		separation distance
		object placement offset
	Foreground Scale	object scale range
	Foreground Rotation	object rotation
	Animation	animations



category	randomizer	parameters
Textures and Colours	Texture	textures
	Hue Offset	hue offset
	Shader Graph Texture	albedo textures
		normal textures
		mask textures
		materials
		hue top clothing
		hue bottom clothing

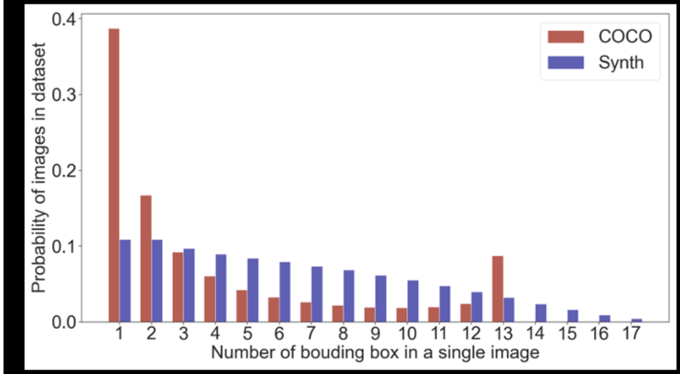
Unity

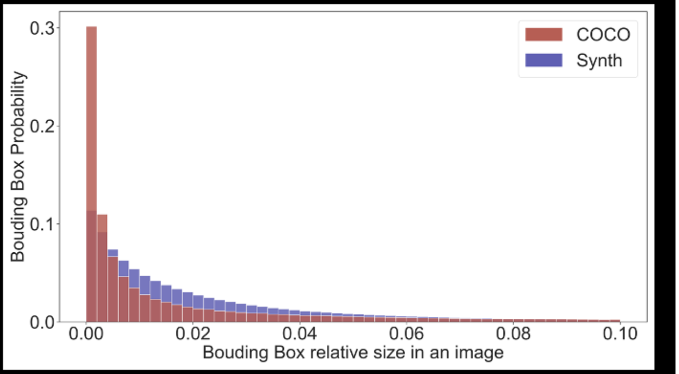
PeopleSansPeople - Exposed Parameters, Rendering

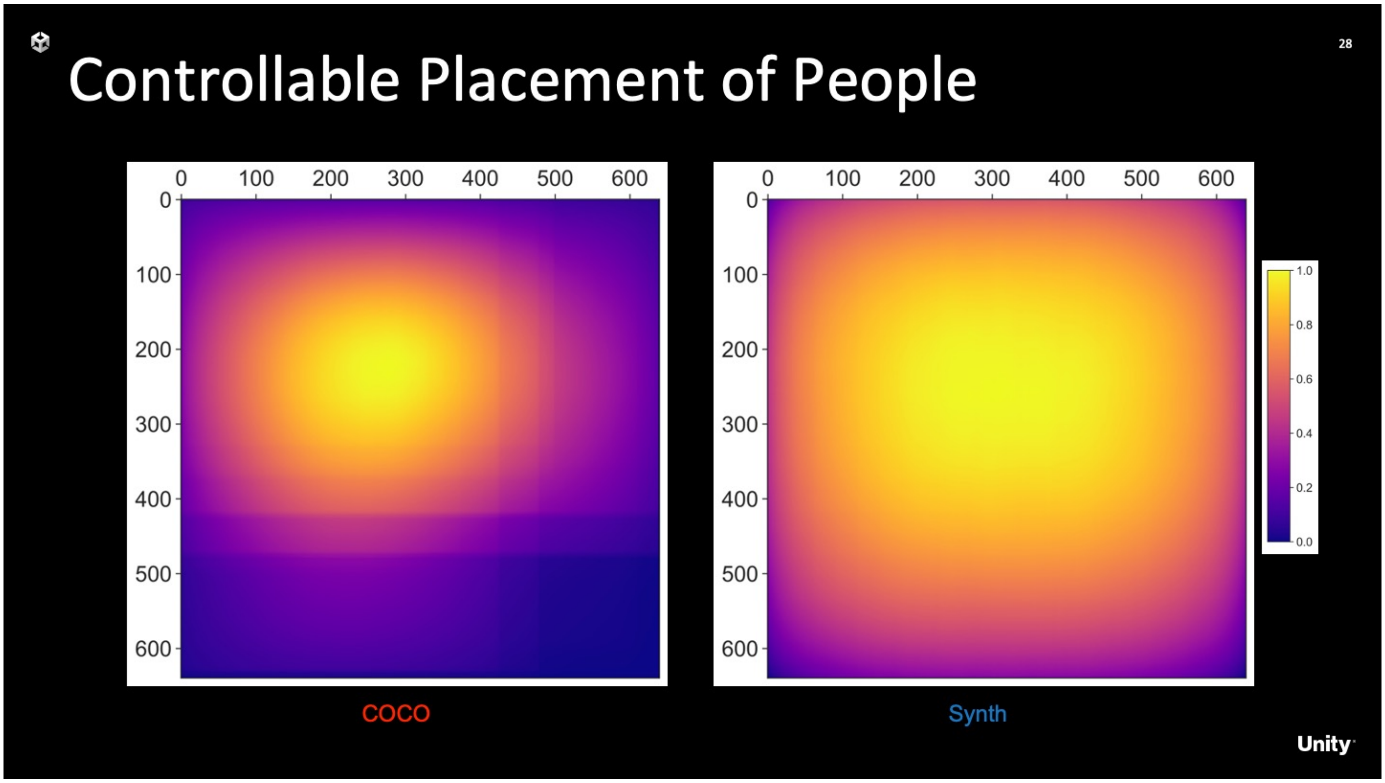


category	randomizer	parameters
Lights	Sun Angle	hour
		day of the year
		lattitude
	Light Intensity and Colour	intensity
		colour
		light switcher enabled probability
	Light Position and Rotation	position offset from initial position
		rotation offset from initial rotation
Camera	Camera	field of view
		focal length
		position offset from initial position
		rotation offset from initial rotation
Post-Processing	Post Process Volume	vignette intensity
		fixed exposure
		white balance temperature
		depth of field focus distance
		colour adjustments: contrast
		colour adjustments: saturation

Controllable Number and Size of People



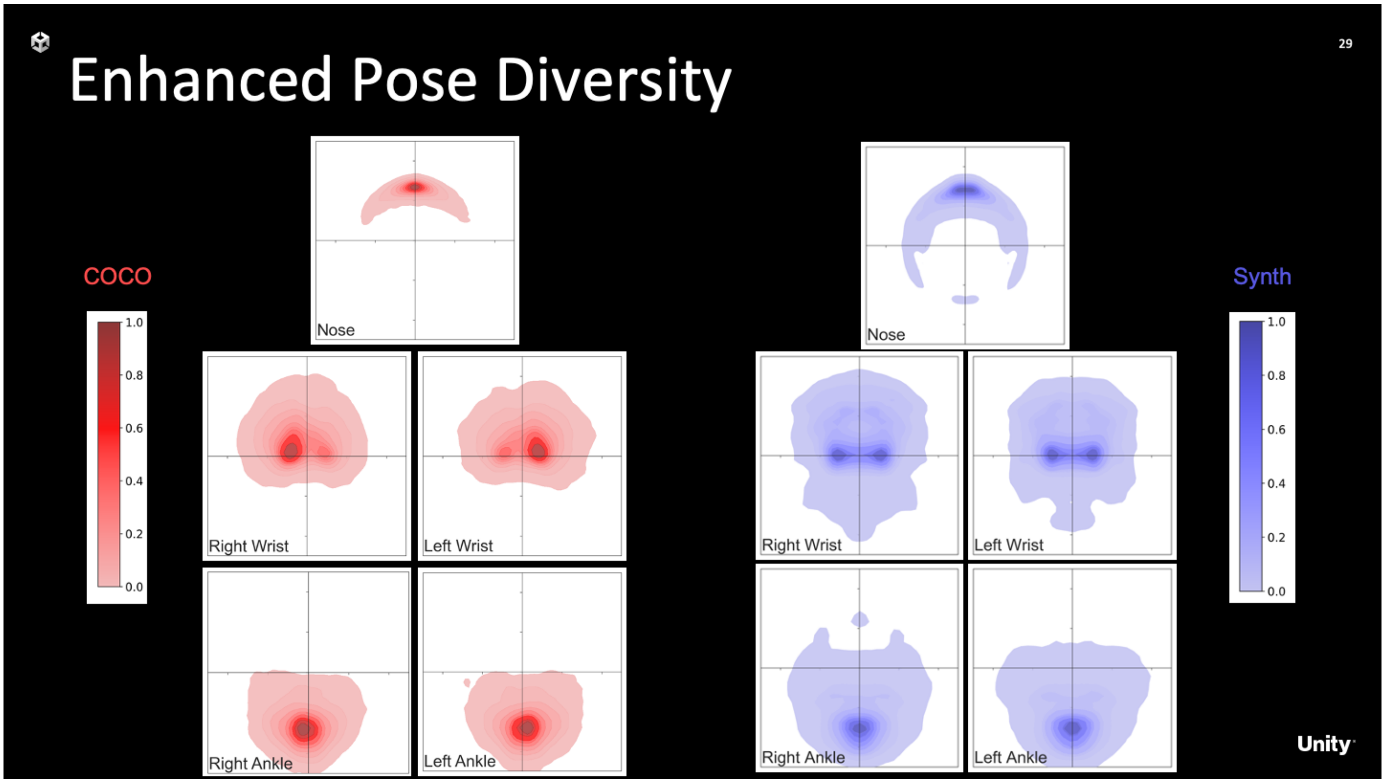




Controllable Placement of People

COCO

Synth

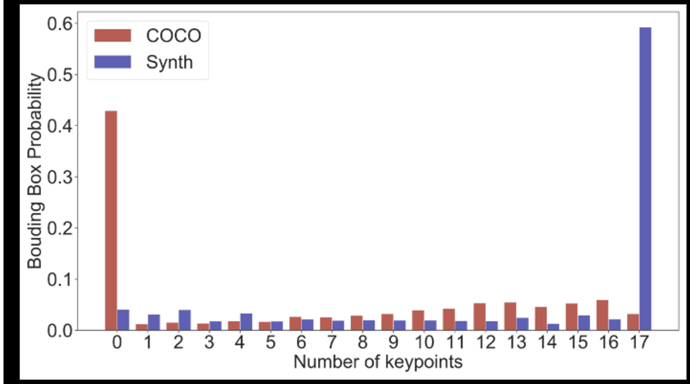


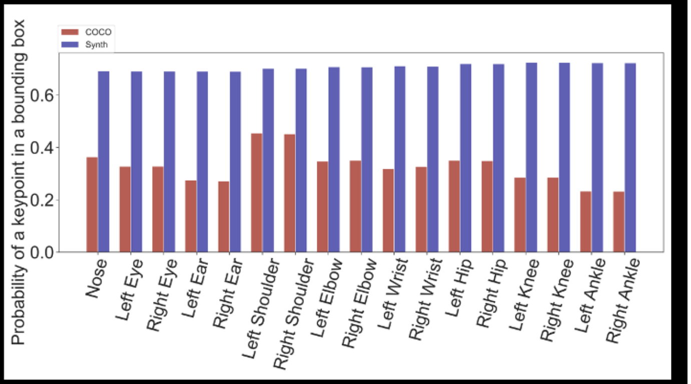
29

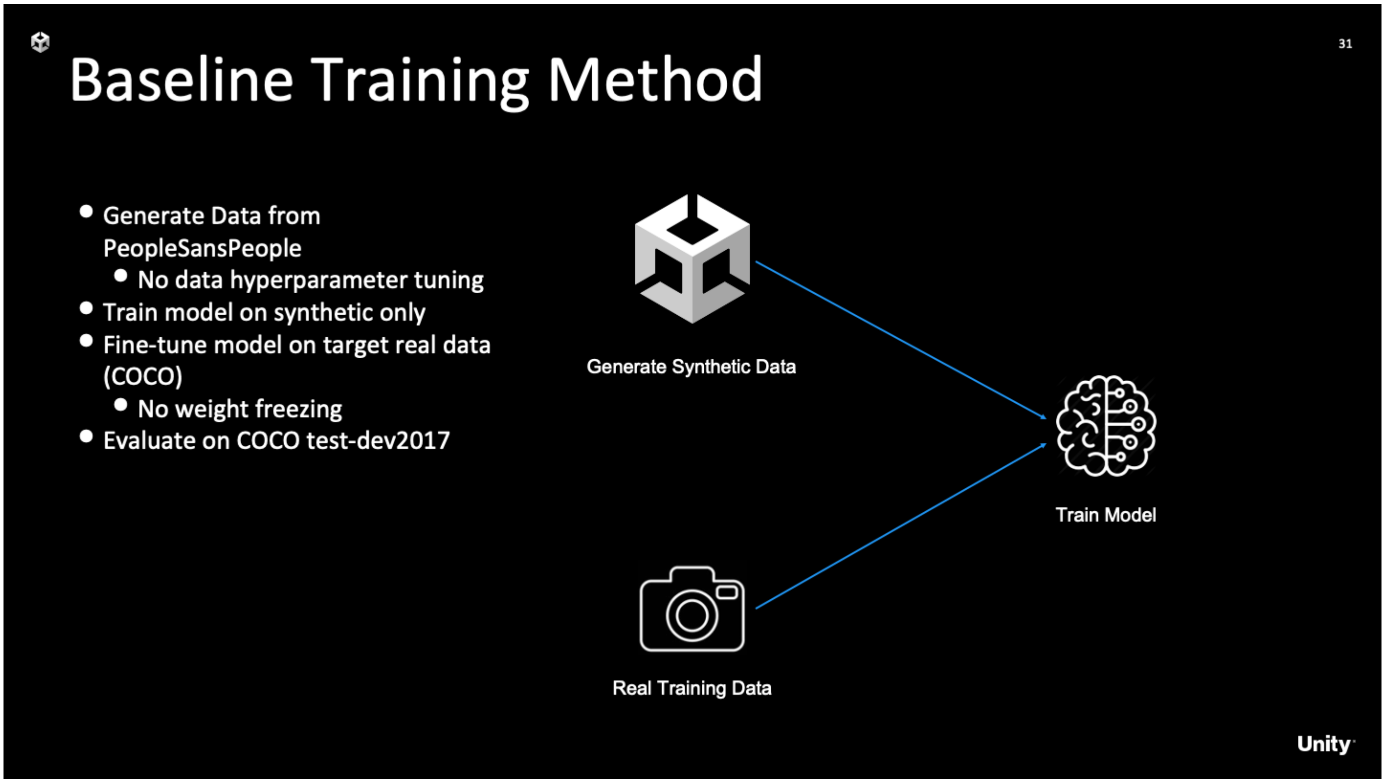
Enhanced Pose Diversity

Unity

Improved Label Consistency







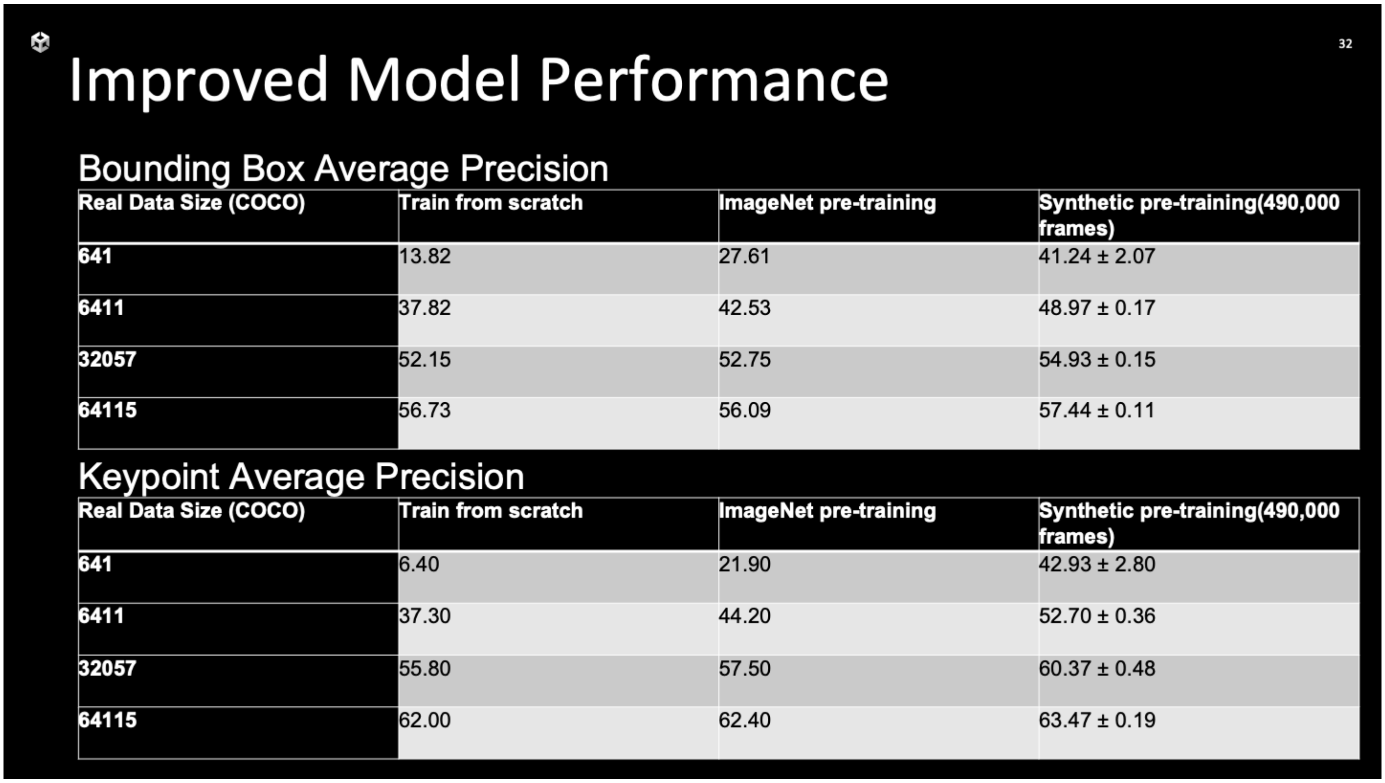
Baseline Training Method

- Generate Data from PeopleSansPeople
  - No data hyperparameter tuning
- Train model on synthetic only
- Fine-tune model on target real data (COCO)
  - No weight freezing
- Evaluate on COCO test-dev2017

Generate Synthetic Data

Train Model

Real Training Data



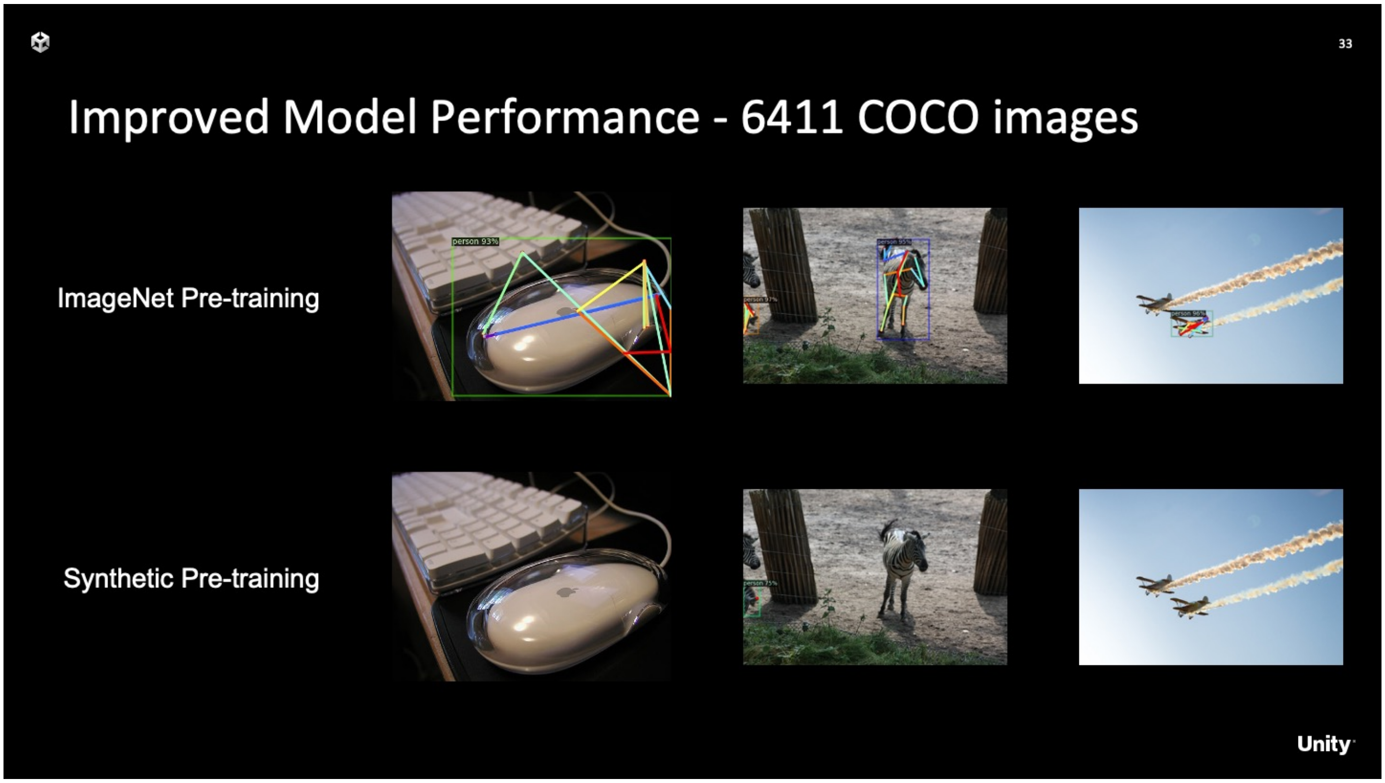
Improved Model Performance

Bounding Box Average Precision

Real Data Size (COCO)	Train from scratch	ImageNet pre-training	Synthetic pre-training(490,000 frames)
641	13.82	27.61	41.24 ± 2.07
6411	37.82	42.53	48.97 ± 0.17
32057	52.15	52.75	54.93 ± 0.15
64115	56.73	56.09	57.44 ± 0.11

Keypoint Average Precision

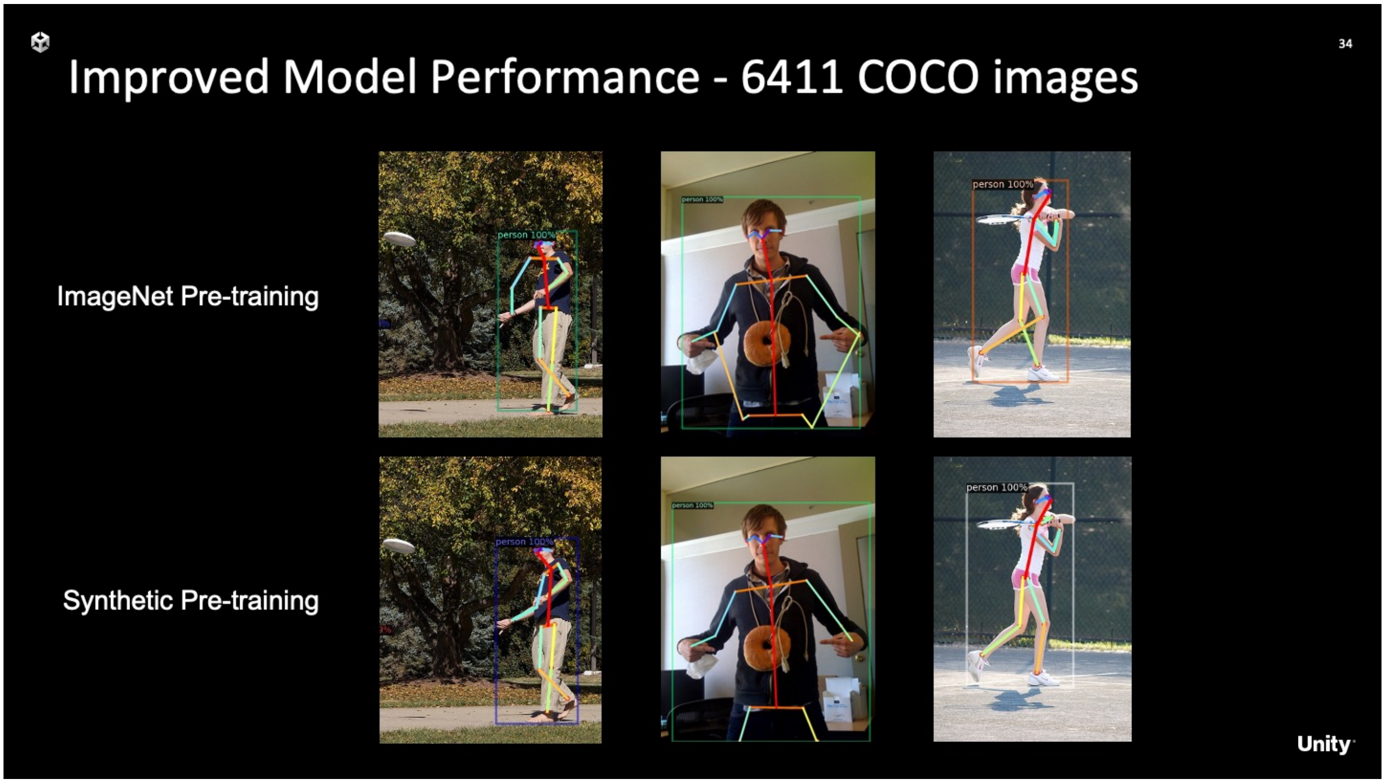
Real Data Size (COCO)	Train from scratch	ImageNet pre-training	Synthetic pre-training(490,000 frames)
641	6.40	21.90	42.93 ± 2.80
6411	37.30	44.20	52.70 ± 0.36
32057	55.80	57.50	60.37 ± 0.48
64115	62.00	62.40	63.47 ± 0.19



Improved Model Performance - 6411 COCO images

ImageNet Pre-training

Synthetic Pre-training



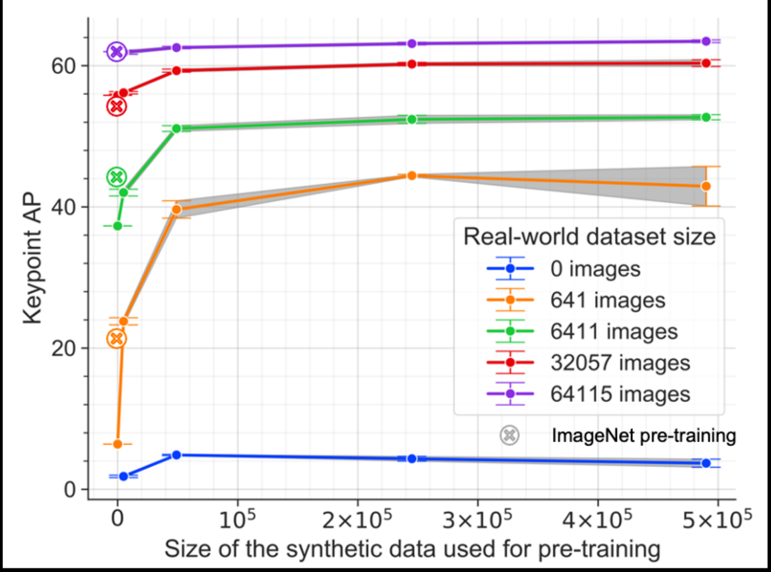
Improved Model Performance - 6411 COCO images

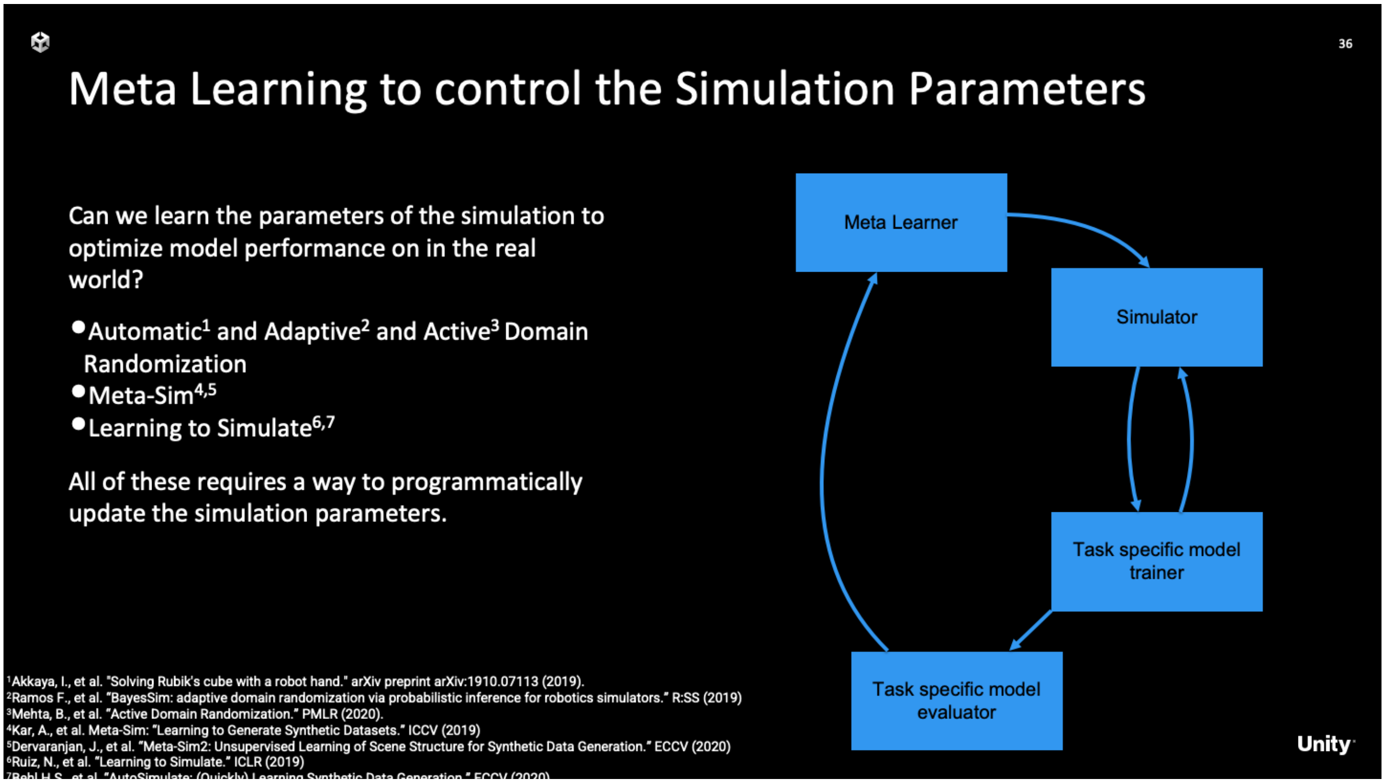
ImageNet Pre-training

Synthetic Pre-training

Improved Model Performance

- Pre-train Detectron2 (KeyPoint-RCNN) on synthetic data
- Fine-tuning performance improves with size of synthetic data
- Poor Zero-Shot performance with wildly randomized data





Meta Learning to control the Simulation Parameters

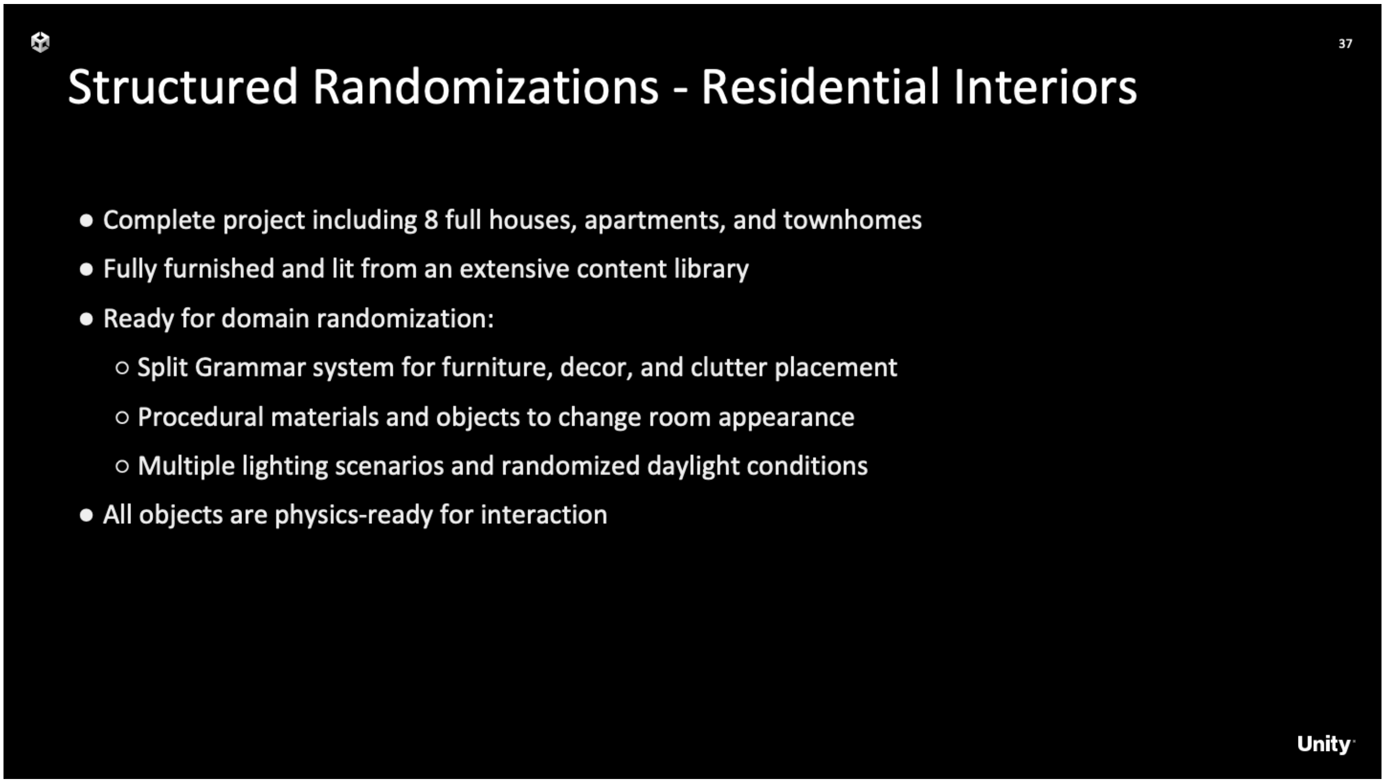
Can we learn the parameters of the simulation to optimize model performance on in the real world?

- Automatic[1] and Adaptive[2] and Active[3] Domain Randomization
- Meta-Sim[4,5]
- Learning to Simulate[6,7]

All of these requires a way to programmatically update the simulation parameters.

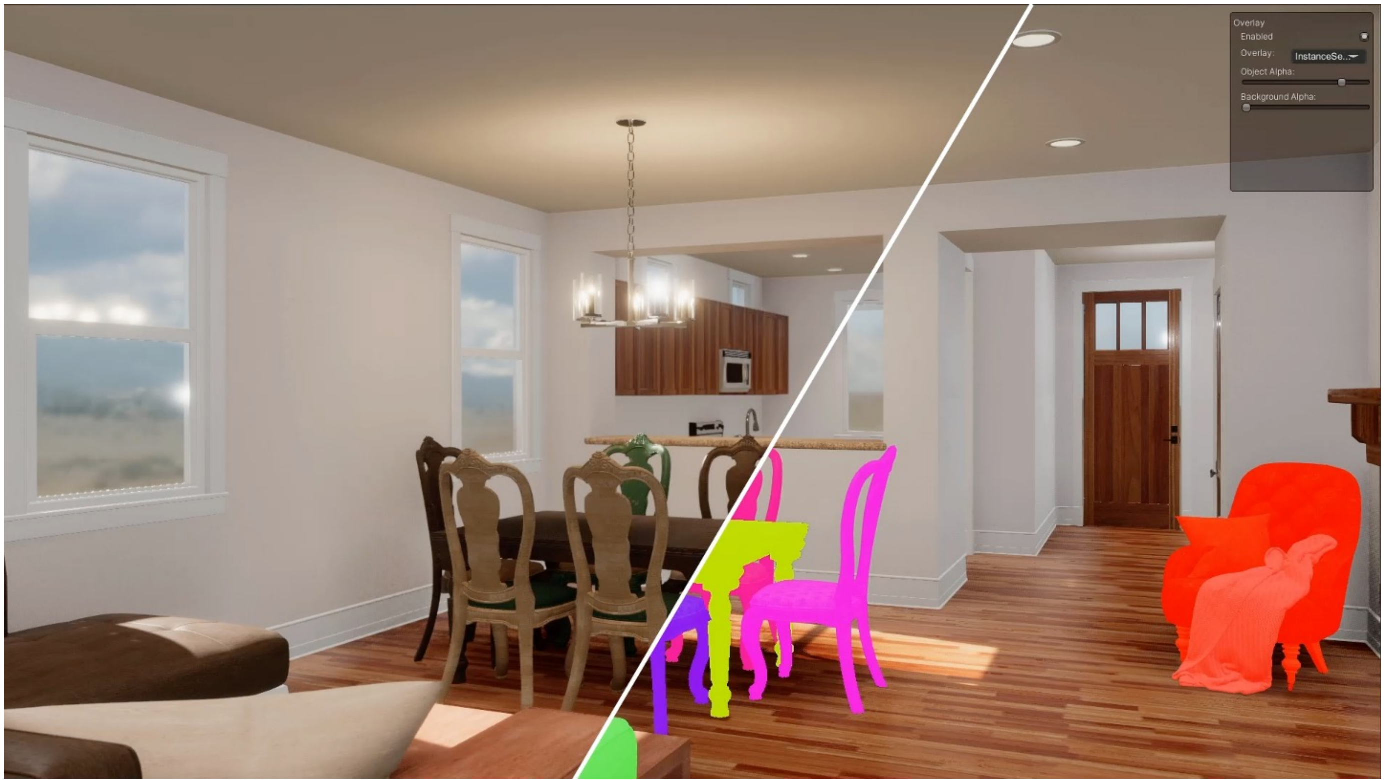
Meta Learner
Simulator
Task specific model trainer
Task specific model evaluator

[1] Akkaya, I., et al. "Solving Rubik's cube with a robot hand." arXiv preprint arXiv:1910.07113 (2019).
[2] Ramos F., et al. "BayesSim: adaptive domain randomization via probabilistic inference for robotics simulators." R:SS (2019)
[3] Mehta, B., et al. "Active Domain Randomization." PMLR (2020).
[4] Kar, A., et al. Meta-Sim: "Learning to Generate Synthetic Datasets." ICCV (2019)
[5] Dervaranjan, J., et al. "Meta-Sim2: Unsupervised Learning of Scene Structure for Synthetic Data Generation." ECCV (2020)
[6] Ruiz, N., et al. "Learning to Simulate." ICLR (2019)
[7] Behl H.S., et al. "AutoSimulate: (Quickly) Learning Synthetic Data Generation." ECCV (2020)



Structured Randomizations - Residential Interiors

- Complete project including 8 full houses, apartments, and townhomes
- Fully furnished and lit from an extensive content library
- Ready for domain randomization:
  - Split Grammar system for furniture, decor, and clutter placement
  - Procedural materials and objects to change room appearance
  - Multiple lighting scenarios and randomized daylight conditions
- All objects are physics-ready for interaction



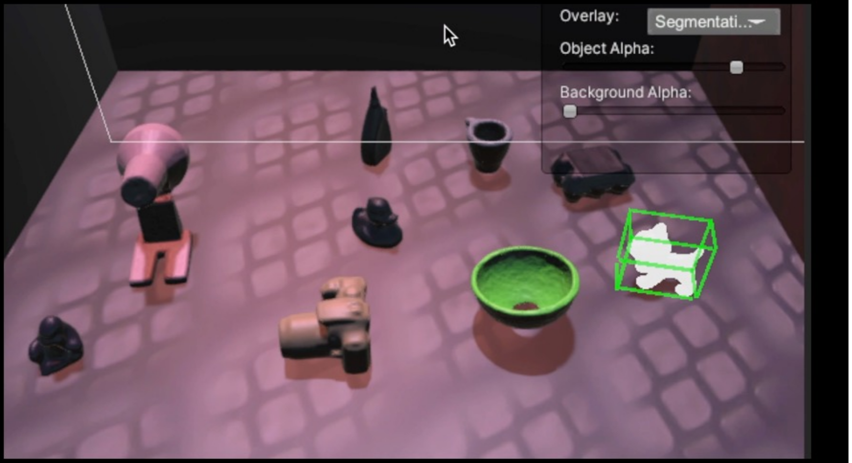
Overlay

Enabled ●

Overlay: InstanceSe...

3D object pose estimation

- RGB-D image capture
- Labeling
  - 3D bounding box
  - Semantic Segmentation
- Camera Intrinsic and Extrinsic Parameters
- LineMod Assets
- Distractor Objects
- Randomizers
  - Camera
  - Object placement
  - Lighting
  - Background Textures

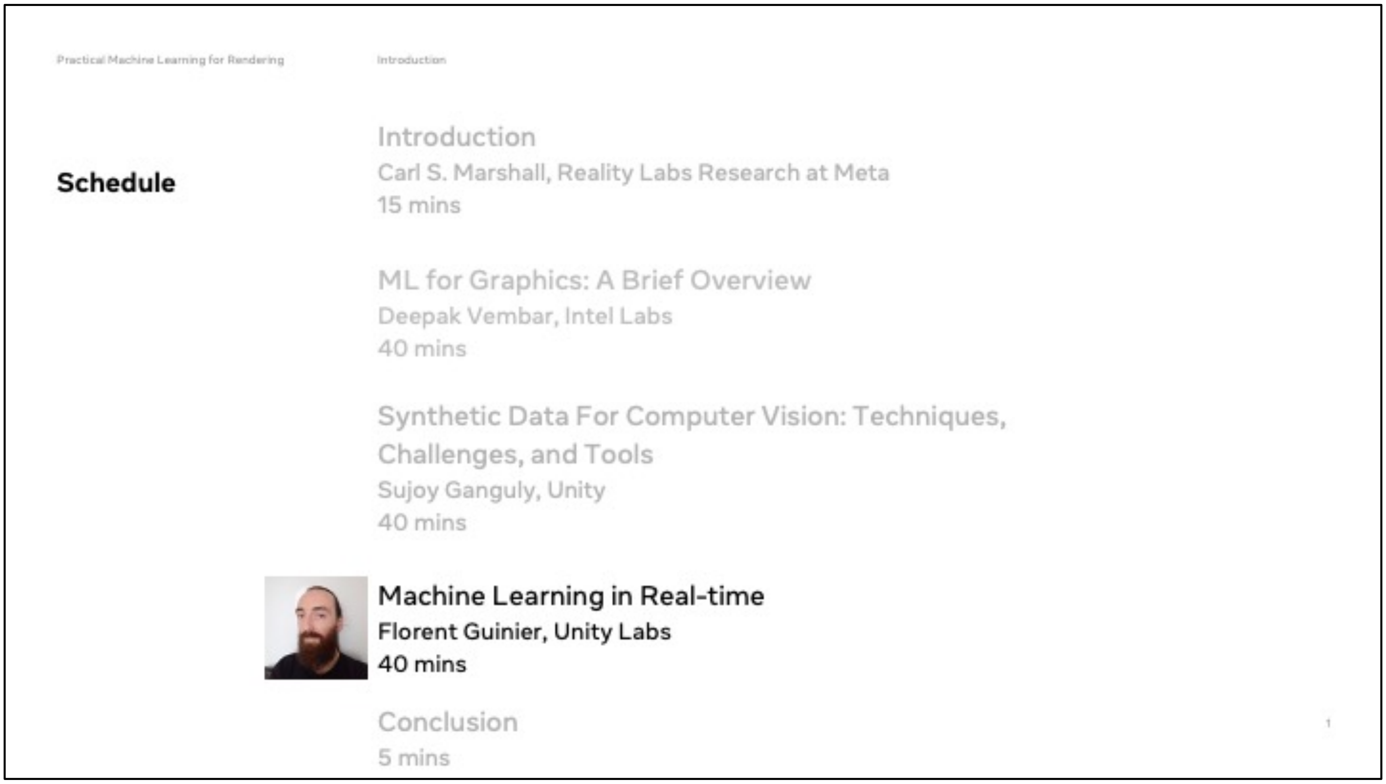


Conclusions

- Synthetic data can be the future of model training, but it is hard to make and use.
- PeopleSansPeople: a free to use synthetic data generator for human-centric computer vision research.
- 3D Object Pose Estimation environment available early next year.
- Synthetic data pre-training out performs real data pre-training.
- Can we learn optimal parameters of synthetic data generators?

Thank you

UNITY.COM



Schedule

Introduction
Carl S. Marshall, Reality Labs Research at Meta
15 mins

ML for Graphics: A Brief Overview
Deepak Vembar, Intel Labs
40 mins

Synthetic Data For Computer Vision: Techniques, Challenges, and Tools
Sujoy Ganguly, Unity
40 mins

Machine Learning in Real-time
Florent Guinier, Unity Labs
40 mins

Conclusion
5 mins

Unity Labs // Intel Labs

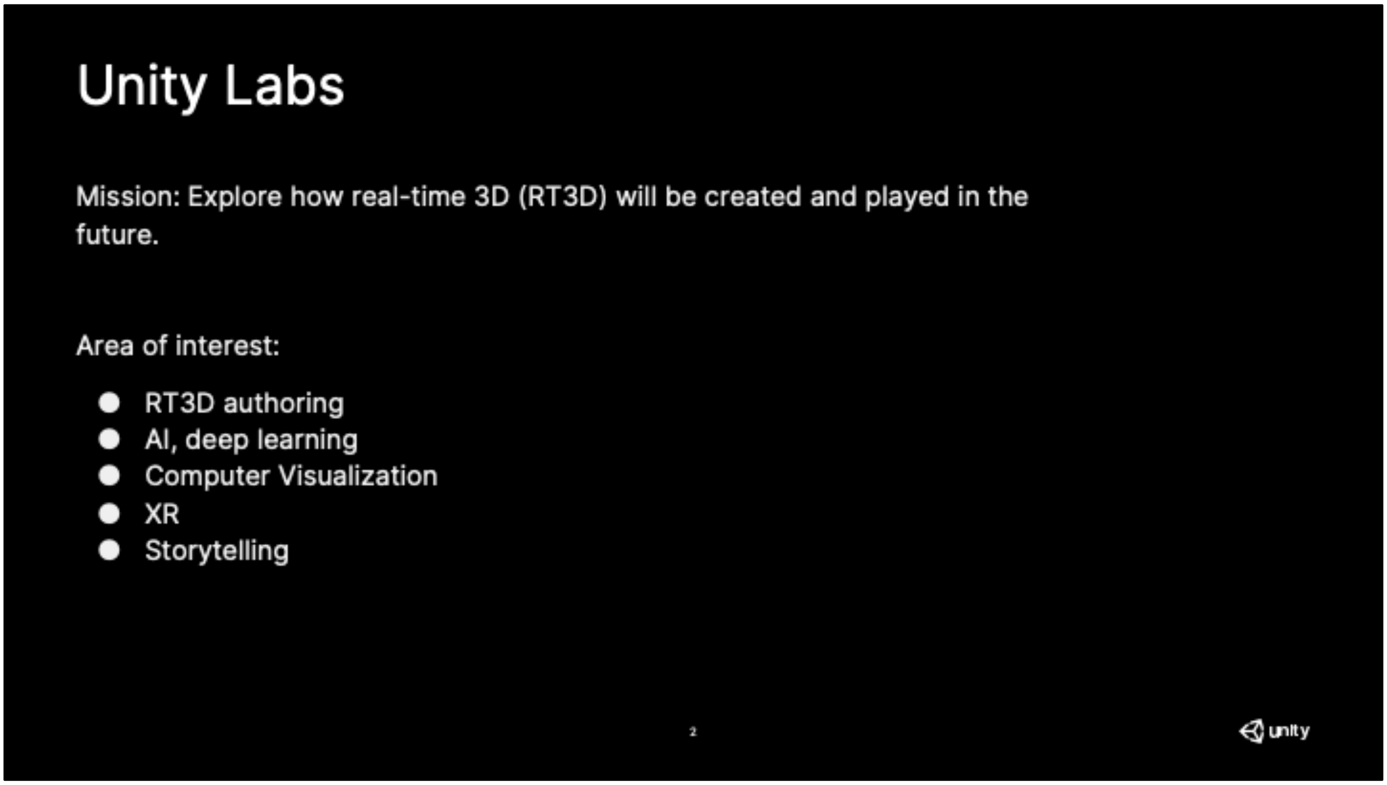
Machine Learning in Real-time

Unity Labs - Barracuda team

Florent Guinier

Generative Art — Made with Unity

Subject: Machine Learning in real-time in the **context of a real-time 3D engine**.



Unity Labs

Mission: Explore how real-time 3D (RT3D) will be created and played in the future.

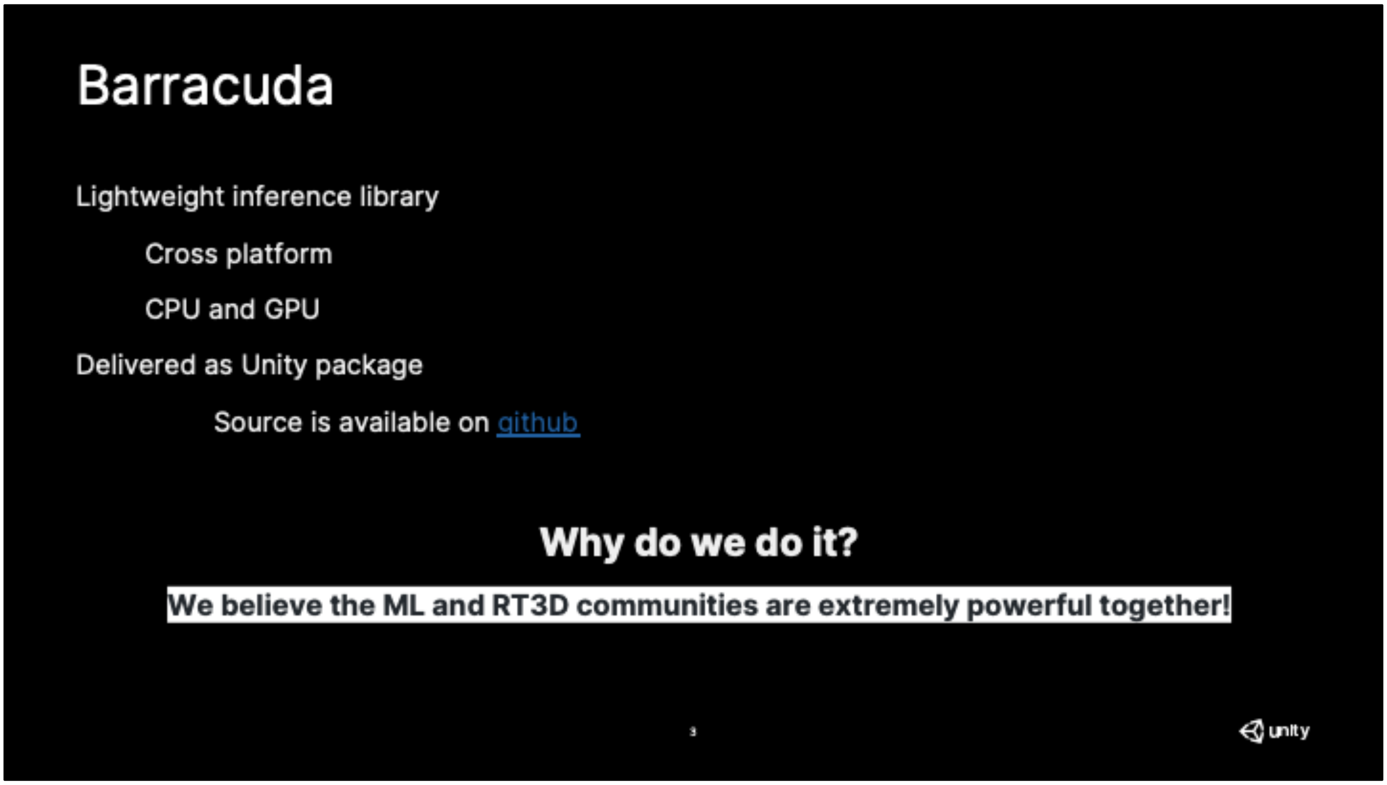
Area of interest:

- RT3D authoring
- AI, deep learning
- Computer Visualization
- XR
- Storytelling

https://unity.com/labs

Publications: https://unity.com/publications

Unity Labs is a part of Unity which mission is to explore how:
Realtime 3D scene authoring,
AI, deep learning,
computer visualization,
VR, AR
and storytelling
will evolve in the next decade to radically transform how realtime 3D applications will be created and experienced.



Barracuda

Lightweight inference library

- Cross platform
- CPU and GPU

Delivered as Unity package

- Source is available on github

Why do we do it?

We believe the ML and RT3D communities are extremely powerful together!

3

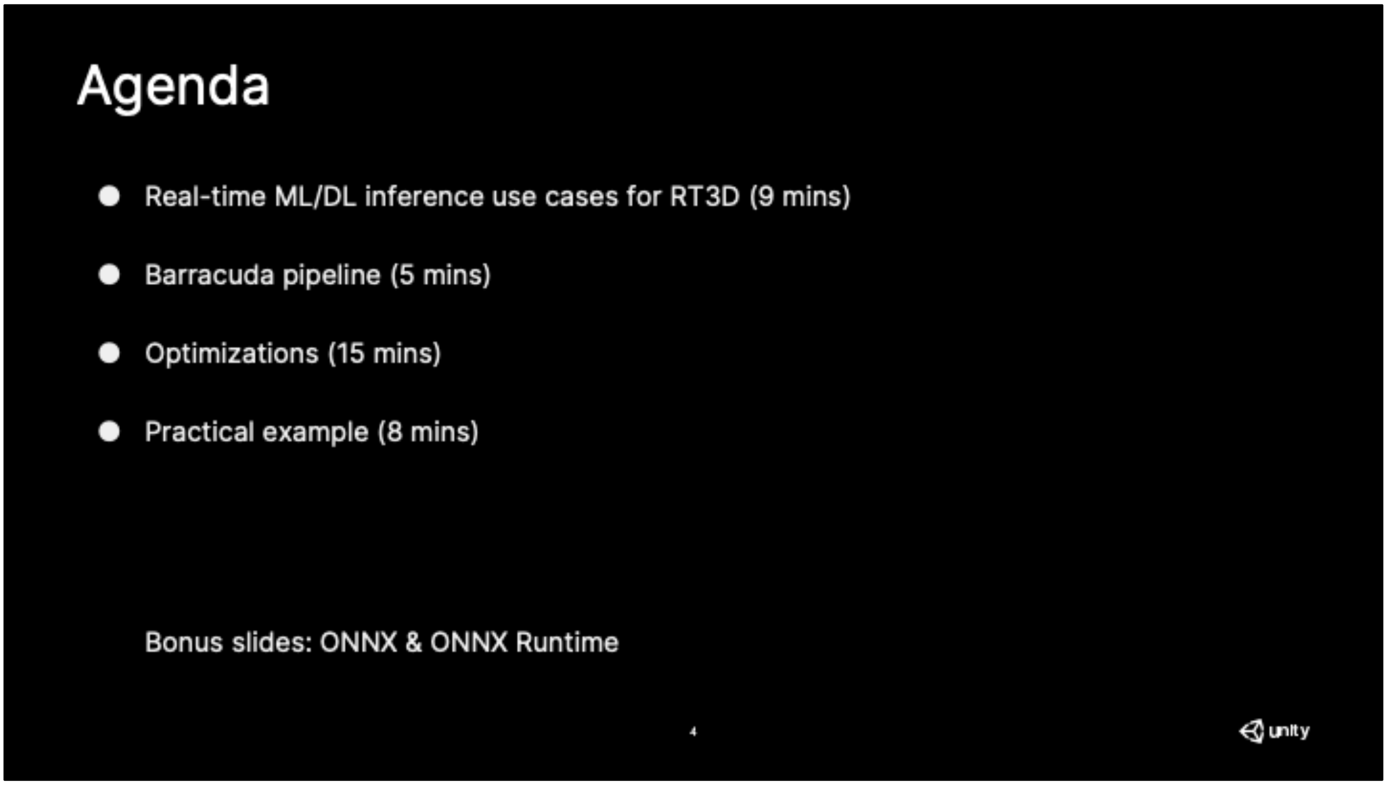
Unity Barracuda is a lightweight Neural Networks inference library for Unity.

It is crossplatform
It can run Neural Networks both on GPU and CPU

Delivered as an Unity package with source available.

Why do we do that?

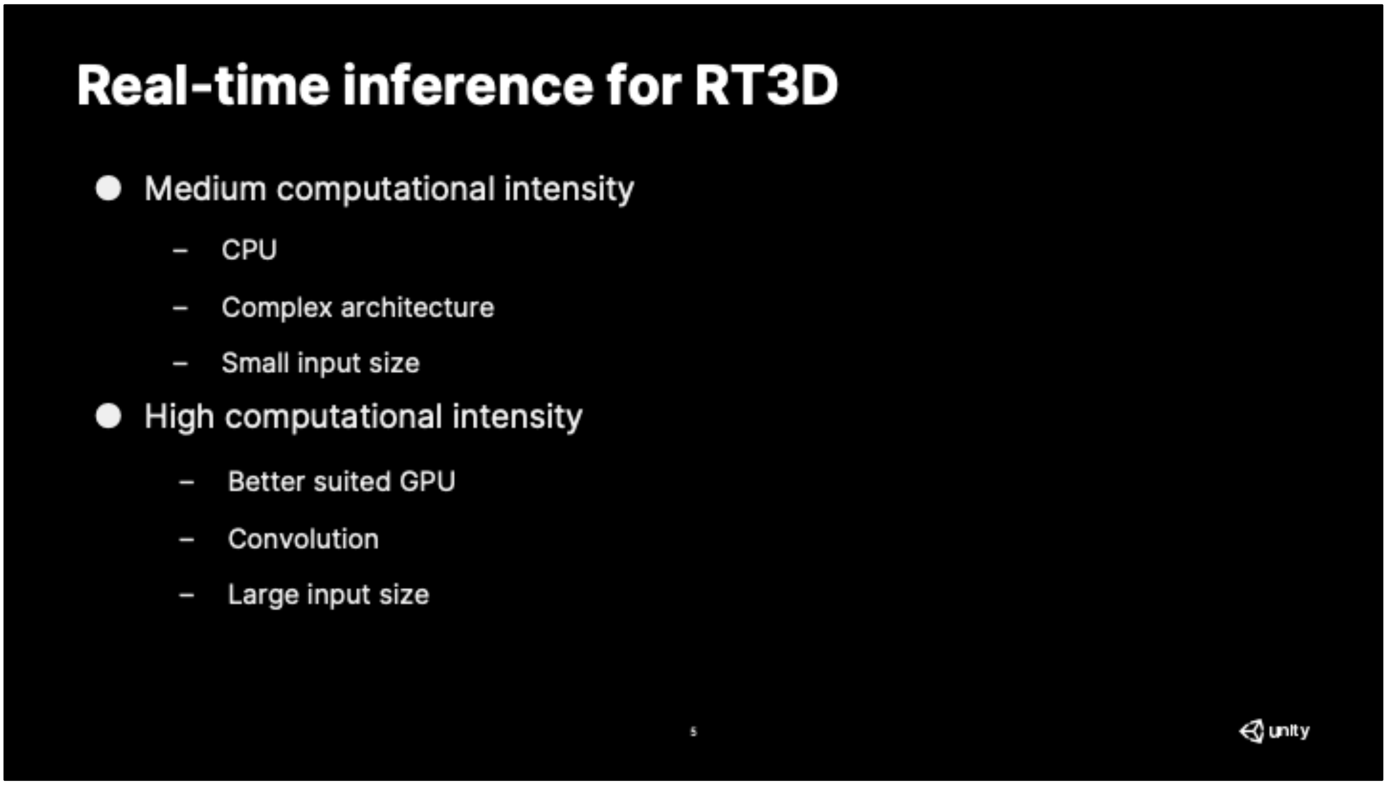
- **We think ML and RT3D communities can achieve awesome things together!**



Agenda

- Real-time ML/DL inference use cases for RT3D (9 mins)
- Barracuda pipeline (5 mins)
- Optimizations (15 mins)
- Practical example (8 mins)

Bonus slides: ONNX & ONNX Runtime

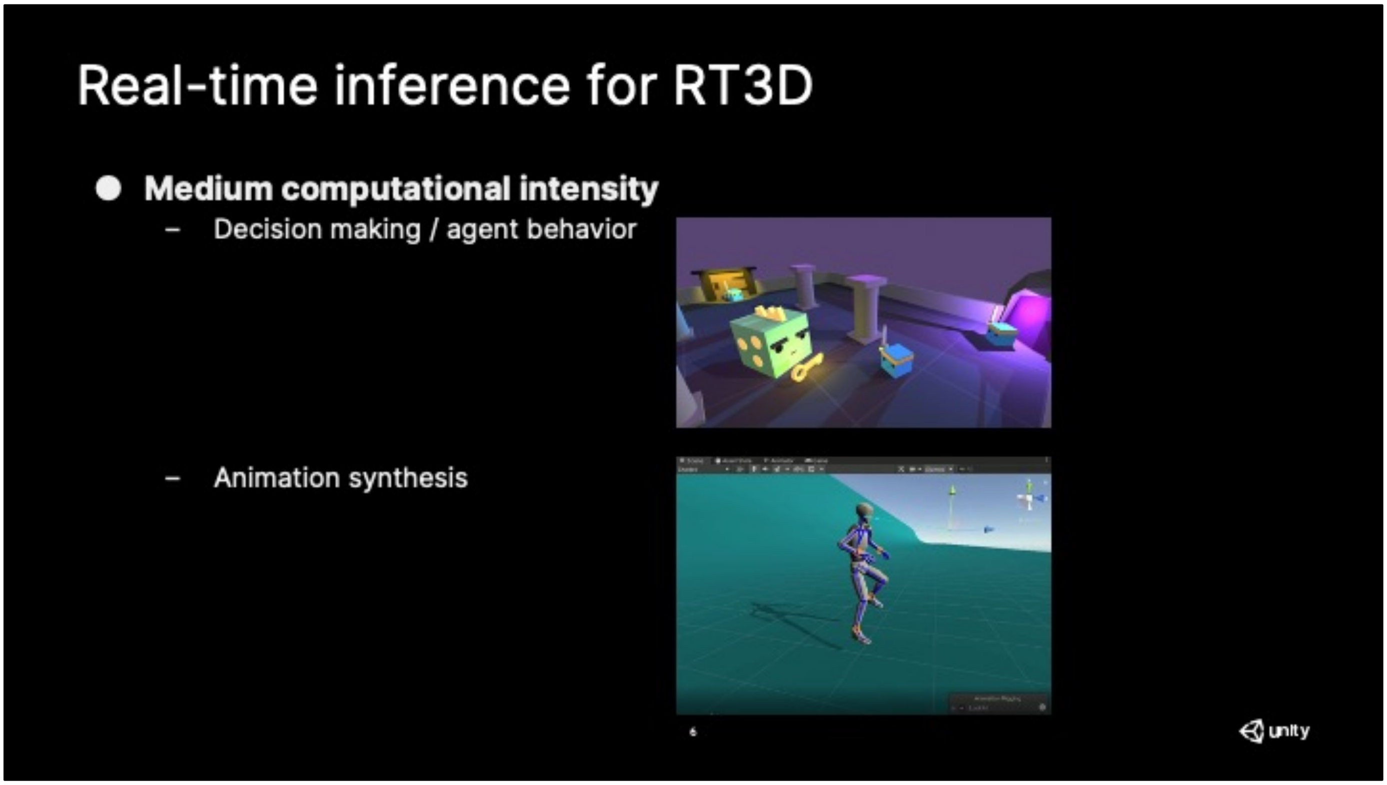


Real-time inference for RT3D

- Medium computational intensity
  - CPU
  - Complex architecture
  - Small input size
- High computational intensity
  - Better suited GPU
  - Convolution
  - Large input size

- Let’s take a look at **some use cases (there are a lot more, and list the list keep growing!)**:

- Two groups:
  - Medium computational intensity.
    - *Ideal for CPU inference*
      - *Low latency to interact with other system (physics or gameplay for example)*
      - *Complexe and/or branching architectures*
      - *Recurrent networks*
      - *Small input size*
  - High computational intensity.
    - *Better suited for GPU inference*
      - *Often driven by convolutional networks*
      - *Large input size*

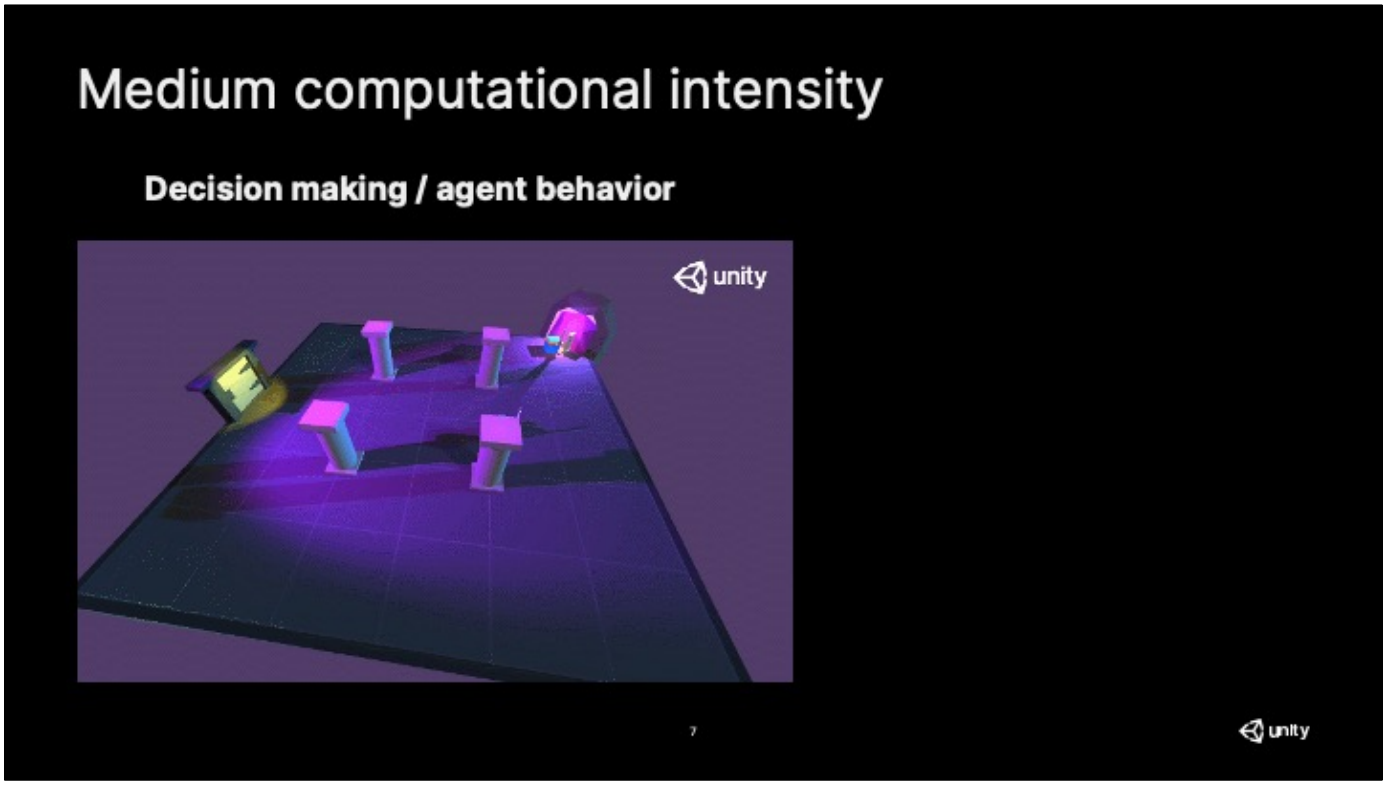


Real-time inference for RT3D

- **Medium computational intensity**
  - Decision making / agent behavior
  - Animation synthesis

6

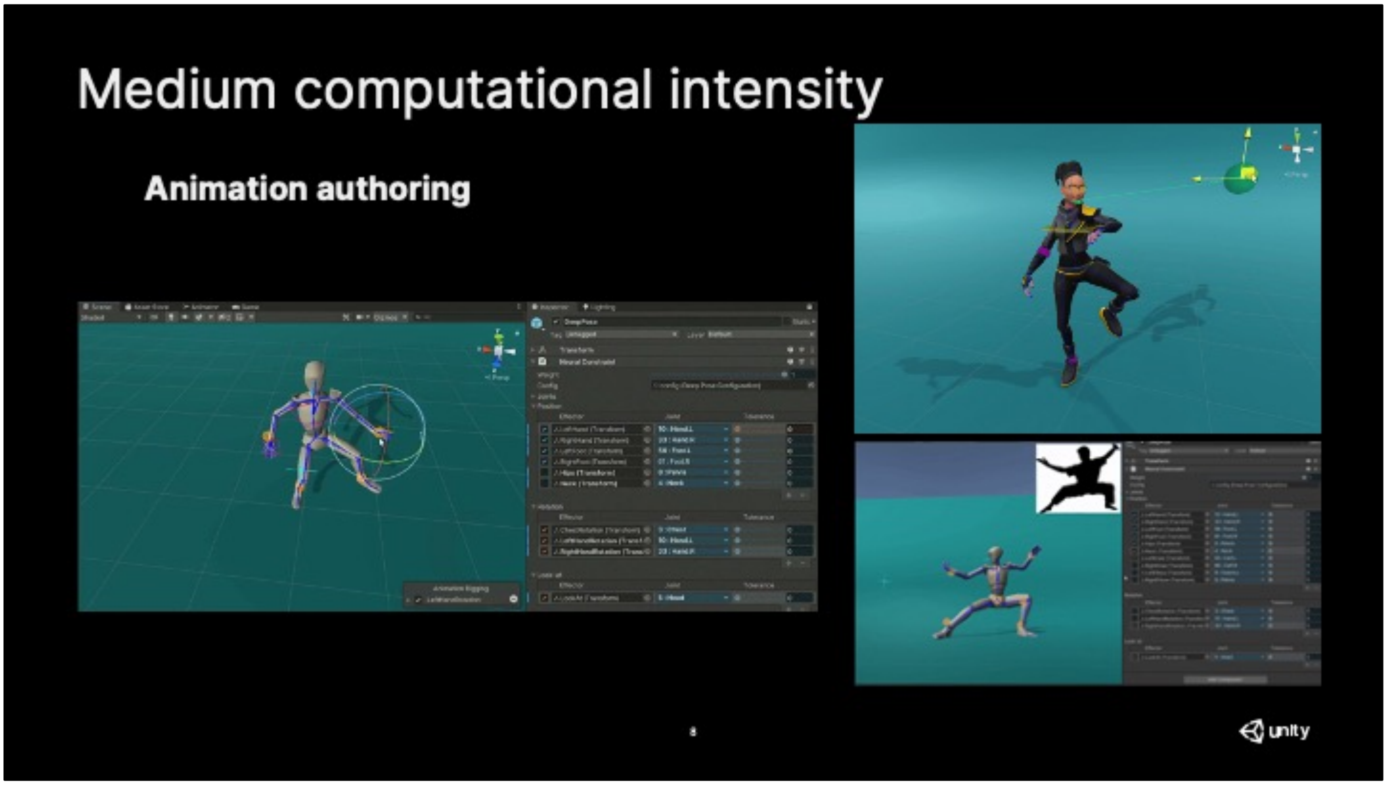
- 1st group: Low latency, medium computational intensity. *Ideal for CPU inference.*



Medium computational intensity

Decision making / agent behavior

- 1st group: Low latency, medium computational intensity. *Ideal for CPU inference.*
  - **Decision making / agent behavior**
    What if AI could be improved using RL (or ML in general). An example is Unity ML-Agents, delivered as an open-source project. It enables both:
    - Developer can easily train behavior ranging from clumsy to superhuman performance.
    - Researchers to access RT3D engine environment to do experiments.
    - ML Agent 2.0 added:
      - Cooperative behaviors
      - Variable amount of observations
      - Task parameterization (helping toward model genericity)
    - Github repo
    - Various training settings:
      - Broad range of task can be handled (from match 3 to parallel parking)

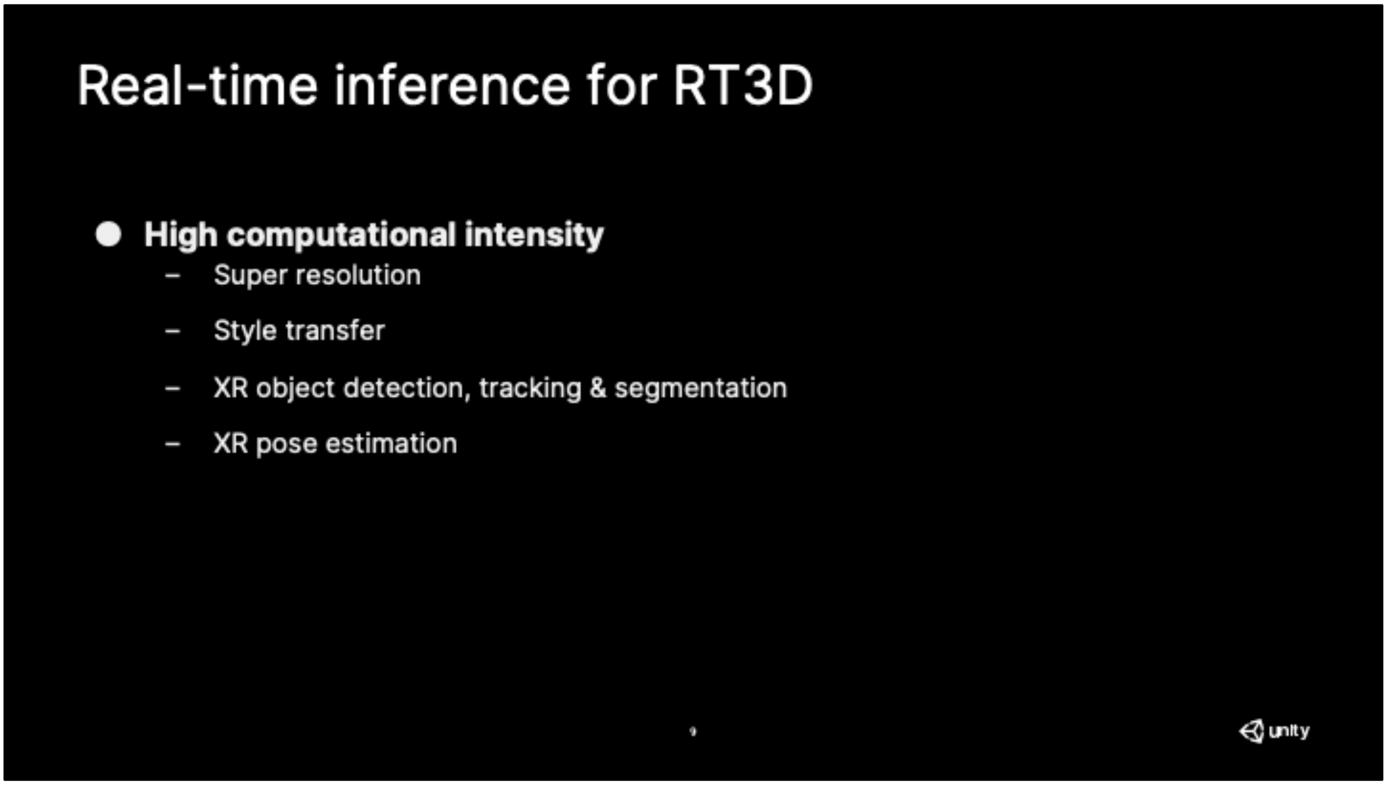


Medium computational intensity

Animation authoring

8

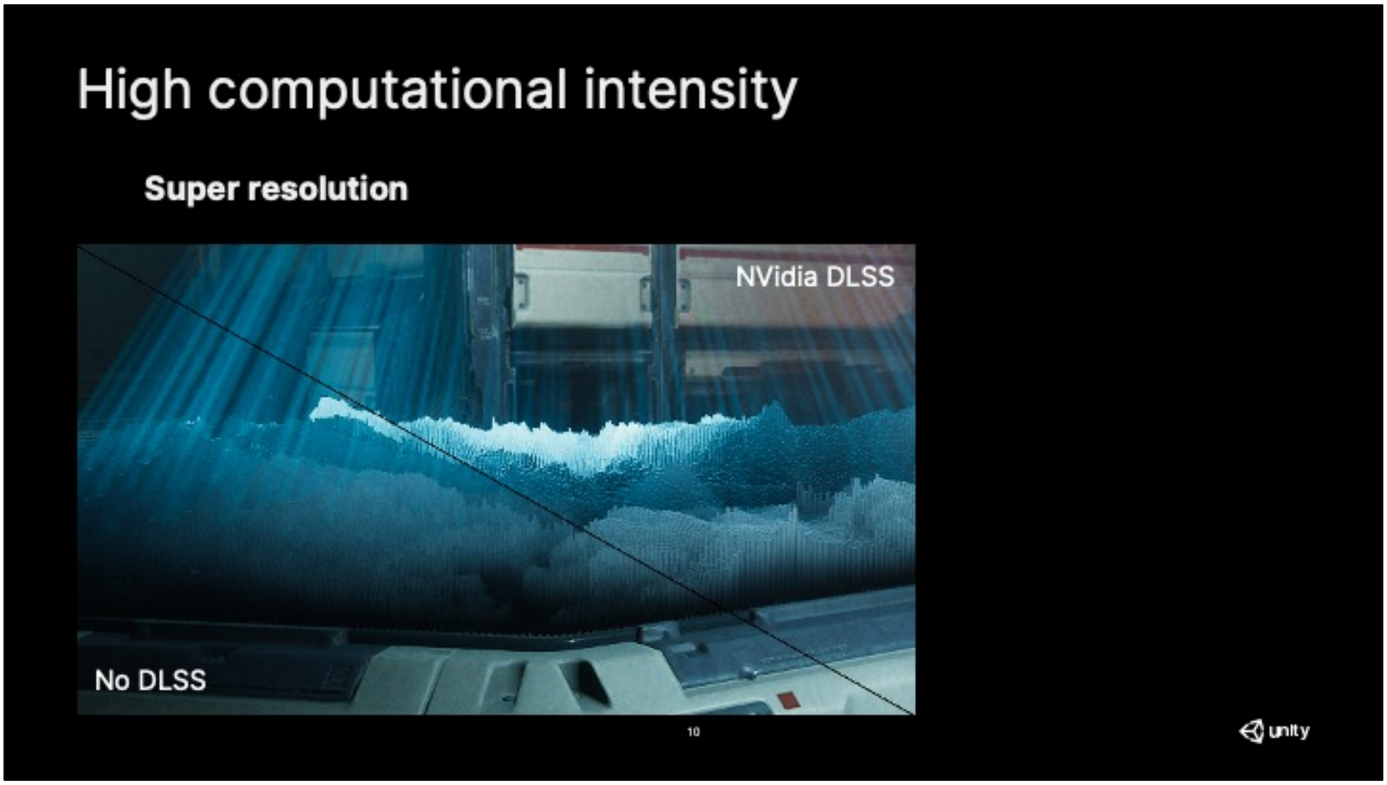
- 1st group: Low latency, medium computational intensity. *Ideal for CPU inference.*
    - **Animation authoring**
      A NN can be trained to understand natural pose to drive a skeletal mesh. This can both improve animation fidelity and greatly lower authoring cost!
        - Some great advances have been done in that regard at Unity Labs and we expect to see a lot of this in the future.
        - Animation authoring can be seen as an example of the broader area of augmented artistry at authoring time.
          In fact Real-time inference take a new meaning when you think of it as tool allowing faster iteration for RT3D content generation!
          After all AI can even generate music!



Real-time inference for RT3D

- **High computational intensity**
  - Super resolution
  - Style transfer
  - XR object detection, tracking & segmentation
  - XR pose estimation

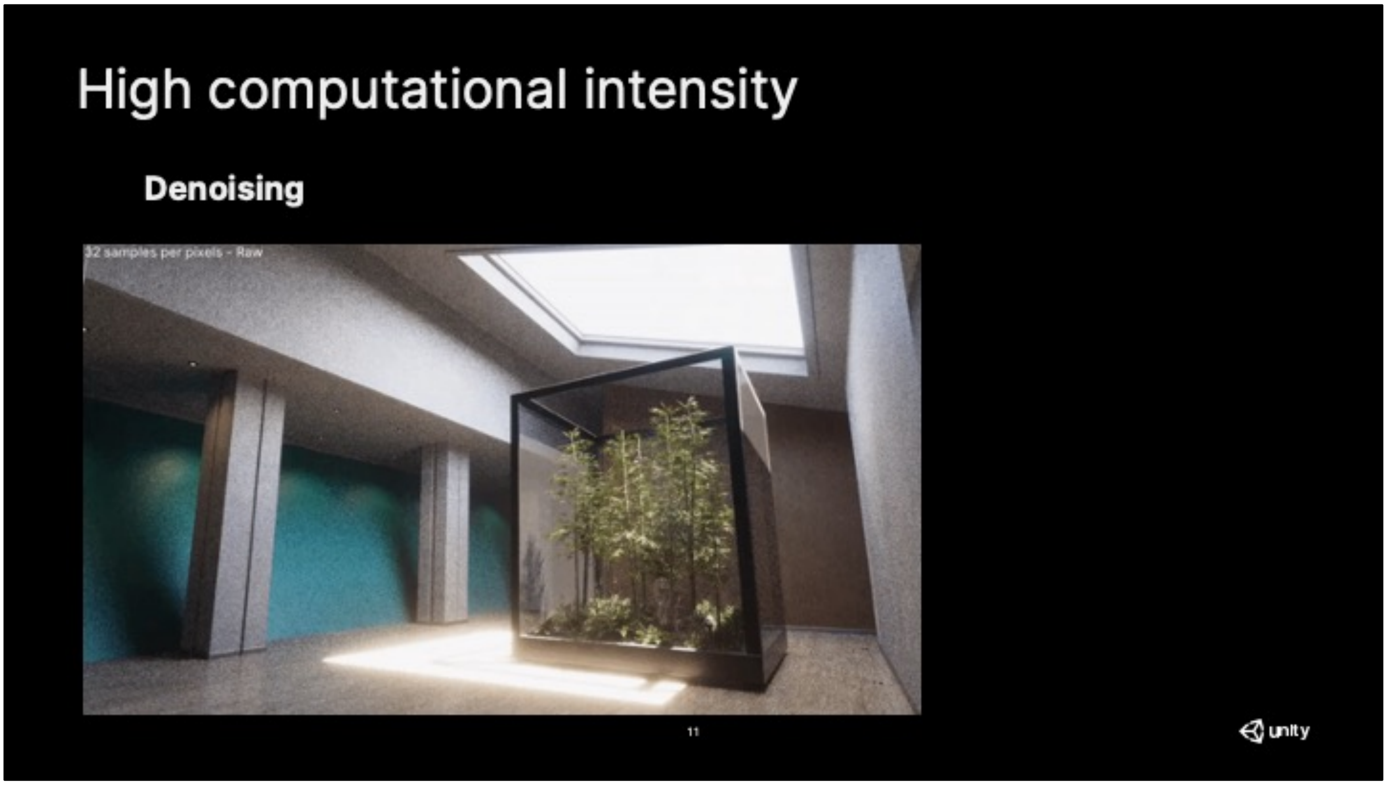
- 2nd group: High computational intensity. *Better suited for GPU inference.*



High computational intensity

Super resolution

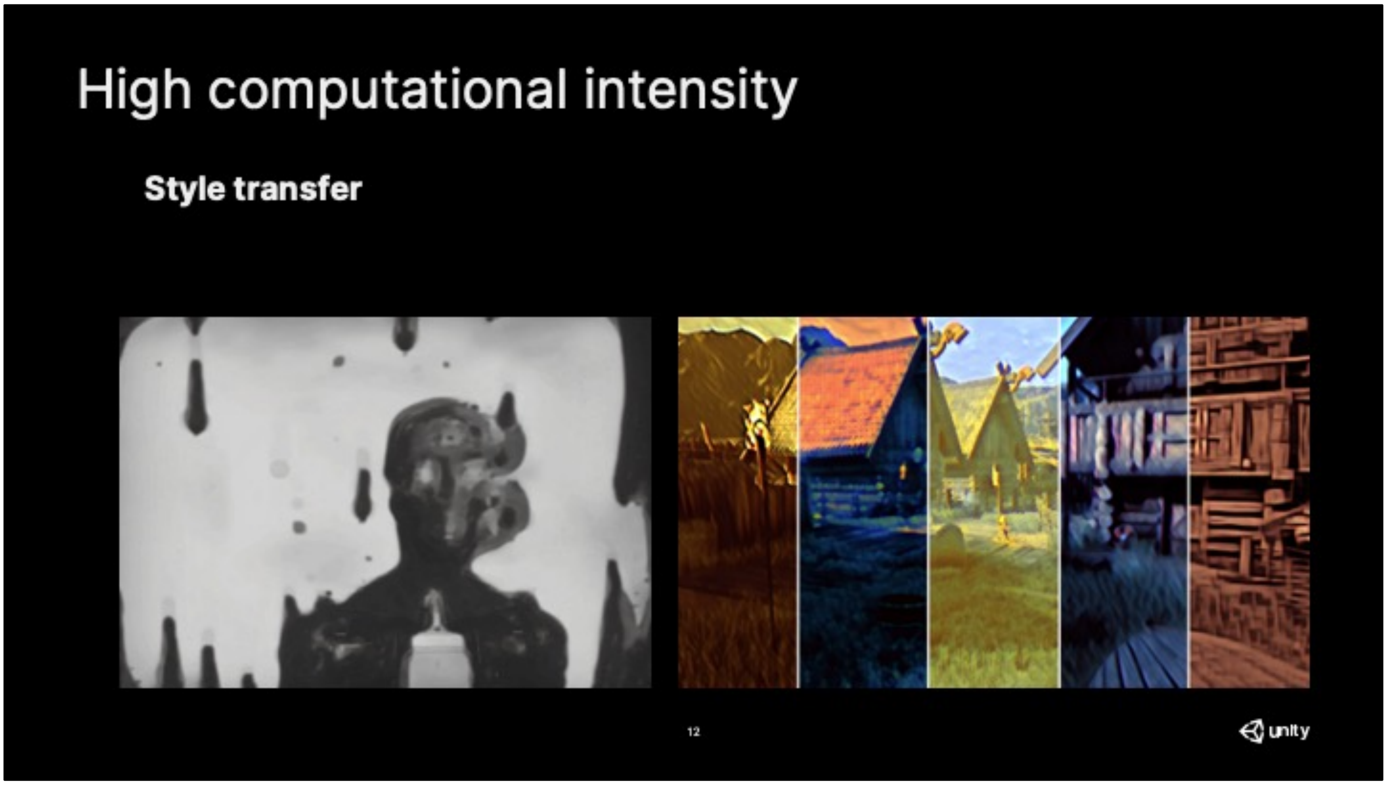
- 2nd group: High computational intensity. *Better suited for GPU inference.*
    - **Super resolution**
      Super resolution have proven to be a superior anti-aliasing and upsampling technique in various use-case.
        - We see super resolution as an essential element to address the either increasing target resolution of the displays in the coming future.
        - Many engine have already integrated DLSS from NVidia, including Unity (example above).



High computational intensity

Denoising

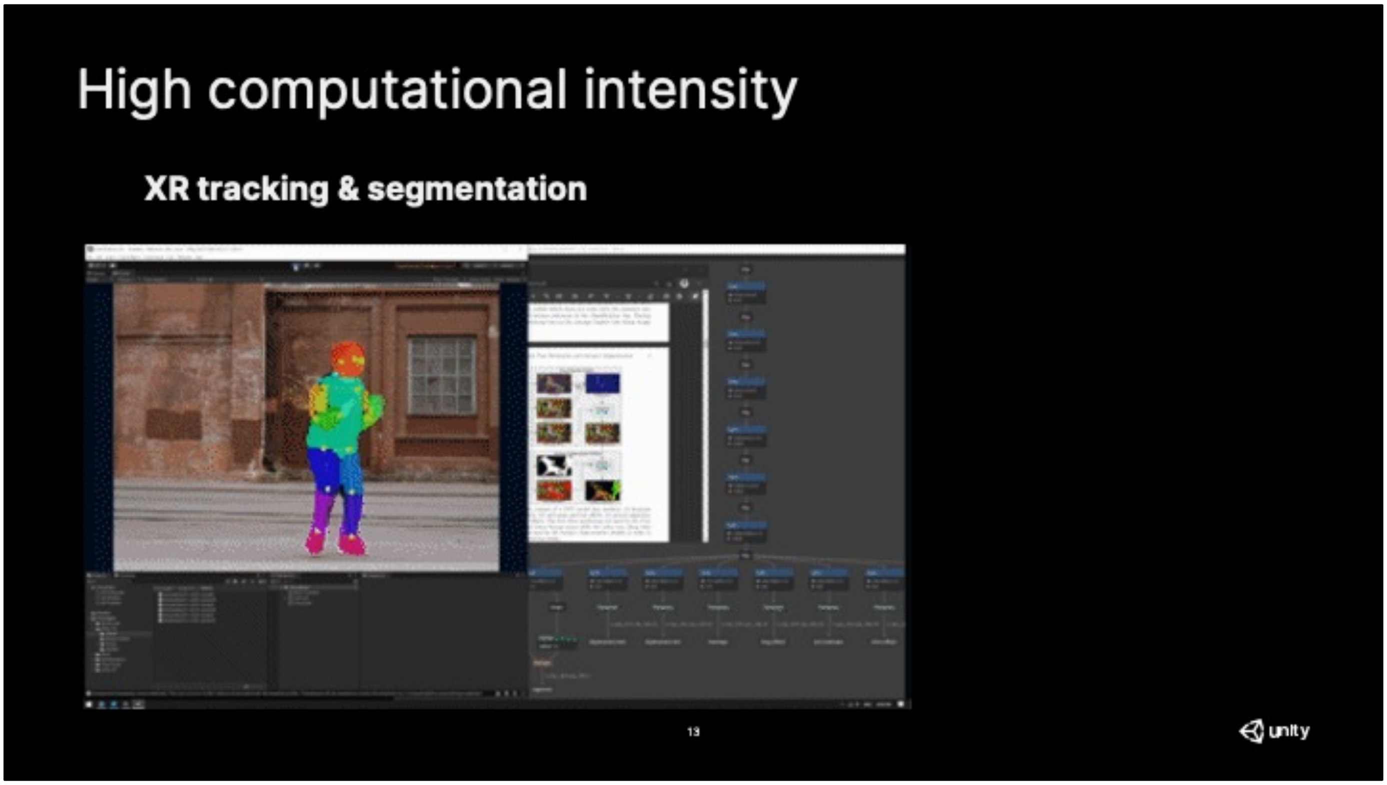
- 2nd group: High computational intensity. *Better suited for GPU inference.*
    - **Denoising**
        - Pathtracing is expensive by nature. AI help bring it closer to real-time performance using denoising.
        - Unity 2022.2 will include AI based denoiser on HDRP denoiser via:
            - Intel OIDN https://www.openimagedenoise.org/
            - NVidia optix https://developer.nvidia.com/optix-denoiser



High computational intensity

Style transfer

- 2nd group: High computational intensity. *Better suited for GPU inference.*
  - **Style transfer**
    Style transfer techniques could lead to emergent gameplays and artistic direction at a fraction of the cost deeply stylized rendering have at the moment.
    We will latter in this presentation dive in more details at at a use case/research we did at Unity Labs in regard to style transfer.



High computational intensity

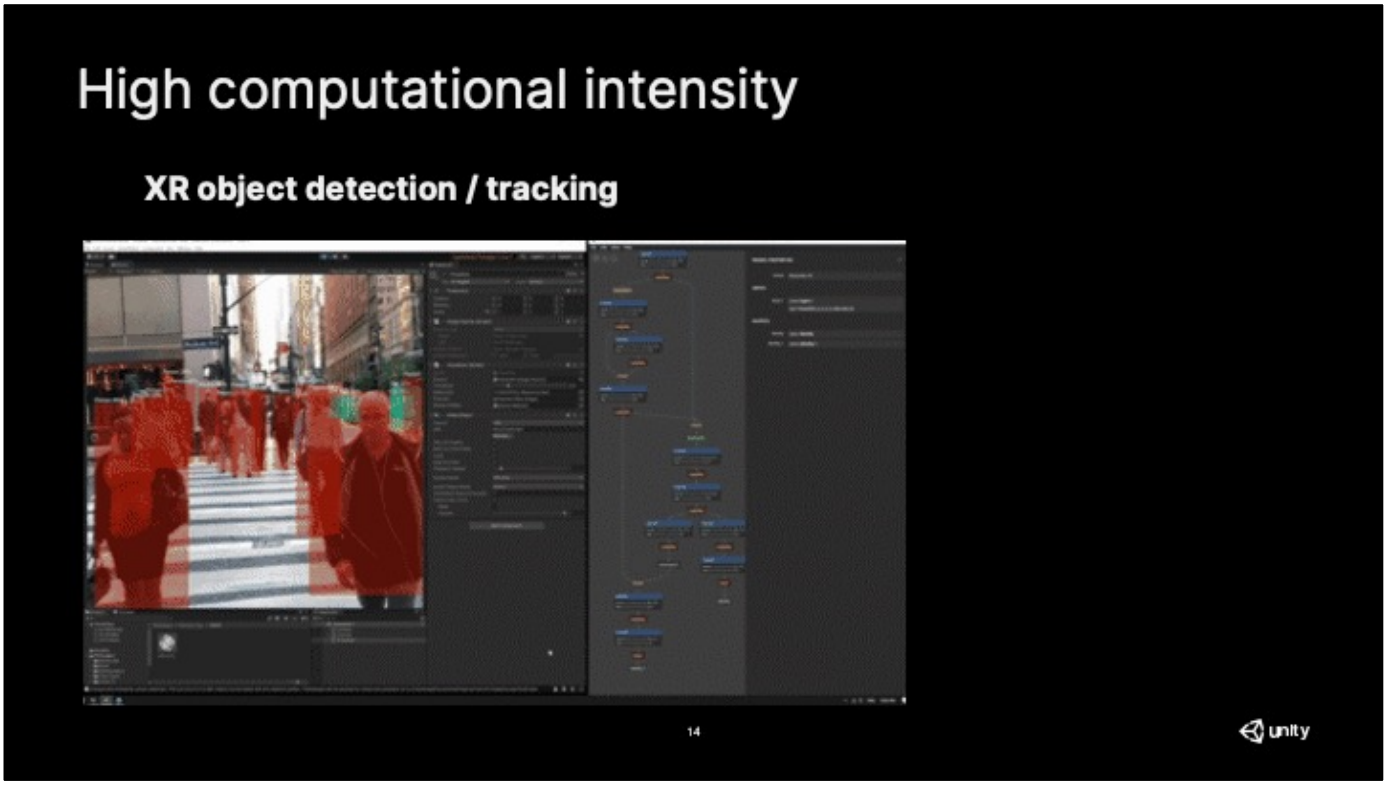
XR tracking & segmentation

13

- 2nd group: High computational intensity. *Better suited for GPU inference.*
  - **XR**
    DL excel in **object detection, tracking, segmentation** and **pose estimation** thanks to the great history of computer vision research. Providing the device have access to a camera those techniques can be used in many creative way.

    It is important to note that performance might be a challenge as XR use cases are often linked to low power hardware (However it is sometimes acceptable to split inference on a few frames).

Demo from Keijiro Takahashi https://twitter.com/i/status/1420742114942406659



High computational intensity

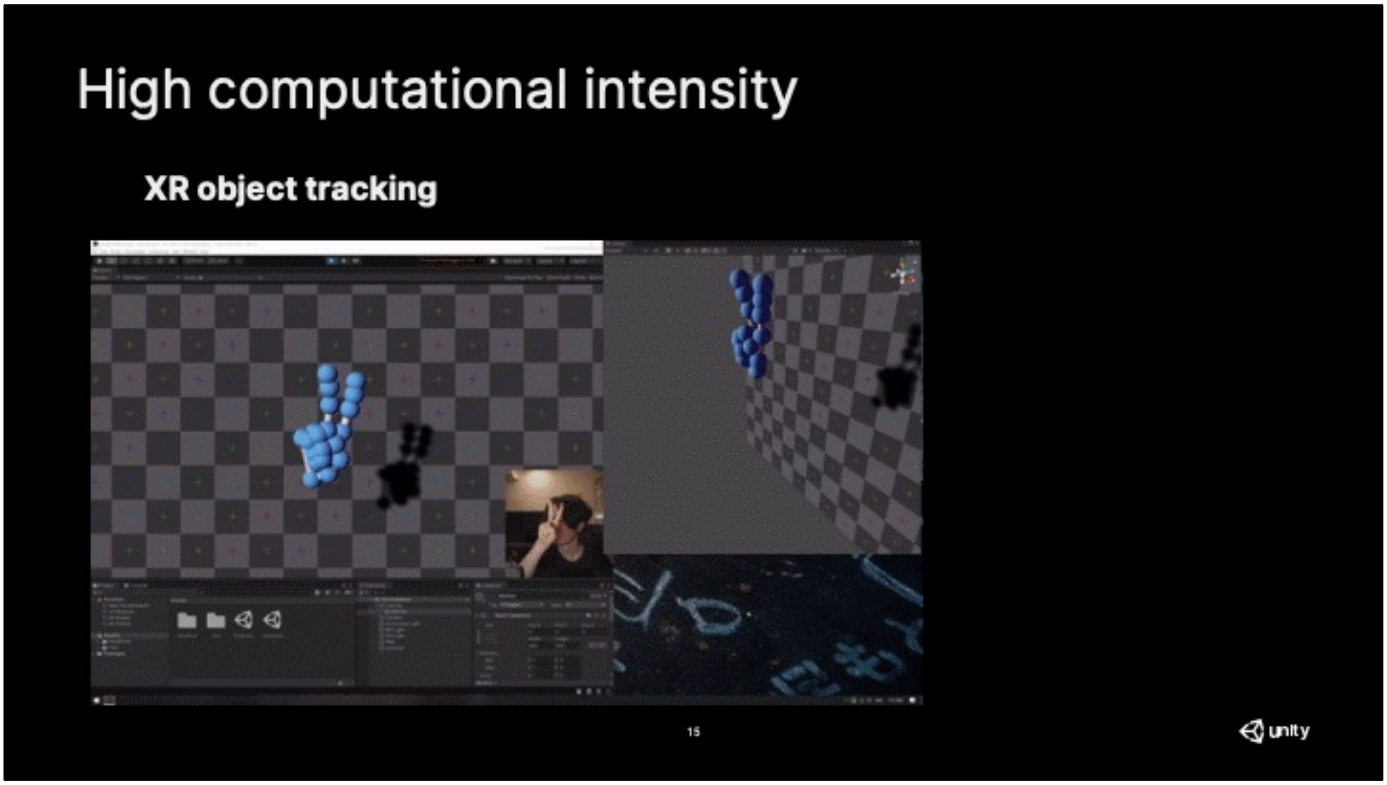
XR object detection / tracking

14

- 2nd group: High computational intensity. *Better suited for GPU inference.*
  - **XR**
    DL excel in **object detection, tracking, segmentation** and **pose estimation** thanks to the great history of computer vision research. Providing the device have access to a camera those techniques can be used in many creative way.

    It is important to note that performance might be a challenge as XR use cases are often linked to low power hardware (However it is sometimes acceptable to split inference on a few frames).

Demo from Keijiro Takahashi https://twitter.com/_kzr/status/1415331937623834629



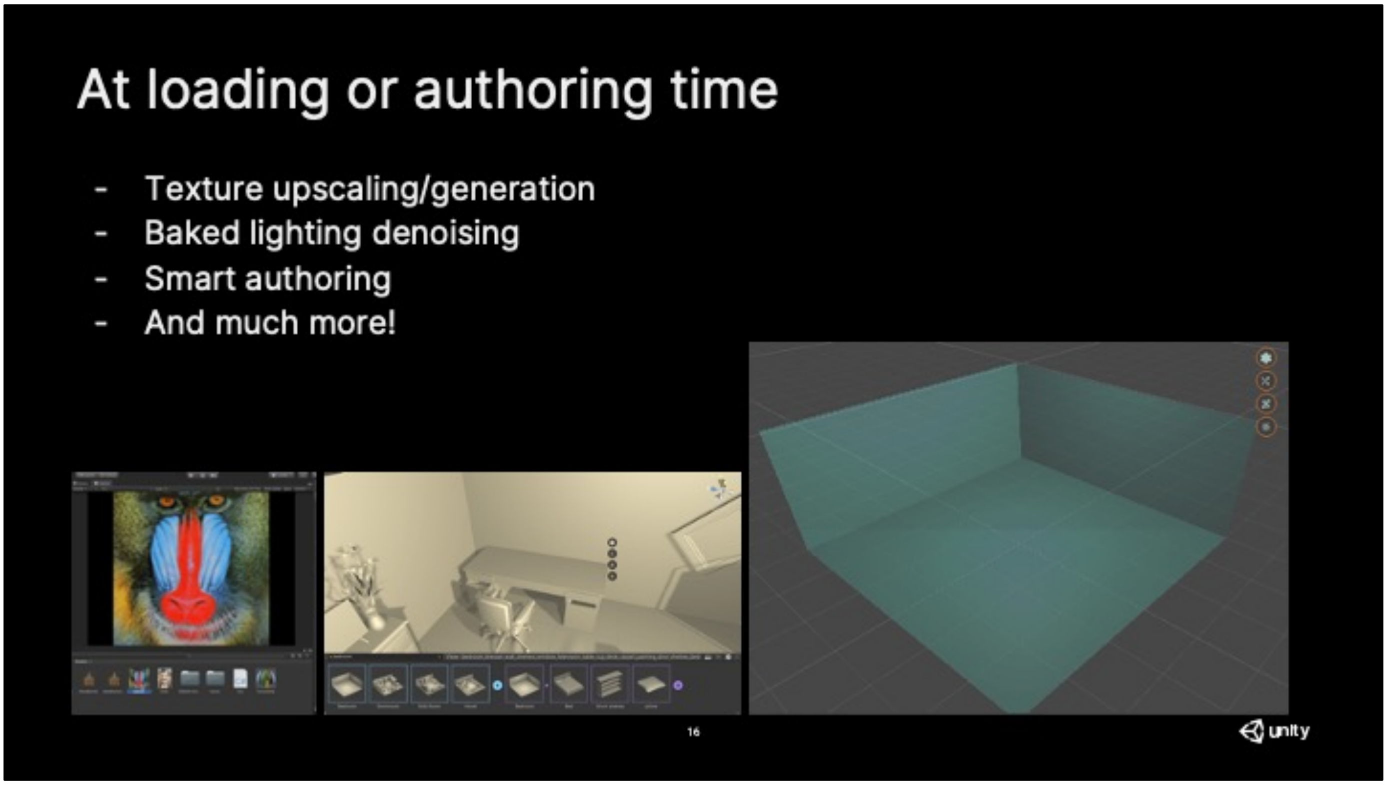
High computational intensity

XR object tracking

- 2nd group: High computational intensity. *Better suited for GPU inference.*
  - ■ **XR**
    DL excel in **object detection, tracking, segmentation** and **pose estimation** thanks to the great history of computer vision research. Providing the device have access to a camera those techniques can be used in many creative way.

    It is important to note that performance might be a challenge as XR use cases are often linked to low power hardware (However it is sometimes acceptable to split inference on a few frames).

Demo from Keijiro Takahashi https://twitter.com/i/status/1386626393723703297

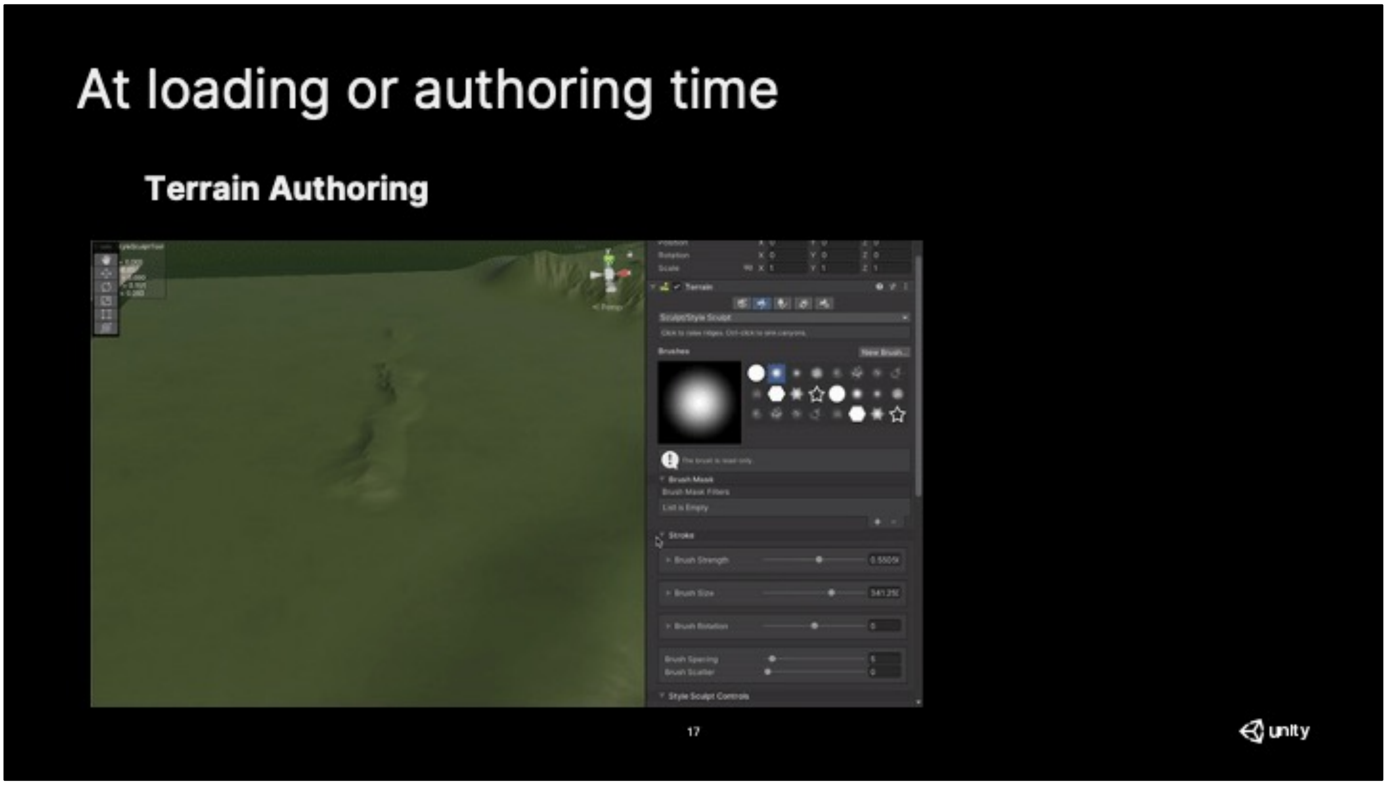


At loading or authoring time

- Texture upscaling/generation
- Baked lighting denoising
- Smart authoring
- And much more!

- Fast in engine inference does not limit itself to **on device** inference, loading or authoring time offer great opportunities too !

- So far we have focused on on-device real-time inference. However a lot can be achieved if one look at the broader capabilities of **in engine inference at loading or authoring time**!
    - **Textures**: A lot of tools are already leveraging ML to generate or upscale textures, this is quite interesting imho! In fact one can also think of ML as a compression/decompression methods! For example with sin networks.
    - **Baked lighting** is traditionally an offline process, however the recent hardware and software improvement around both ray tracing and ML denoising have bringed us to interactive iteration speeds if not more!
    - **AI assisted content authoring for faster workflow**. On the right are two examples showing AI smartly placing furniture inside a room, for a very fast workflow. Much more AI assisted authoring workflow can be thought off!



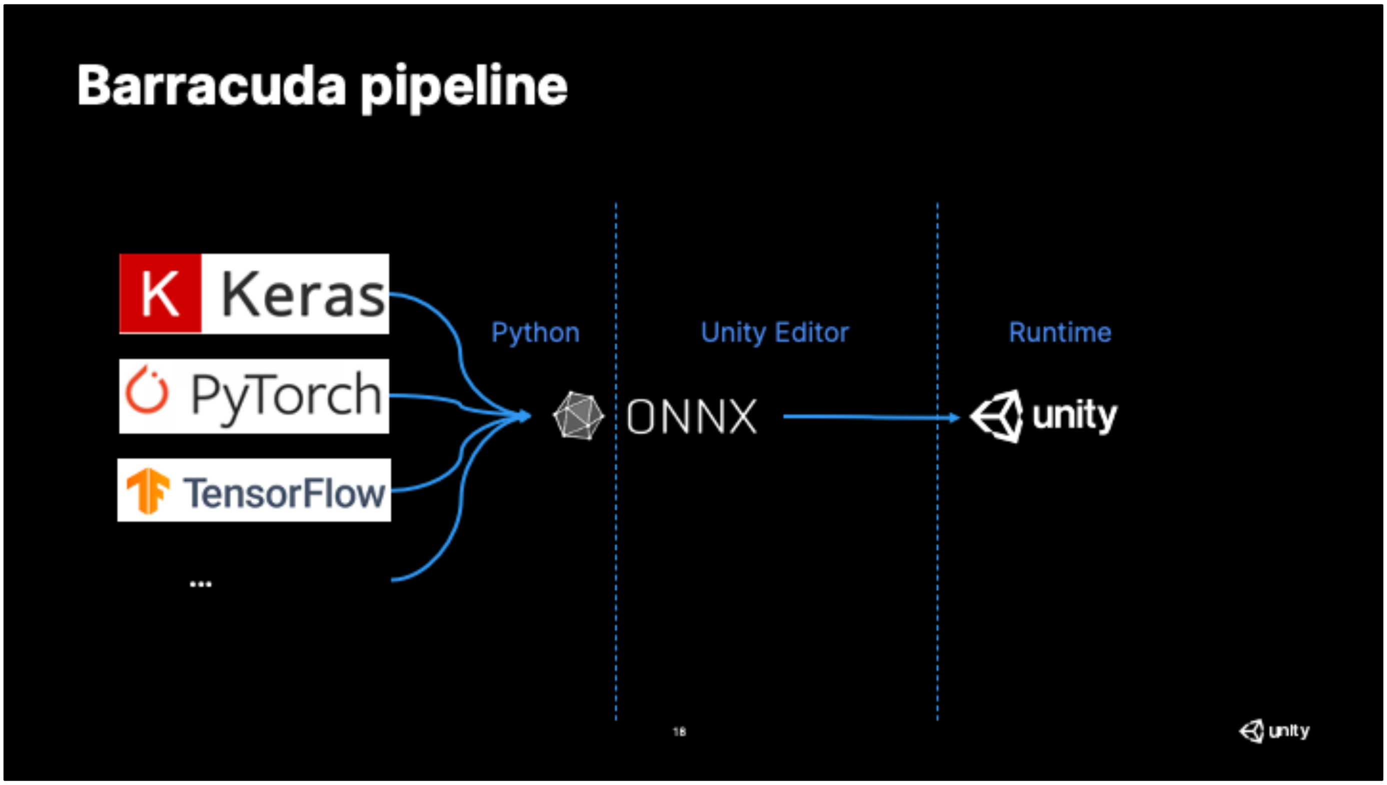
At loading or authoring time

Terrain Authoring

17

- ○ **AI assisted content authoring for faster workflow**:
  Continuing in the direction of ML assisted authoring here is an example of terrain authoring directly from the ML-Artistry world building team at Unity Labs. As you can see ML can drastically improve iteration time, allowing to generate high quality content in a fraction of the time.

- Finally : We hope these use cases illustrate well why we think ***native in engine inference can help bridge the gap between ML and RT3D communities***. A lot can be achieved with both working hand in hand.



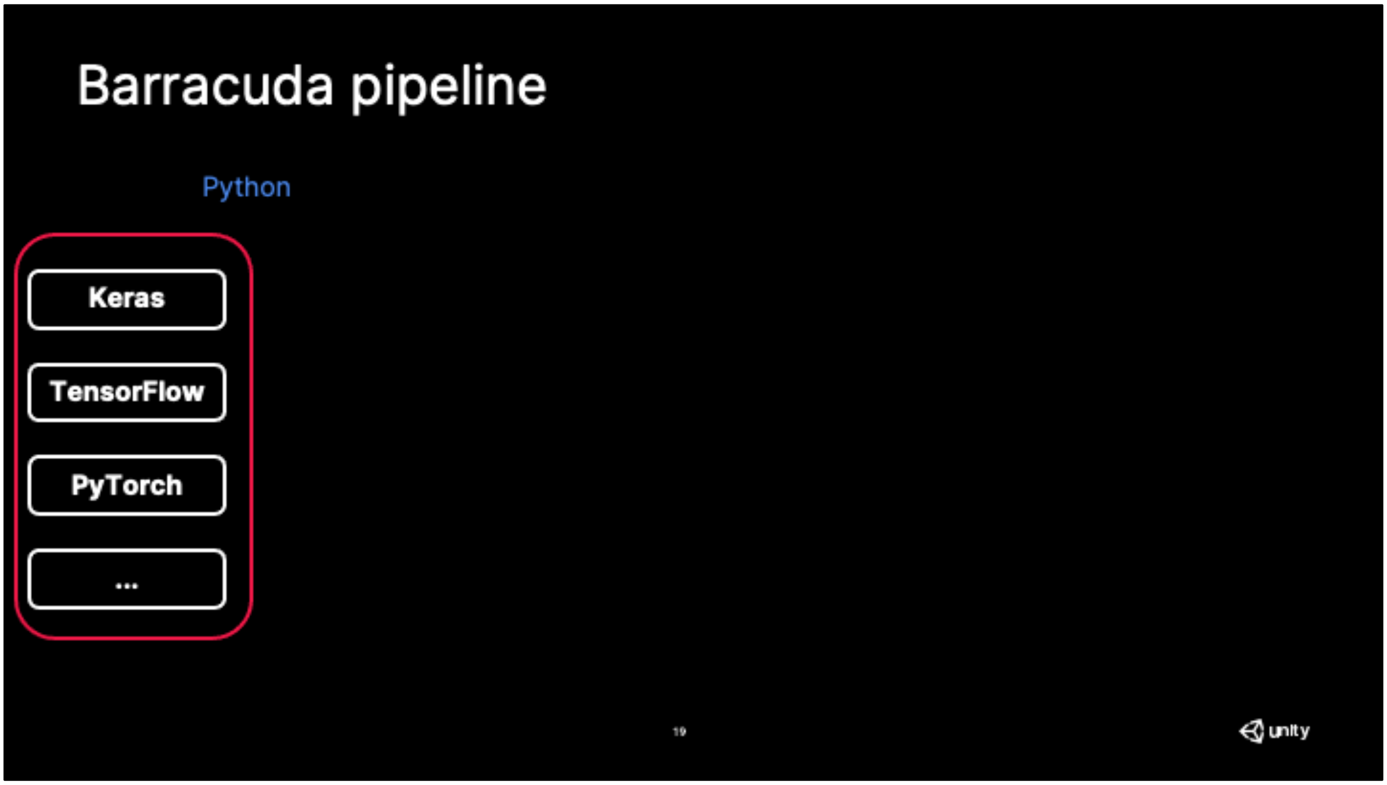
Barracuda pipeline

Now that we have taken a look at how powerful in engine inference can be let's take a look at how to achieve it.

The flow is the following:

- NN is trained in library of choice, PyTorch, Tensorflow, etc.
- NN is frozen/exported to an ONNX file
- file is dropped into Unity editor.
- At runtime the Barracuda model can be loaded and scheduled

Let's dive a bit further.



Barracuda pipeline

Python

Keras

TensorFlow

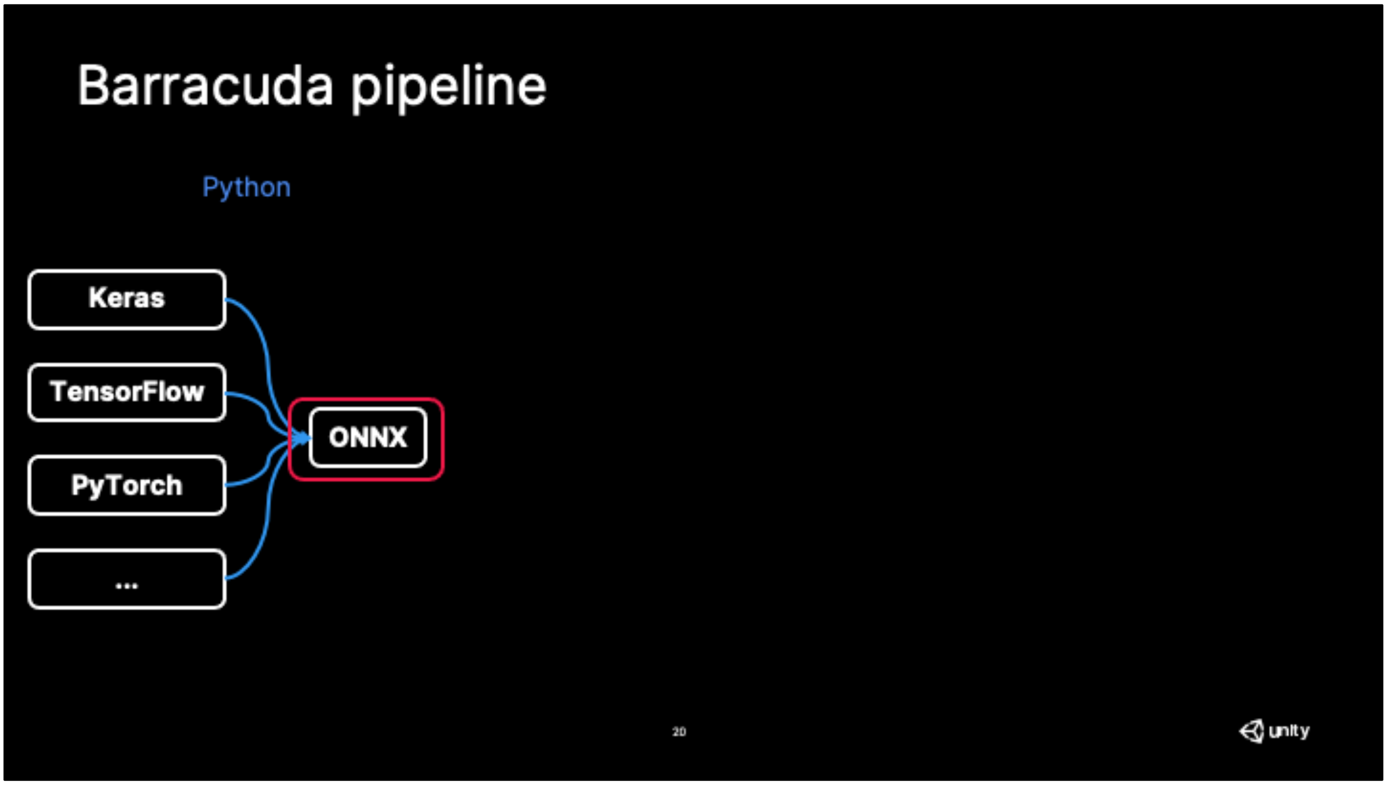
PyTorch

...

19

- Now that we have taken a look at how powerful in engine inference can be let’s take a look at how to achieve it.

  The flow is the following:
  - **NN is trained in library of choice, PyTorch, TF, etc.**



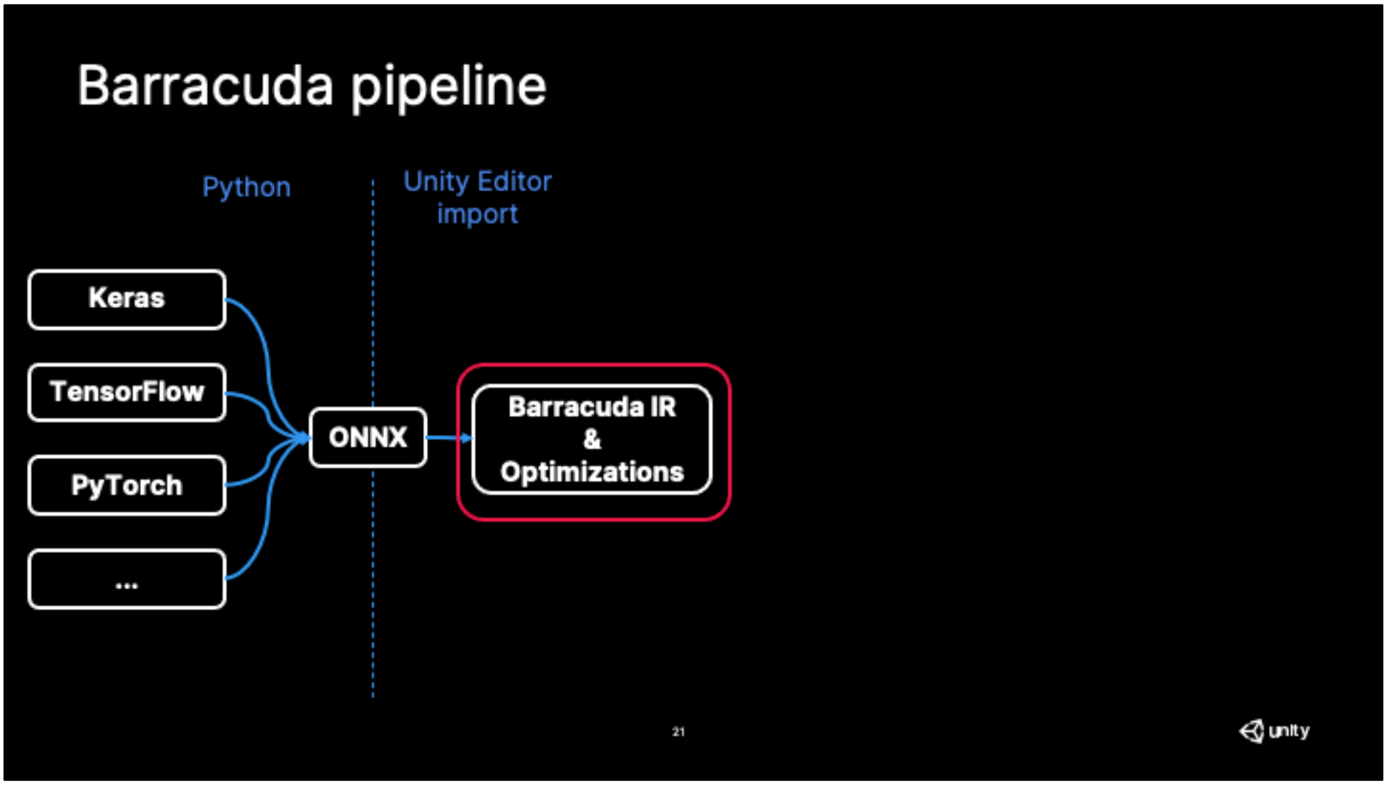
Barracuda pipeline

- Now that we have taken a look at how powerful in engine inference can be let's take a look at how to achieve it.

  The flow is the following:
  - NN is trained in library of choice, PyTorch, TF, etc.
  - **NN is frozen/exported to an ONNX file**
    - Open format to represent NN.
      - Defines a common set of DL operators: such as convolutions, activations, etc
    - Active community, well maintained and updated.
      - Many popular ML frameworks export to it (Pytorch, TF, etc).

      ⇒ See bonus slide for more on ONNX and how to export to it:

      Short story: ***It is often easy, sometime a one liner.***



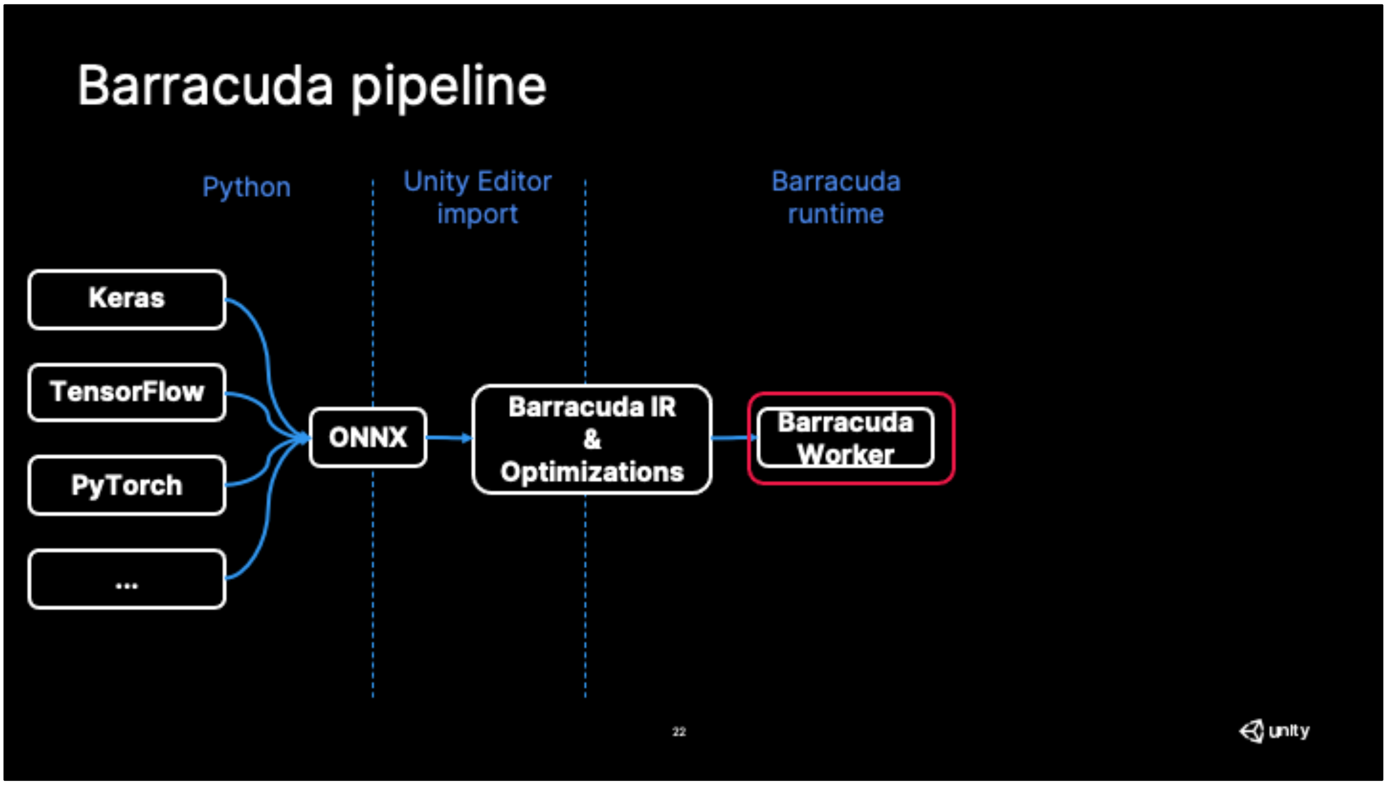
Barracuda pipeline

- Now that we have taken a look at how powerful in engine inference can be let’s take a look at how to achieve it.

  The flow is the following:
  - NN is trained in library of choice, PyTorch, TF, etc.
  - NN is frozen/exported to an ONNX file (more on this latter)
  - **ONNX file is dropped into Unity editor, witch:**
    - Translates to Barracuda internal representation (IR)
      - A bit different from ONNX in term of granularity sometime for performance reasons sometime for legacy reasons, also we don’t support all of the ops, import problem will be reported to the user here.
    - Applies offline optimizations (more on that latter)

    - After import (and actually at any point) user code can alter the Barracuda IR representation, ie add or remove layers, change weights or even build model from scratch.

      This can be useful when python and app code are not expecting the same inputs because of normalisation or color space format for exemple.



Barracuda pipeline

Python

Unity Editor import

Barracuda runtime

Keras

TensorFlow

PyTorch

...

ONNX

Barracuda IR & Optimizations

Barracuda Worker

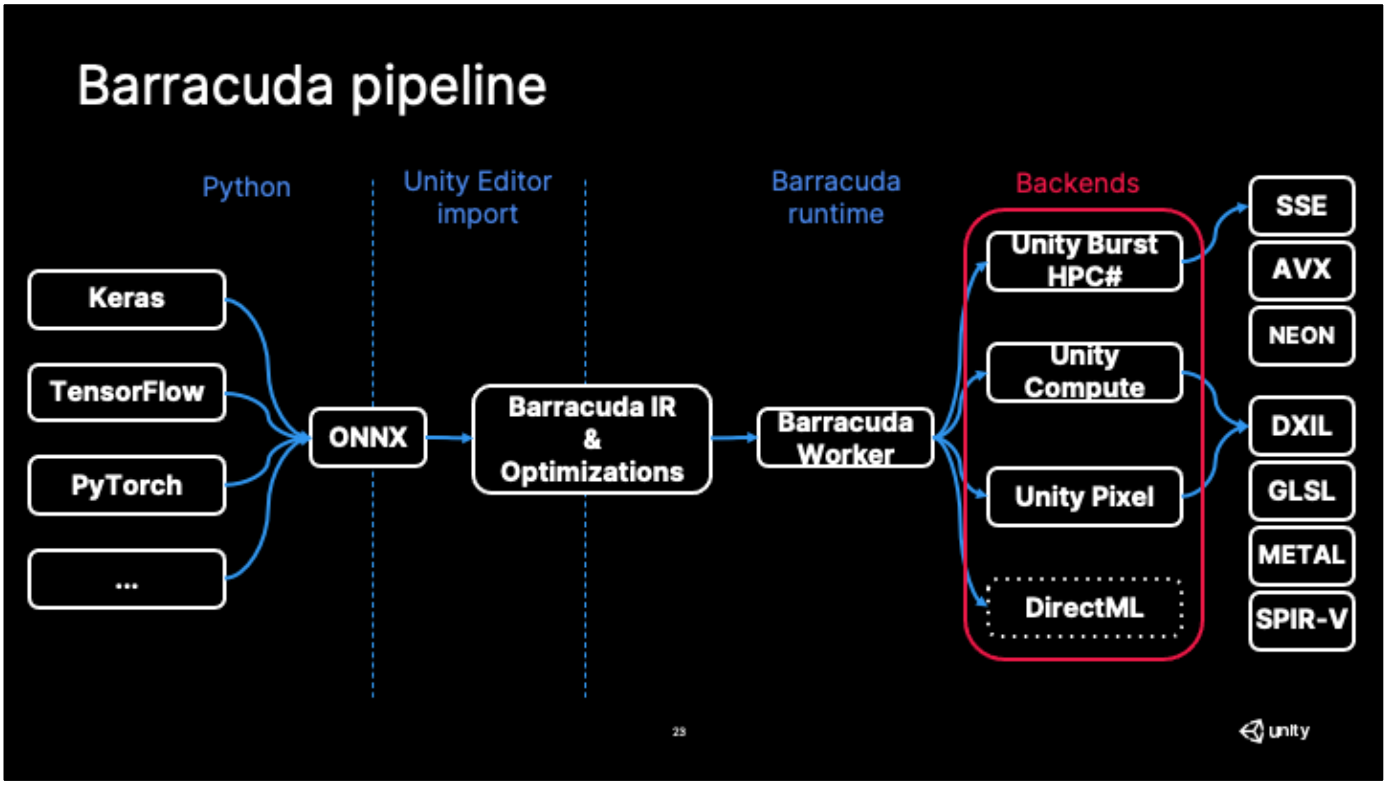
22

- Now that we have taken a look at how powerful in engine inference can be let's take a look at how to achieve it.

  The flow is the following:
  - NN is trained in library of choice, PyTorch, TF, etc.
  - NN is frozen/exported to an ONNX file
  - ONNX file is dropped into Unity editor
  - **At runtime the Barracuda model can be loaded and scheduled:**
    - Loading and scheduling of network is left to the application code for flexibility reasons. For example it can be really useful to split network inference in a smart way knowing performance profile of the various layer on target hardware.

    - However Barracuda take care of all internal states, memory and asynchronous behavior.

    - General idea is that you can load and schedule a network in a few line of code but can also deep dive and take control if your are looking for optimal inference performance for your application.



Barracuda pipeline

Python

Unity Editor import

Barracuda runtime

Backends

Keras

TensorFlow

PyTorch

...

ONNX

Barracuda IR & Optimizations

Barracuda Worker

Unity Burst HPC#

Unity Compute

Unity Pixel

DirectML

SSE

AVX

NEON

DXIL

GLSL

METAL

SPIR-V

23

- Now that we have taken a look at how powerful in engine inference can be let’s take a look at how to achieve it.

  The flow is the following:
  - NN is trained in library of choice, PyTorch, TF, etc.
  - NN is frozen/exported to an ONNX file
  - ONNX file is dropped into Unity editor
  - **At runtime the Barracuda model can be loaded and scheduled:**
    - **About the backend:**
      - CPU using the Burst compiler + job system → allow to compile to extremely optimized and scalable native code.
        - Burst allow to use SIMD register and instruction from carefully written C# (a subset of C# to be exact).

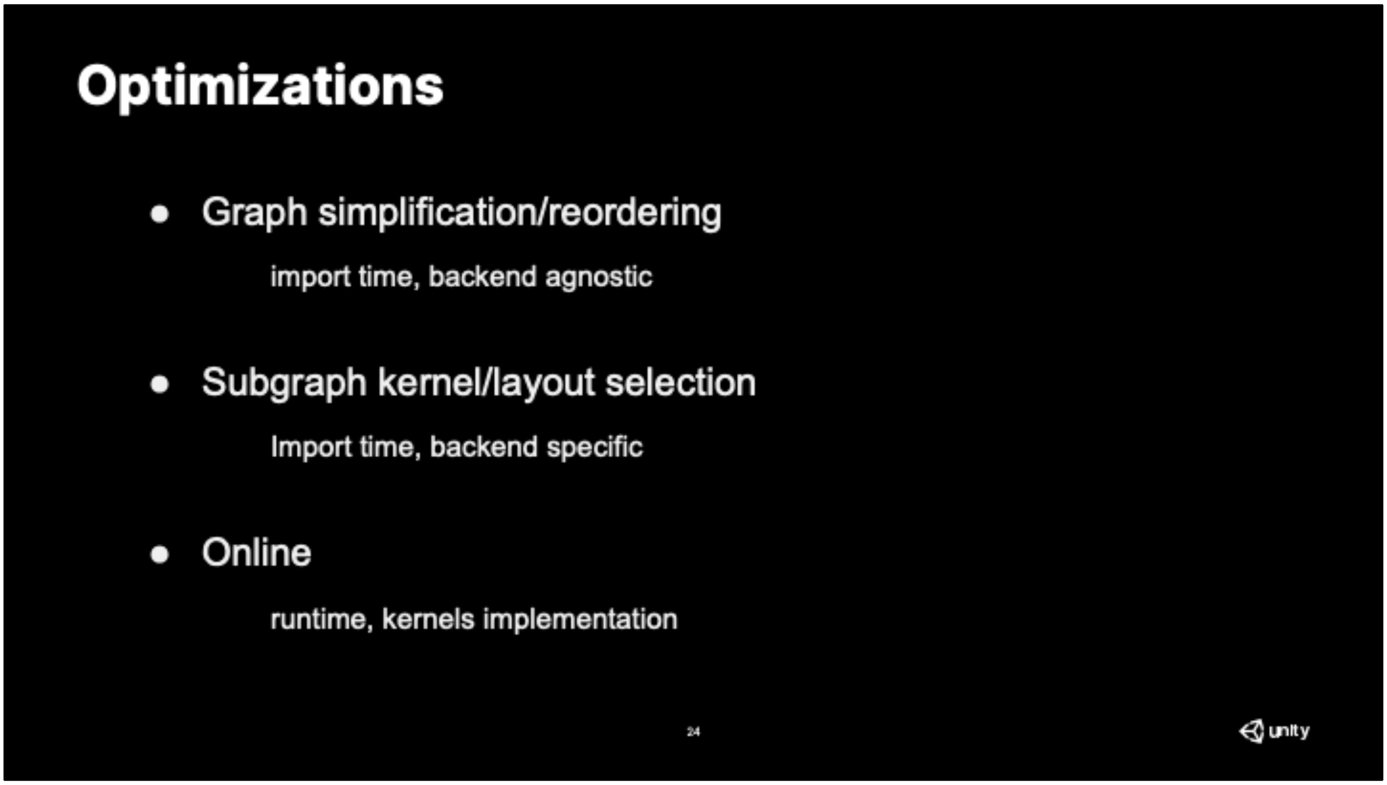
      - GPU using Unity compute shaders system → HLSL based language cross compilated to any platform supporting compute shader.
        - Barracuda support Metal, DX11, DX12 &

Vulkan. We are not actively supporting OpenGLES as driver quality is not as great as we would like on some devices in regard to compute shaders (we advice vulkan there), also WebGL does not supported compute shader.

  - Compiler chain is currently based on FXC however DXC is currently being introduced to Unity (see 2021.2 beta).

- This architecture also allows building against dedicated ML hardware acceleration API such as DirectML / CoreML / NNAPI.
  Those backend are not currently shipped with Barracuda, we have however successfully experimented with all the three of them internally.

  A note: DirectML requires DX12 and according to our stats DX11 is still insanely popular among RT3D developers.



Optimizations

- Graph simplification/reordering

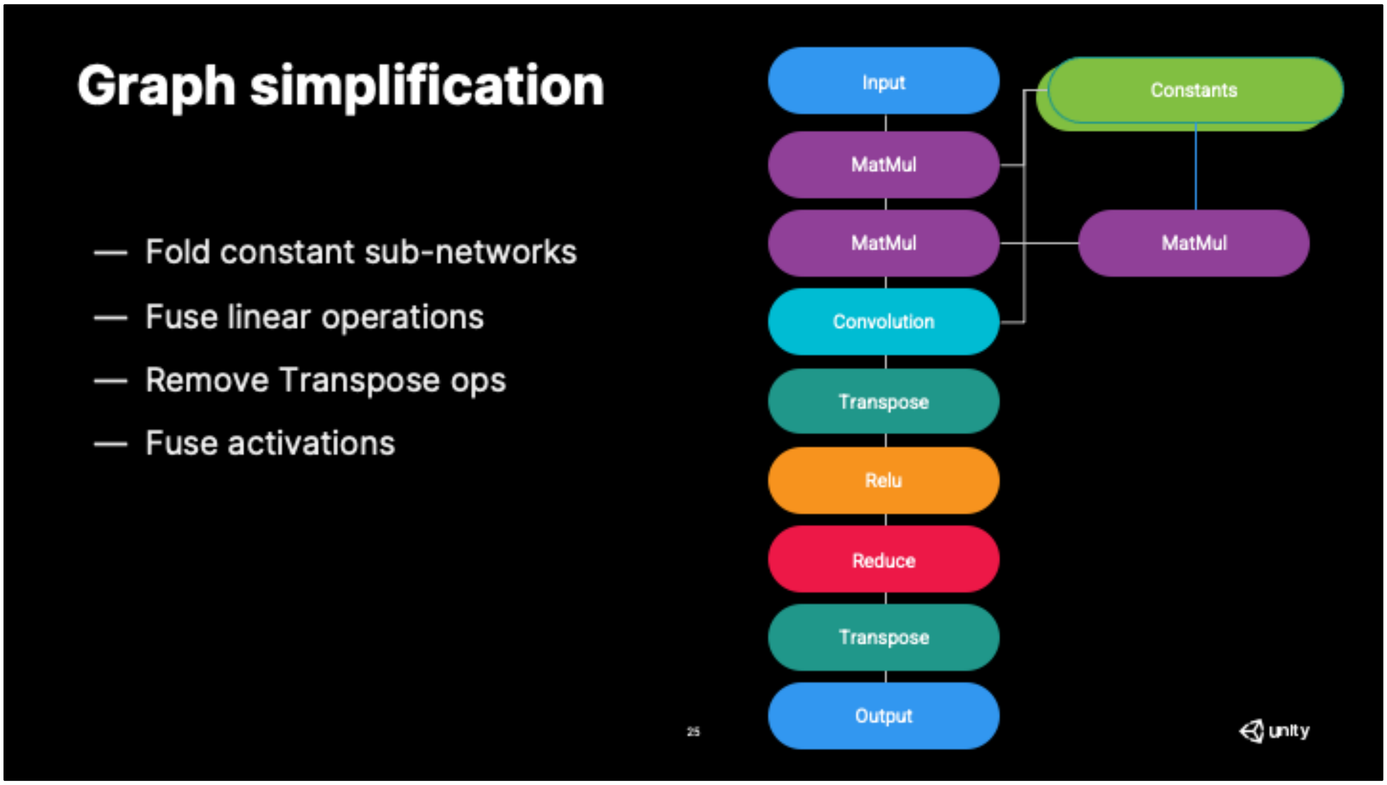
  import time, backend agnostic

- Subgraph kernel/layout selection

  Import time, backend specific

- Online

  runtime, kernels implementation

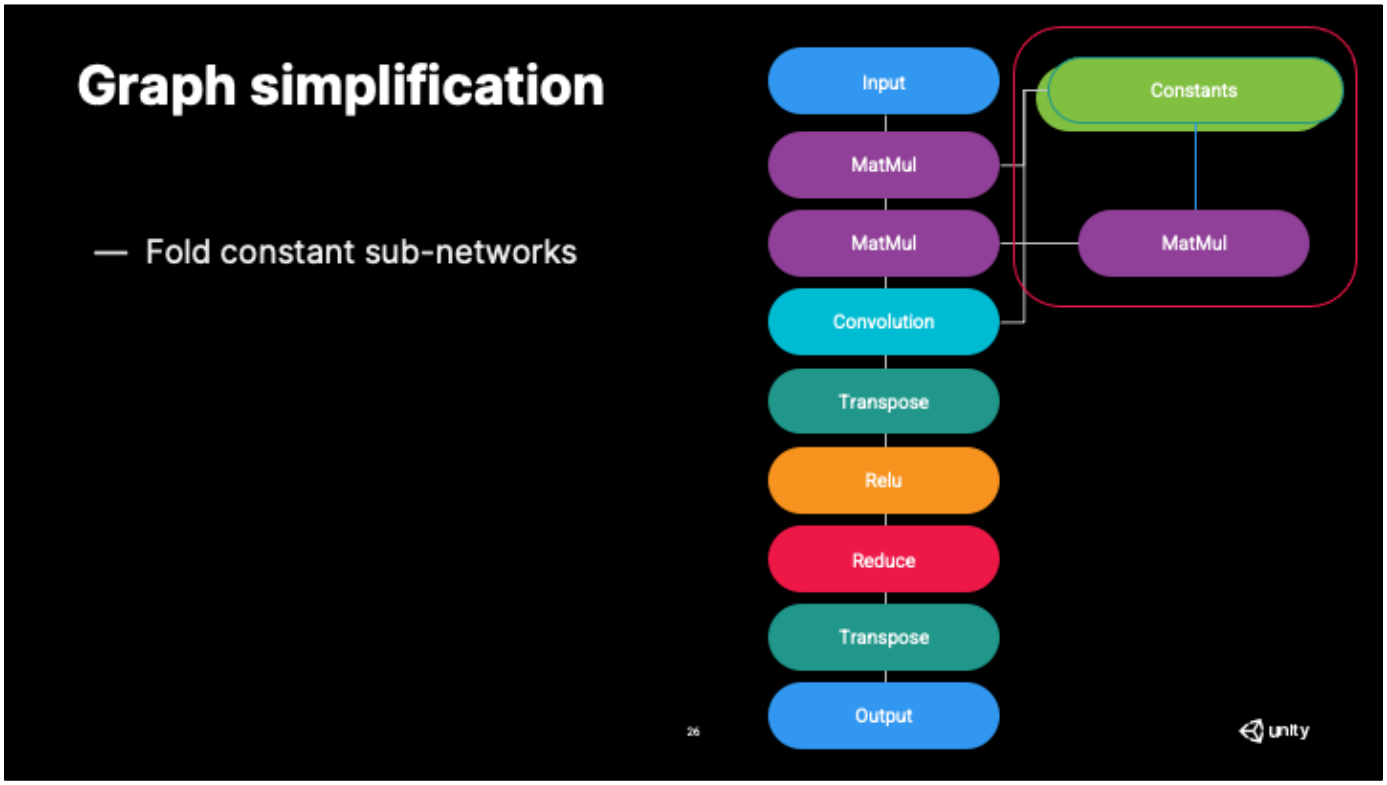


Graph simplification

- Fold constant sub-networks
- Fuse linear operations
- Remove Transpose ops
- Fuse activations

Let’s take a toy example and see some examples of model level optimization (offline).

NOTE: This model would not be legit without at least a reshape especially between MatMul and convolution.
It is omitted here for simplicity as it result in most case as a no-op.

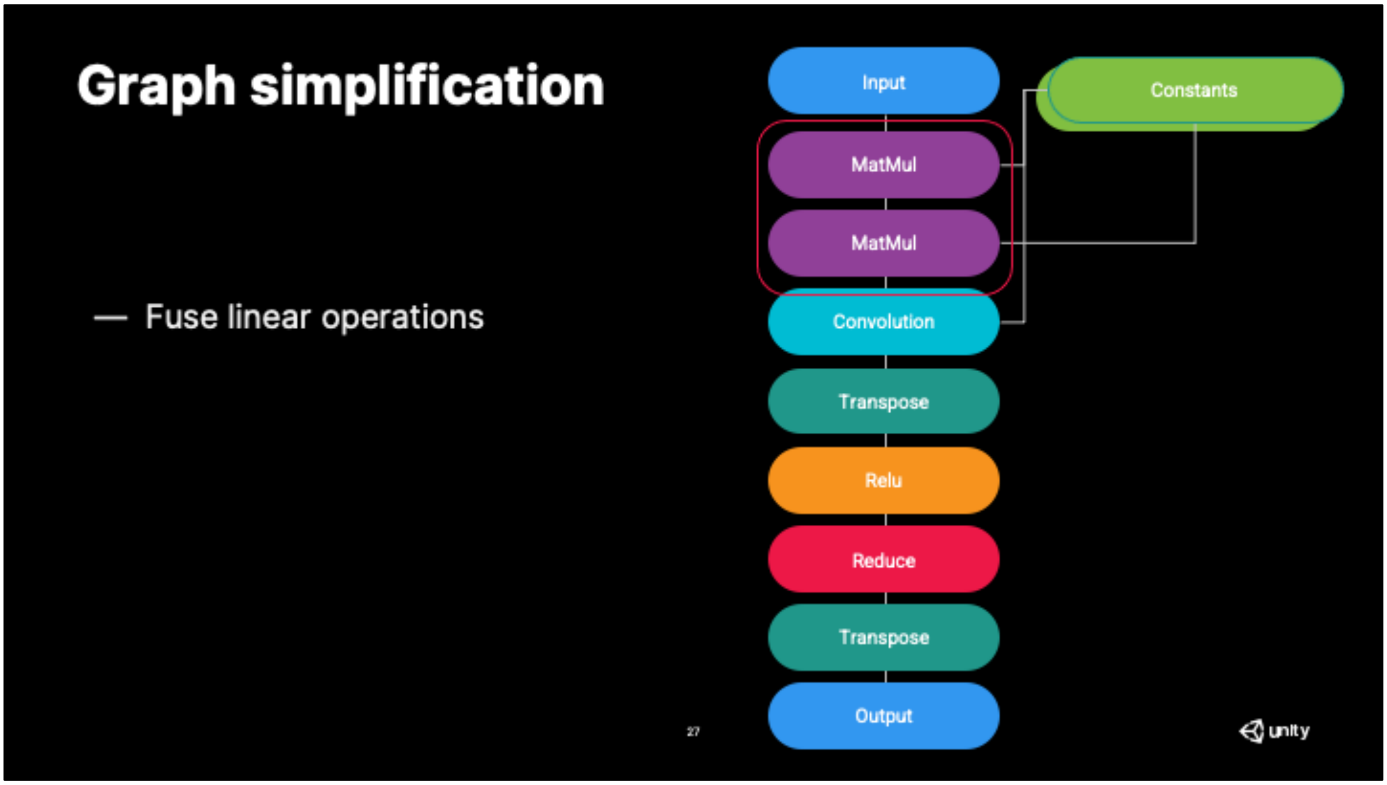


Graph simplification

— Fold constant sub-networks

26

NN often have branches which have constant input (i.e. the input of those branches don’t change at runtime), in those cases we compute those branches at import time and fold them to constants for inference to use.



Graph simplification

- Fuse linear operations

Input

MatMul

MatMul

Convolution

Transpose

Relu

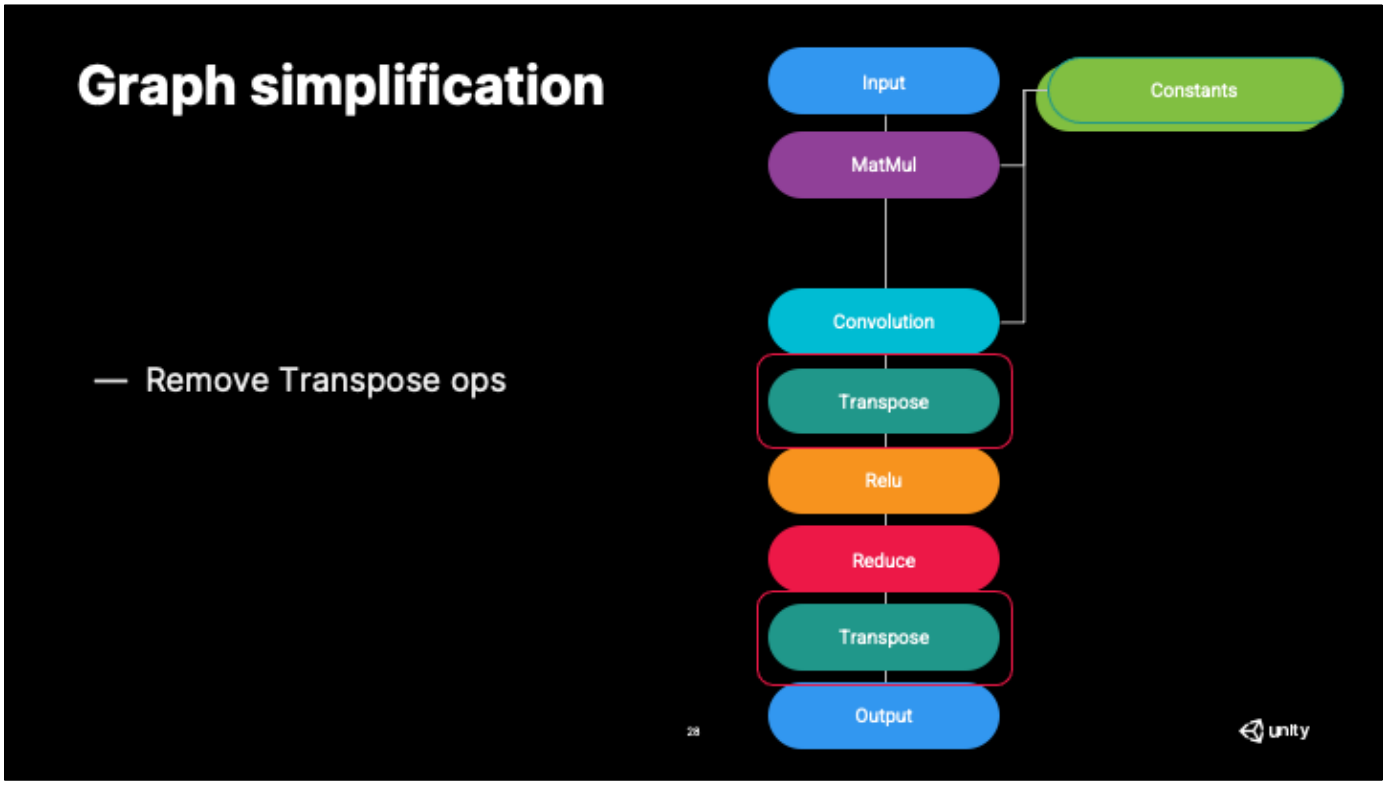
Reduce

Transpose

Output

Constants

The final goal is to be fully capped by the compute capabilities of our devices (in terms of FLOPS) however before that we also want to reduce the amount of operations we do for a given network. When possible we fused linear operations together, here the two matmul can be expressed as only one matrix multiplication.

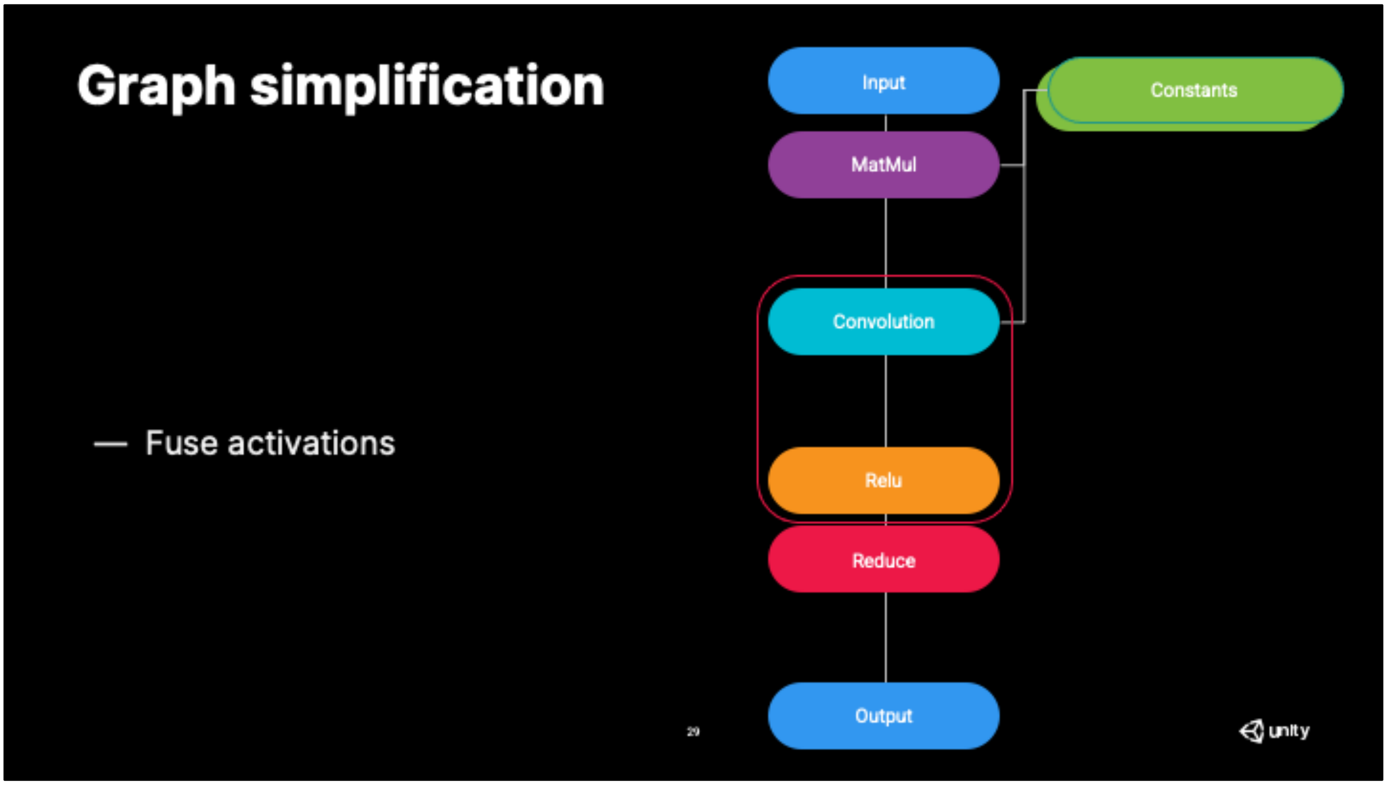


Graph simplification

— Remove Transpose ops

The various framework and NN models can be expressed in terms of various memory layout (more on this latter).
Going from one memory layout to another can lead to extra transpose being included in the model, resulting in undesired memory shuffling.

When possible we **detect those cases** and **merge or wipe the transposes** out of the model for extra performance.
In this example:
Relu is element wise and thus not impacted by the removal of the transposes.
Reduce axis(ies) will need to be updated at import time to keep the expected behavior.

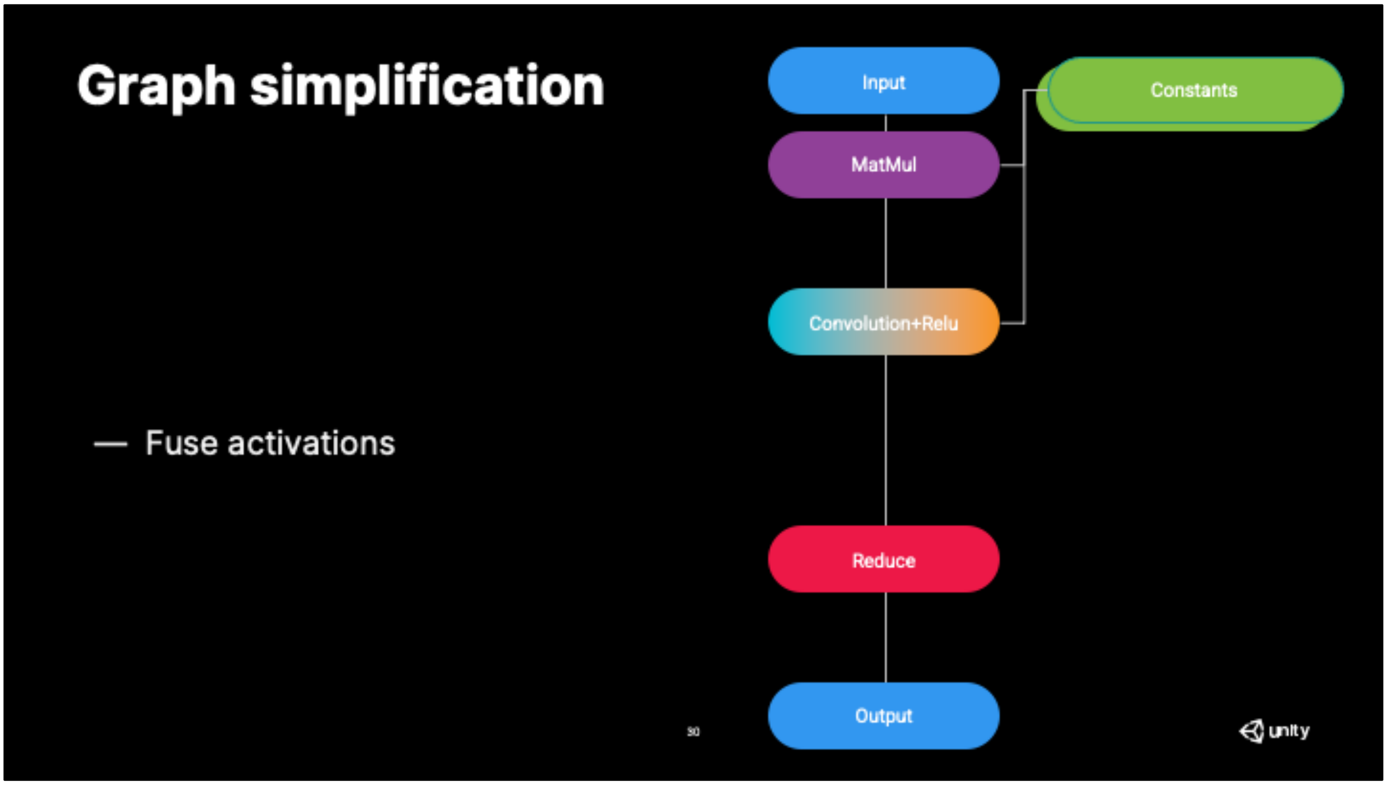


Graph simplification

— Fuse activations

In general **memory access** is way slower and power hungry than any on-chip operation, so want to avoid unnecessary load-store operations.
This includes **fusing** activations and simple linear operations such as ScaleBias when possible.

Here the Relu bandwidth is saved as the Relu will be applied in-place along the Convolution.

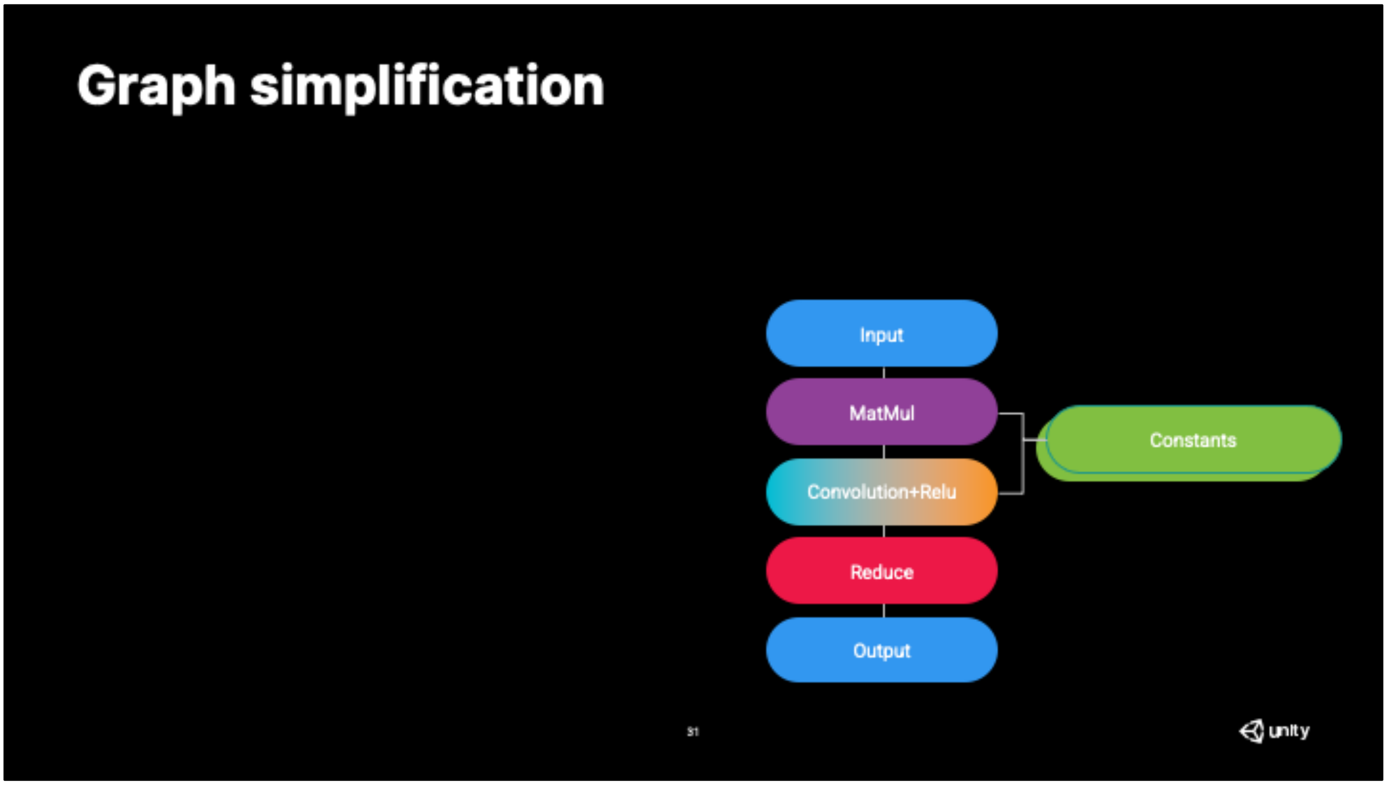


Graph simplification

— Fuse activations

50

This would be the optimized model as described by Barracuda IR.



Graph simplification

Input

MatMul

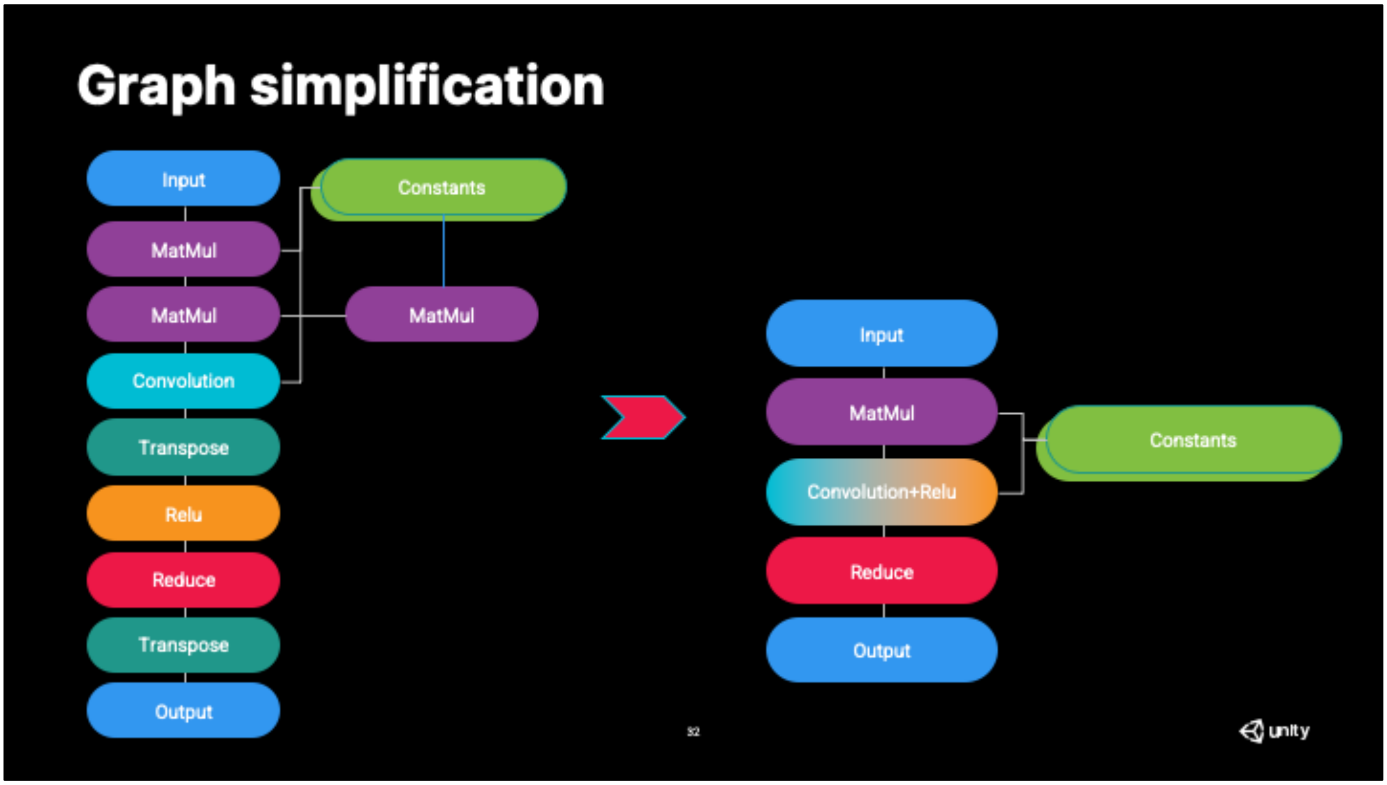
Convolution+Relu

Reduce

Output

Constants

Repacked for readability



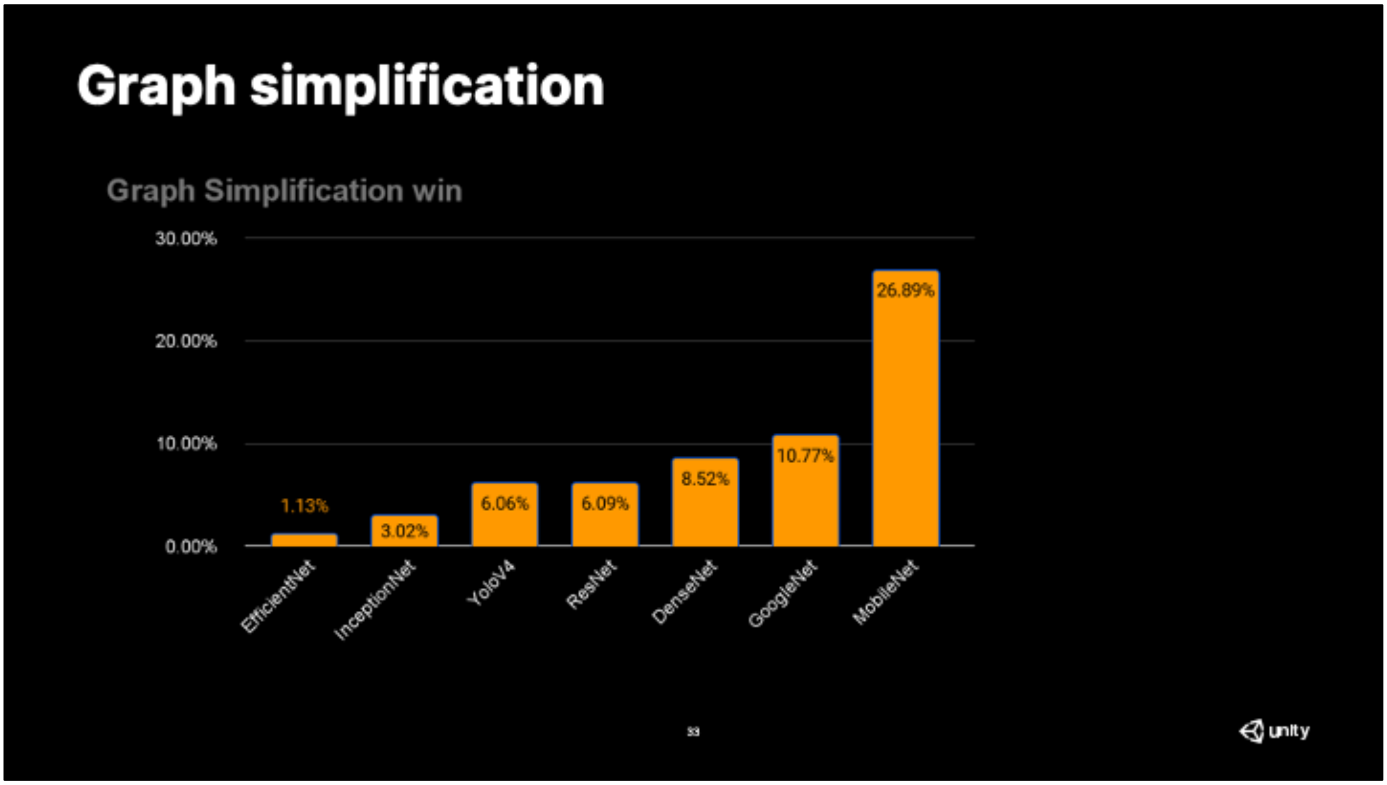
Graph simplification

This seems much better however actual gain from graph simplification vary quite a lot:

In the end if all depend on the network and use cases:

- If 95% of the time was spend in the convolution, optimized model is not gonna be much faster (max 5%).
- In practice we have seen various gain: from negligible to very good.

Let’s see some examples on popular architectures.



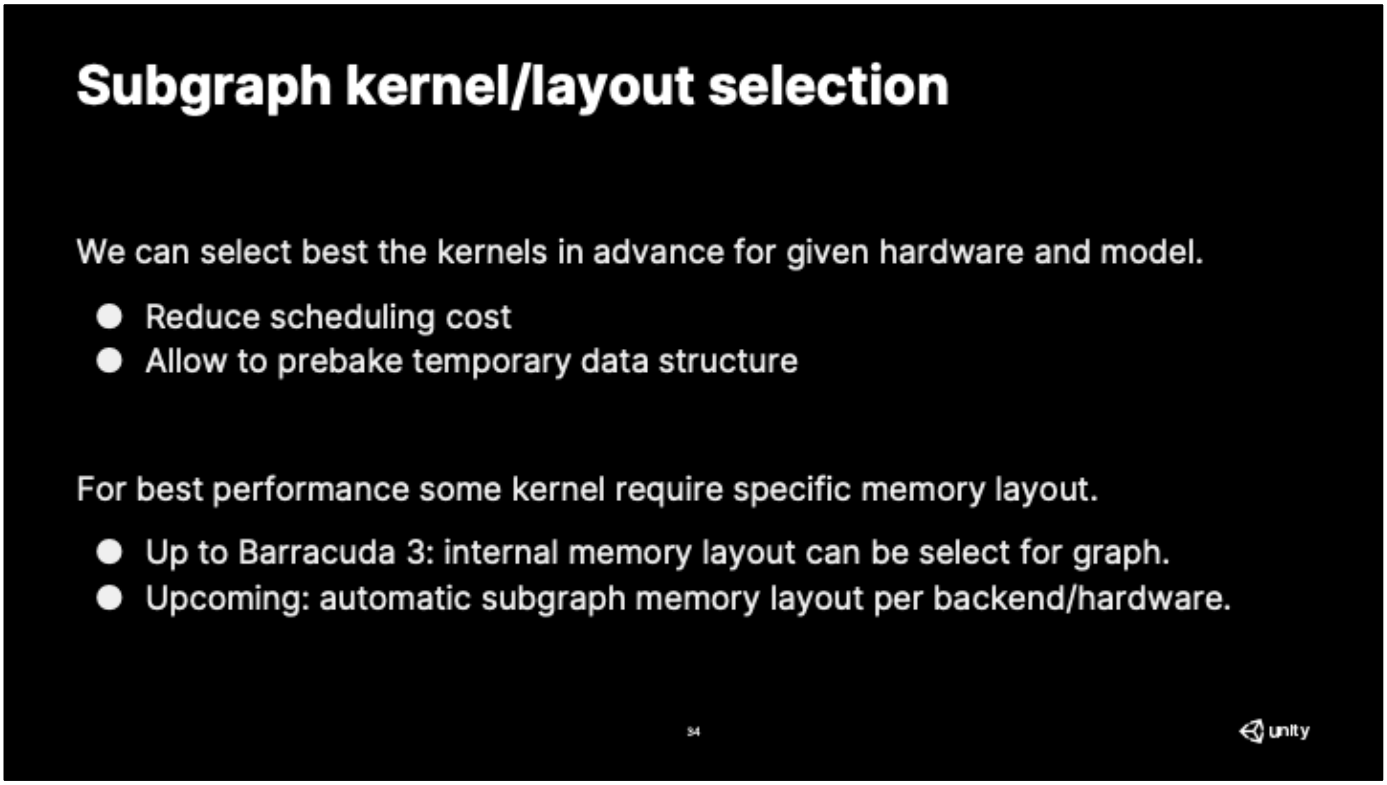
Graph simplification

From **EfficientNet around 1%**
Up to **MobileNet around 26%** gain

Exact gain will for sure depend on the backend (CPU/GPU) and the device performance characteristics.

In that regard it is interesting to note that those simplification at the graph level are especially interesting on GPU where:

- bigger workload are needed to hide memory latency.
- dispatches have an inherent cost (as GPU occupancy won’t be perfect at end of dispatches, especially on linear models).



Subgraph kernel/layout selection

We can select best the kernels in advance for given hardware and model.

- Reduce scheduling cost
- Allow to prebake temporary data structure

For best performance some kernel require specific memory layout.

- Up to Barracuda 3: internal memory layout can be select for graph.
- Upcoming: automatic subgraph memory layout per backend/hardware.

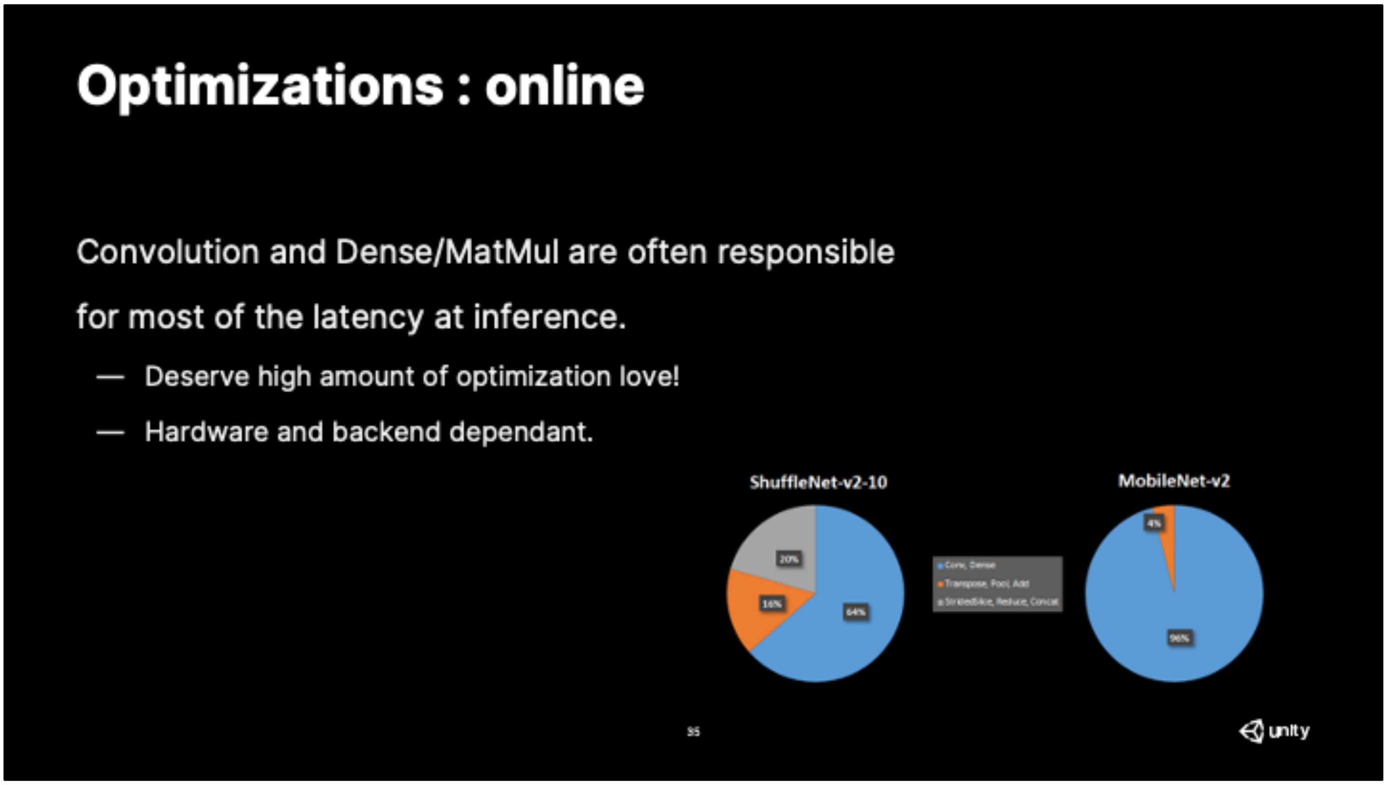
54

If one use the PrecompiledCompute backend on Barracuda:

- We select kernel by advance based on input shape, operator parameters and target hardware
- We prebake temporary data for example: updated kernel weights in the case of a winograd based convolution kernel.

Up to Barracuda 3 on GPU we offer the user with the ability to change the internal memory representation from NHWC to NCHW for performance purpose, this impact the full graph.

We are currently working on automating subgraph level optimization allowing to be more granular with those optimization and have if fully automatic based on target backend and hardware.



Optimizations : online

Convolution and Dense/MatMul are often responsible for most of the latency at inference.

- Deserve high amount of optimization love!
- Hardware and backend dependant.

Once the ***NN is real time friendly*** in term of ***architecture***, the last piece of the puzzle is online optimization

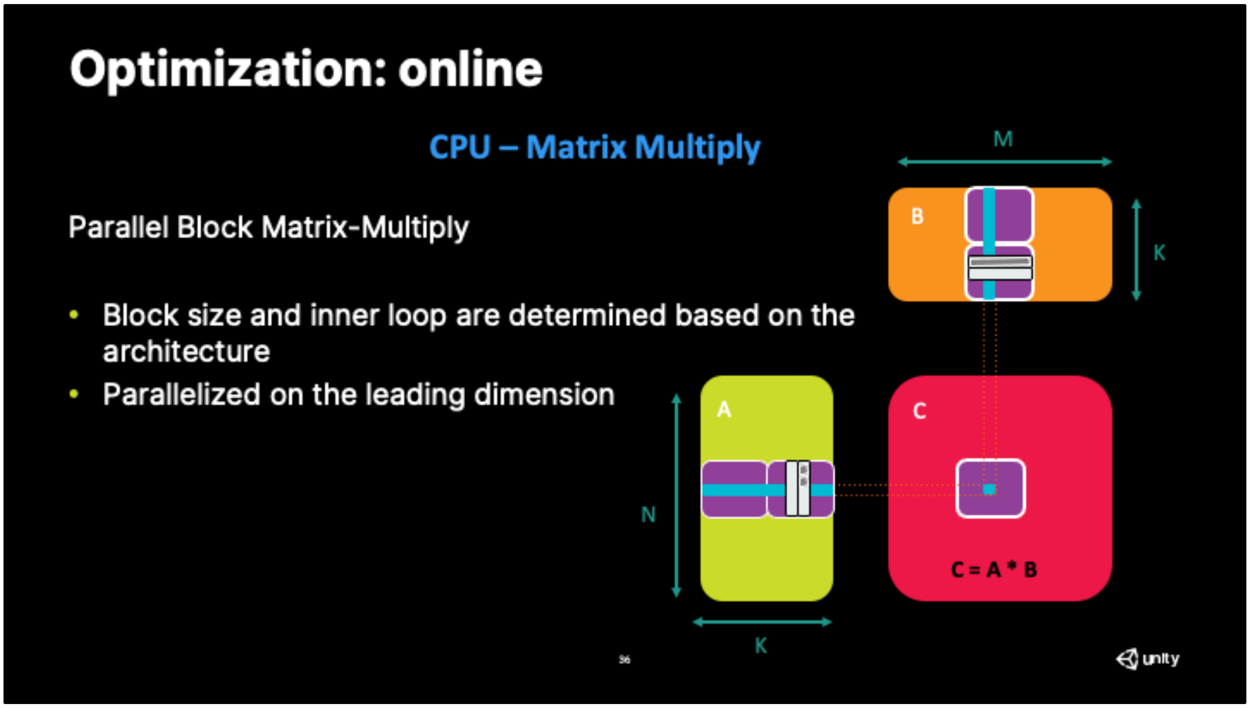
Here two examples (GPU inference)

MobileNet-v2 after offline optimization contain 68 layers
54 Convolution/DepthwiseConvolution
10 Add
2 Transpose
1 GlobalAvgPool2D
1 Reshape (noop)
=> 80% of them are convolution for 96% of total latency.

Shufflenet-v2-10 after offline optimization contain 182 layers
56 Convolution/DepthwiseConvolution
48 Transpose
26 StridedSlice
16 Concat
2 Reduce
1 MaxPool
1 Dense
32 Reshape (noop)
Dense 10%

=> 31% of them are convolution or dense for 64% of total latency.

⇒ Those optimization are however backend and hardware dependant, they thus require careful optimization.



Optimization: online

CPU – Matrix Multiply

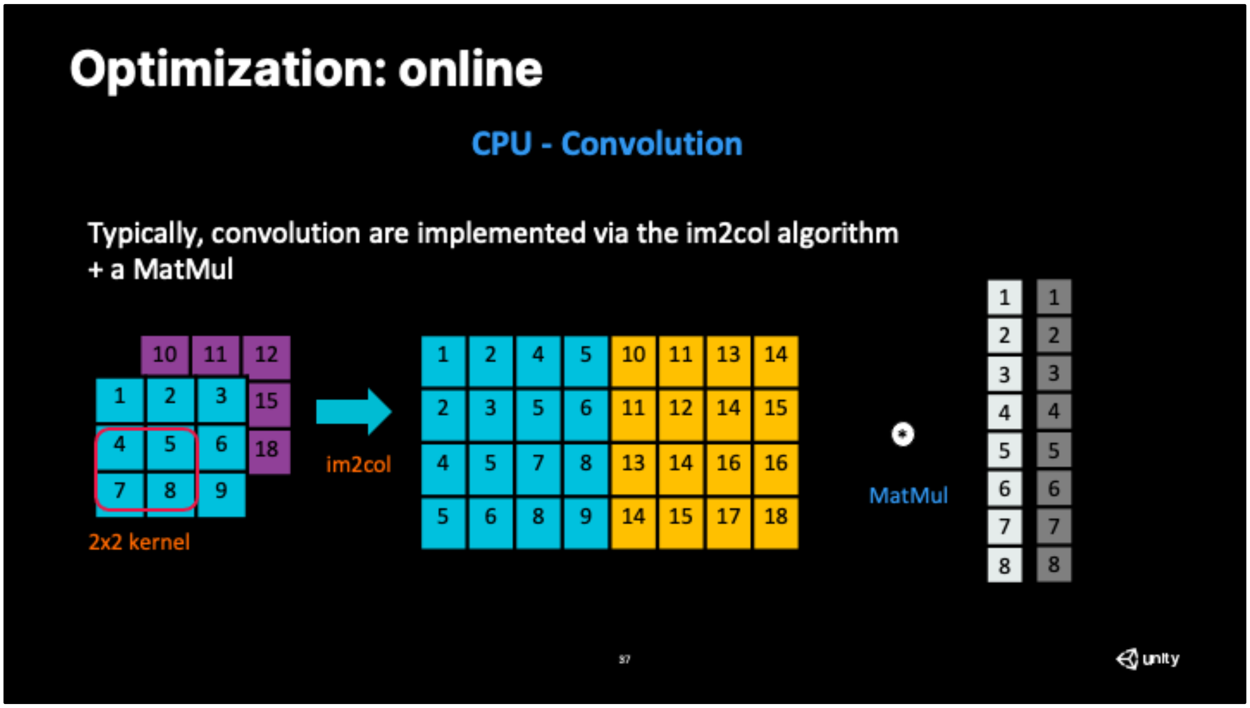
Parallel Block Matrix-Multiply

- Block size and inner loop are determined based on the architecture
- Parallelized on the leading dimension

Our CPU Matrix Multiplication is a block-wise MatMul. We follow pretty closely the work of GOTO and BLISS

One noteworthy thing is that it is parallelized on the leading dimension

Optimal block sizes is determined based on the device architecture



Optimization: online

CPU - Convolution

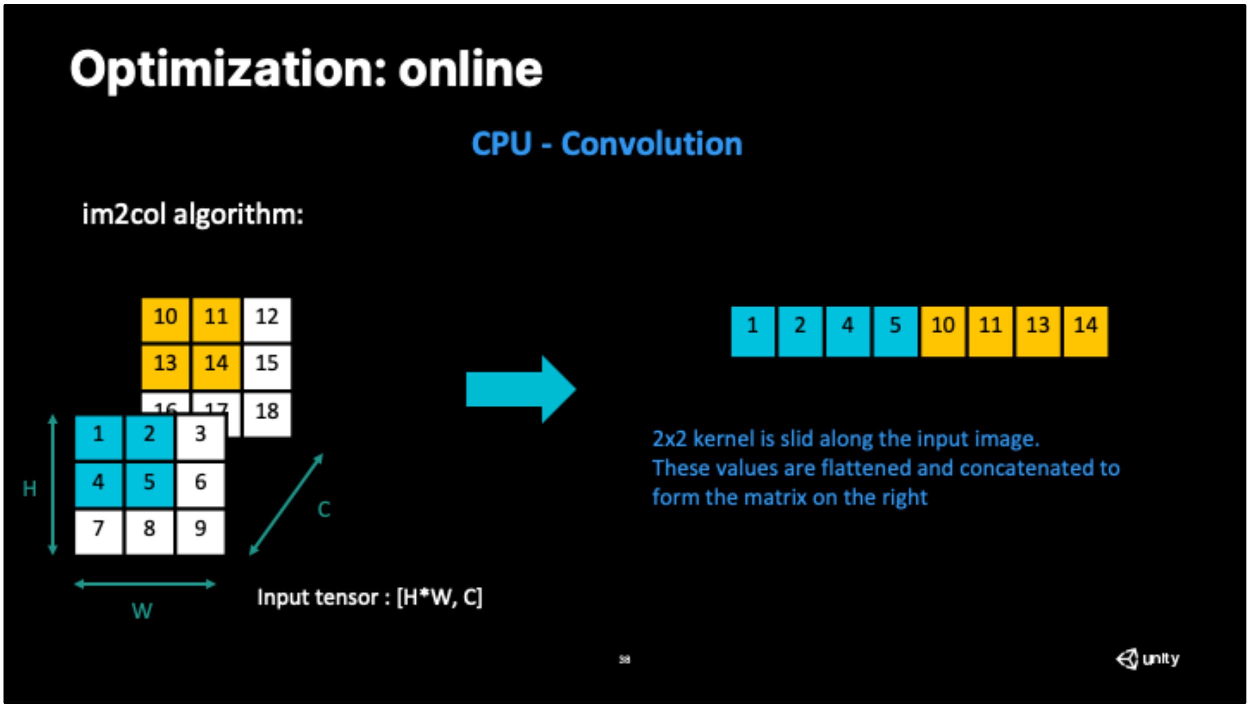
Typically, convolution are implemented via the im2col algorithm + a MatMul

1	2	4	5	10	11	13	14
2	3	5	6	11	12	14	15
4	5	7	8	13	14	16	16
5	6	8	9	14	15	17	18

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8

Convolutions are typically done via the use of the im2col algorithm followed by a MatMul
Let’s go briefly over that algorithm and the potential drawbacks

In this example we are looking at at 3x3 input with 2 channels and two output features and a 2x2 kernel



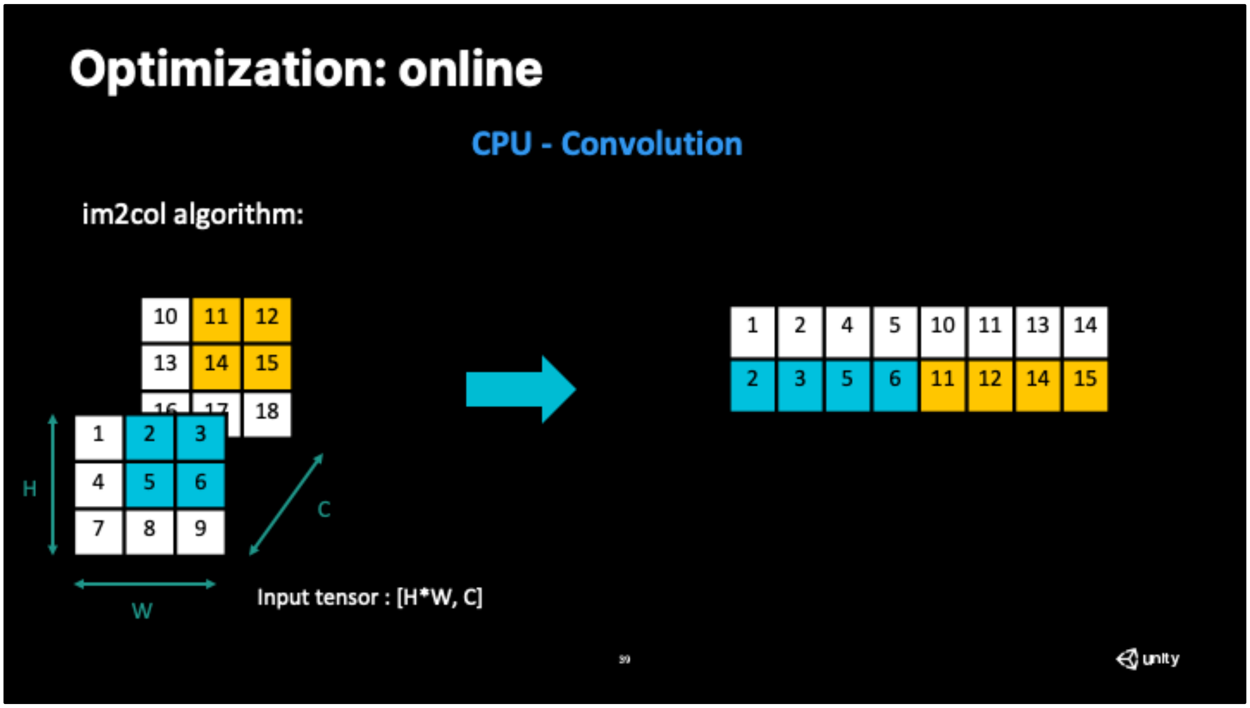
Optimization: online

CPU - Convolution

im2col algorithm:

Input tensor : [H*W, C]

2x2 kernel is slid along the input image.
These values are flattened and concatenated to form the matrix on the right



Optimization: online

CPU - Convolution

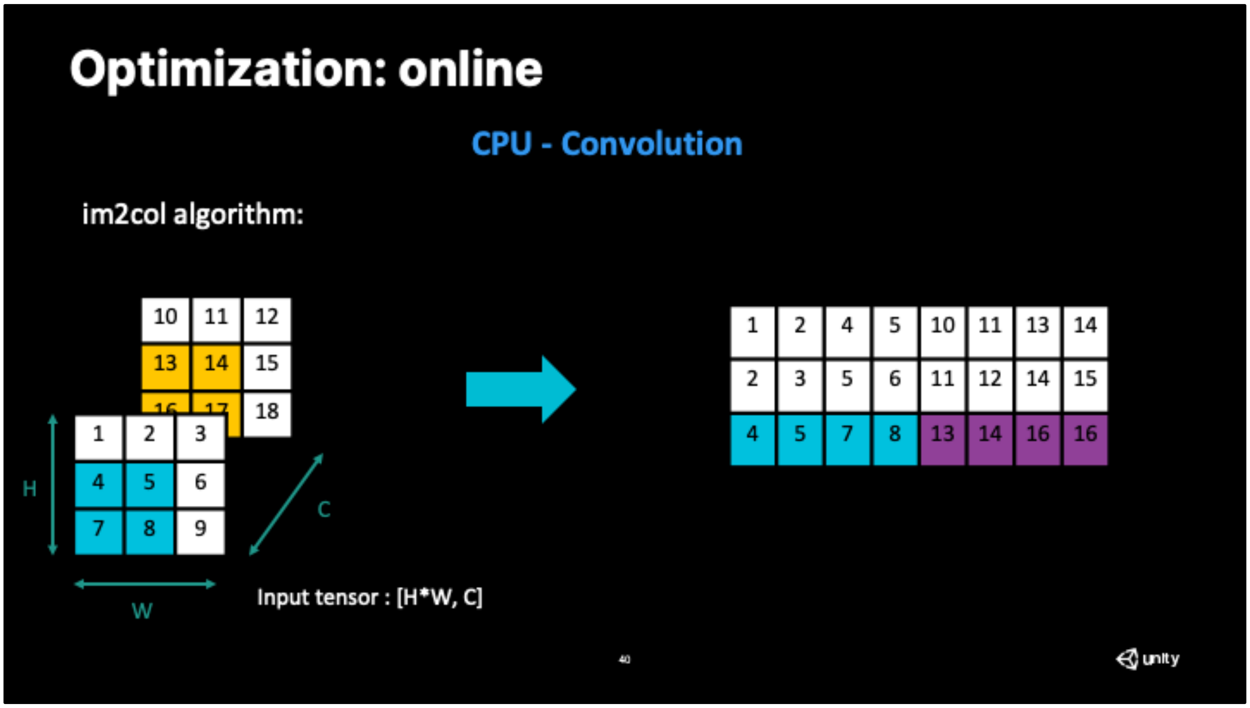
im2col algorithm:

1	2	3
4	5	6
7	8	9

10	11	12
13	14	15
16	17	18

Input tensor : [H*W, C]

1	2	4	5	10	11	13	14
2	3	5	6	11	12	14	15



Optimization: online

CPU - Convolution

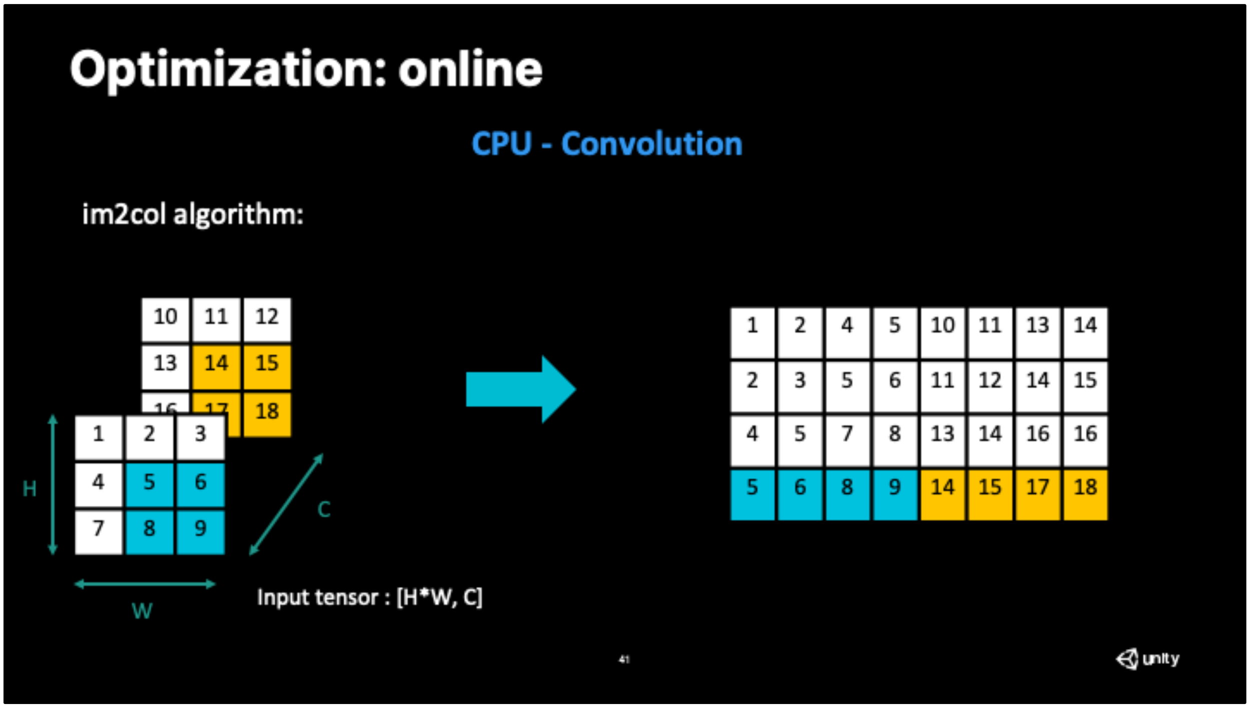
im2col algorithm:

10	11	12
13	14	15
16	17	18

1	2	3
4	5	6
7	8	9

Input tensor : [H*W, C]

1	2	4	5	10	11	13	14
2	3	5	6	11	12	14	15
4	5	7	8	13	14	16	16



Optimization: online

CPU - Convolution

im2col algorithm:

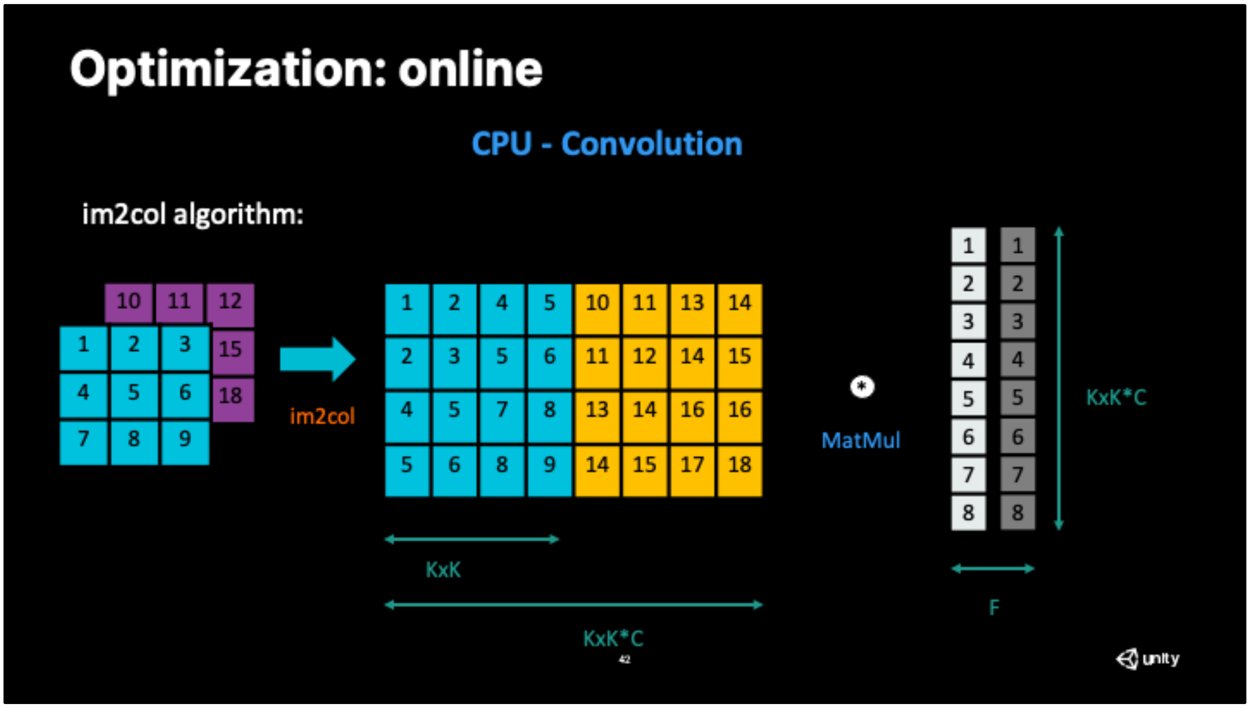
10	11	12
13	14	15
16	17	18

1	2	3
4	5	6
7	8	9

Input tensor : [H*W, C]

1	2	4	5	10	11	13	14
2	3	5	6	11	12	14	15
4	5	7	8	13	14	16	16
5	6	8	9	14	15	17	18

41



Optimization: online

CPU - Convolution

im2col algorithm:

1	2	4	5	10	11	13	14
2	3	5	6	11	12	14	15
4	5	7	8	13	14	16	16
5	6	8	9	14	15	17	18

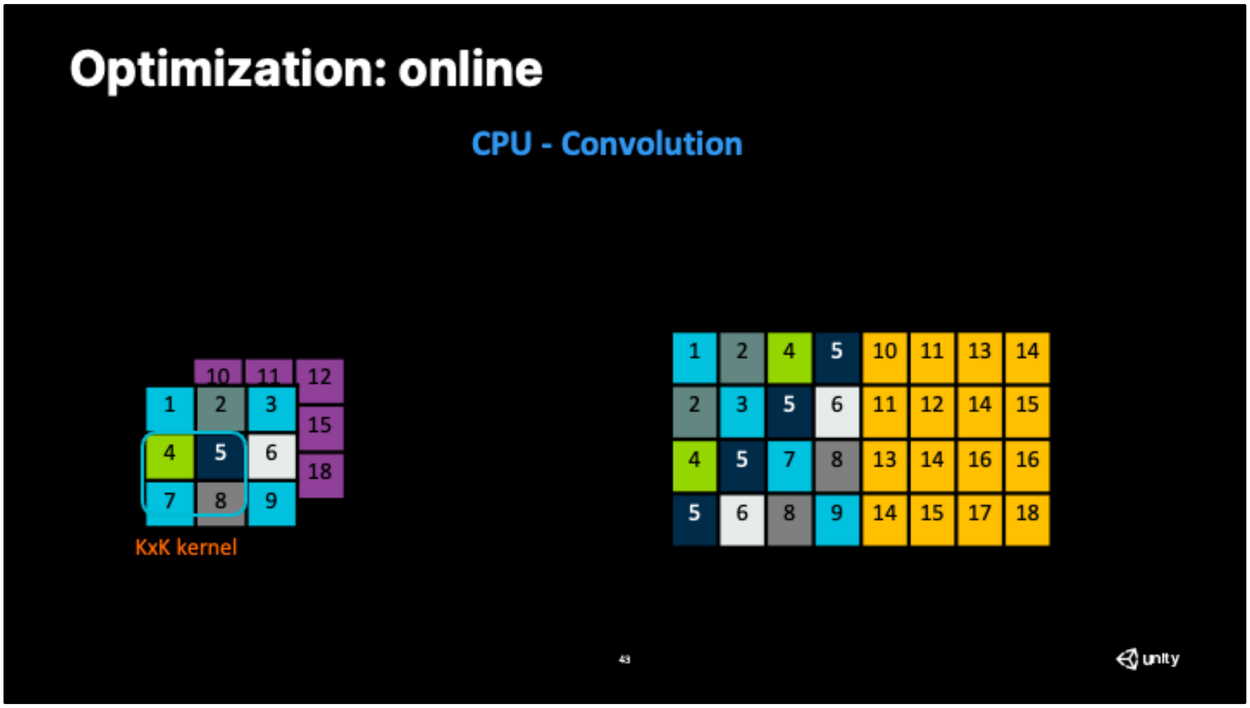
This is the result.
The 3x3x2 input is transformed into a 8x4 matrix which is multiplied by the 8x2 filter

In this case we went from :

- Input 3x3x2 => 18 floats

To :

- become 8x4 => 32 floats



Optimization: online

CPU - Convolution

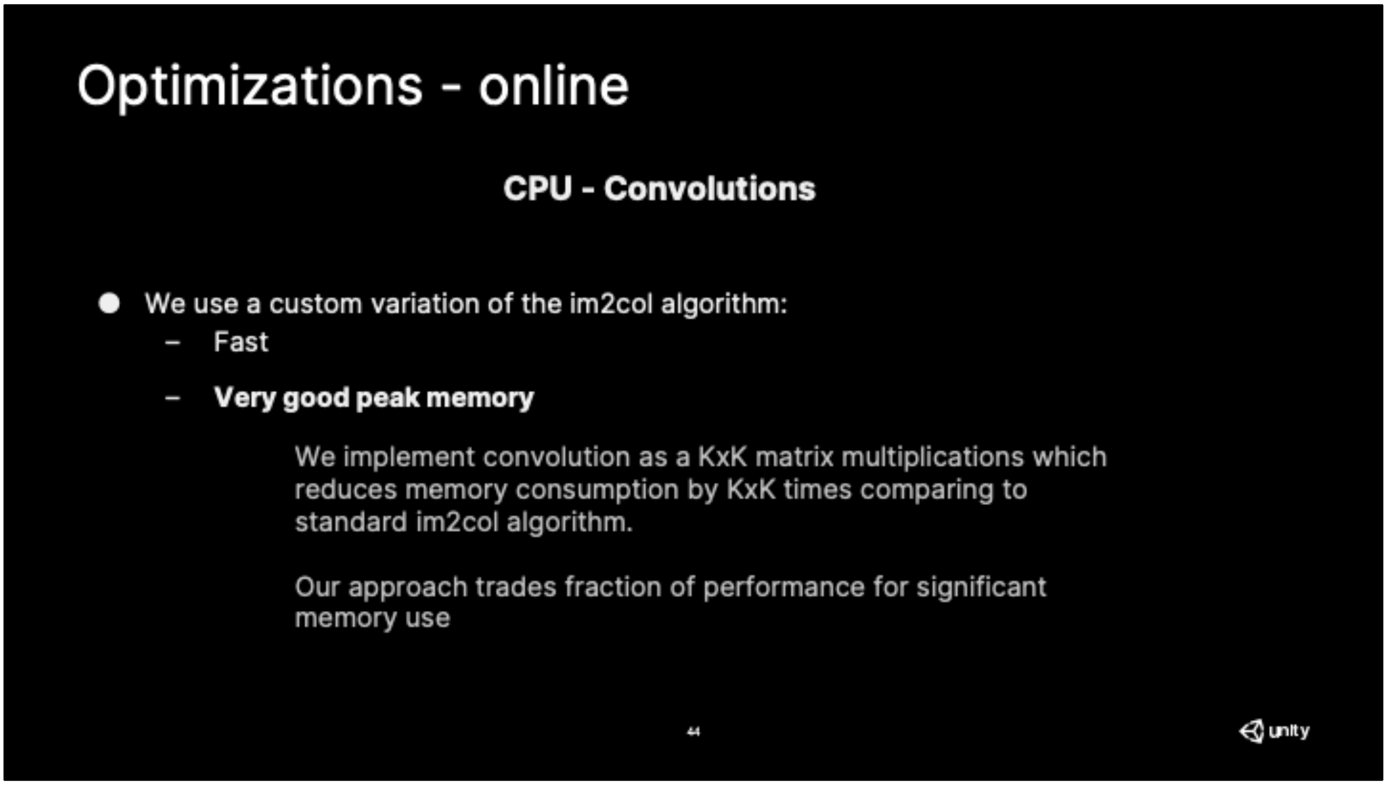
1	2	4	5	10	11	13	14
2	3	5	6	11	12	14	15
4	5	7	8	13	14	16	16
5	6	8	9	14	15	17	18

The draw back is a large memory overhead

Input size: H*W*C
im2col size : (H-K+1)*(W-K+1)*C*K*K (note: simplified considering 0 padding and stride of 1)

For 3x3x2 input, 2x2 kernels
Input size: 3*3*2 = 18
im2col size : (3-2+1)*(3-2+1)*2*2*2 = 32
→ 1.7x source data

However for a more common case 256x256x3 input, 3x3 kernels
Input size: 256*256*3 = 169608
im2col size : (256-3+1)*(256-3+1)*3*3*3 = 1741932
→ More than 10x source data



Optimizations - online

CPU - Convolutions

- We use a custom variation of the im2col algorithm:
  - Fast
  - **Very good peak memory**

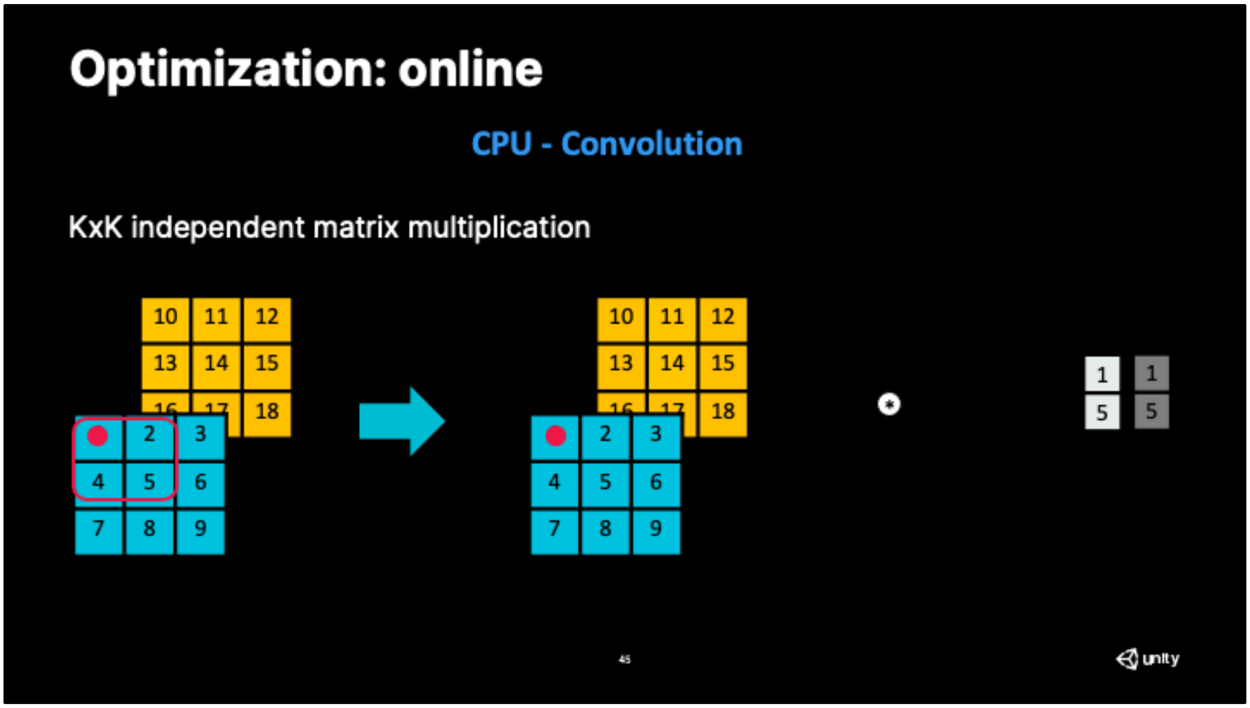
    We implement convolution as a KxK matrix multiplications which reduces memory consumption by KxK times comparing to standard im2col algorithm.

    Our approach trades fraction of performance for significant memory use

It’s not only about raw speed, in RT3D inference means memory is on a tight budget. Thus our custom variation of im2col have very good peak memory properties.

It’s public along all of Barracuda, check it out! Code is available here: https://github.com/Unity-Technologies/barracuda-release/blob/76077c0b2de7254b4f559a398a622cda072e7bf5/Barracuda/Runtime/Core/Backends/BarracudaBurstCPU.Ops.cs#L306

Let’s take a look at how it works.



Optimization: online

CPU - Convolution

KxK independent matrix multiplication

We implement convolution as a KxK independent matrix multiplication.
For each filter index, we copy a strided version of the input. We flatten it on the spatial dim and get a H*W*C Matrix which we can then multiply with the C*F filter

flops im2col : (H-K+1)*(W-K+1)*K*K * C * F

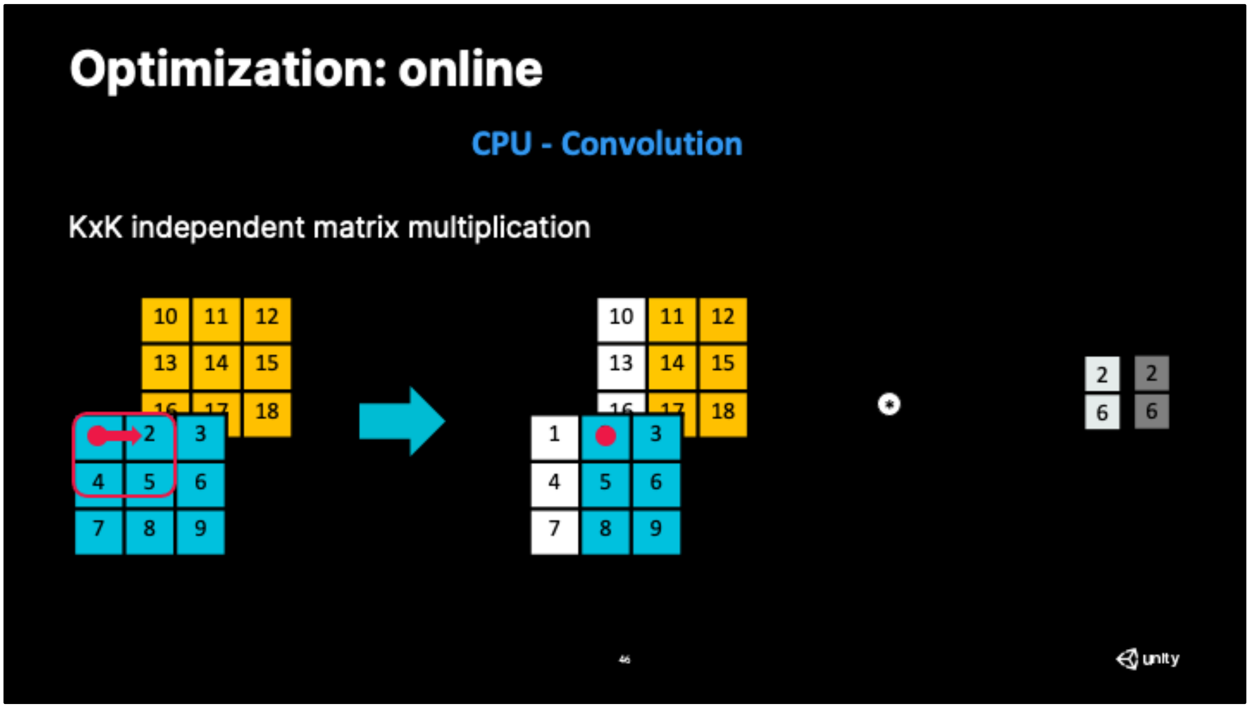
flops our implementation : (H*K * C * F) * (K*K)

flop overhead (H-K+1)*(W-K-1)/(H*W) ~ 1

Here is the pseudo code:

```
// We can solve convolution by iteratively accumulating
// matrix multiplication of X' and K' for each positon in kernel where:
//   X' is input X repeatedly shifted according to kernel position,
//   K' is slice of weights K according to kernel position.
//
// Pseudocode:
```

```
//  X :: Input

//  T :: Temporary

//  K :: Kernel

//  O :: Output

//  foreach ky in kernelHeight:

//      foreach kx in kernelWidth:

//          Temporary = shift(Input, horizontal_shift = kx,
vertical_shift = ky)

//          Temporary = pad(Temporary)

//          Temporary = stride(Temporary)

//          Output += Temporary * Kernel[dy, dx, :, :]

//

// Note for functions above that:

//  1) shift() can be implemented by copying data from n to T in a
linear fashion.

//  2) stride() can be implemented by copying data every Nth pixel in
a linear fashion.

//  3) pad() can be optimized for top and bottom of the tensor by
writing 0s across the whole row.
```

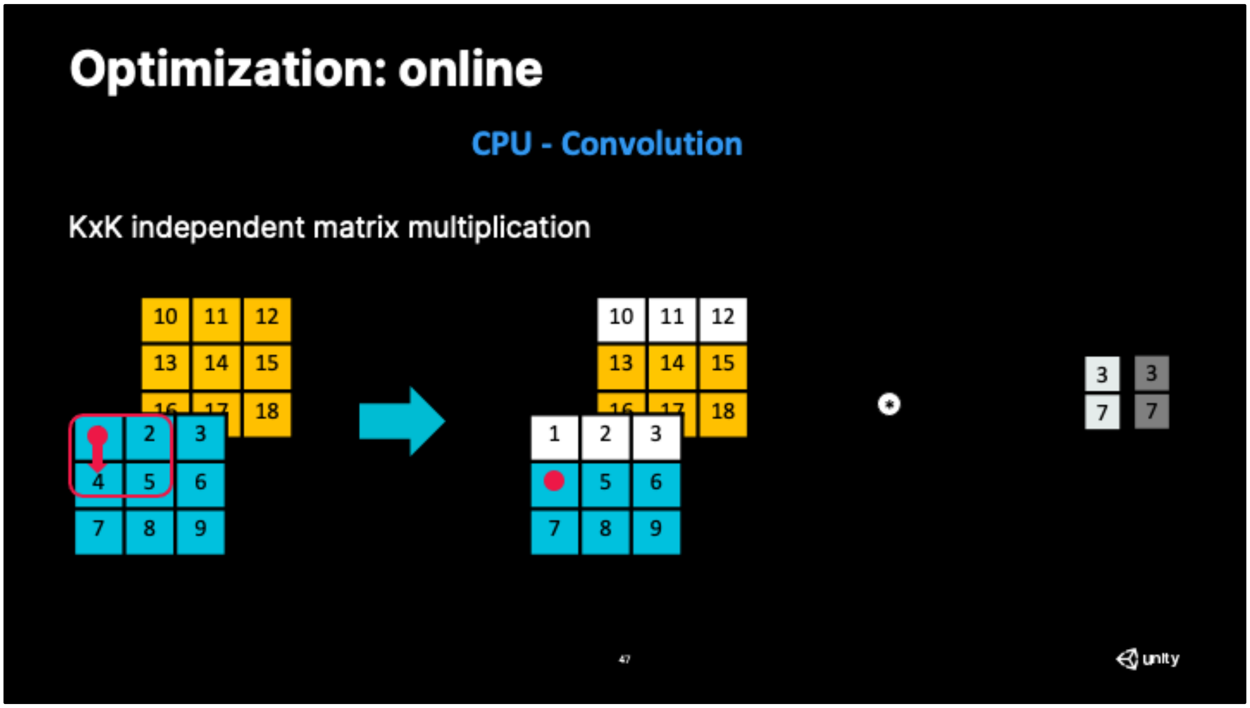


Optimization: online

CPU - Convolution

KxK independent matrix multiplication

Here you see we sample from the input strided to the left

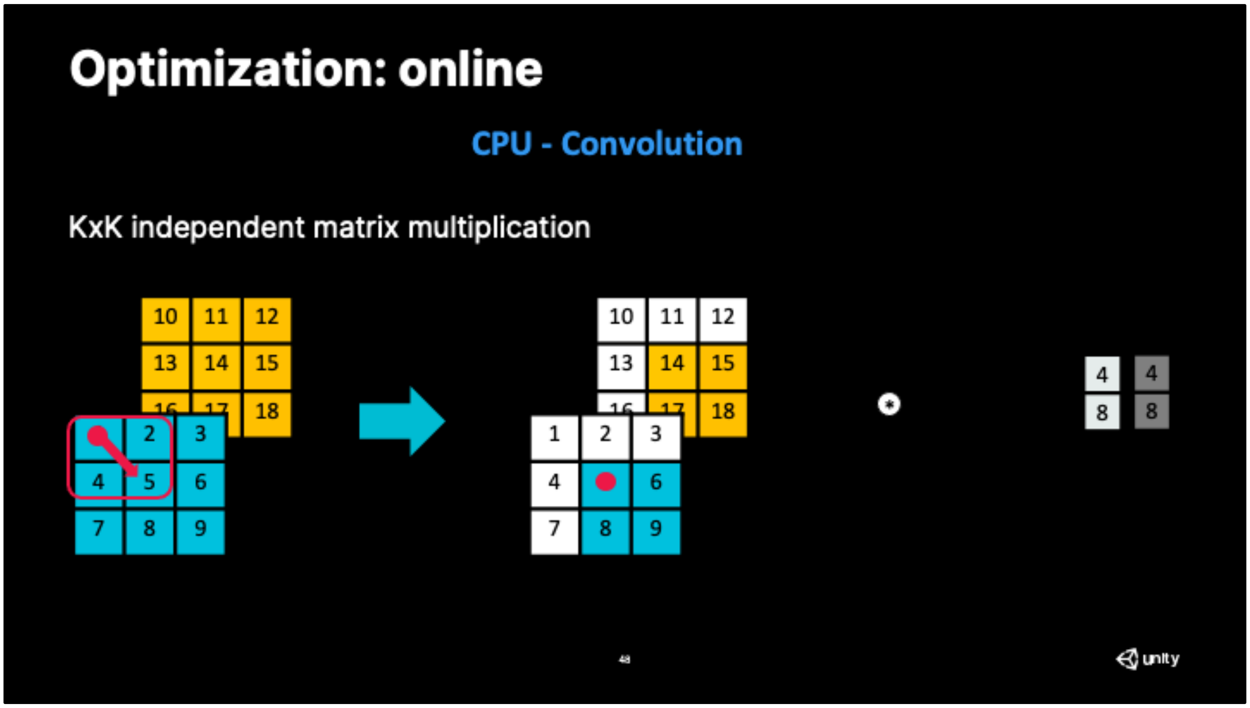


Optimization: online

CPU - Convolution

KxK independent matrix multiplication

We iterate, each time striding in the weight direction



Optimization: online

CPU - Convolution

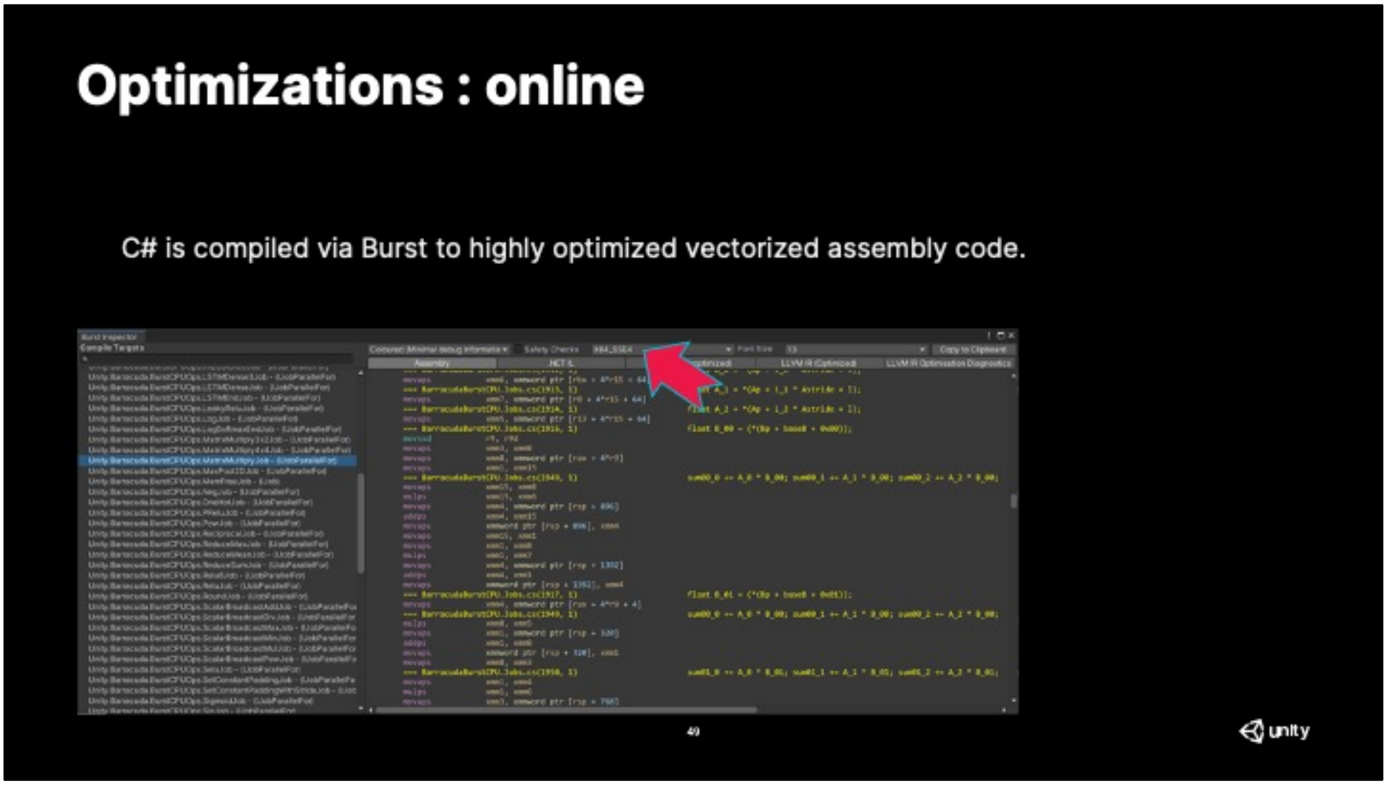
KxK independent matrix multiplication

Ect…
Finally, you can see we perform K*K matmuls

flops im2col : (H-K+1)*(W-K+1)*K*K * C * F

flops our implementation : (H*K * C * F) * (K*K)

flop overhead (H-K+1)*(W-K-1)/(H*W) ~ 1



Optimizations : online

C# is compiled via Burst to highly optimized vectorized assembly code.

Coloured (Minimal debug informatio | Safety Checks | X64_SSE4 | Font Size 13

```
Unity.Barracuda.BurstCPUOps.LSTMDense3Job - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.LSTMDenseJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.LSTMEndJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.LeakyReluJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.LogJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.LogSoftmaxEndJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.MatrixMultiply3x2Job - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.MatrixMultiply4x4Job - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.MatrixMultiplyJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.MaxPool2DJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.MemFreeJob - (IJob)
Unity.Barracuda.BurstCPUOps.NegJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.OneHotJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.PReluJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.PowJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.ReciprocalJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.ReduceMaxJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.ReduceMeanJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.ReduceSumJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.Relu6Job - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.ReluJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.RoundJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.ScalarBroadcastAddJob - (IJobParallelFo
Unity.Barracuda.BurstCPUOps.ScalarBroadcastDivJob - (IJobParallelFor
Unity.Barracuda.BurstCPUOps.ScalarBroadcastMaxJob - (IJobParallelFo
Unity.Barracuda.BurstCPUOps.ScalarBroadcastMinJob - (IJobParallelFor
Unity.Barracuda.BurstCPUOps.ScalarBroadcastMulJob - (IJobParallelFor
Unity.Barracuda.BurstCPUOps.ScalarBroadcastPowJob - (IJobParallelFo
Unity.Barracuda.BurstCPUOps.SeluJob - (IJobParallelFor)
Unity.Barracuda.BurstCPUOps.SetConstantPaddingJob - (IJobParallelFo
Unity.Barracuda.BurstCPUOps.SetConstantPaddingWithStrideJob - (IJob
Unity.Barracuda.BurstCPUOps.SigmoidJob - (IJobParallelFor)
```

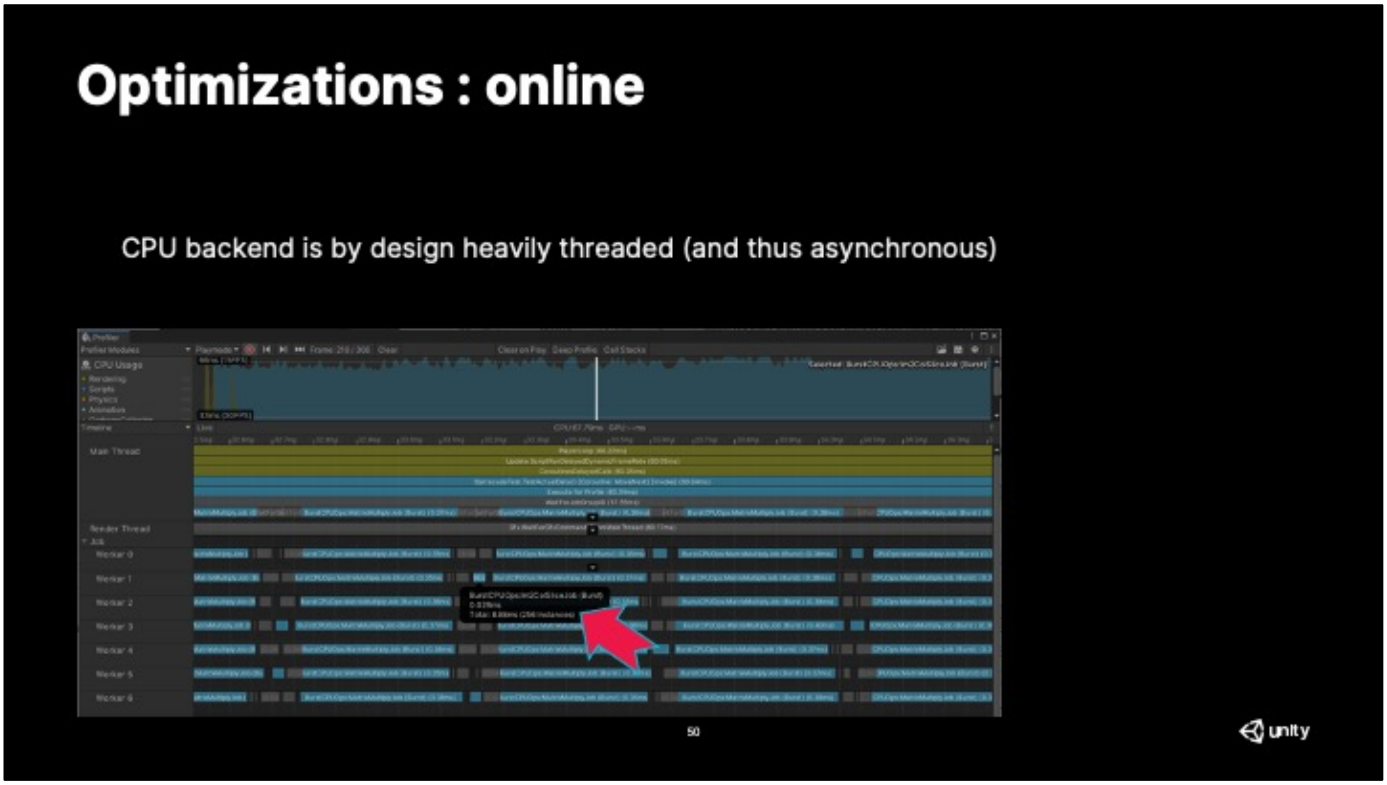
```
        movups          xmm6, xmmword ptr [rbx + 4*r15 + 64]
        === BarracudaBurstCPU.Jobs.cs(1915, 1)          float A_1 = *(Ap + I_1 * Astride + I);
        movups          xmm7, xmmword ptr [r8 + 4*r15 + 64]
        === BarracudaBurstCPU.Jobs.cs(1914, 1)          float A_2 = *(Ap + I_2 * Astride + I);
        movups          xmm5, xmmword ptr [r13 + 4*r15 + 64]
        === BarracudaBurstCPU.Jobs.cs(1916, 1)          float B_00 = (*(Bp + base0 + 0x00));
        movsxd          r9, r9d
        movaps          xmm1, xmm8
        movups          xmm8, xmmword ptr [rax + 4*r9]
        movaps          xmm1, xmm15
        === BarracudaBurstCPU.Jobs.cs(1949, 1)          sum00_0 += A_0 * B_00; sum00_1 += A_1 * B_00; sum00_2 += A_2 * B_00;
        movaps          xmm15, xmm8
        mulps           xmm15, xmm6
        movaps          xmm4, xmmword ptr [rsp + 896]
        addps           xmm4, xmm15
        movaps          xmmword ptr [rsp + 896], xmm4
        movaps          xmm15, xmm1
        movaps          xmm1, xmm8
        mulps           xmm1, xmm7
        movaps          xmm4, xmmword ptr [rsp + 1392]
        addps           xmm4, xmm1
        movaps          xmmword ptr [rsp + 1392], xmm4
        === BarracudaBurstCPU.Jobs.cs(1917, 1)          float B_01 = (*(Bp + base0 + 0x01));
        movups          xmm4, xmmword ptr [rax + 4*r9 + 4]
        === BarracudaBurstCPU.Jobs.cs(1949, 1)          sum00_0 += A_0 * B_00; sum00_1 += A_1 * B_00; sum00_2 += A_2 * B_00;
        mulps           xmm8, xmm5
        movaps          xmm1, xmmword ptr [rsp + 320]
        addps           xmm1, xmm8
        movaps          xmmword ptr [rsp + 320], xmm1
        movaps          xmm8, xmm3
        === BarracudaBurstCPU.Jobs.cs(1950, 1)          sum01_0 += A_0 * B_01; sum01_1 += A_1 * B_01; sum01_2 += A_2 * B_01;
        movaps          xmm1, xmm4
        mulps           xmm1, xmm6
        movaps          xmm3, xmmword ptr [rsp + 768]
```

49

On CPU we use Burst (see https://docs.unity3d.com/Packages/com.unity.burst@1.7/manual/index.html)

It is a compiler that translates from a subset of C# to highly optimized native code on all platform that Unity support using LLVM. It is released as a unity package.

Here you can see assembly for the MatrixMultiply job compiled for Intel X64 SSE4.



Optimizations : online

CPU backend is by design heavily threaded (and thus asynchronous)

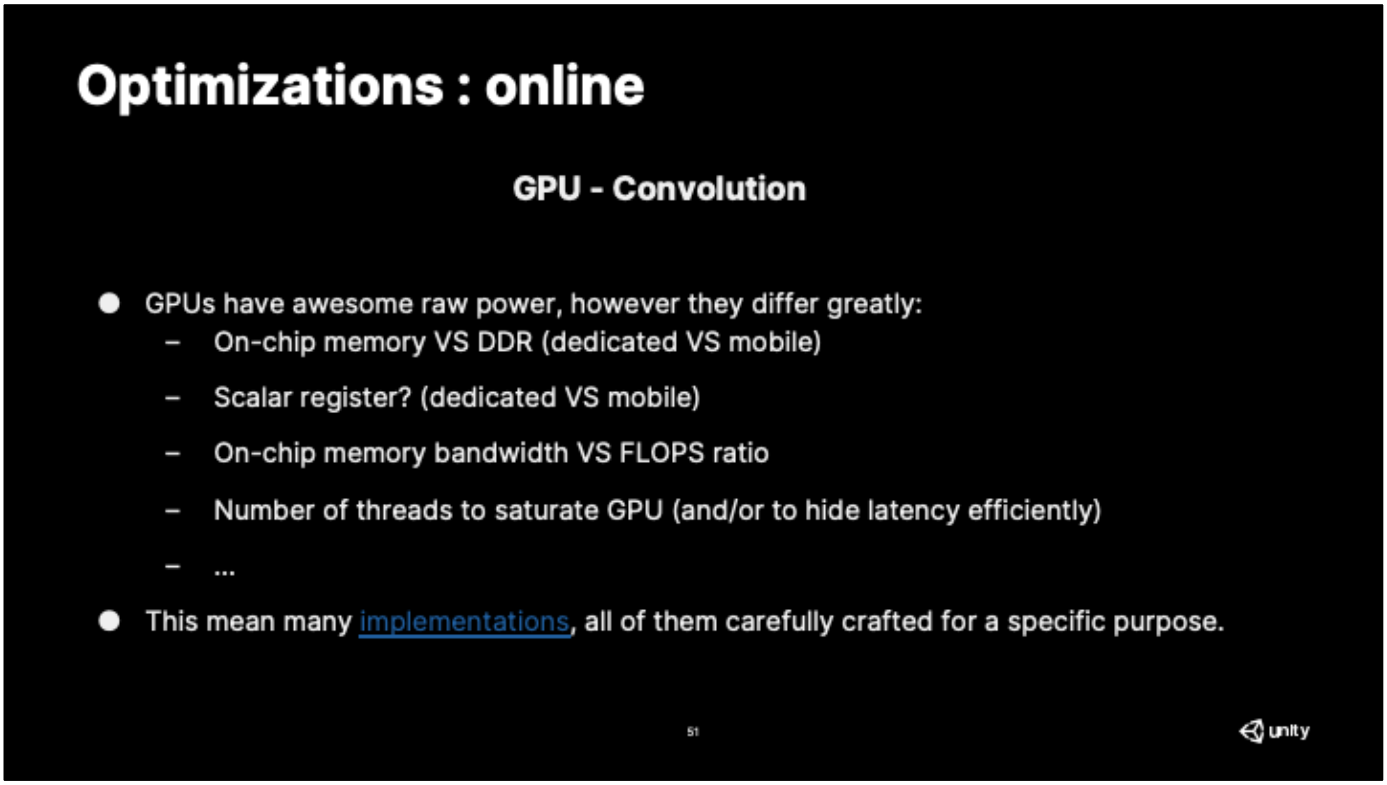
50

On top of Burst native code work is jobified:

Here you can see the im2col job spawning many MatMul jobs, creating a job dependency chain for a given model.

⇒ Despite all of these optimization for heavy duty model CPU inference latency might not be enough for use case, this is where GPU backend come into play.

NOTE: Also if your device does not have unified memory if might be interesting to run on GPU if input is a texture or rendertarget.



Optimizations : online

GPU - Convolution

- GPUs have awesome raw power, however they differ greatly:
  - On-chip memory VS DDR (dedicated VS mobile)
  - Scalar register? (dedicated VS mobile)
  - On-chip memory bandwidth VS FLOPS ratio
  - Number of threads to saturate GPU (and/or to hide latency efficiently)
  - ...
- This mean many implementations, all of them carefully crafted for a specific purpose.

- On-chip memory VS DDR (desktop VS mobile)

  → Group shared memory might get dedicated hardware on mobile. Depending on the case it might then not be a good idea to use this feature as it is might be backed by DDR and could rather trash the cache.

  → Also when using on-ship memory on dedicated GPU you will want have custom memory access pattern to avoid bank conflict, those pattern will likely require indexing math + could trash regular DDR cash even more if there is not dedicated shared memory.

  → The amount of on-ship memory is limited in size, influencing the choice of possible algorithms. For example our current winograd implementations are used up to 3x3 for spatial kernels.

- On-chip memory VS DDR (desktop VS mobile)

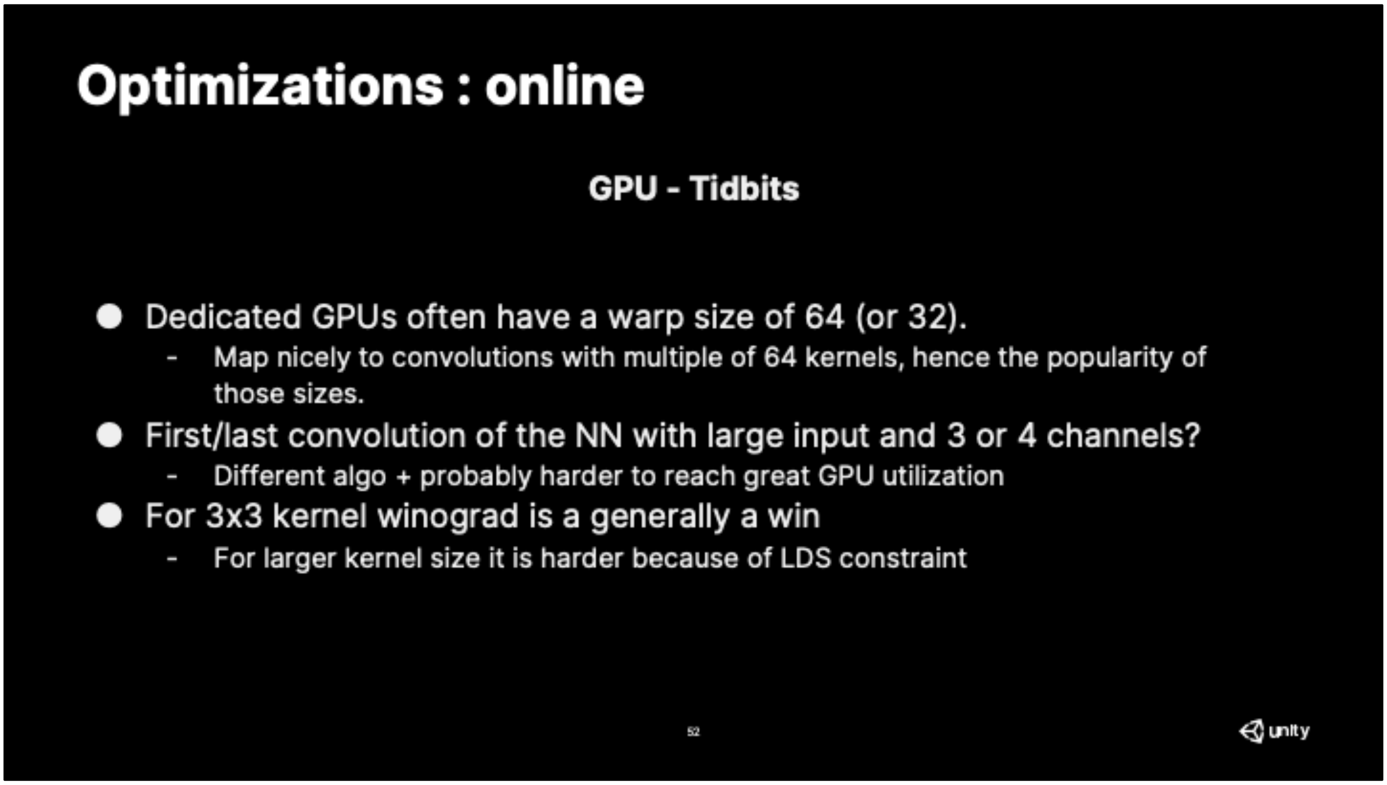
  → Scalar registers are a huge help on dedicated GPU to help with register pressure and occupancy however on mobile devices they will probably not exist.

- On-chip memory bandwidth VS FLOPS ratio

  → When designing your convolution algorithm you ideally want to align the bandwidth from your memory (on-ship if possible) with your inner loop in term of flops. AMD hardware usually have a ratio of 2 ALU per float of bandwidth, while NVidia have 4. This typically change the number of register you need to work with in the inner loop of convolution from 16 to 64.

- Number of threads to saturate GPU (and/or to hide latency efficiently)
  → Depending on your GPU the amount of hardware thread can vary from very few to thousands of threads! You thus need to design an algorithm that can go wide enough considering your kernel size, your input shape and your target hardware.

This mean a lot of convolution implementation, to cover the various use cases out there from the many models. Code is here https://github.com/Unity-Technologies/barracuda-release/tree/76077c0b2de7254b4f559a398a622cda072e7bf5/Barracuda/Runtime/Core/Resources/Barracuda look for Conv*.compute.



Optimizations : online

GPU - Tidbits

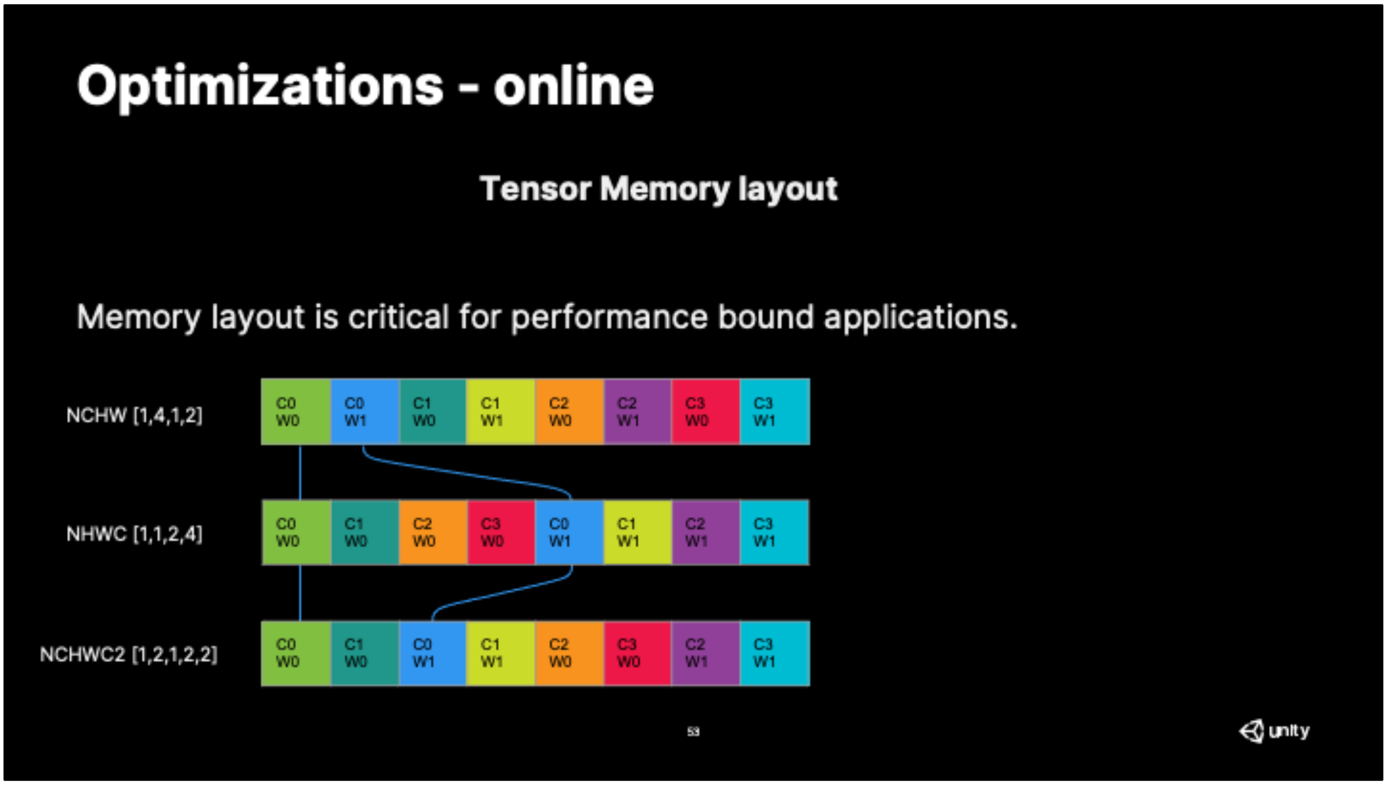
- Dedicated GPUs often have a warp size of 64 (or 32).
  - Map nicely to convolutions with multiple of 64 kernels, hence the popularity of those sizes.
- First/last convolution of the NN with large input and 3 or 4 channels?
  - Different algo + probably harder to reach great GPU utilization
- For 3x3 kernel winograd is a generally a win
  - For larger kernel size it is harder because of LDS constraint

- Multiple of 64 kernels map nicely to desktop hardware and console, however for real time inference one might rather want a narrower network, requiring different tradeoffs for the algorithm.

- The first and last convolution of a network can often map to texture meaning 3 or 4 channels, sometimes at very high resolution, again a different algorithm, can be thought of.

- A note here: to avoid too much details in slide i have mixed up kernel and channel count, however having a low channel count vs a low kernel count is actually a different constraint for the algorithm, in the case of Barracuda we are a more flexible in term of channel than kernels count with current implementations.
  - That’s where understanding the hardware and the associated algorithms/kernel implementation start to be important to design fast model! A great example are the recently presented fully fused NN, a beauty!

→ **Going forward we expect and hope to see more and more successful models that were designed in consideration or in conjunction with the hardware.**



Optimizations - online

Tensor Memory layout

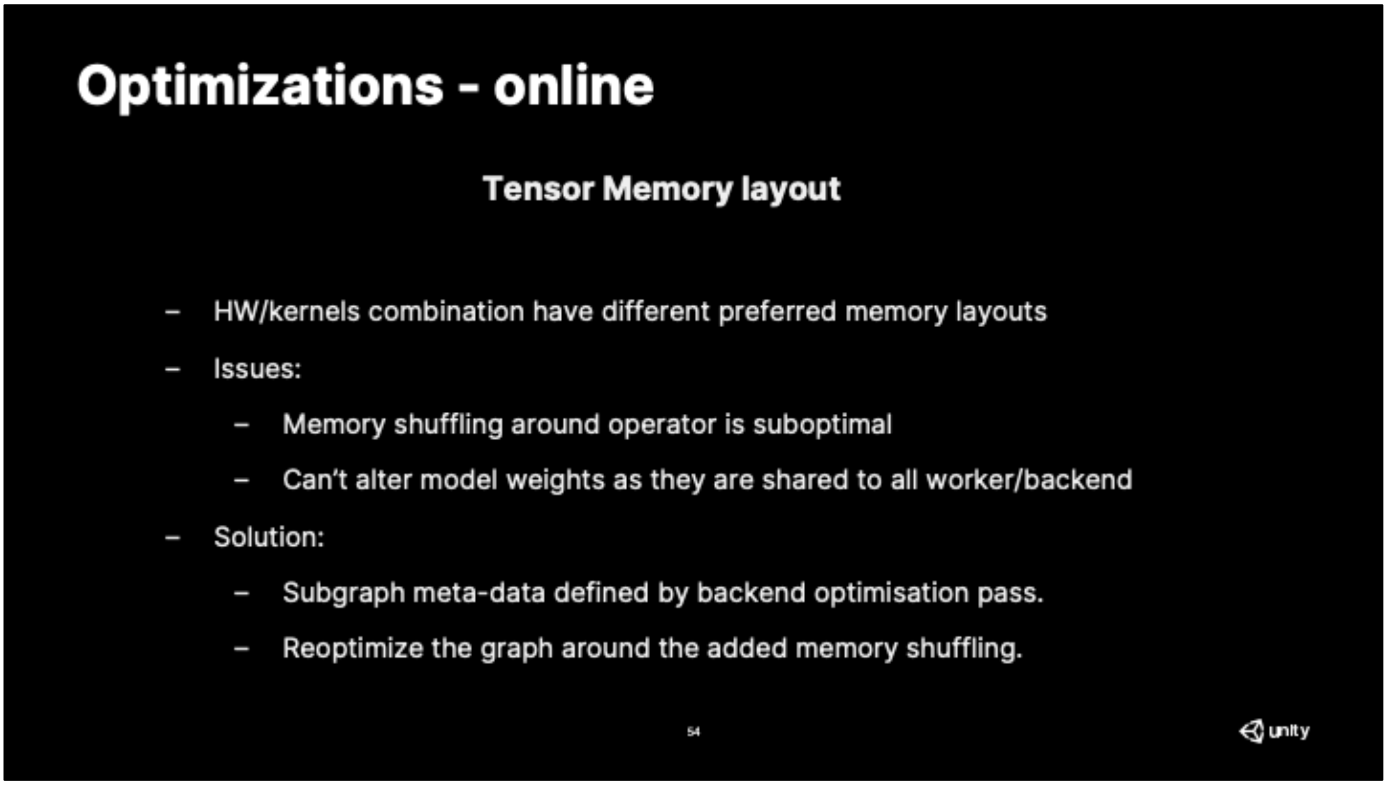
Memory layout is critical for performance bound applications.

NCHW [1,4,1,2]	C0 W0	C0 W1	C1 W0	C1 W1	C2 W0	C2 W1	C3 W0	C3 W1
NHWC [1,1,2,4]	C0 W0	C1 W0	C2 W0	C3 W0	C0 W1	C1 W1	C2 W1	C3 W1
NCHWC2 [1,2,1,2,2]	C0 W0	C1 W0	C0 W1	C1 W1	C2 W0	C3 W0	C2 W1	C3 W1

In general performance sensitive application are sensible to memory access pattern, as memory is slow compared to processing power.

The problem here is that for kernel implementations (themselves targeting a given hardware) will need a favored memory layout to be as fast as possible:

- For example in our tests NCHW is advantageous on dedicated GPU with on ship-memory compare to NHWC.
    - This is especially true at lower channel/kernel count, which you might require for real time performance reasons
- On the other hand GPU with TensorCore or API with NPU support such as NNAPI might prefer (or maybe only support!) channel last.
- On CPU you will likely prefer NCHWC8 to get the best of SIMD hardware.
- On GPU if using texture as input NCHWC4 could be interesting to leverage the texture cache (especially on mobile).



Optimizations - online

Tensor Memory layout

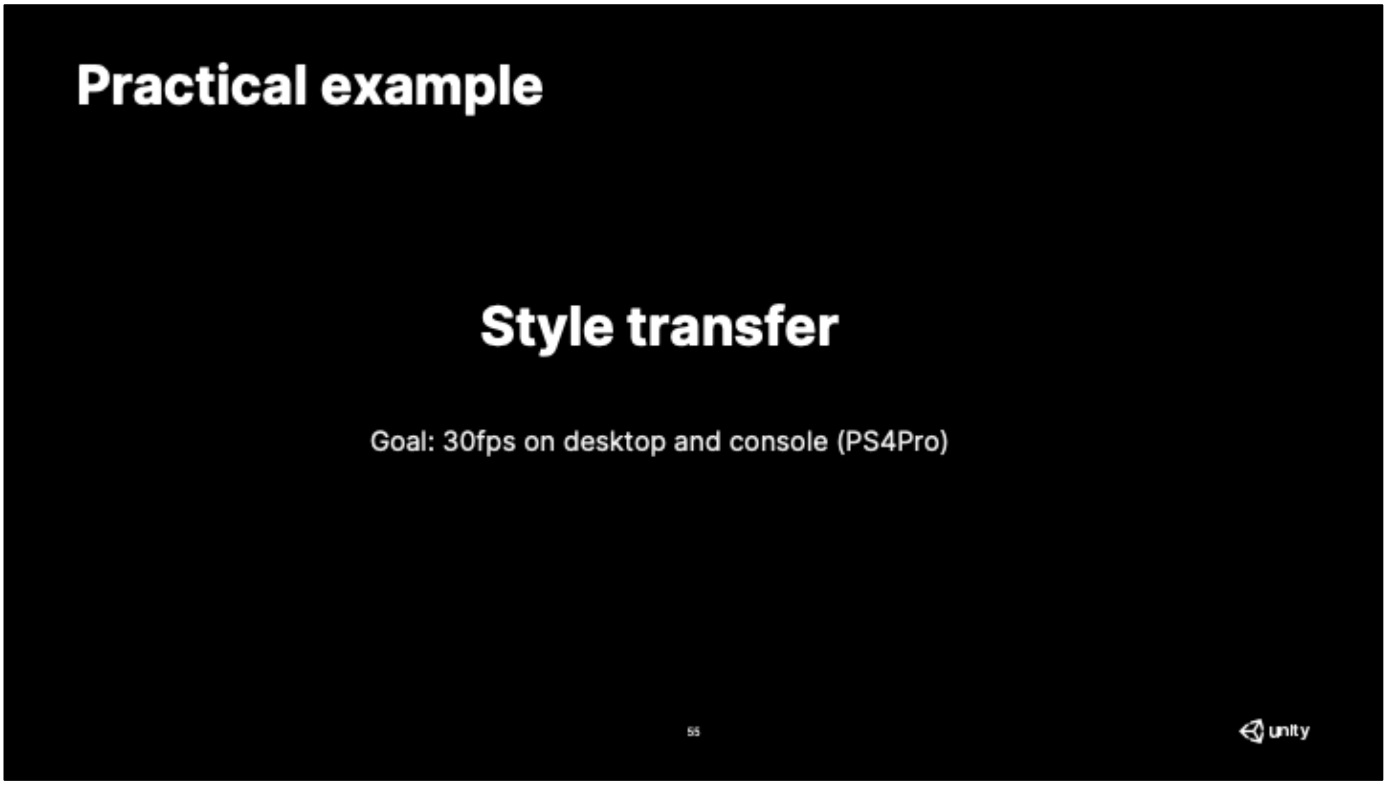
- HW/kernels combination have different preferred memory layouts
- Issues:
  - Memory shuffling around operator is suboptimal
  - Can’t alter model weights as they are shared to all worker/backend
- Solution:
  - Subgraph meta-data defined by backend optimisation pass.
  - Reoptimize the graph around the added memory shuffling.

So we have seen the memory layout plays a crucial role in achieving the best performance.

Thus the ideal would be to design a model from the ground up with target hardware in mind. However what if you target multiple platform or backend. Also the NN library out there each have their own choice of memory layout and in any case ONNX at the moment express the graph with convolution as NCHW.

So we want to select the best memory with the knowledge of the target hardware/backend and used operators, we thus alter/enrich the graph with meta-data to do so, adding memory shuffling to respect the graph initial behavior.

However those extra memory shuffling have a cost adding them around each critical operators is suboptimal, the idea is thus to reoptimize the graph moving/grouping and ideally removing those memory layout as much as possible at the subgraph level.

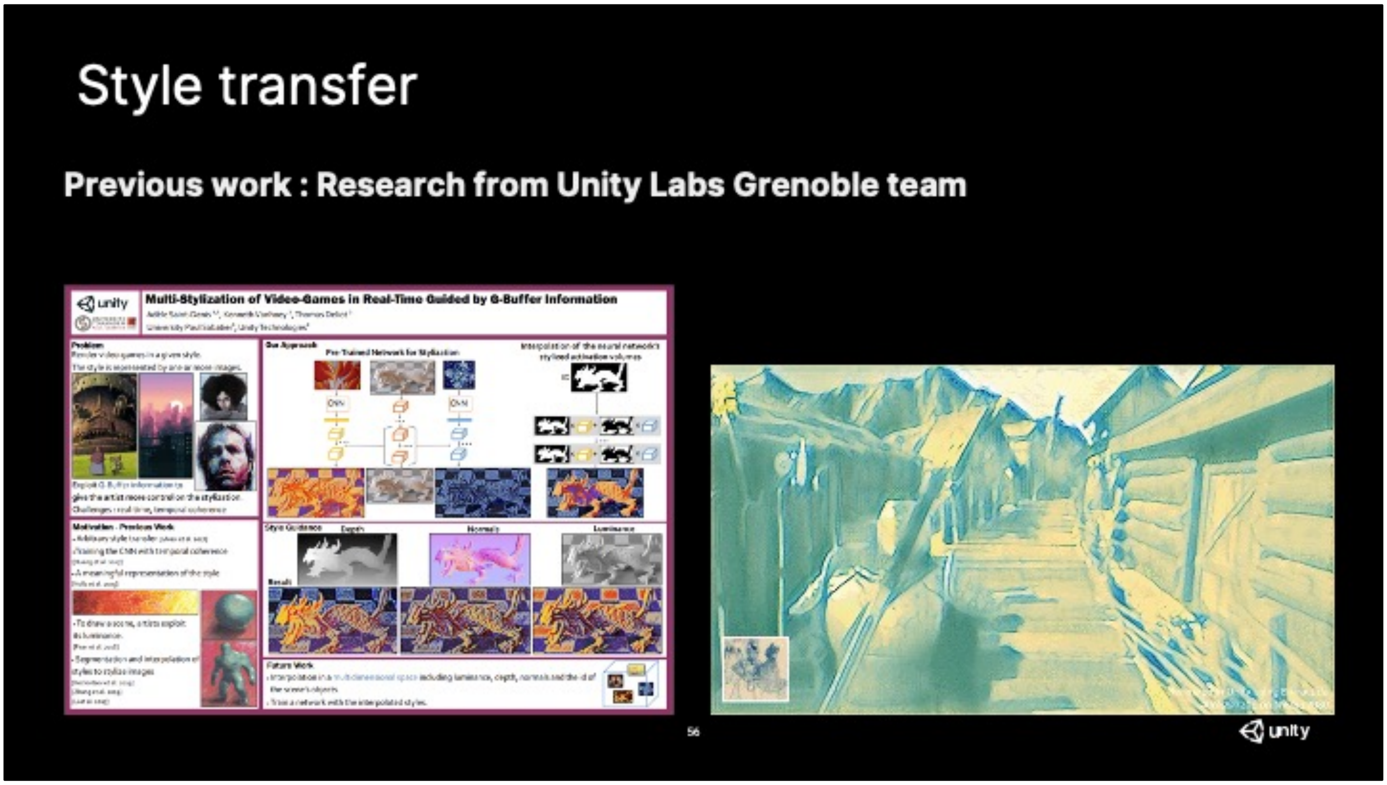


Practical example

Style transfer

Goal: 30fps on desktop and console (PS4Pro)

The goal was to take the existing research from Unity Labs Grenoble team and push it further, toward in real-time performance on desktop and console (PS4Pro). 30fps @ 1080p



Style transfer

Previous work : Research from Unity Labs Grenoble team

Multi-Stylization of Video-Games in Real-Time Guided by G-Buffer Information

Adèle Saint-Denis [1,2], Kenneth Vanhoey [2], Thomas Deliot [2]

University Paul Sabatier [1], Unity Technologies [2]

Problem

Render video games in a given style.

The style is represented by one or more images.

Exploit G-Buffer information to give the artist more control on the stylization.

Challenges : real-time, temporal coherence

Motivation - Previous Work

- Arbitrary style transfer
- Training the CNN with temporal coherence
- A meaningful representation of the style
- To draw a scene, artists exploit its luminance.
- Segmentation and interpolation of styles to stylize images

Our Approach

Pre-Trained Network for Stylization

Interpolation of the neural network's stylized activation volumes

Style Guidance

Depth

Normals

Luminance

Result

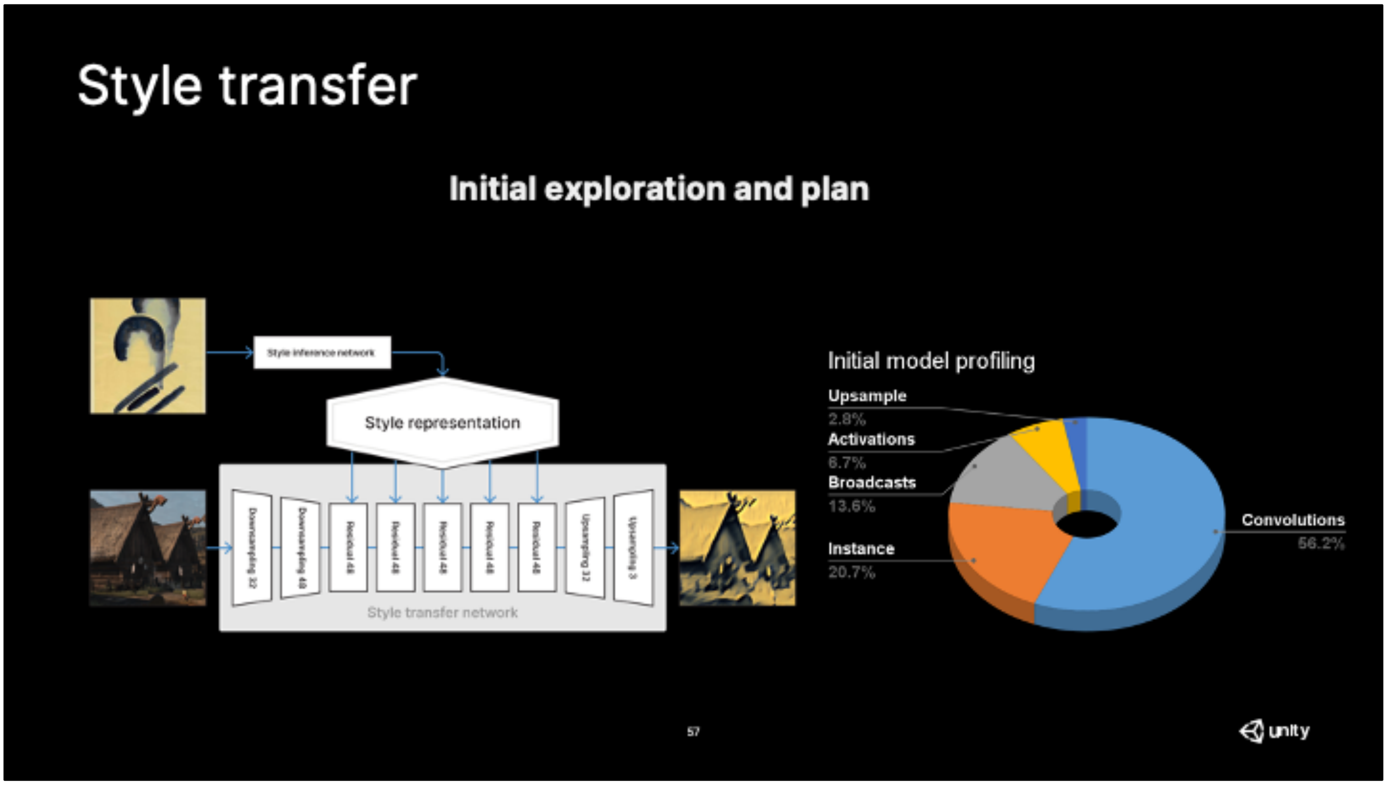
Future Work

- Interpolation in a multi-dimensional space including luminance, depth, normals and the id of the scene's objects
- Train a network with the interpolated styles.

Initial perf where 40ms @ 720p on NVidia RTX2080.

This particular style transfer technique is unique in the sense that a single model is able to generalize to many different style. Also the style can be swapped at runtime!

https://unity-grenoble.github.io/website/publication/2019/07/05/publication-styletransfer.html



Style transfer

Initial exploration and plan

57

We looked at the model as well as performance on device (both desktop and PS4Pro thus) and discovered a few interesting things.

- Convolution performance could be improved (even if we had far from naive implementation already!):
    - Start and stop of network.
        - Model optim : Using strided convolutions to reduce input size as fast as possible.
        - Implement faster kernel for low channels/kernels count.

    - Residual part where width is 48 and does not map to hardware well.
        - Model optim: Switch to 64 kernel but reduce number of layer.
        - We need to implement faster kernel in general

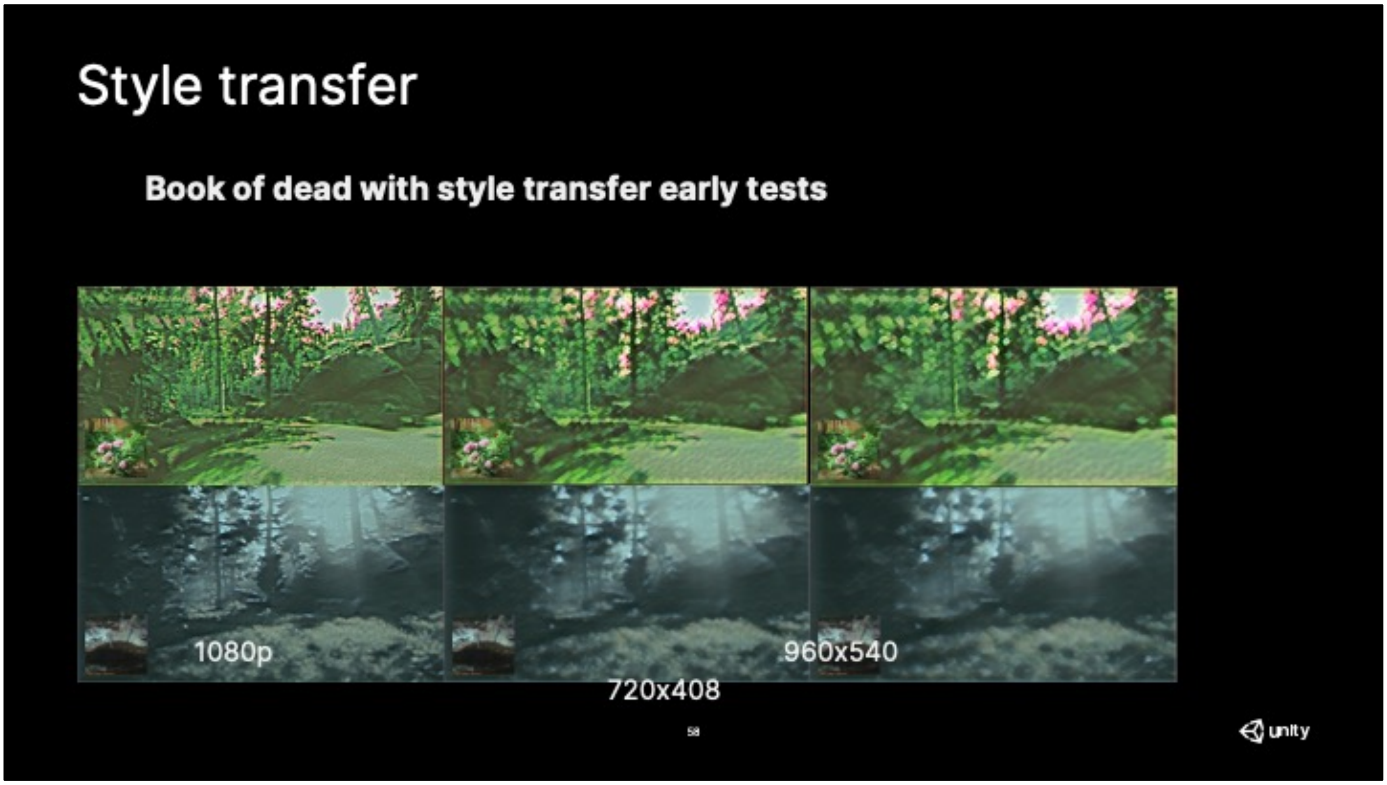
- Weirdly however convolution performances were only around 56% of the time? That was very surprising!
    - We had some inefficient code leading to the network being memory bound for a bunch of operators.
        - Model optim: for training reason the model was doing texture normalization, however this could just be skipped in the context of the engine.
        - At import time we needed more folding and fusing (offline optimization)

    - Instance normalization needed to be improved (heavily used by model).

- However it appeared very quickly that even when all planned optim would be in, performance budget would be very tight. So we also started to look at :
    - Applying Style transfer on downsampled target
    - Do tiled inference and apply temporal reprojection other frames.

- Finally we would like to run this demo on some great content and wished to target the Book of the dead demo from Unity demo team as a base.

The lesson here is probably not new for the game dev community or any community that care about performance: **Always profile the exact use case on device.**



Style transfer

Book of dead with style transfer early tests

- After a bunch of iterations both on optimizations and visuals we discovered:
    - Network was not reacting well at lower resolution especially on the high frequency content such as Book of the Dead. Look was very dreamy and not high quality enough.
    - At lower resolution it was harder to occupy the GPUs making us lose some of the performance benefit we hoped for.
    - As training the network is a long process, we could not try to train it at various resolutions to experiment if visual quality on downsampled input would be improved.

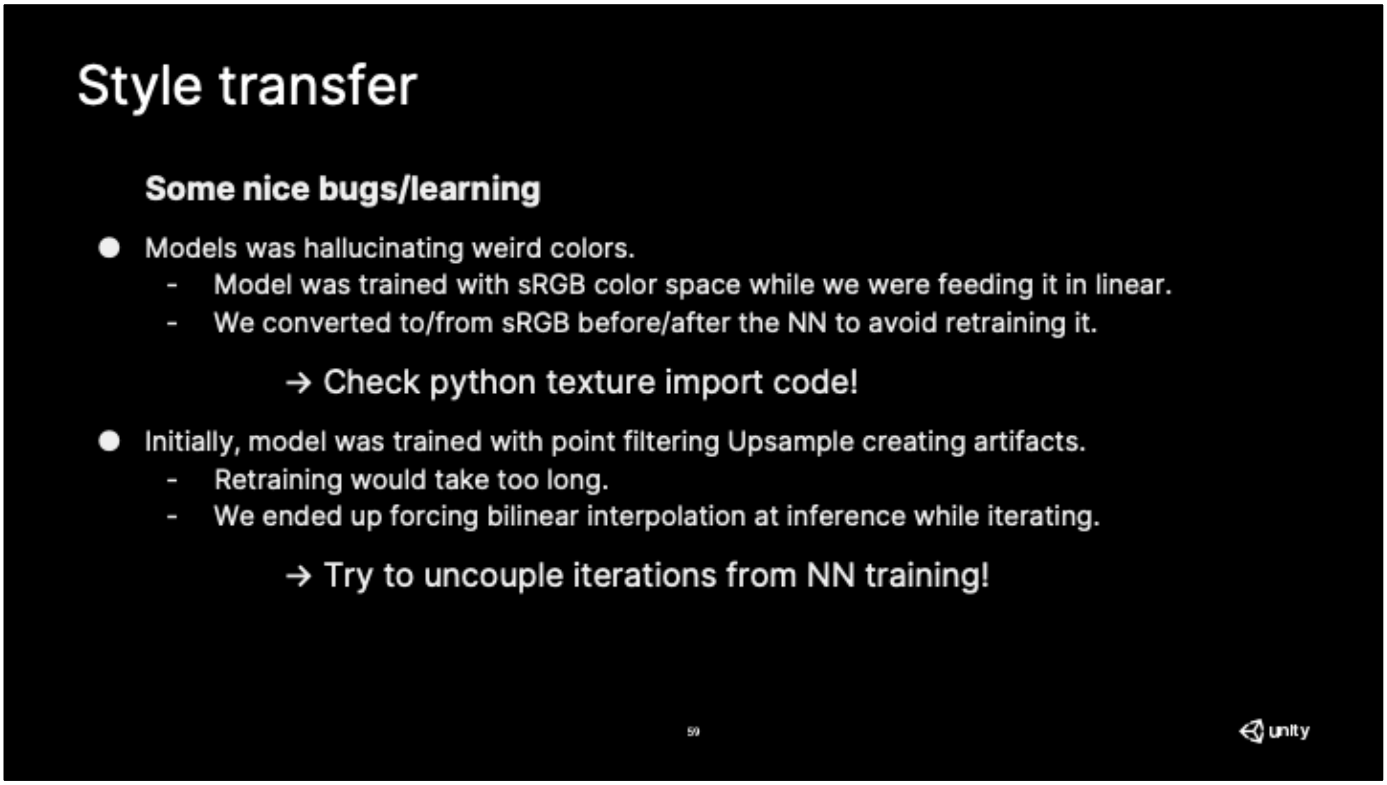
  → Downsampling before style transfer is not such a great idea finally, we need to run style transfer at 1080p

    - Style transfer quality was needing the depth of the network much more than it was needing the width of it. And even with optimisations we probably could not afford 64 kernels in the residual part.
    - Artistics feedback was that the style transfer was too heavy and would be tiring to the eyes, we needed a lighter effect.

  → What if we tried 32 kernels for residual part of network and same depth.

- Book of the Dead run well on PS4Pro @ 1080p 30fps, however frame time is already close to 33ms on some scenes.

  → Finally we decided to run:
  Book of the dead @1080p + style transfer @1080p on PC.
  Viking village @1080p + tiled style transfer @1080/4 with temporal reprojection on 4 frames on PS4Pro.



Style transfer

Some nice bugs/learning

- Models was hallucinating weird colors.
  - Model was trained with sRGB color space while we were feeding it in linear.
  - We converted to/from sRGB before/after the NN to avoid retraining it.

→ Check python texture import code!

- Initially, model was trained with point filtering Upsample creating artifacts.
  - Retraining would take too long.
  - We ended up forcing bilinear interpolation at inference while iterating.

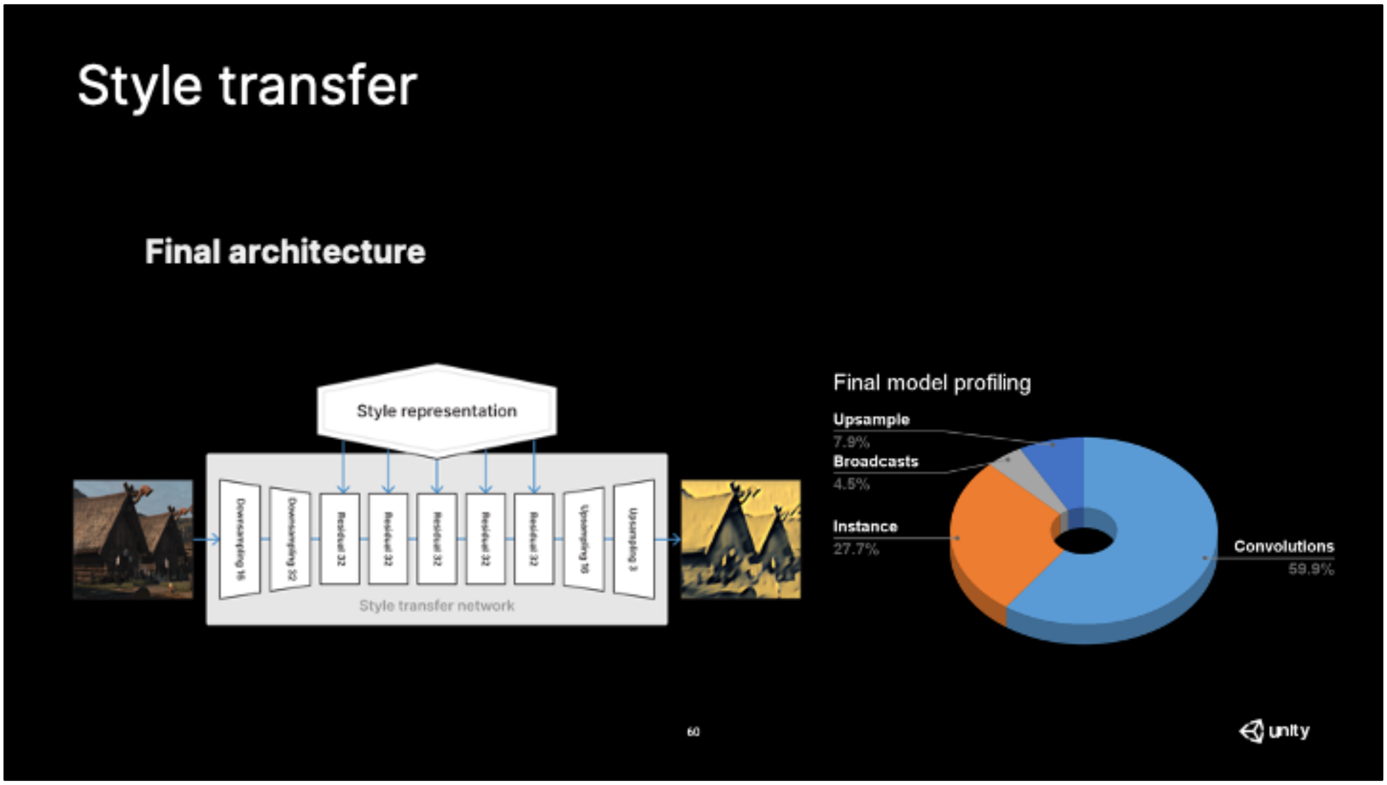
→ Try to uncouple iterations from NN training!

Bugs happens that’s for sure and NN are no stranger to that rule

Lessons here:

- When training a network in Python if one load images using some python packages they usually do not explicitly say what they will do about color space, need to check the code!
  Some package do only sRGB, some others only linear! And worst some will select based on the file format or extension! This is definitely something one should pay attention when planning to use the model as an effect in a RT3D engine!

- Important to pick what you want to experiment and what you won’t in term of model training, you probably can’t do all of the training experiments you dream of.
  Also Iterating on the visual need to ideally be a fast process but training the model is a slow one.
  It’s is good to try to uncouple both by taking all possible visual controls out of the network.
  For example we iterated with LUT tables after the network but ended up not using those in the end.



Style transfer

Final architecture

What we ended up using is a network where :

- Up and downsampling part have been changed to be more lightweight.
- Residual kernel width is 32.

Some perf numbers:

PC

Book of the dead @1080p + style transfer @1080p

RTX 2080 → 23ms (6-9 ms rendering, 12ms inference).
(Before was 40ms @ 720p, different model however!)

More details, videos and perf number on the associated blog post here (older version of Barracuda perf a bit lower) :
https://blogs.unity3d.com/2020/11/25/real-time-style-transfer-in-unity-using-deep-neural-networks/

Some very interesting observation:
After both model and code optimization if performance are much improved we funnily ended up in a somewhat similar situation (percentage wise):

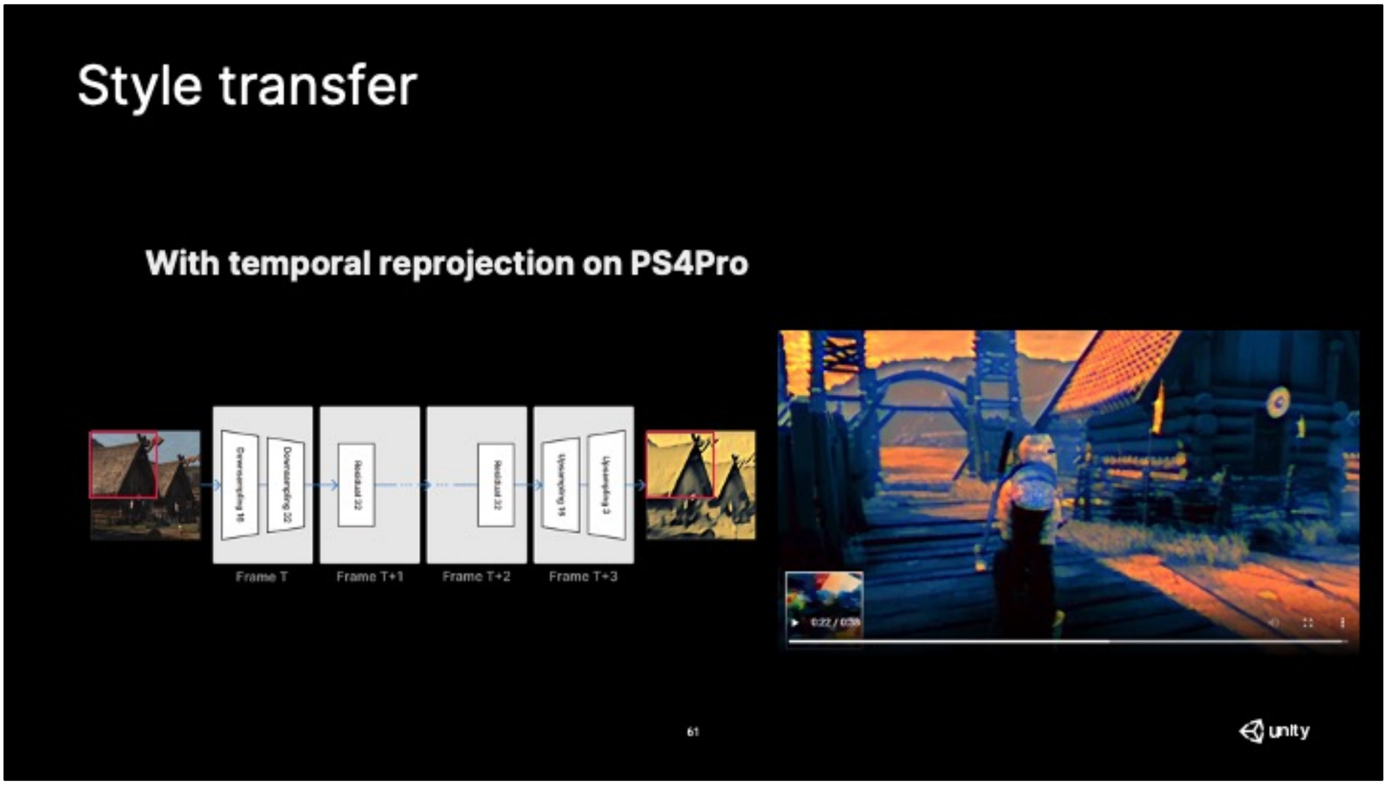
- Convolution are 59.9 % of the inference time (was 56.2%).
- Upsample are 8% (was 2.8%)

- Instance normalization are 27% (was 20.7%)
- Broadcast are 4.5% (was 13.6%)
- Activation are 0% (was 6.7%)

Also there are plenty of great opportunity to further optimized this effect:

- Faster Instance normalizations
  - kernel level
  - model level
- Convolution:
  - for this demo we didn't harness the power of fp16!
  - also we optimized for 64 kernels while final model is actually 32! Could be interesting to dig especially for hardware with a warp size of 32.
- Upsample
  - Kernel level optimization
  - Could transposed convolution be used instead (or maybe merge upsample and conv at kernel level?)
- More operation fusing!

Finally this effect focus on changing the style at runtime? But what if we don't need that feature, could we have a cheaper network?



Style transfer

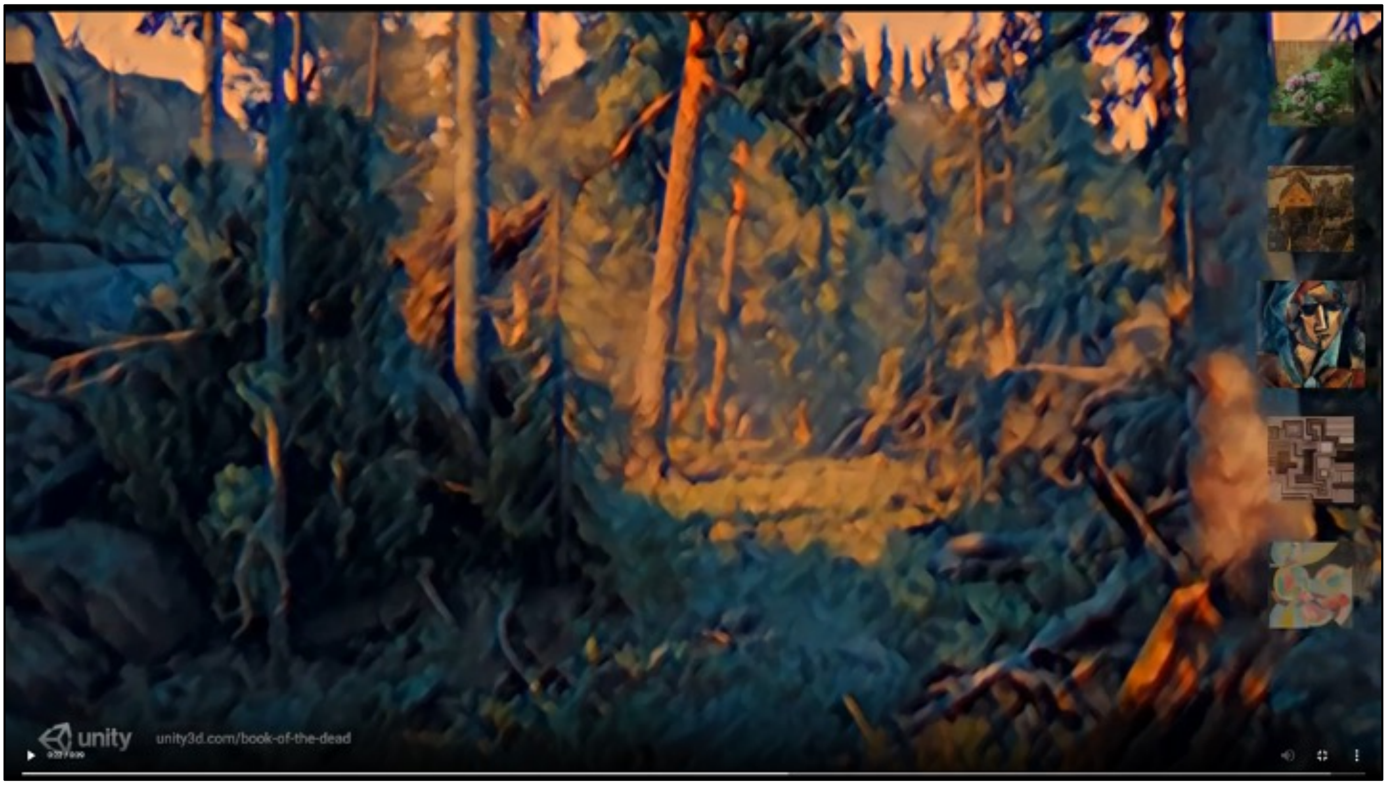
With temporal reprojection on PS4Pro

(on the right: THIS IS A VIDEO, please check course video or the blog post below to watch it)

PS4Pro
Viking village @1080p + tiled style transfer @1080p/4 with temporal reprojection on 4 frames
28ms (10ms rendering, 14ms per frame for sliced inference + 4ms temporal reprojection).

The general idea here is to stylized a quarter of the screen every frame and temporarily reproject over 4 frames, this is however tricky as typically reprojection technique use the depth buffer to detect occlusion and disocclusion.
However style transfer don’t write to depth while still affecting the shape of the objects sometime almost as a volumetric effect.
All credit to Thomas Deliot.

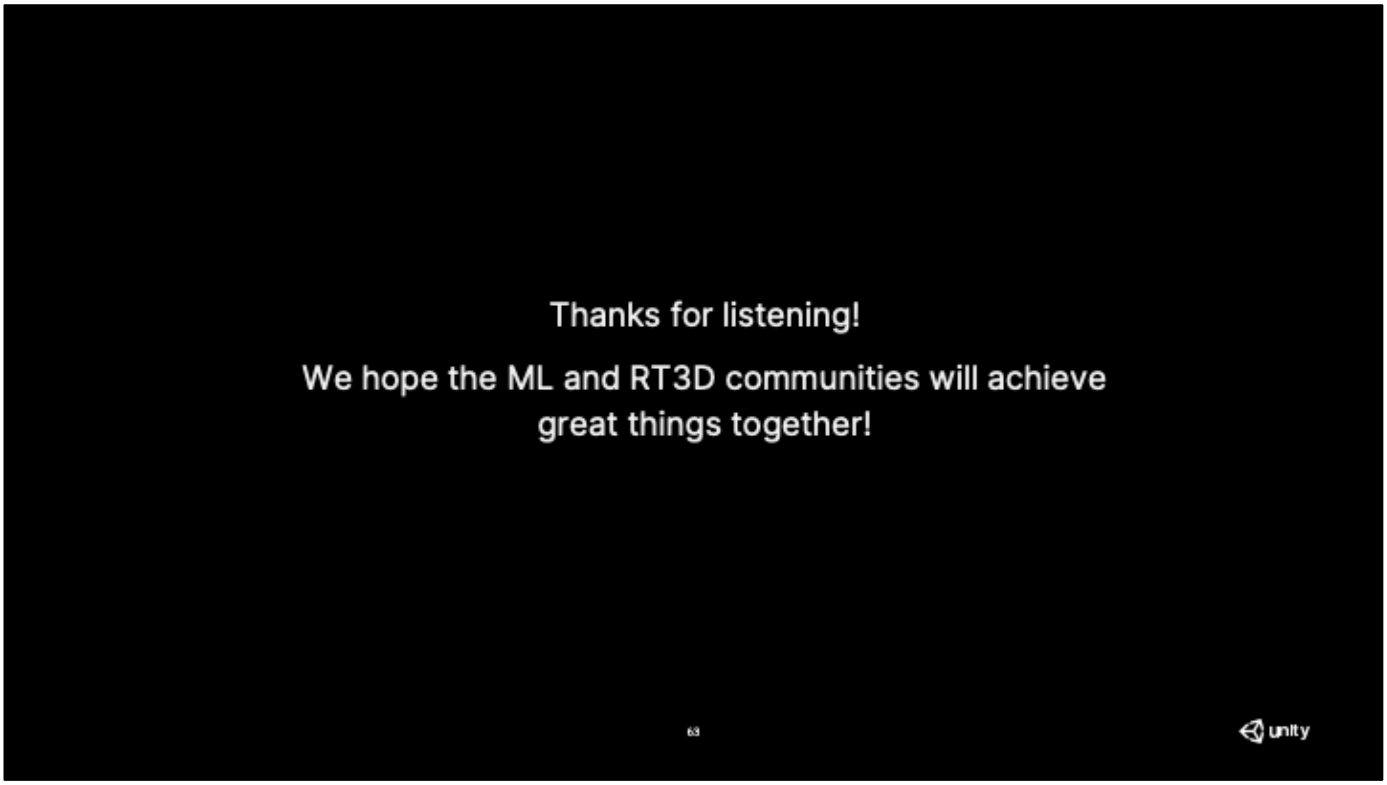
More details in the blog post.
https://blogs.unity3d.com/2020/11/25/real-time-style-transfer-in-unity-using-deep-neural-networks/



Here we can see how the style can be dynamically changed at runtime (THIS IS A VIDEO, please check course video or the blog post below to watch it)

The trick here is to only evaluate the style evaluation part of the network once for each style and save the resulting embedding on disk, then just hot swap those weight in memory at runtime.

More details, videos and perf numbers on the associated blog post here: https://blogs.unity3d.com/2020/11/25/real-time-style-transfer-in-unity-using-deep-neural-networks/



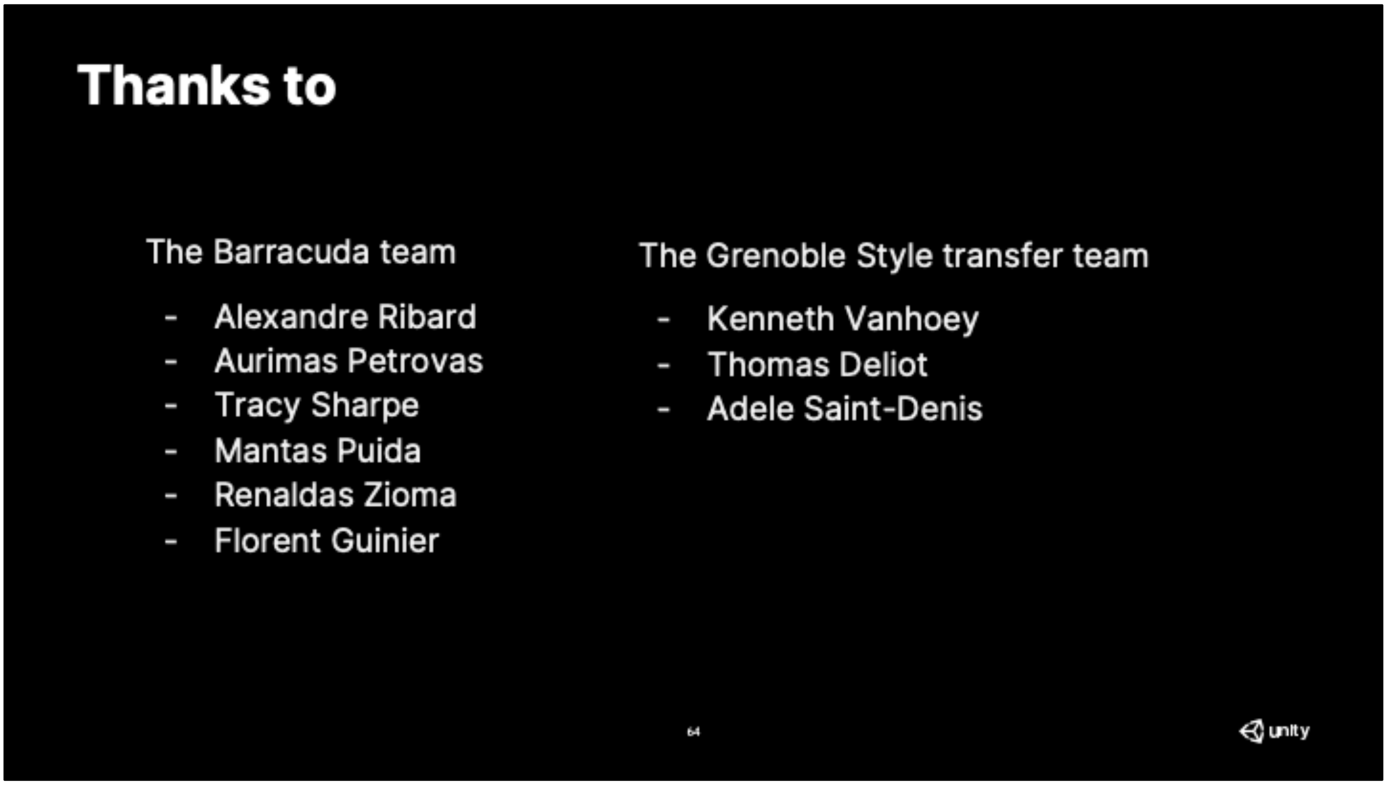
Thanks for listening!

We hope the ML and RT3D communities will achieve great things together!

Thanks for listening, we hope this was useful for you!

On behalf of the wonderful Barracuda team we hope that the **ML and game dev communities will join forces and create amazing things together**!

Have a great day!



Thanks to

The Barracuda team

- Alexandre Ribard
- Aurimas Petrovas
- Tracy Sharpe
- Mantas Puida
- Renaldas Zioma
- Florent Guinier

The Grenoble Style transfer team

- Kenneth Vanhoey
- Thomas Deliot
- Adele Saint-Denis

https://github.com/Unity-Technologies/barracuda-release

[redacted]

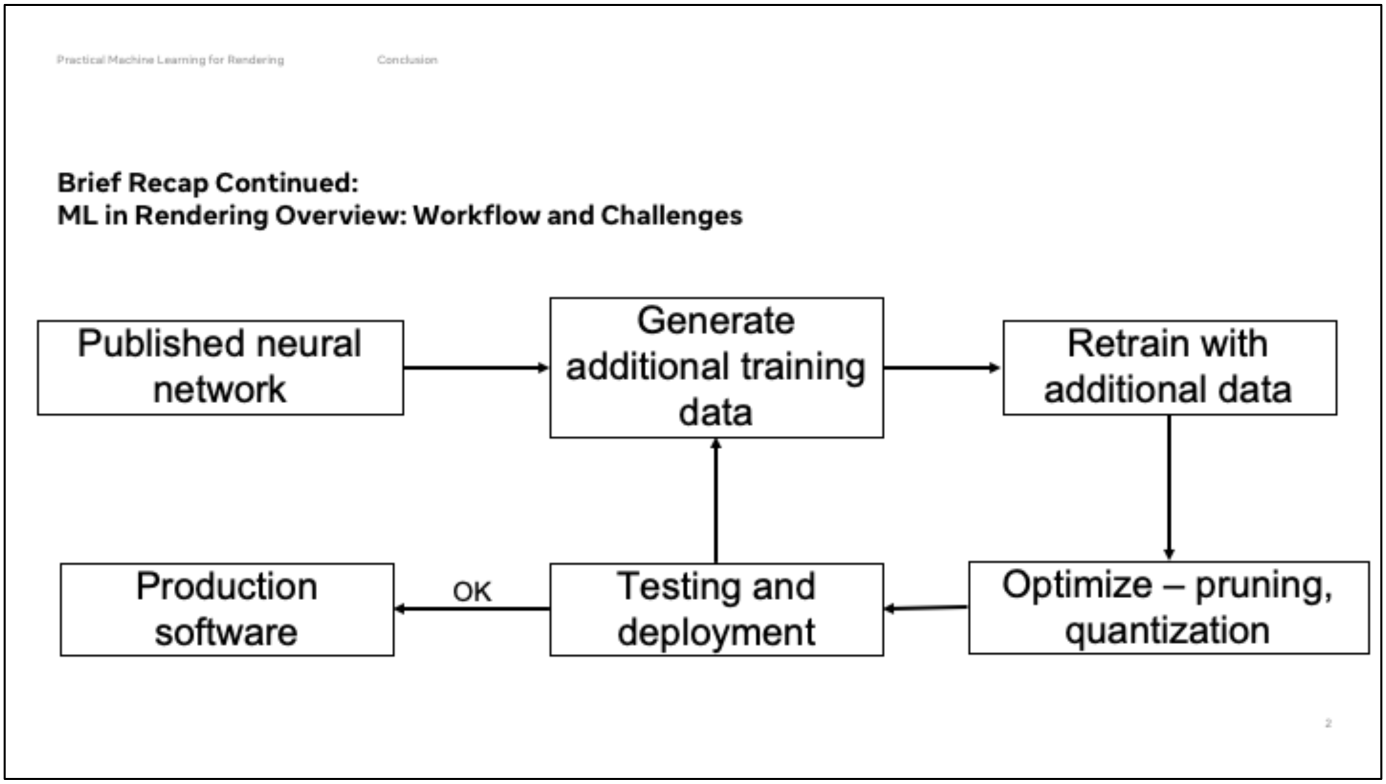
Carl S. Marshall | April 25, 2022 | Eurographics

Conclusion:
Practical Machine Learning for Rendering

From Research to Deployment

Unity intel.

Thank you for attending our course on Practical Machine Learning for Rendering.   It has been a great pleasure working with our collaborators at Unity and my colleagues at Intel. In this section, I will provide a brief overview from each of the sessions of this course and a call to action at the end of this section.



Brief Recap Continued:
ML in Rendering Overview: Workflow and Challenges

Published neural network

Generate additional training data

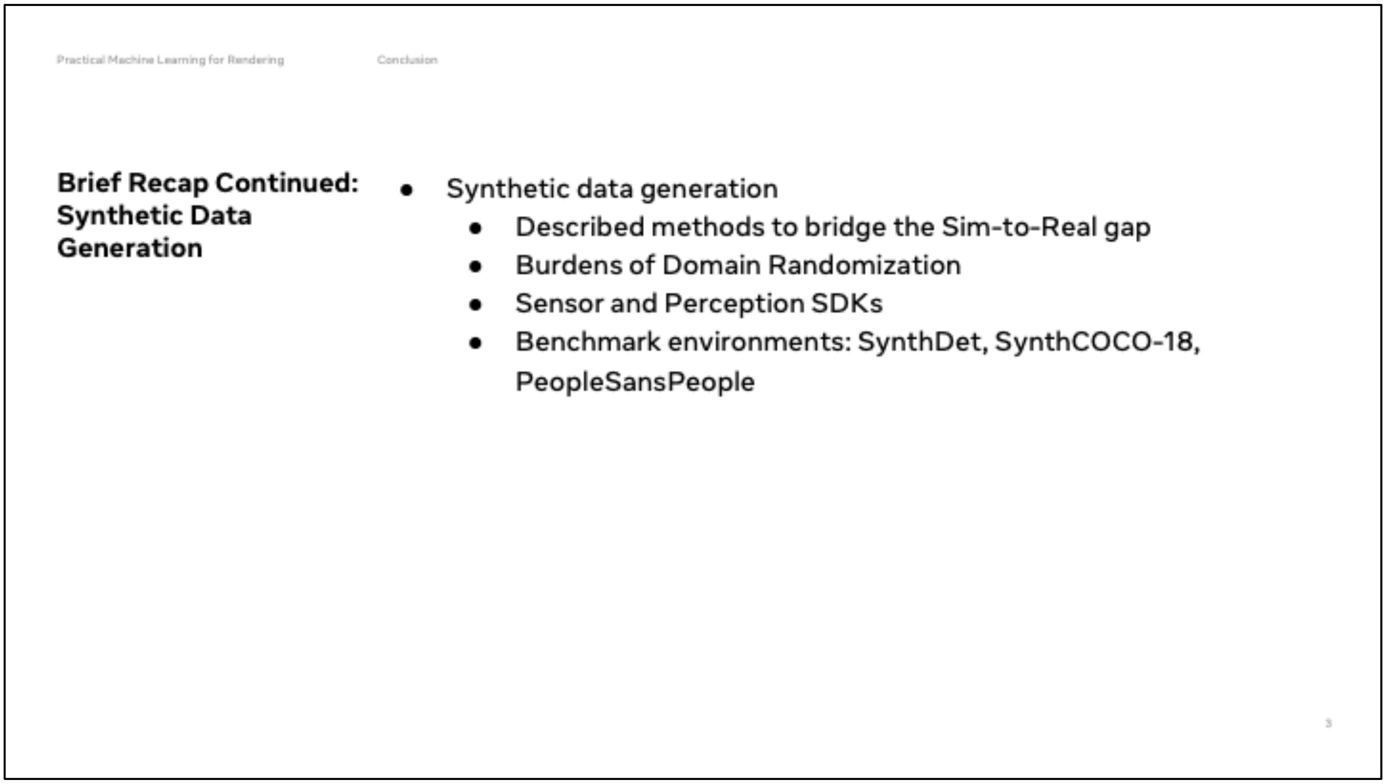
Retrain with additional data

Optimize – pruning, quantization

Testing and deployment

OK

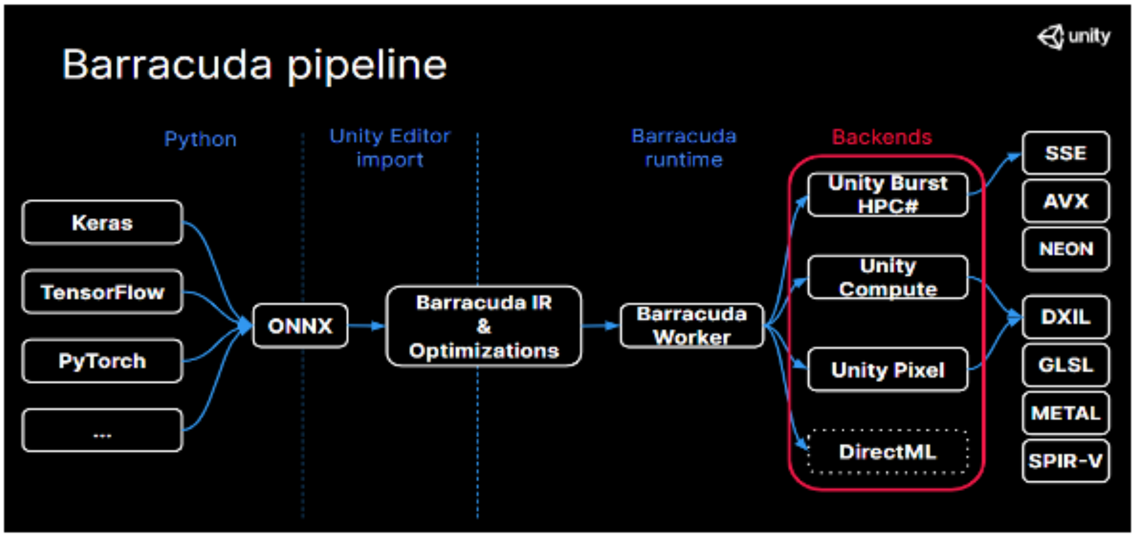
Production software



Brief Recap Continued: Synthetic Data Generation

- Synthetic data generation
  - Described methods to bridge the Sim-to-Real gap
  - Burdens of Domain Randomization
  - Sensor and Perception SDKs
  - Benchmark environments: SynthDet, SynthCOCO-18, PeopleSansPeople

Brief Recap Continued:
Machine Learning in Real-time



Barracuda pipeline

unity

Python

Unity Editor import

Barracuda runtime

Backends

Keras

TensorFlow

PyTorch

...

ONNX

Barracuda IR & Optimizations

Barracuda Worker

Unity Burst HPC#

Unity Compute

Unity Pixel

DirectML

SSE

AVX

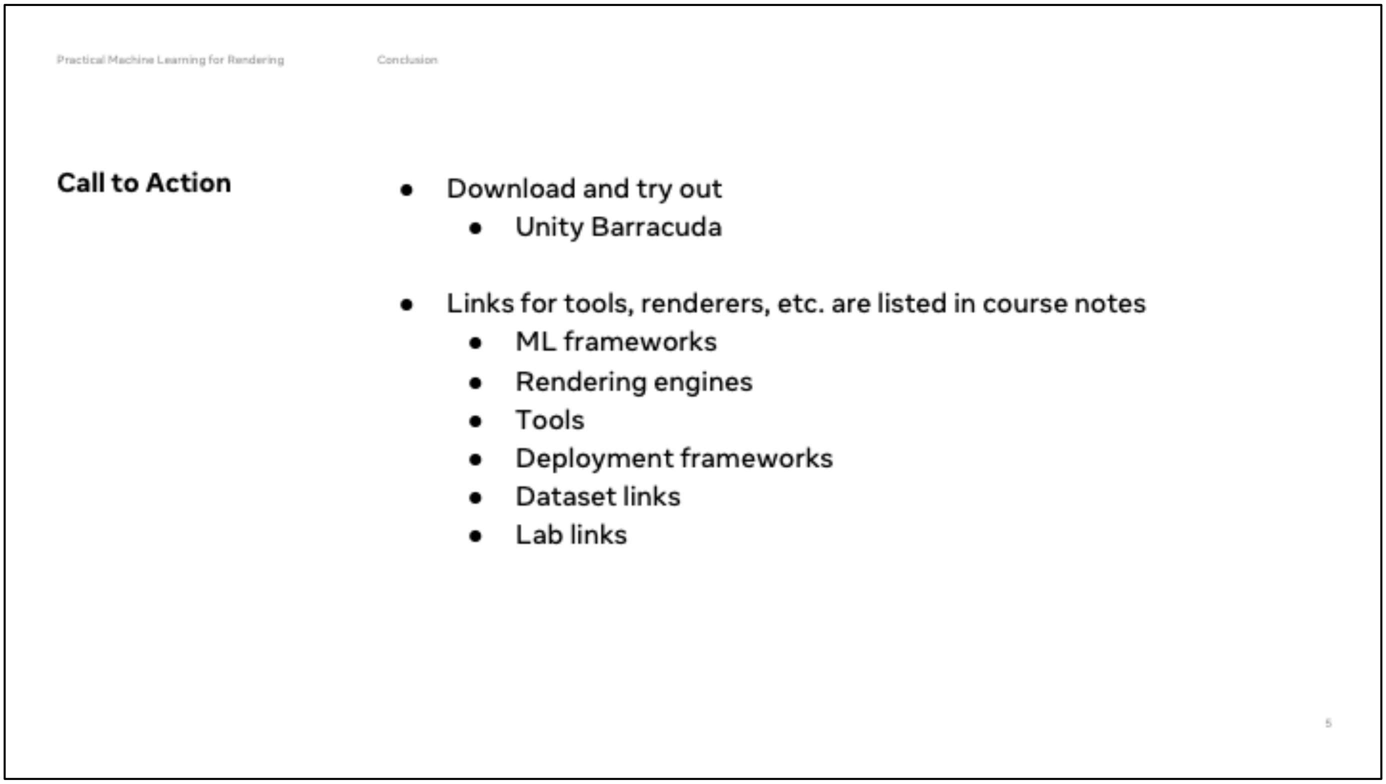
NEON

DXIL

GLSL

METAL

SPIR-V



Practical Machine Learning for Rendering Conclusion

Call to Action

- Download and try out
  - Unity Barracuda
- Links for tools, renderers, etc. are listed in course notes
  - ML frameworks
  - Rendering engines
  - Tools
  - Deployment frameworks
  - Dataset links
  - Lab links

5

Resources:

- Machine learning frameworks: pytorch.org, tensorflow.org, keras.io
- Rendering engines: Unity.com, blender.org, pbrt.org, unrealengine.com

Tool Links:

- Intel® oneAPI Toolkit - https://software.intel.com/content/www/us/en/develop/tools/oneapi/base-toolkit.html - foundational base toolkit enables the building, testing, and optimizing of data-centric applications across XPUs
- Intel® oneAPI Deep Neural Network Library - https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onednn.html - Increase Deep Learning Framework Performance on CPUs and GPUs
- Intel® Distribution of OpenVINO™ Toolkit - https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html – optimizing deep learning networks
- Intel® Graphics Performance Analyzers - https://software.intel.com/content/www/us/en/develop/tools/graphics-performance-analyzers.html
- NVIDIA® TensorRT™ - https://developer.nvidia.com/tensorrt
- NVIDIA® Nsight™ - https://developer.nvidia.com/tools-overview
- ONNX format - www.onnx.ai

Deployment:

- Unity Barracuda: https://github.com/Unity-Technologies/barracuda-release
- Onnx Runtime: onnxruntime.ai
- DirectML: https://github.com/microsoft/DirectML

Datasets links:

- Turbosquid – www.turbosquid.com
- Unity Assets - https://assetstore.unity.com/
- Open 3D models - https://open3dmodel.com/
- Free3D – https://free3D.com
- Kaggle datasets - https://www.kaggle.com/datasets
- Disney - https://studios.disneyresearch.com/data-sets/

Lab links:

- Intel – https://www.intel.com/content/www/us/en/research/overview.html
- Unity – https://unity.com/labs
- Meta RL Research - https://research.facebook.com/research-areas/augmented-reality-virtual-reality/
- Nvidia – https://www.nvidia.com/en-us/research/
- Disney – https://www.disneyresearch.com/

Contact Info

Carl S. Marshall, Reality Labs Research at Meta
csmarshall@fb.com

Deepak Vembar, Intel Labs
deepak.s.vembar@intel.com

Sujoy Ganguly, Unity
sujoy.ganguly@unity3d.com

Florent Guinier, Unity Labs
florent@unity3d.com

April 25, 2022

Thank you.