Lecture 17:

Rendering (and Simulation) for Learning

Visual Computing Systems
Stanford CS348K, Spring 2021

Think back to earlier in course

What was the biggest practical bottleneck to training good models?

Snorkel: Rapid Training Data Creation with Weak Supervision

Alexander Ratner Stephen H. Bach Henry Ehrenberg
Jason Fries Sen Wu Christopher Ré
Stanford University
Stanford, CA, USA
{ajratner, bach, henryre, jfries, senwu, chrismre}@cs.stanford.edu

ABSTRACT

Labeling training data is increasingly the largest bottleneck in deploying machine learning systems. We present Snorkel, a first-of-its-kind system that enables users to train state-of-the-art models without hand labeling any training data. Instead, users write labeling functions that express arbitrary heuristics, which can have unknown accuracies and correlations. Snorkel denoises their outputs without access to ground truth by incorporating the first end-to-end implementation of our recently proposed machine learning paradigm, data programming. We present a flexible interface layer for writing labeling functions based on our experience over the past year collaborating with companies, agencies, and research labs. In a user study, subject matter experts build models 2.8× faster and increase predictive performance an average 45.5% versus seven hours of hand labeling. We study the modeling tradeoffs in this new setting and propose an optimizer for automating tradeoff decisions that gives up to 1.8× speedup per pipeline execution. In two collaborations, with the U.S. Department of Veterans Affairs and the U.S. Food and Drug Administration, and on four open-source text and image data sets representative of other deployments, Snorkel provides 132% average

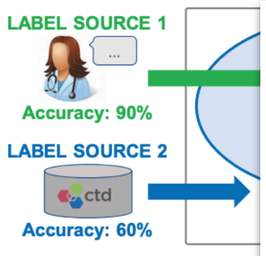


Figure 1: In Example 1 by sources of differing ac key challenges arise in u effectively. First, we need known source accuracies Second, we need to pass formation to the end mo

advent of *deep learning* tech specific representations of in to be the most time-consum engineering. These learned r effective for tasks like natural

Overton: A Data System for Monitoring and Improving Machine-Learned Products

Christopher Ré Apple
Feng Niu Apple
Pallavi Gudipati Apple
Charles Srisuwananukorn Apple

September 13, 2019

Abstract

We describe a system called Overton, whose main design goal is to support engineers in building, monitoring, and improving production machine learning systems. Key challenges engineers face are monitoring fine-grained quality, diagnosing errors in sophisticated applications, and handling contradictory or incomplete supervision data. Overton automates the life cycle of model construction, deployment, and monitoring by providing a set of novel high-level, declarative abstractions. Overton's vision is to shift developers to these higher-level tasks instead of lower-level machine learning tasks. In fact, using Overton, engineers can build deep-learning-based applications without writing any code in frameworks like TensorFlow. For over a year, Overton has been used in production to support multiple applications in both near-real-time applications and back-of-house processing. In that time, Overton-based applications have answered billions of queries in multiple languages and processed trillions of records reducing errors $1.7-2.9\times$ versus production systems.

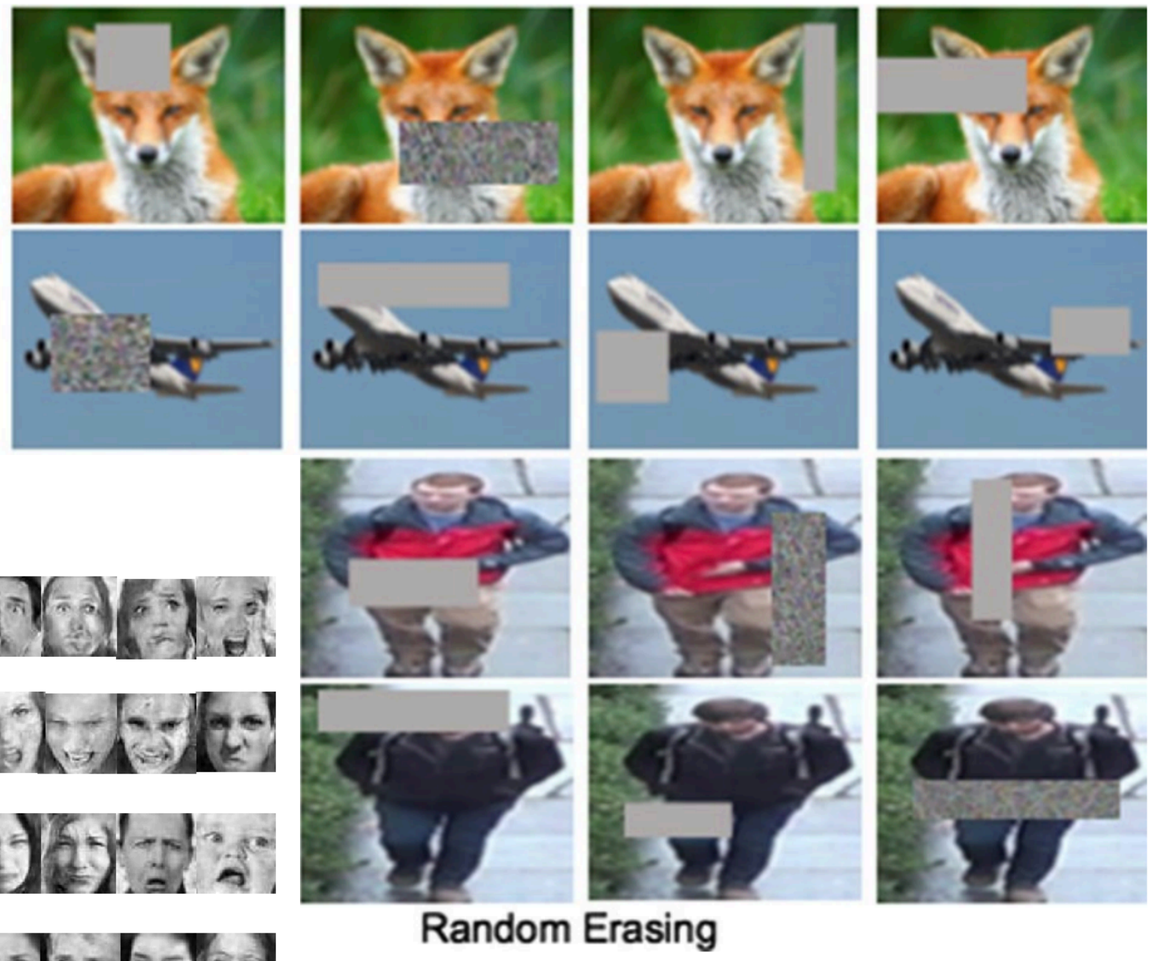
1 Introduction

In the life cycle of many production machine-learning applications, maintaining and improving deployed models is the dominant factor in their total cost and effectiveness–much greater than the cost of *de novo* model construction. Yet, there is little tooling for model life-cycle support. For such applications, a key task for supporting engineers is to improve and maintain the quality in the face of changes to the input distribution and new production features. This work describes a new style of data management system called Overton that provides abstractions to support the model life cycle by helping build models, manage supervision, and monitor application quality.[1]

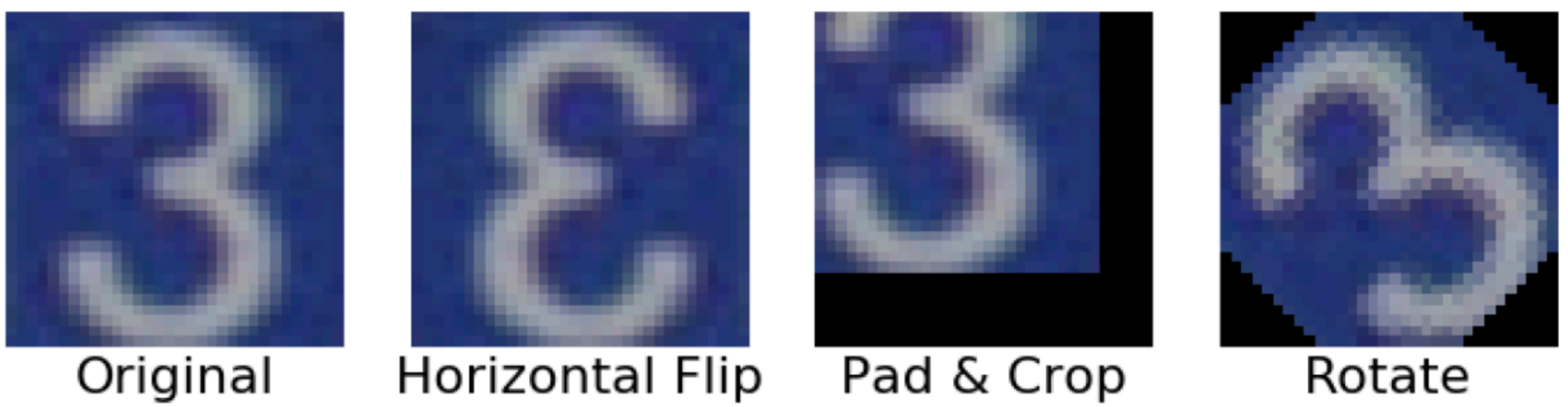
Overton is used in both near-real-time and backend production applications. However, for concreteness, our running example is a product that answers factoid queries, such as *"how tall is the president of the united states?"* In our experience, the engineers who maintain such machine learning products face several challenges on which they spend the bulk of their time.

Data-augmentation

A common strategy for automatically generating new labeled training data from a small number of labeled examples (as long as augmentations don't change classification result)



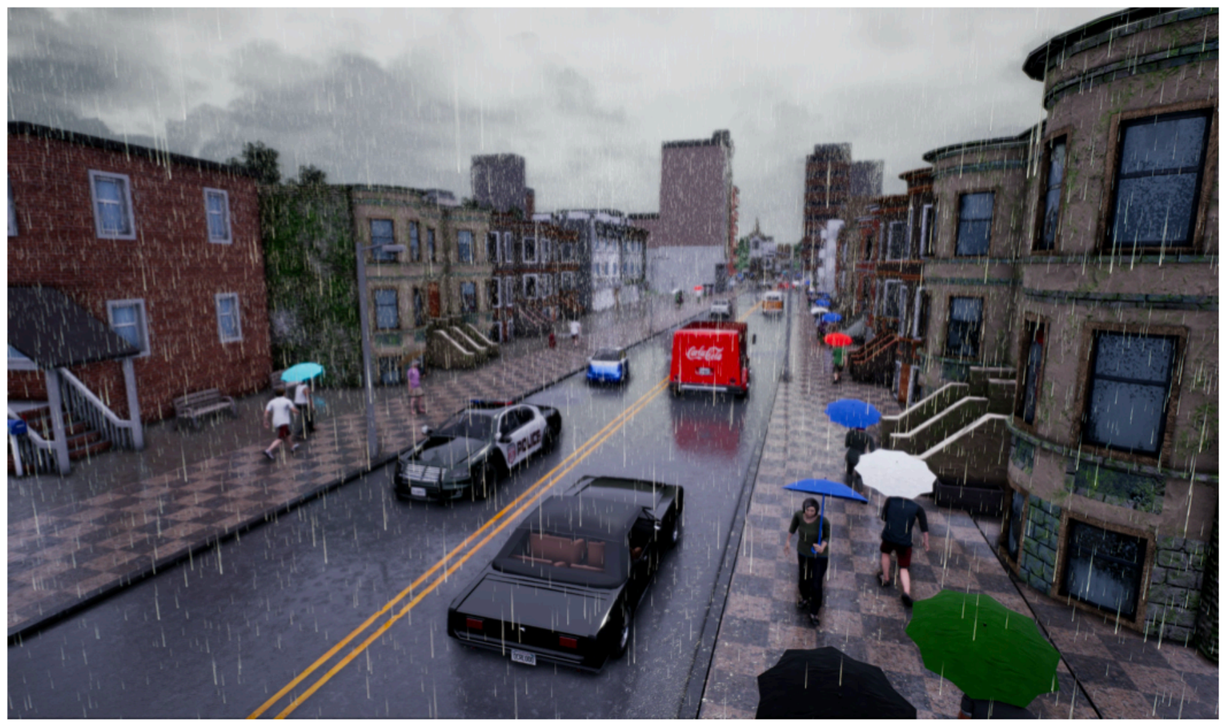
[Credit: Zhu et al. 2017]



[Credit: Ho et al. 2019]

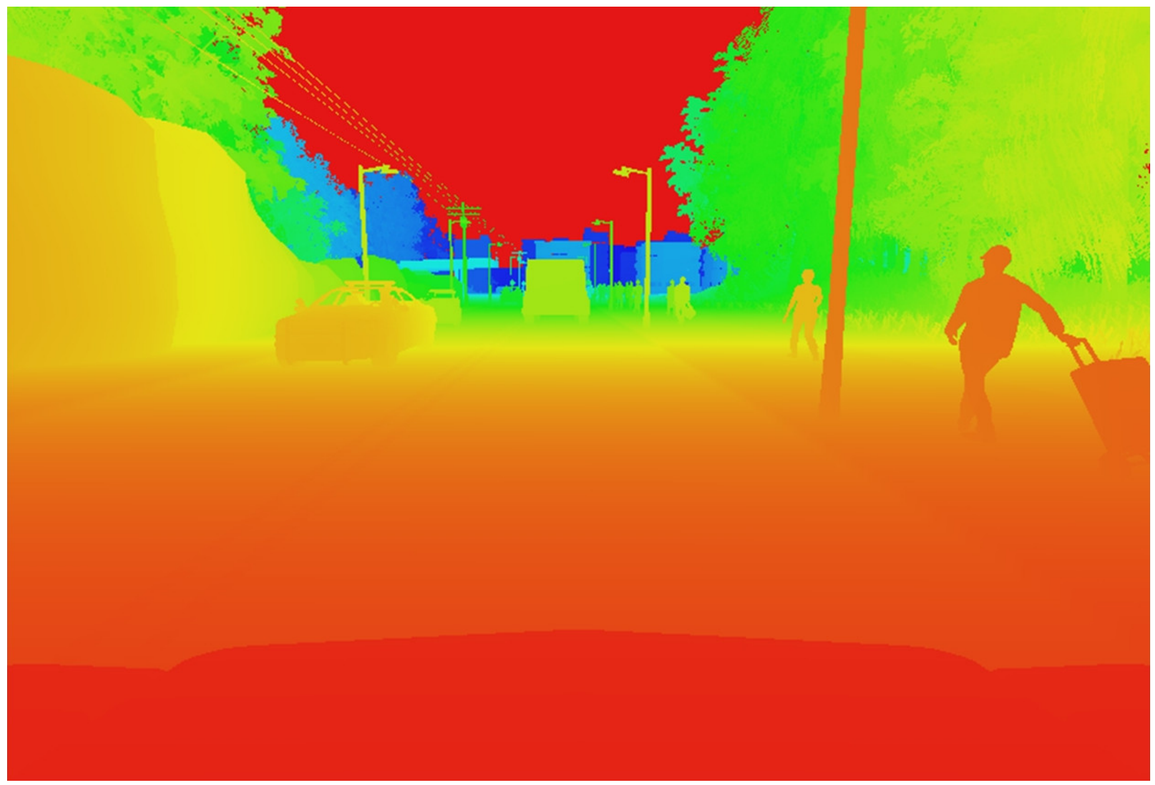
Using advanced rendering/simulation to train better models

Carla: urban driving simulator based on Unreal Engine

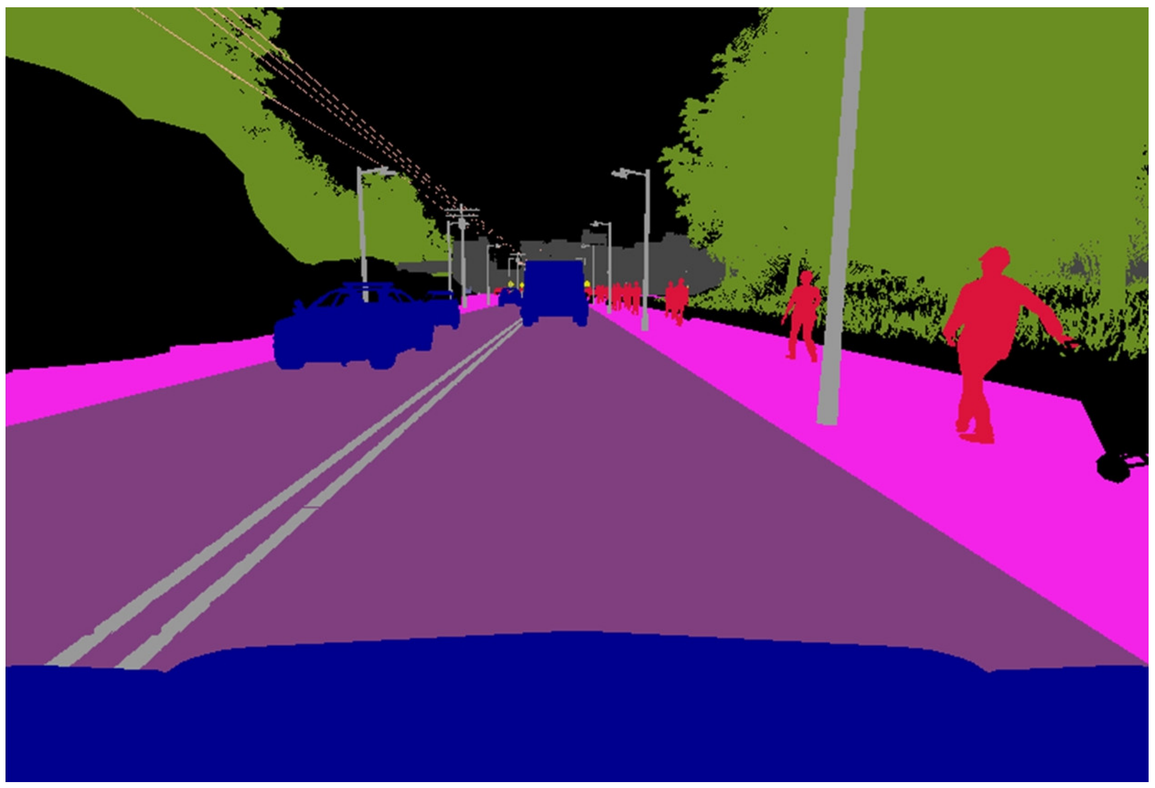


Example Carla outputs

RGB



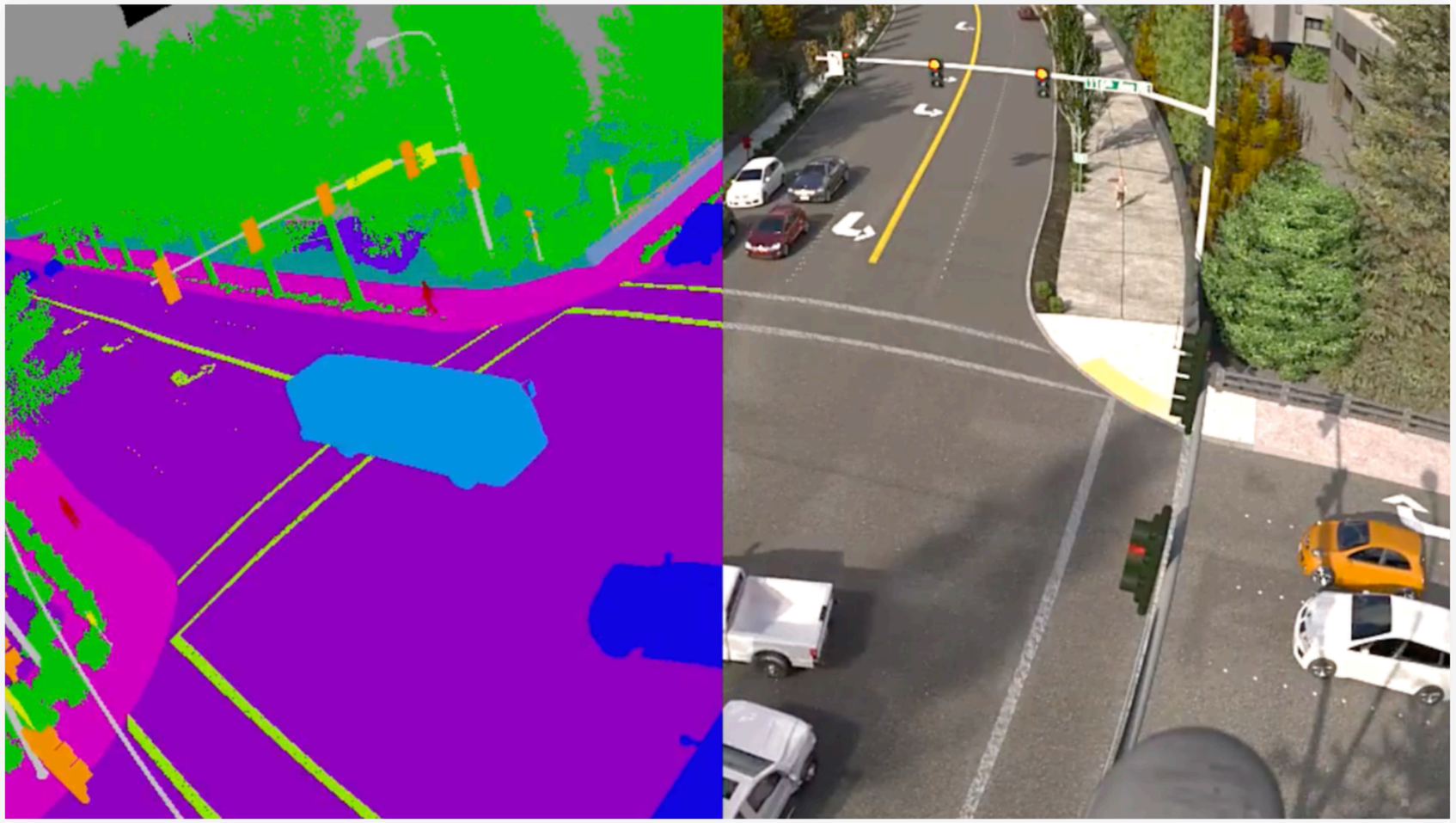
Depth



Object type

Since renderer has complete description of scene, it can output detailed, fine-grained labels as well as RGB image.

(would be laborious to annotate)



Synthetic data: Simulating myriad possibilities to train robust machine learning models

Srinivas Annambhotla, Cesar Romero and Alex Thaman, May 1, 2020

Machine Learning | Manufacturing

Share

Categories

- **All**
- **AEC**
- **Asset Store**
- **Community**
- **Events**
- **Machine Learning**
- **Made With Unity**
- **Manufacturing**
- **Monetize**
- **Services**
- **Technology**

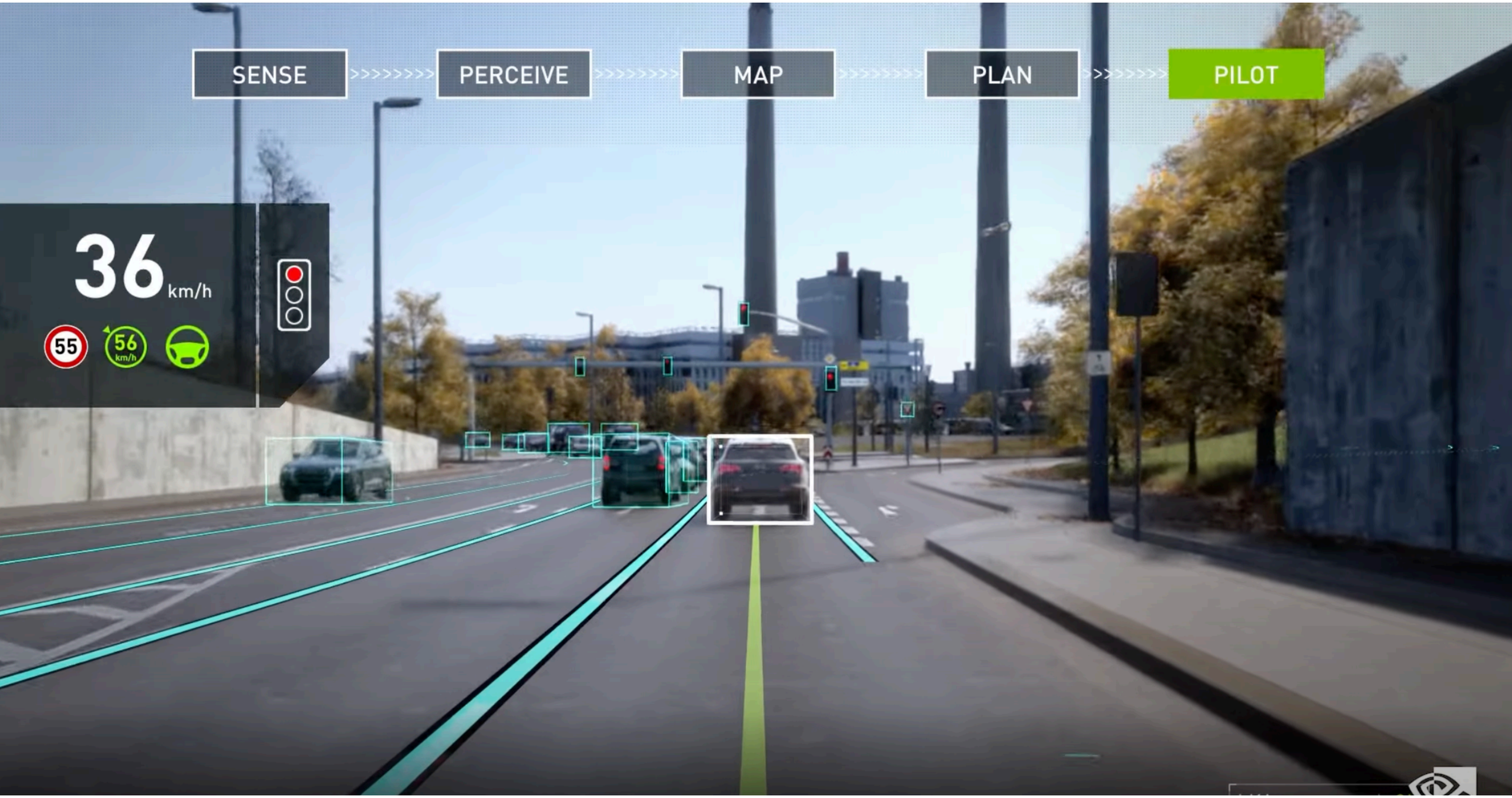
Popular posts

Introducing the new Input System
October 14, 2019

2D Pixel Perfect: How to set up your Unity project for retro 8-bit games
March 13, 2019

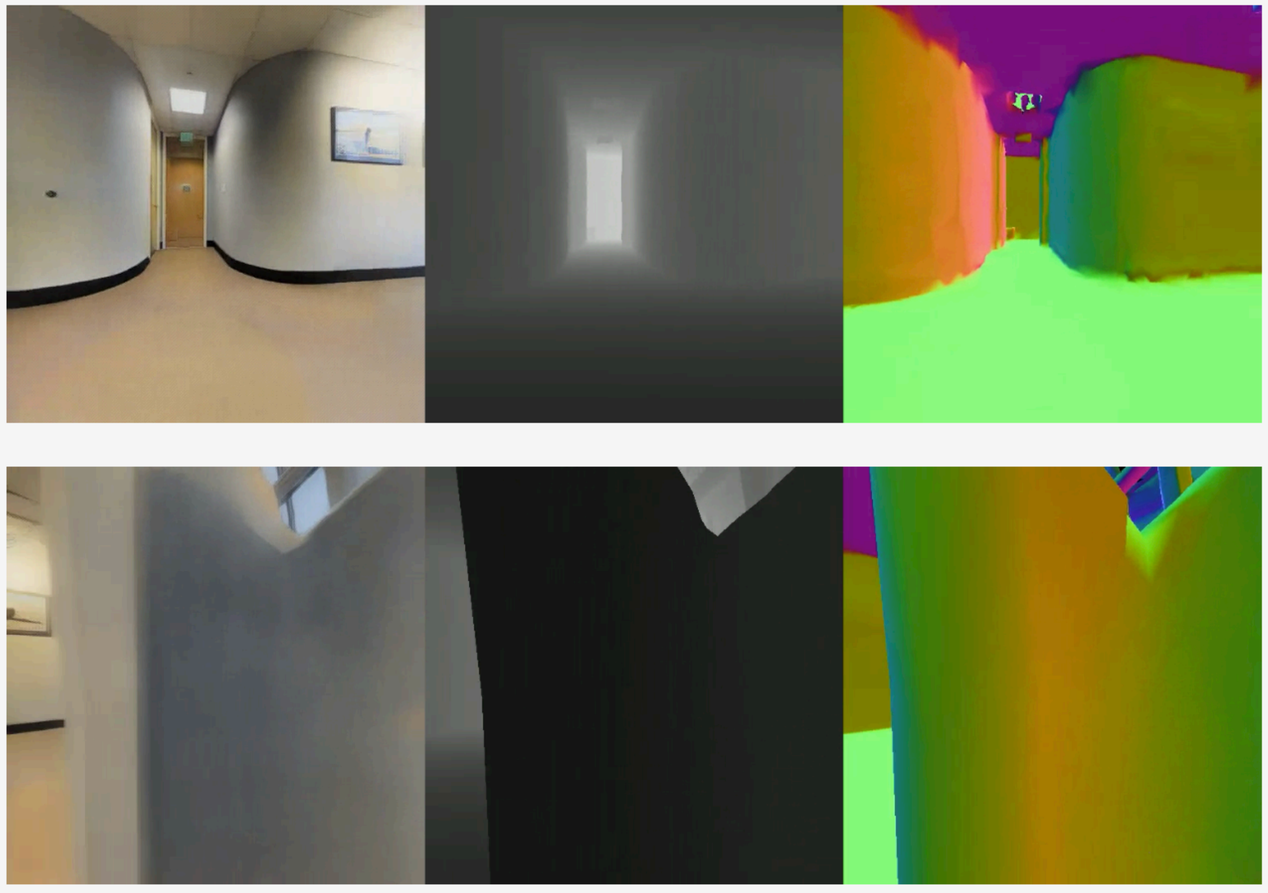
The High Definition Render Pipeline: Getting Started Guide for Artists
September 24, 2018

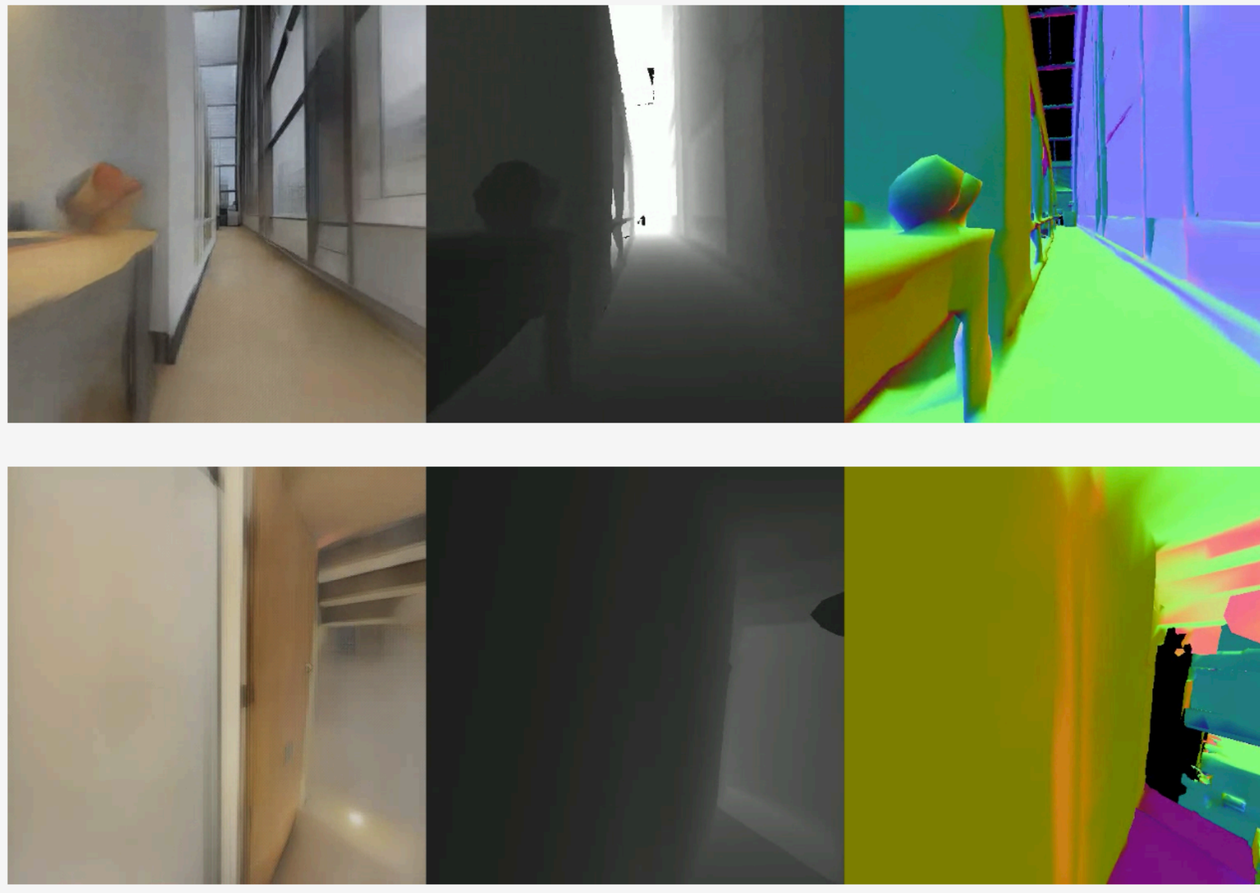
NVIDIA Drive Sim

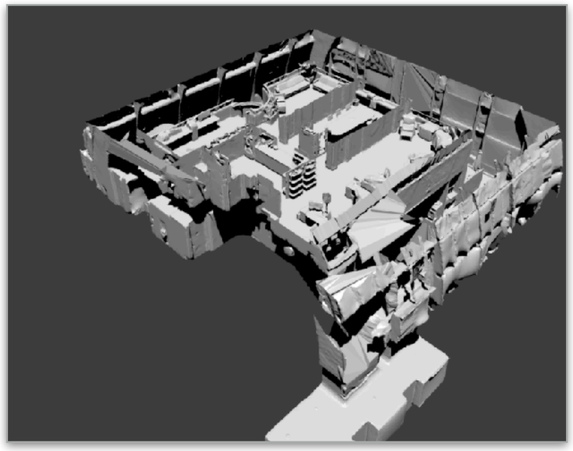


Gibson: acquire/render real world data

- Dataset acquired via 3D scanning (3D mesh + texture)
- Geometry, normals, semantics, + “photorealistic” 3D

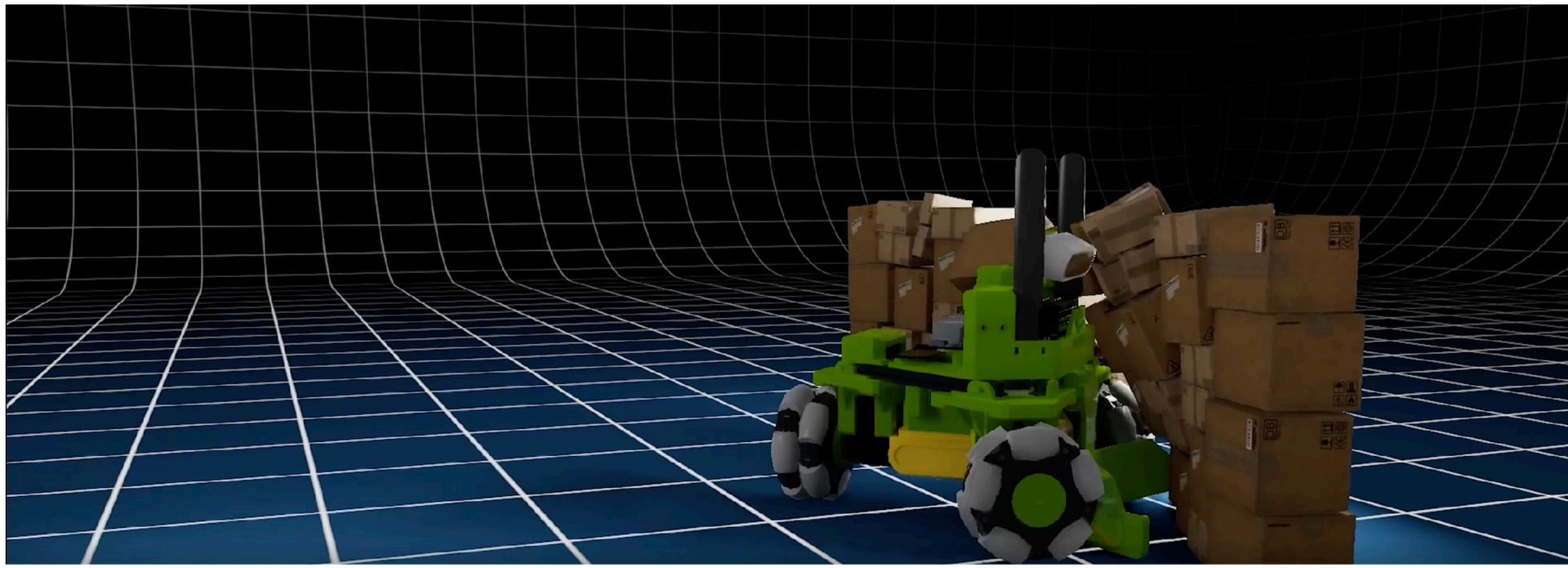


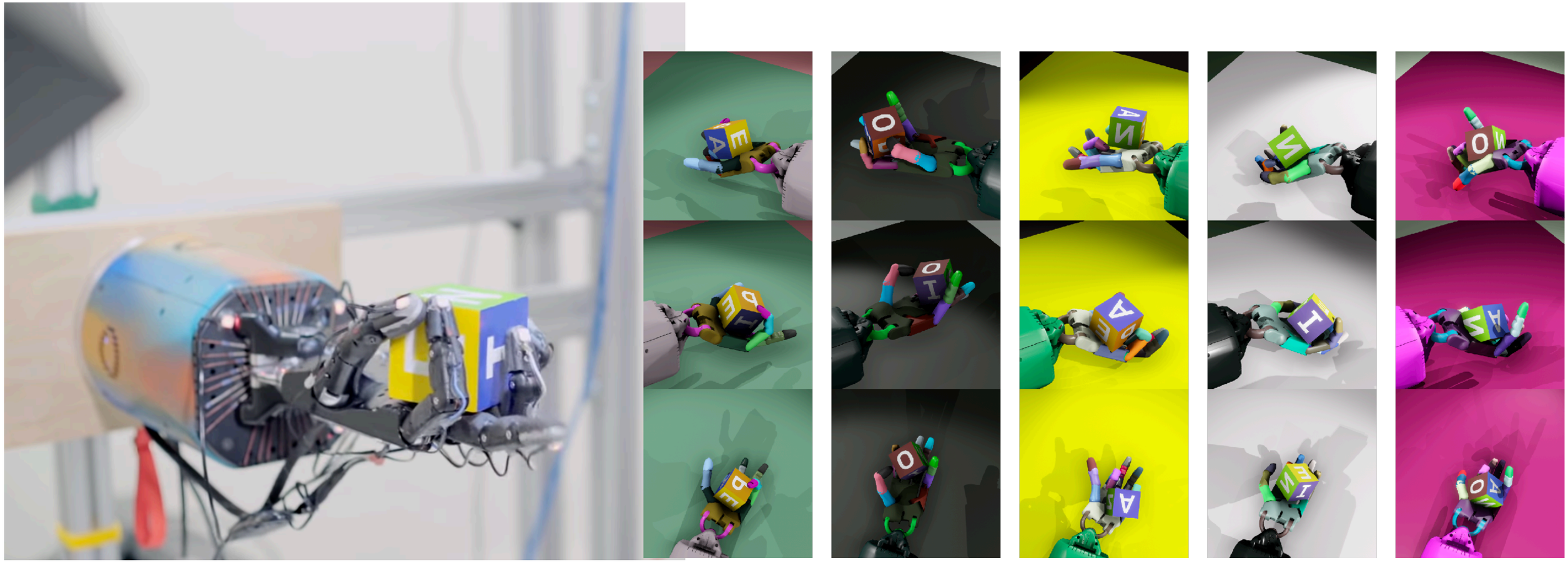




Enhancing CG images using learned image-to-image transfer

Physics simulation





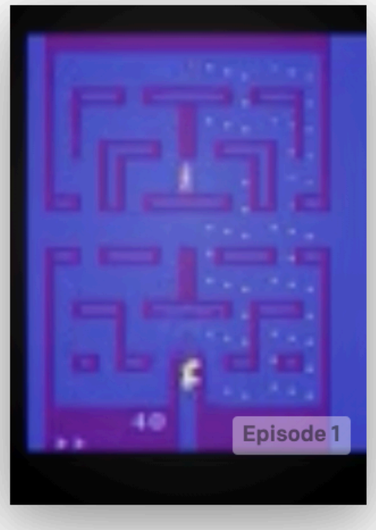
OpenAI gym: Atari games

Atari

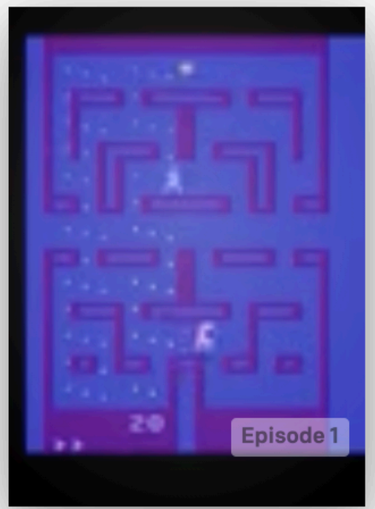
Reach high scores in Atari 2600 games.

AirRaid-ram-v0
Maximize score in the game AirRaid, with RAM as input

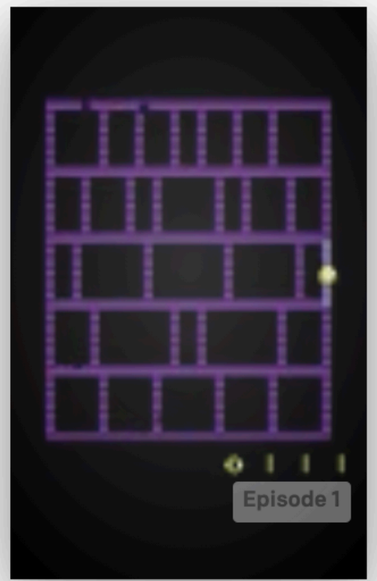
AirRaid-v0
Maximize score in the game AirRaid, with screen images as input



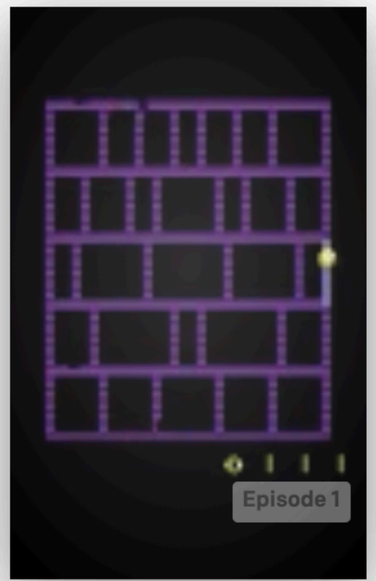
Alien-ram-v0
Maximize score in the game Alien, with RAM as input



Alien-v0
Maximize score in the game Alien, with screen images as input



Amidar-ram-v0
Maximize score in the game Amidar, with RAM as input



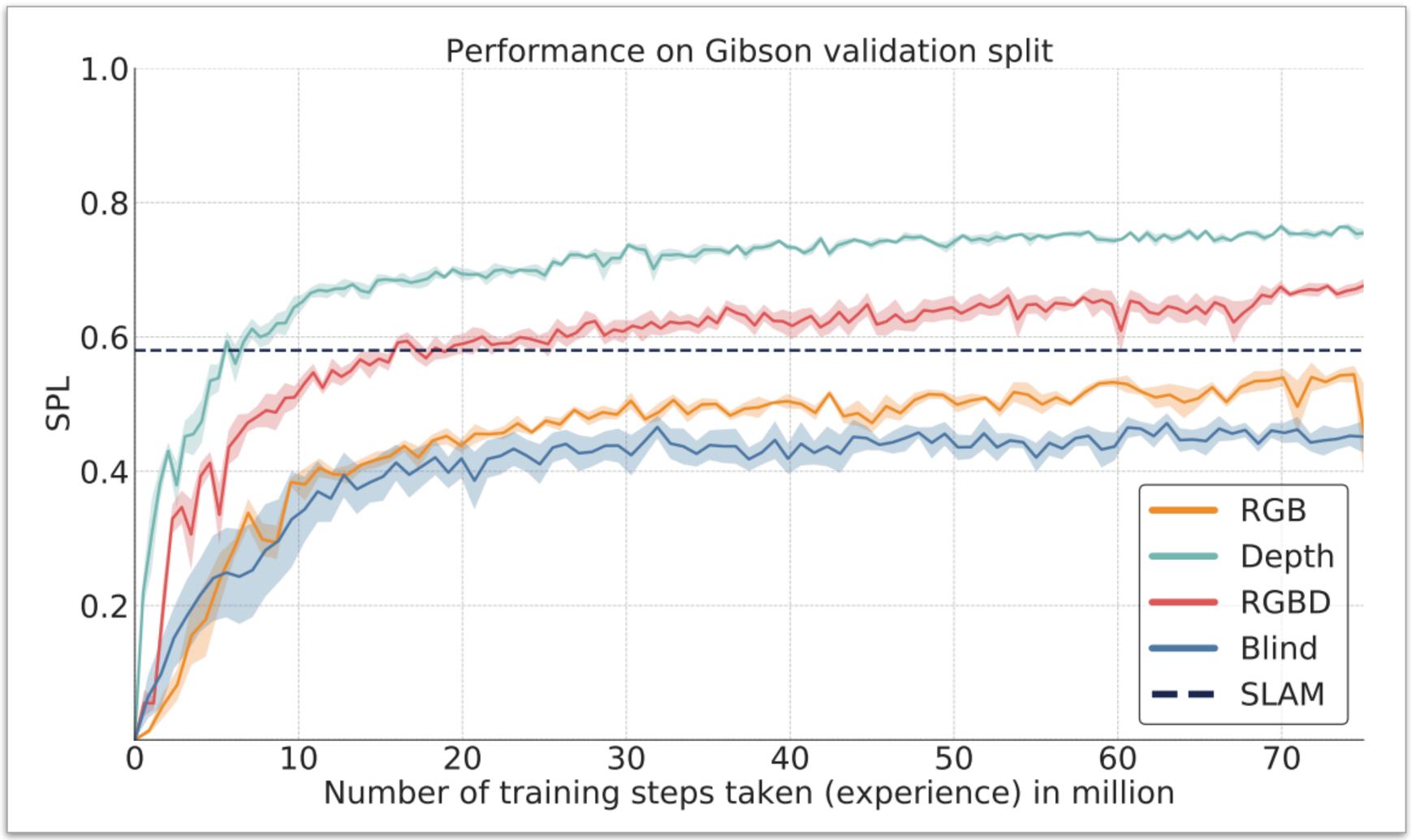
Amidar-v0
Maximize score in the game Amidar, with screen images as input

OpenAI's "OpenAI 5" Dota 2 bot

	OPENAI FIVE
CPUs	128,000 preemptible CPU cores on GCP
GPUs	256 P100 GPUs on GCP
Experience collected	~180 years per day (~900 years per day counting each hero separately)
Size of observation	~36.8 kB
Observations per second of gameplay	7.5
Batch size	1,048,576 observations
Batches per minute	~60

Need significant amounts of simulated experience

Example: even for simple PointGoal navigation task: need billions of steps of "experience" to exceed traditional non-learned approaches



Deeper dive:

Accelerating reinforcement learning

RL in 30 seconds

Model Inference

environment observation
e.g. RGB image → π_θ → agent action

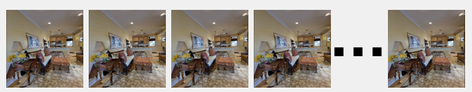
RL in 30 seconds

Model Inference

environment observation e.g. RGB image → π_θ → agent action

Model Training

sequence of observations



sequence of agent actions

compute loss gradients

π_θ

update model via SGD

Reward: change in distance from goal

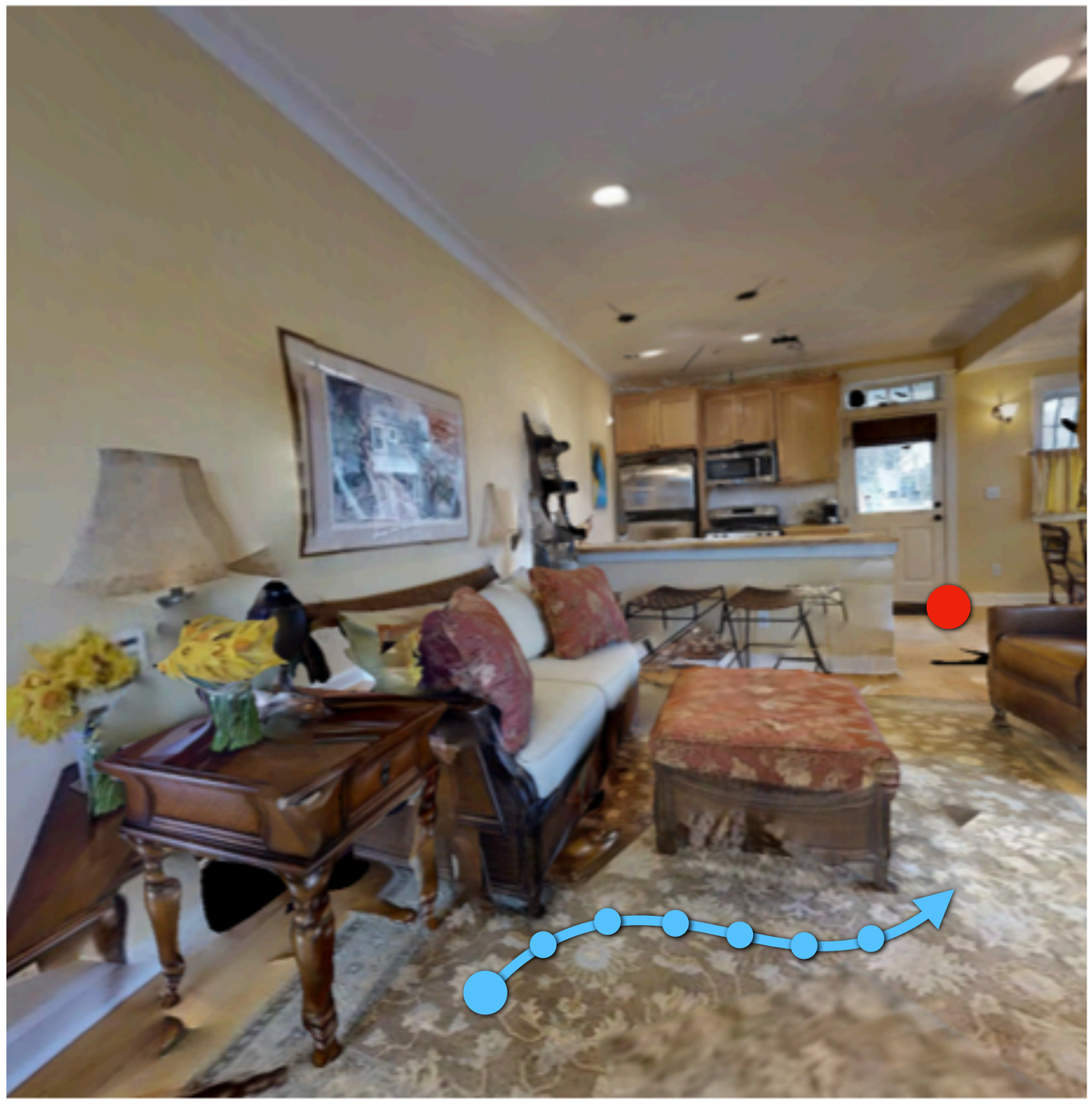
RL in 30 seconds

Model Inference

environment observation e.g. RGB image → π_θ → agent action

Model Training

Rollout → compute loss gradients π_θ → update model via SGD

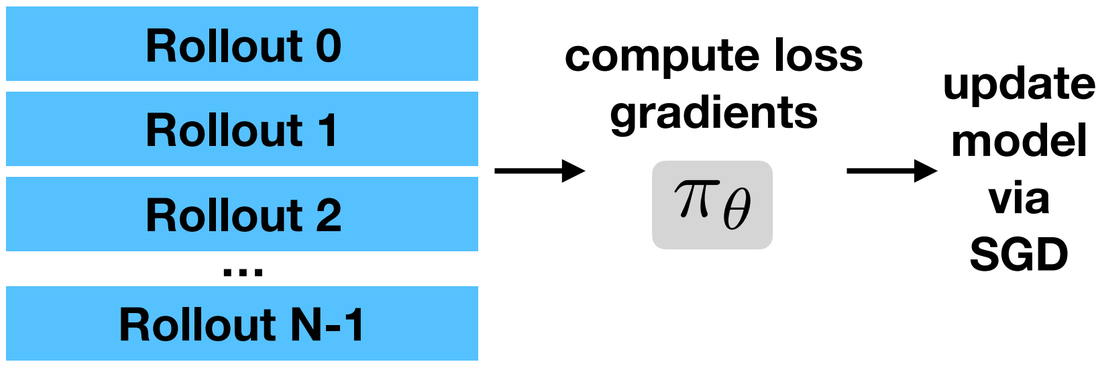


RL in 30 seconds

Many rollouts:

- **Agents independently navigating same environments**

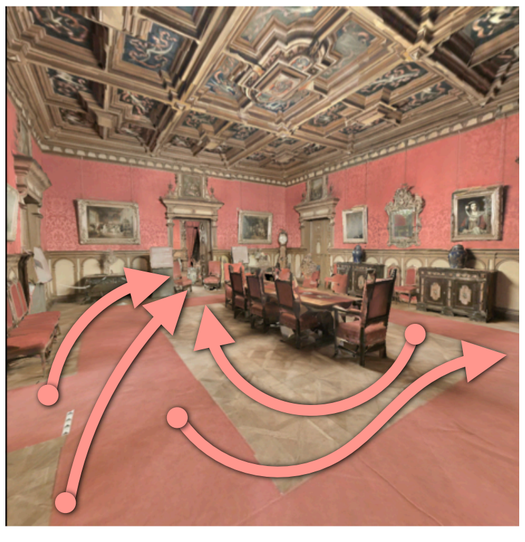
Batch Model Training



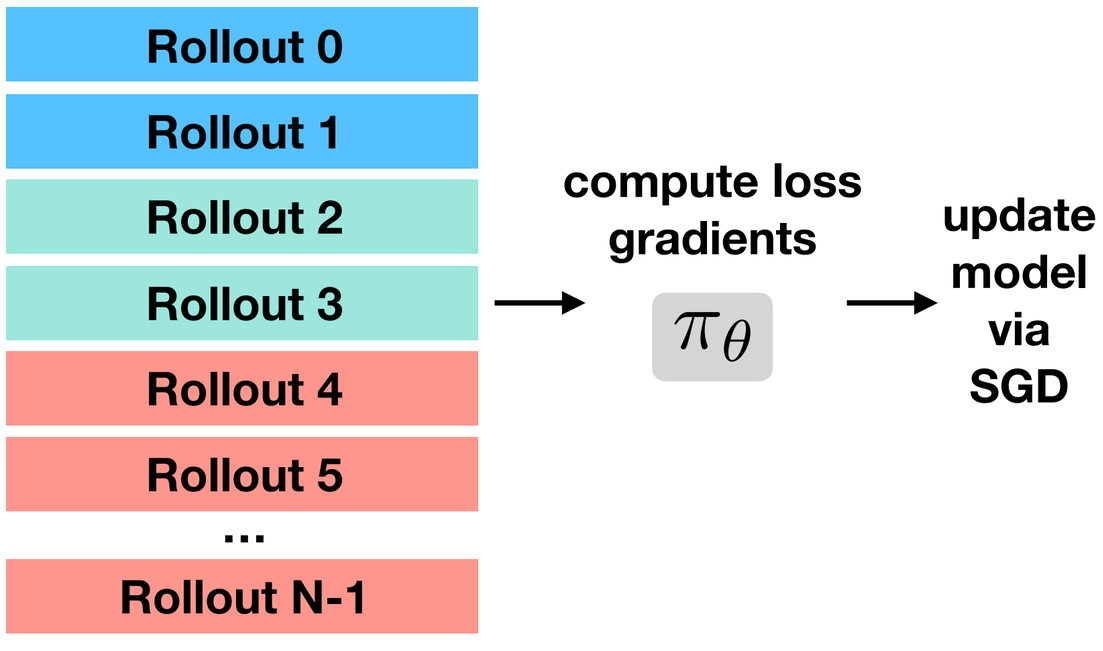
RL in 30 seconds

Many rollouts:

- **Agents independently navigating same environments**
- **Or different environments**



Batch Model Training



Workload summary

- Within a rollout
  - For each step of a rollout:
  - Render -> Execute policy inference -> simulate next world state

- Across *many* independent rollouts
  - Simulated agents may (or may not) share scene state
  - Diversity in scenes in a batch of rollouts is desirable to avoid overfitting, sample efficiency of learning

System components

Database of 3D assets (meshes, textures collision meshes)

Viewpoints, scene object positions

“Simulator”
(updates position of agent in scene,
detects collisions with scene geometry)

Renderer
(render scene from viewpoint of agent)

Non-rendered state: position, compass…

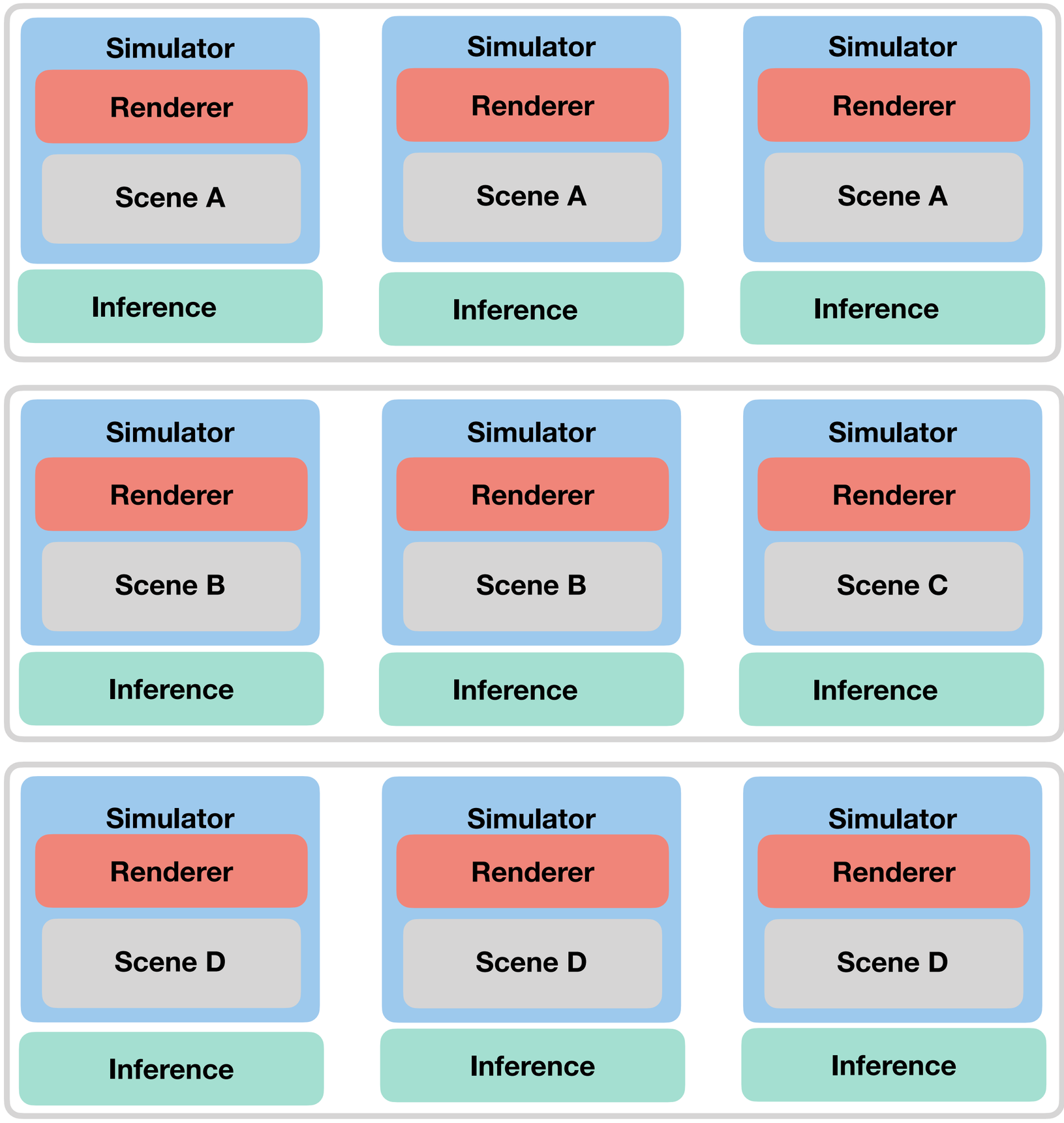
Rendered frames

Inference/Learning
(inference: action from rendered image,
learning: update policy model from rollouts)

π_θ

Next action

Basic design: parallelize over workers



Ask yourself:

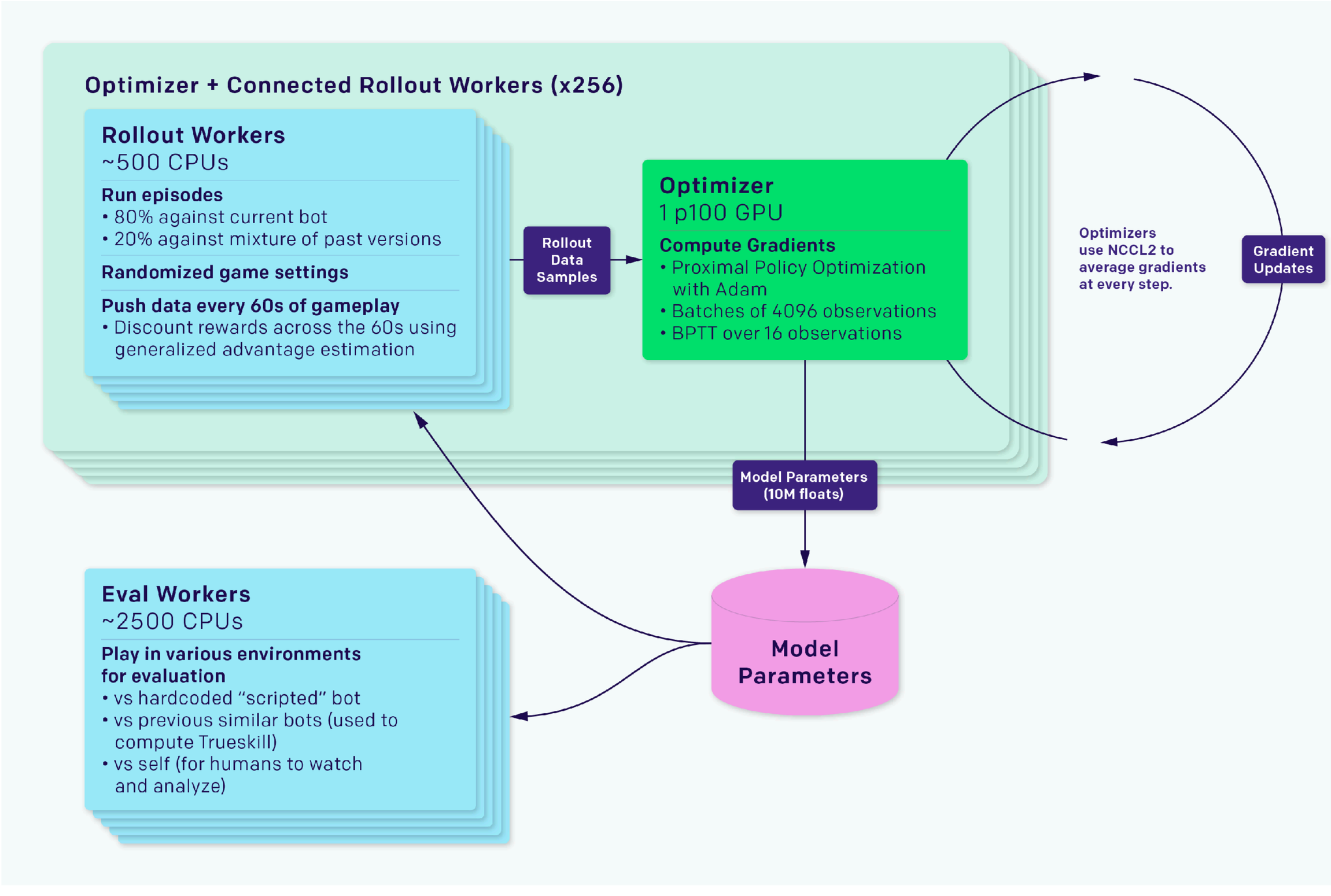
1. **What data gets communicated?**
2. **Can the system scale to sufficient parallelism?**
3. **Are there sync bottlenecks**

Learning

learning: update policy model from rollouts)

π_θ

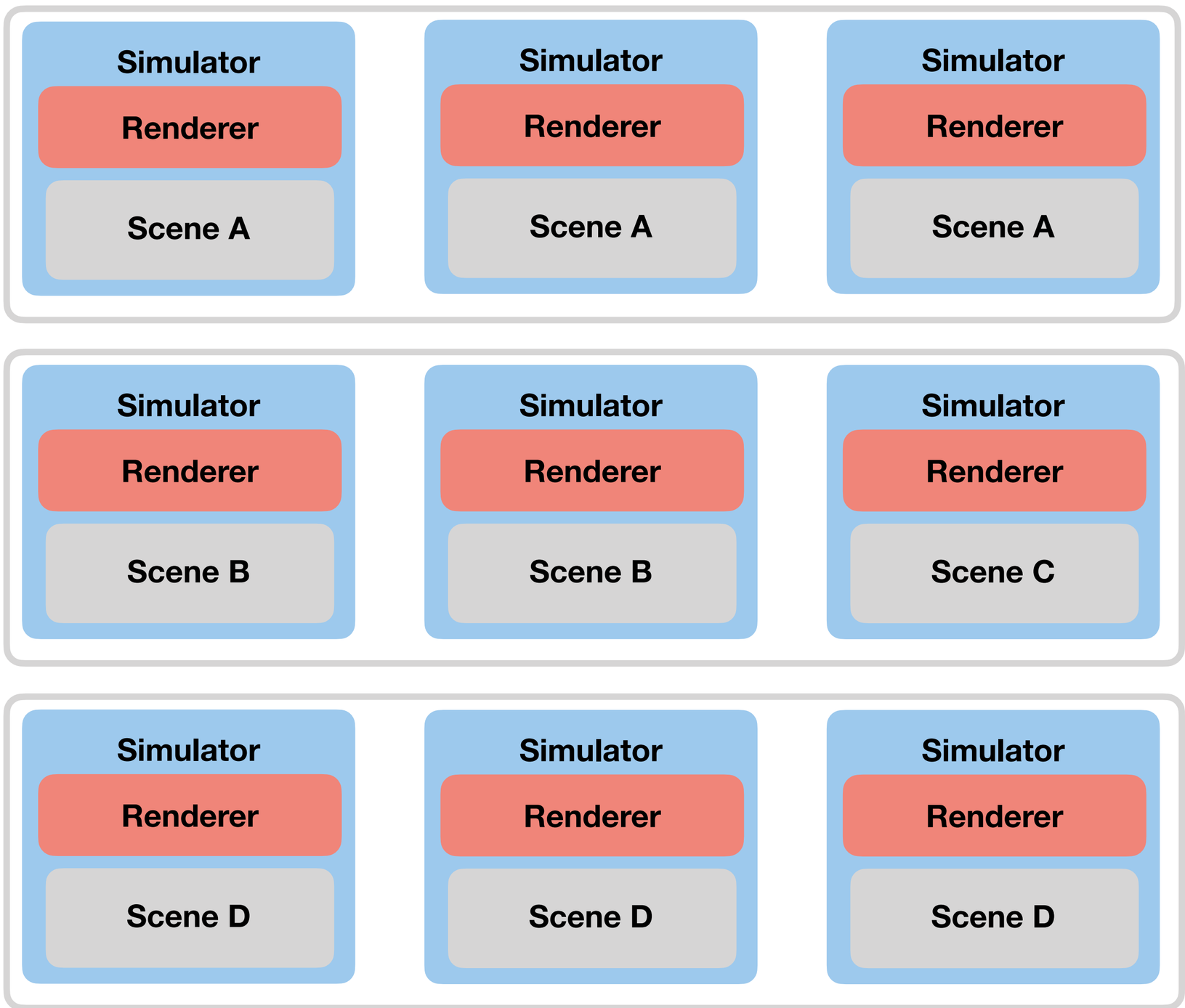
Example: Rapid (OpenAI)

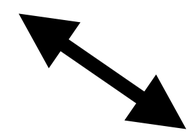


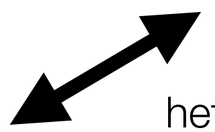
Design issues

- Expensive communication of weights from learner node to workers
- Worker nodes inefficiently run inference
  - May run on CPU if simulation code on workers doesn't require GPU (use cheap worker nodes that don't feature GPUs)
  - Run inference on small batches since each worker is running one rollout sim

Centralize inference AND training







Batch Inference/ Learning

(inference: action from rendered image, learning: update policy model from rollouts)

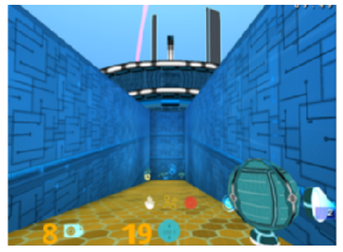
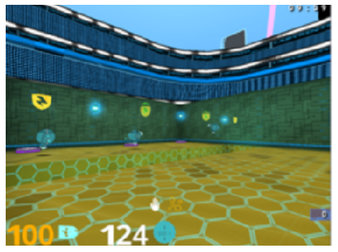
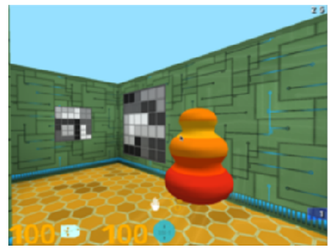
π_θ

Efficient batch inference/training
Centralization enables heterogeneity (e.g., use TPU for training)

Advantages

- No communication of model weights between workers and learner
- Must communicate simulation state — surprisingly this can be compact (object locations, smaller rendered image)
- Can use efficient batch inference in a centralized location (batch over rollouts from many workers)
- Can use machine optimized for DNN operations in centralized location — e.g., run on a TPU

REED RL

Architecture	Accelerators	Environments	Actor CPUs	Batch Size	FPS	Ratio
DeepMind Lab						
IMPALA	Nvidia P100	176	176	32	30K	—
SEED	Nvidia P100	176	44	32	19K	**0.63x**
SEED	TPU v3, 2 cores	312	104	32	74K	**2.5x**
SEED	TPU v3, 8 cores	1560	520	48[1]	330K	**11.0x**
SEED	TPU v3, 64 cores	12,480	4,160	384[1]	2.4M	**80.0x**
Google Research Football						
IMPALA, Default	2 x Nvidia P100	400	400	128	11K	—
SEED, Default	TPU v3, 2 cores	624	416	128	18K	**1.6x**
SEED, Default	TPU v3, 8 cores	2,496	1,664	160[3]	71K	**6.5x**
SEED, Medium	TPU v3, 8 cores	1,550	1,032	160[3]	44K	—
SEED, Large	TPU v3, 8 cores	1,260	840	160[3]	29K	—
SEED, Large	TPU v3, 32 cores	5,040	3,360	640[3]	114K	**3.9x**
Arcade Learning Environment						
R2D2	Nvidia V100	256	N/A	64	85K[2]	—
SEED	Nvidia V100	256	55	64	67K	**0.79x**
SEED	TPU v3, 8 cores	610	213	64	260K	**3.1x**
SEED	TPU v3, 8 cores	1200	419	256	440K[4]	**5.2x**

Design issues

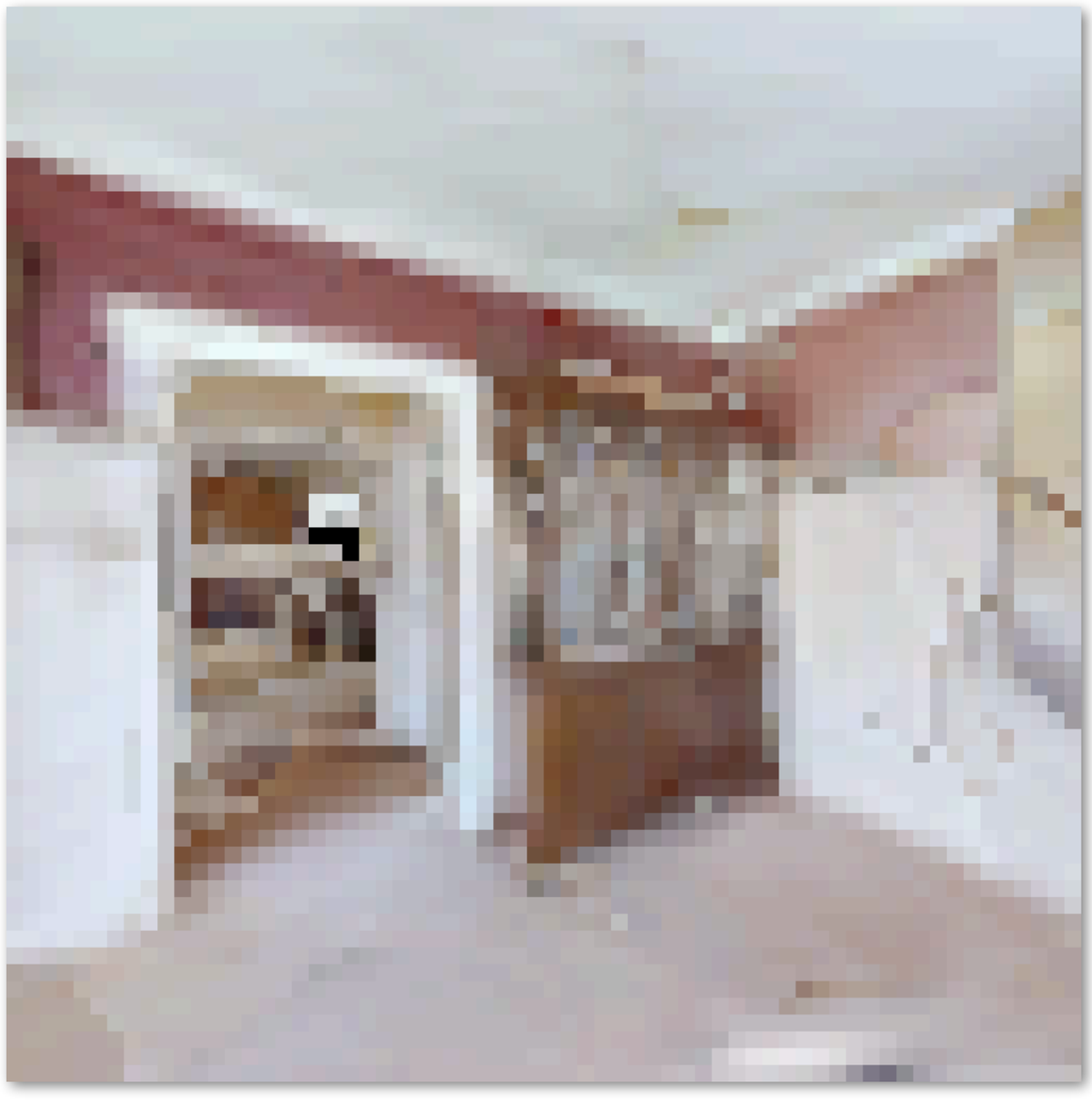
- **Inefficient simulation/rendering: rendering a small image does not make good use of a modern GPU (rendering throughput is low)**

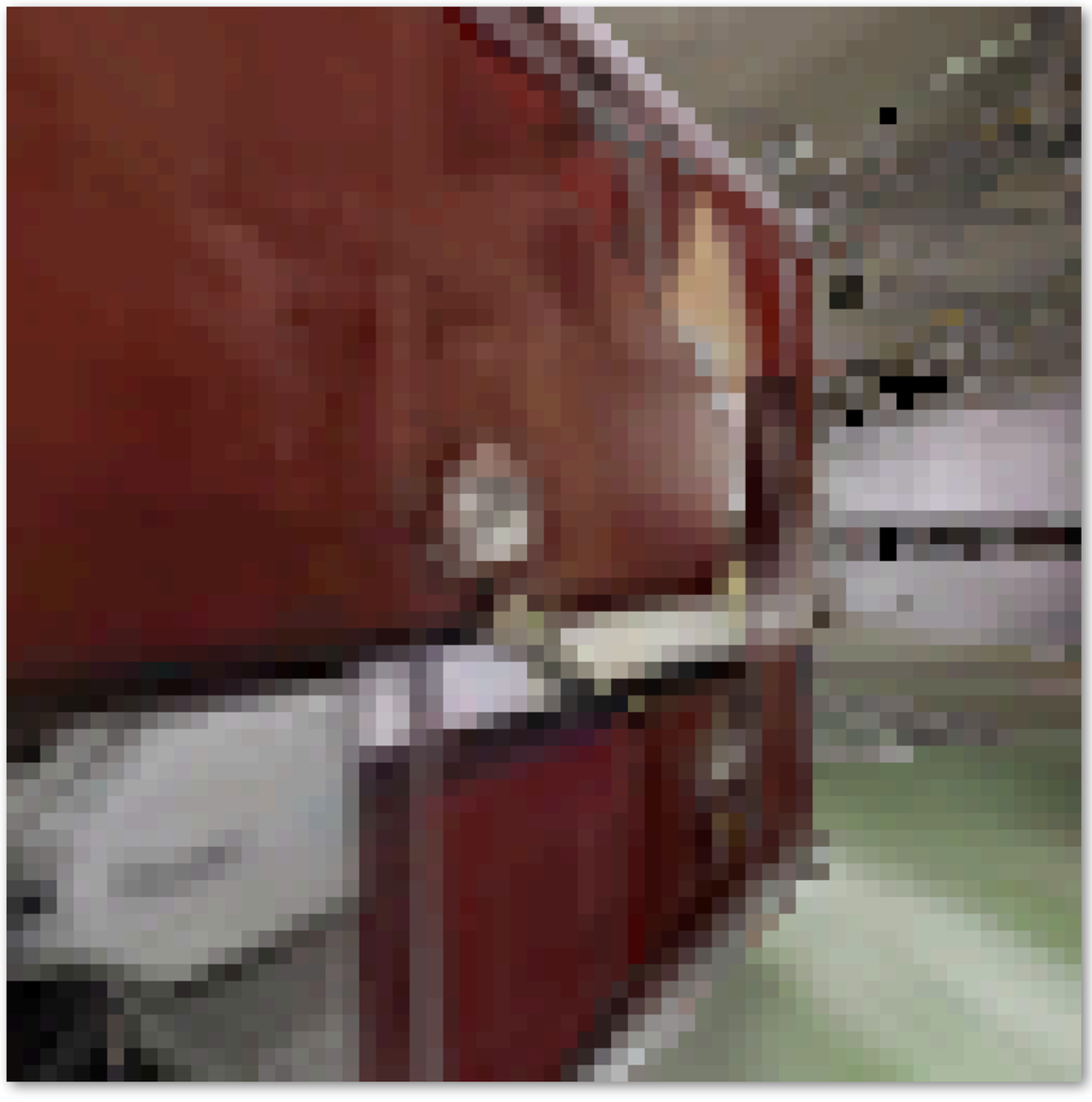
- **Duplication of computation and memory footprint (for scene data) across renderer/simulator instances**

What modern renderers are designed to render

(complex scenes at high resolution)

Low-resolution images with pre-captured lighting (from Gibson): clearly not state-of-the-art rendering! ;-)





Often the best way to reduce communication / increase efficiency is often to make the best possible use out of one node

Can we make simulation faster?

AI Habitat

- **Focus on high-performance rendering/simulation to enable order of magnitude longer RL training runs**

The table below reports performance statistics for a test scene from the Matterport3D dataset (id `17DRP5sb8fy`) on a `Xeon E5-2690 v4 CPU` and `Nvidia Titan Xp`. Single-thread performance reaches several thousand frames per second, while multi-process operation with several independent simulation backends can reach more than 10,000 frames per second on a single GPU!

	1 proc			3 procs			5 procs		
Sensors / Resolution	**128**	**256**	**512**	**128**	**256**	**512**	**128**	**256**	**512**
RGB	4093	1987	848	10638	3428	2068	10592	3574	2629
RGB + depth	2050	1042	423	5024	1715	1042	5223	1774	1348
RGB + depth + semantics*	709	596	394	1312	1219	979	1521	1429	1291

Previous simulation platforms that have operated on similar datasets typically produce on the order of a couple hundred frames per second. For example Gibson reports up to about 150 fps with 8 processes, and MINOS reports up to about 167 fps with 4 threads.

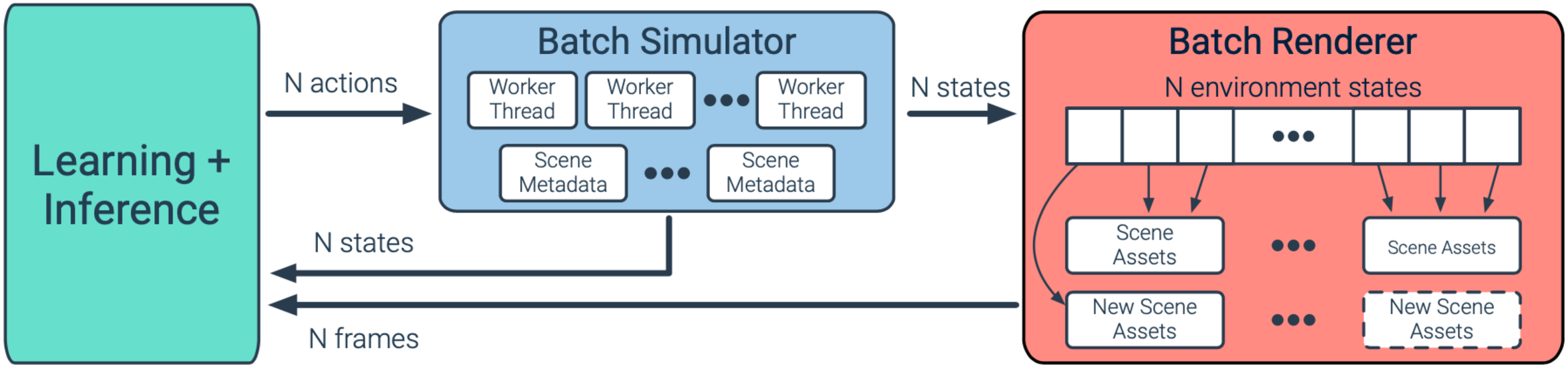
Prior was still using simulators (game engines) designed to render large high-resolution images for human eyes.

How would you design an engine "from the ground up" for the RL workload?

Main idea: design a renderer that executes rendering for 100s-1000's of unique rollouts in a single request

Inference/training, simulation, and rendering all operate on batches of N requests (rollouts)

Efficient bulk communication between three components



Example renderer output (PointNav task)

Opportunities provided by a batch rendering interface

- Wide parallelism: rendering each scene in a batch is independent
  - "Fill up" large parallel GPU with rendering work
  - Enables graphics optimizations like pipelining frustum culling (removing off-screen geometry before drawing it) for one environment with rendering of another

- Footprint optimizations: rendering requests in a batch can share same geometry assets
  - Significantly reduces memory footprint, enables large batch size
  - N ~ 256-1024 (per GPU) in our experiments: fills up large GPU
  - Limit number of unique scenes in a batch to $K \ll N$ scenes.
    - GPU RAM and scene size determines K

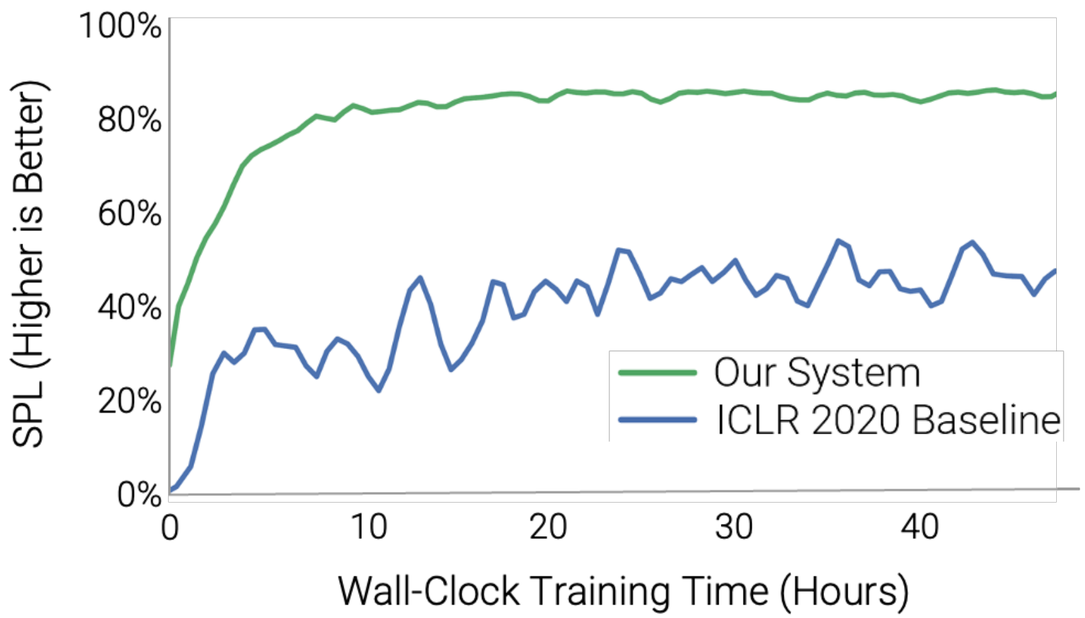
- Amortize communication: rendering requests in a batch can be packaged and drawn together
  - Render frames in batch to tiles in a single large frame buffer to avoid state update

Also, simultaneously optimize policy DNN

- DNN design/engineering (DNN encoder followed by policy LSTM)
- Reduce resolution of rendered input to from 128x128 to 64x64
- Move to ResNet9-based visual encoder from ResNet50
- Replace key layers with performant alternatives (e.g. replace normalization with Fixup Initialization)
- Adjust learning rates and use Lamb optimization

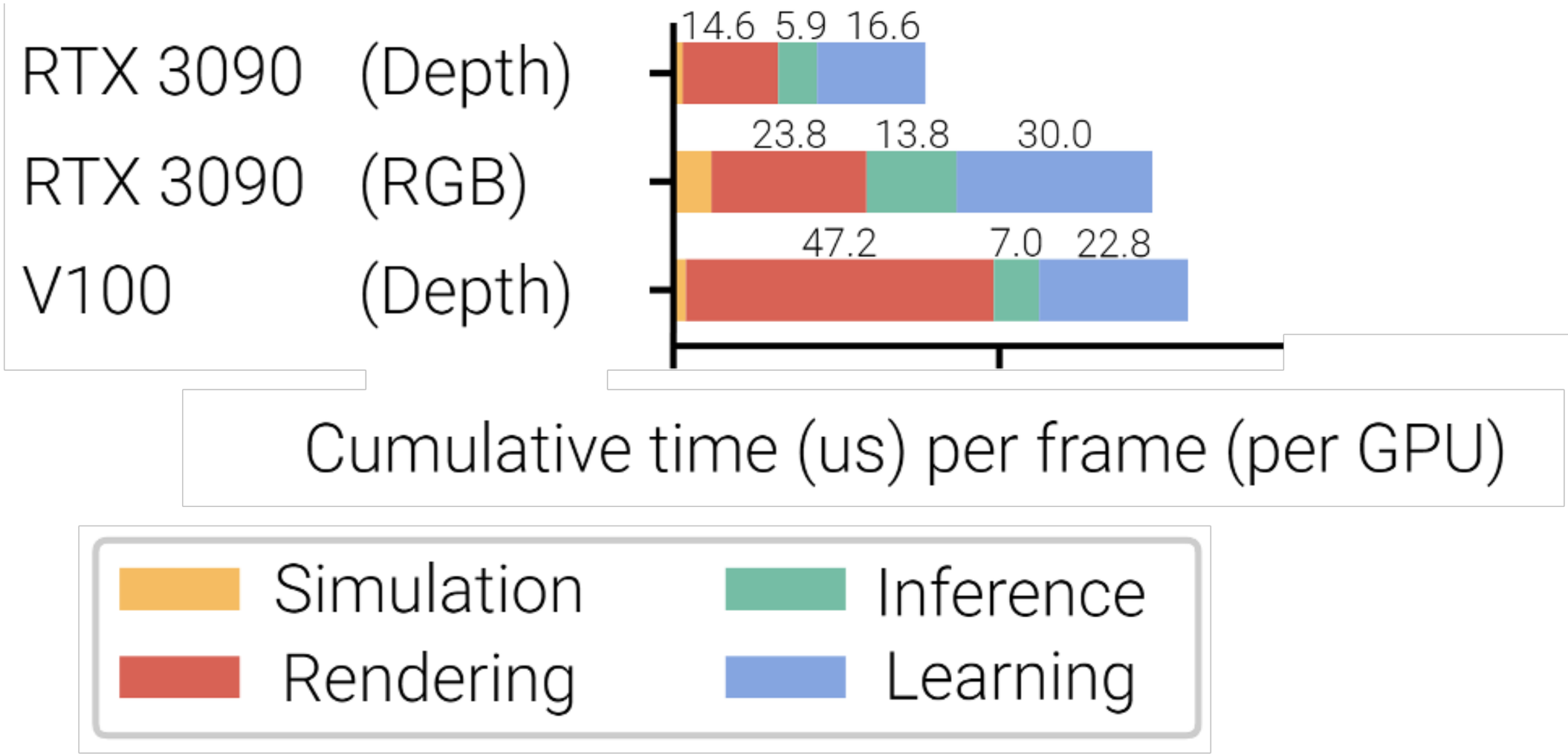
Example: 10,000+ FPS render→infer→train on a single GPU *

Sensor	System	CNN	Agent Res.	RTX 3090	RTX 2080Ti	Tesla V100	8×2080Ti	8×V100
Depth	BPS	SE-ResNet9	64	19900	12900	12600	72000	46900
	BPS-R50	ResNet50	128	2300	1400	2500	10800	18400
	WIJMANS++	SE-ResNet9	64	2800	2800	2100	9300	13100
	WIJMANS20	ResNet50	128	180	230	200	1600	1360
RGB	BPS	SE-ResNet9	64	13300	8400	9000	43000	37800
	BPS-R50	ResNet50	128	2000	1050	2200	6800	14300
	WIJMANS++	SE-ResNet9	64	990	860	1500	4600	8400
	WIJMANS20	ResNet50	128	140	OOM	190	OOM	1320



*** But low resolution: 64x64 rendered output resolution**

Performance breakdown



Interesting (open) rendering/simulation systems research questions

- If you had to design a rendering/simulation system "from the ground up" to support ML model training, what would you do differently from a modern high-performance game engine?

- What new opportunities for performance optimization are there? (amortize rendering across multiple virtual sensors, agents, etc.)
  - What should the architecture/API to the renderer be?

- How much visual fidelity is needed to train models that transfer into the real-world?
  - Do we even need photorealistic quality to train policies that work in the real world?
  - If so, does ML-based image manipulation provide new opportunities to bridge the simulation to real world gap?

Project presentation expectations

Presentation day (next Thursday)

- Each group will present ~8 minutes
  - Remember: write-ups are due on Friday at 5pm
- Short talks are tricky, so here are some tips

- As this point:
  - If you are a performance-oriented project: are all your "baselines" in place
    - You can run a script and produce "us" vs. "baseline" graphs
  - If you are an applications project, do you have "any" result yet?
  - For all projects: do you have the right test cases (datasets) to show off success?

Benefit TO YOU of a good (clear) talk

- Non-linear increase in the impact of your work
  - Others are more likely to remember and build upon your work
  - Others are more likely to adopt your ideas
  - Others are more likely to come up to you after the talk
- Clarity is highly prized in the world: the audience will remember clear communicators
  - “Hey, that was a great talk yesterday… are you looking for a job anytime soon?”
  - “Hey, that was a great talk, I’m working on something that you might find helpful.”

Tip 1

Identify your audience

Easy: the people in this class!

Your #1 priority should be to be clear, rather than be comprehensive

(your project writeup is the place for completeness)

Everything you say should be understandable by someone in this class.
If you don't think the audience will understand, leave it out (or change).
(spend the time saying something we will understand)

This will be much harder than it seems.

Consider your audience

- Everyone in the audience knows about course readings/topics
  - Terminology/concepts we all know about need not defined (just say "remember we talked about X")

- Most of the audience knows little-to-nothing about the specific application domain or problem you are trying to solve
  - Application-specific terminology should be defined or avoided

- Everyone wants to know the "most interesting" thing that you found out or accomplished (your job is to define most interesting for them)

2.

Pick a focus.

Figure out what you want to say. Then say it.

(and nothing more)

A good speaking philosophy: “every sentence matters”

Tip: for each sentence, ask yourself:

What is the point I am trying to make?

Did the sentence I just say make that point?

Pick a focus

- In this class, different projects should stress different results
- Some projects may wish to show a flashy demo and describe how it works (proof by "it works")
- Other projects may wish to show a sequence of graphs (path of progressive optimization) and describe the optimization that took system from performance A to B to C
- Other projects may wish to clearly contrast parallel CPU vs. parallel GPU performance for a workload

Your job is not to explain what you did, but to explain what you think we should know

Tip 3

Set up the problem: establish inputs, outputs, and constraints (goals and assumptions)

Establish goals and assumptions early

- Given these inputs, we wish to generate these outputs
- We are working under the following constraints
  - Example: the outputs should have these properties
  - Example: the computer graphics algorithm…
    - Should run in real time
    - Should be parallelizable so it can run on a GPU
    - Should not require artist intervention to get good output
- Example: the system…
  - Need not compile all of Python, only this subset that we care about…
  - Should realize about 90% of the performance of hand-tuned code, with much lower development time

Why is knowing the goals and constraints important?

Your contribution is typically a system or algorithm that meets the stated goals under the stated constraints.

Understanding whether a solution is "good" requires having this problem context.

Example: 3D rendering problem

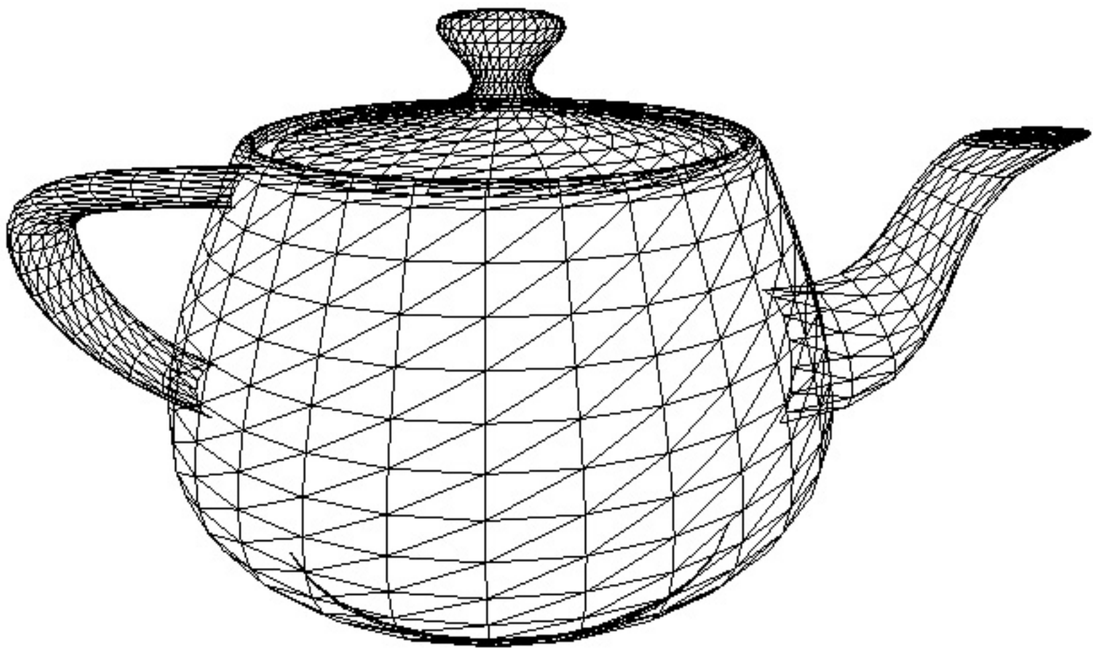


Image credit: Henrik Wann Jensen

Input: description of a scene:

3D surface geometry (e.g., triangle mesh)
surface materials, lights, camera, etc.

Output: image of the scene

Simple definition of rendering task: computing how each triangle in 3D mesh contributes to appearance of each pixel in the image?

Tip 4

Show, don't tell

It's much easier to communicate with figures/images than text

(And it saves the speaker a lot of work explaining)

Example:

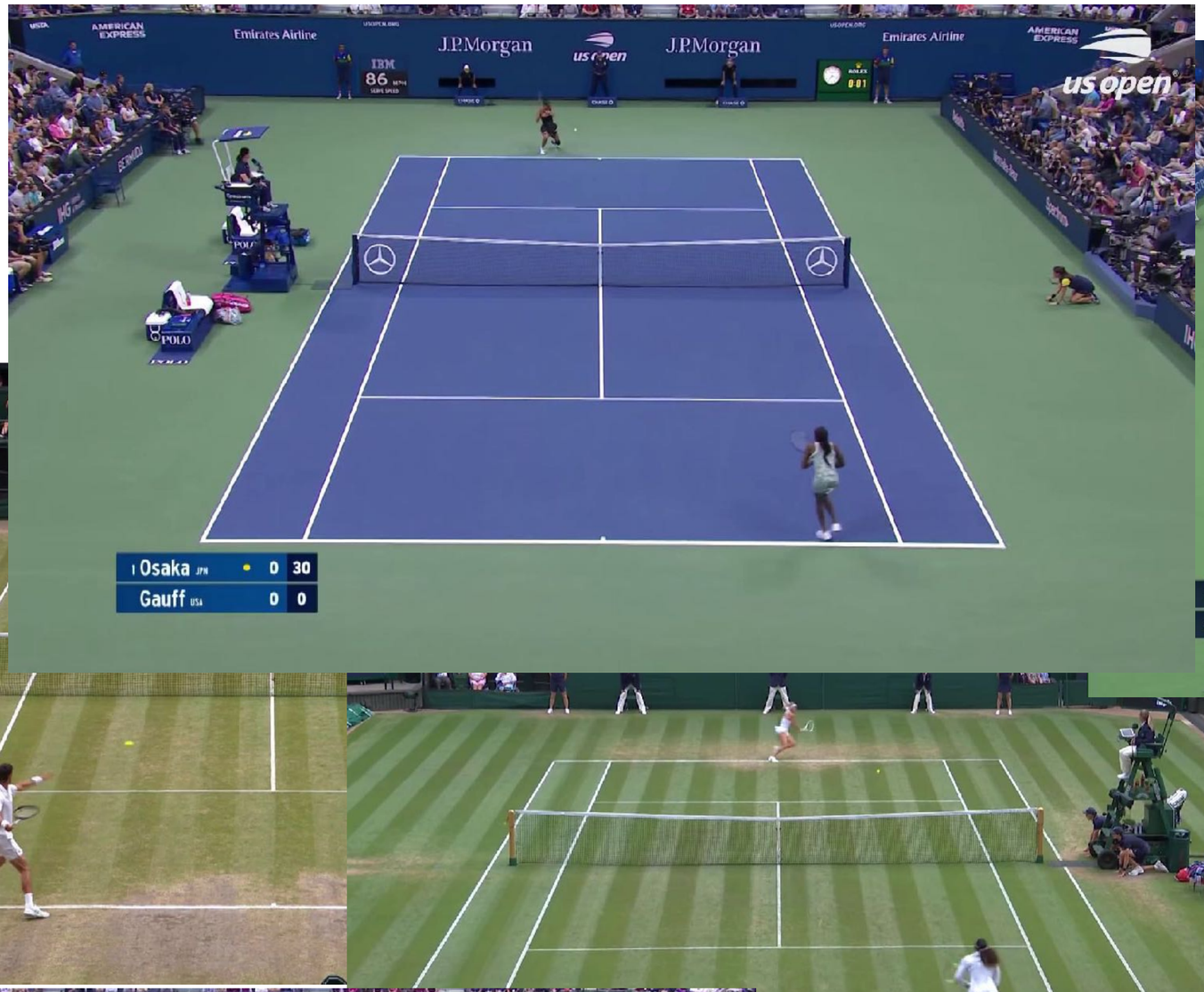
- In a recent project, we asked the question… given enough video of tennis matches of a professional athlete, could we come up with an algorithm for turning all this input video into a controllable video game character?

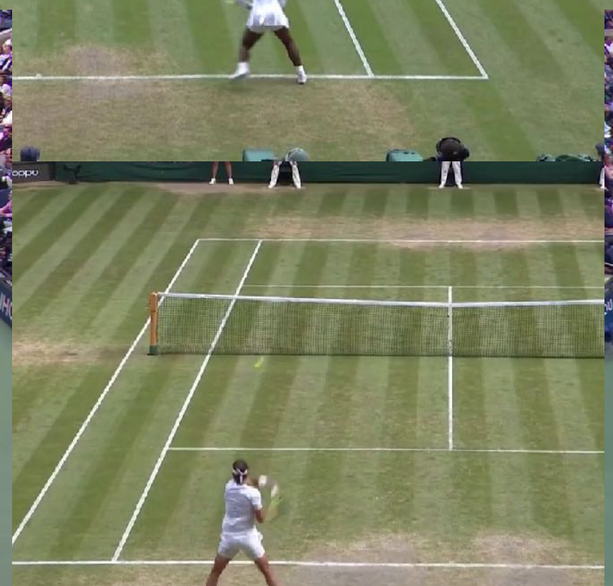
Compare the description above to the following sequence…

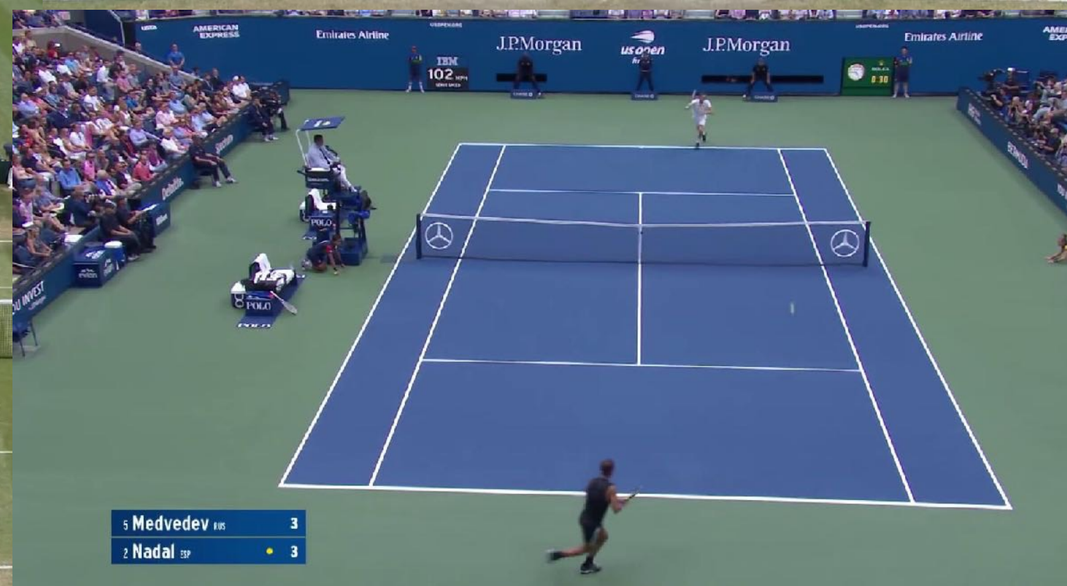
Here's an example of that source video

The best way to describe the input data than to just show it!

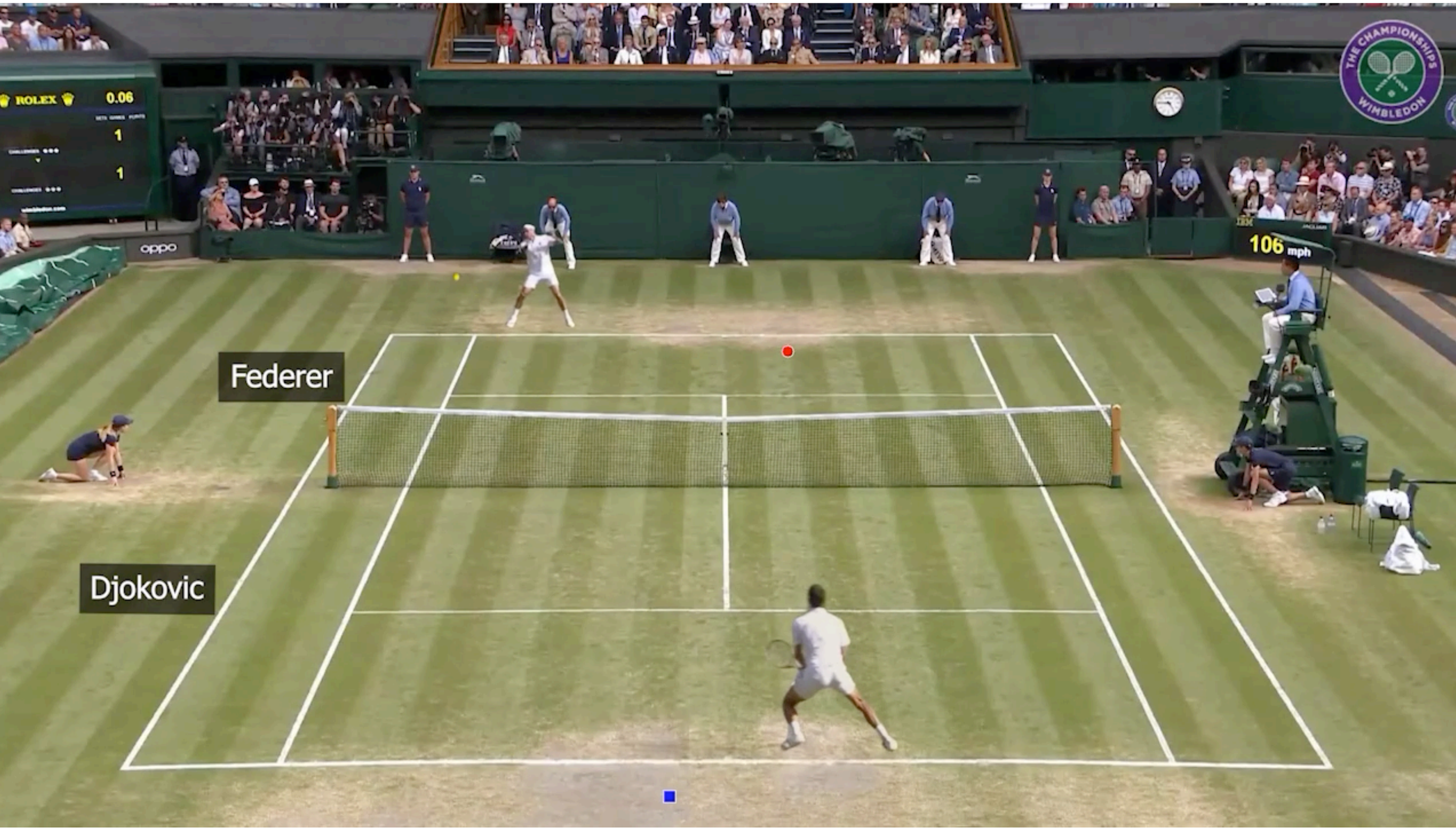
And there's a lot of it!







And here's an example of controllable output



The best way to describe the output we seek is just show the result of the system!

Another example:

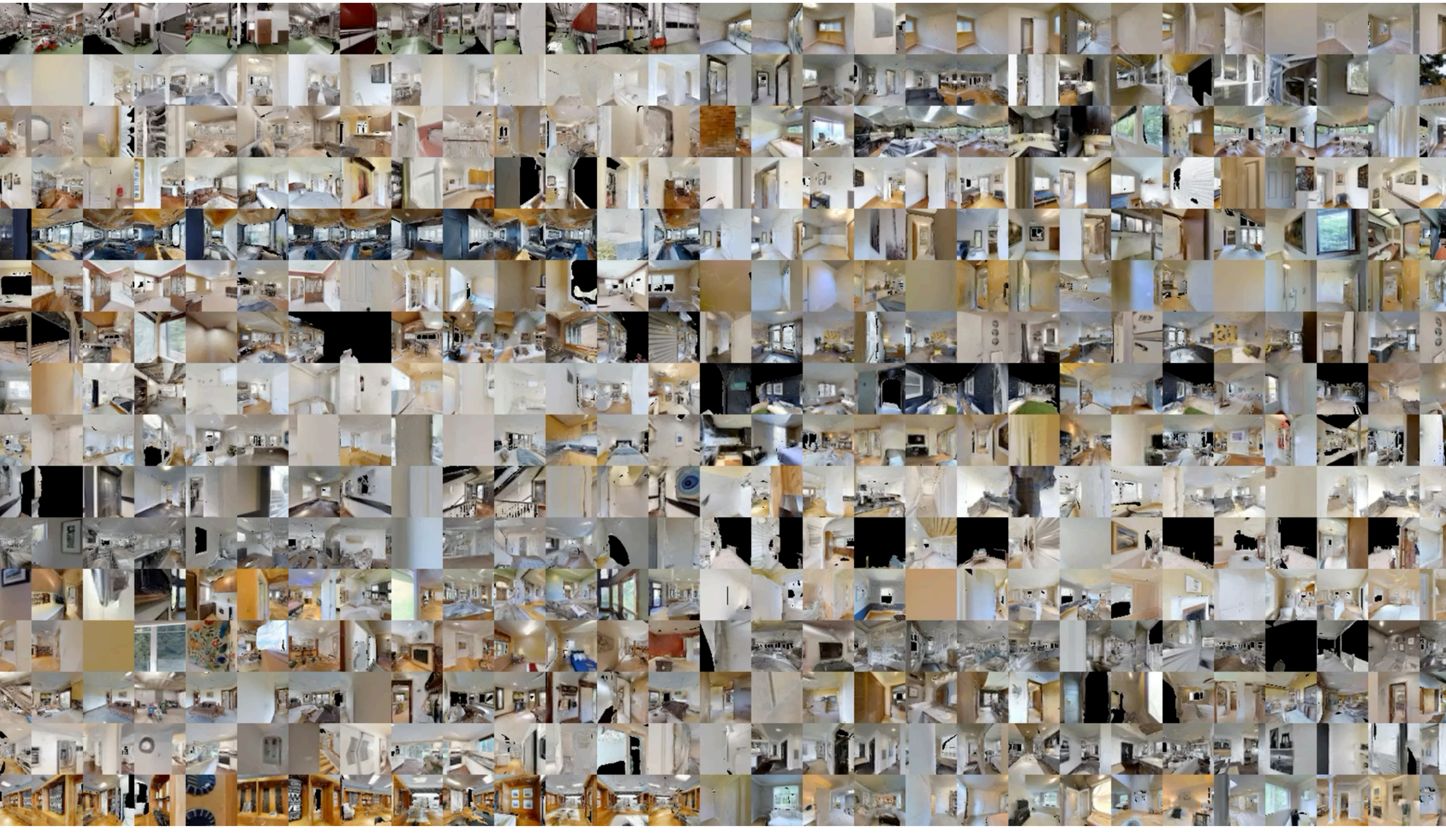
- We had a problem in this project: input videos were taken in different lighting conditions, and these lighting differences were the cause of bad results.

- An anticipated audience question: "what do you mean by lighting differences?"

The problem (lighting differences)

After the fix

Another example: a renderer that renders many views of the scene at the same time



Another example: it is difficult to train detectors

= positive examples

= negative examples

Tip 5

The audience prefers not to think (much)

An audience has a finite supply of mental effort

- **The audience <u>does not</u> want to burn mental effort about things you know and can just tell them.**
  - They want to be led by hand through the major steps of your story
  - They <u>do not</u> want to interpret any of your figures or graphs, they want to be directly told how to interpret them (e.g., what to look for in a graph).
  - They want to be told about your key assumptions

- **The audience <u>does</u> want to spend their energy thinking about:**
  - Potential problems/limitations with what you did (did you consider all edge cases? Is your evaluation sound?)
  - Implications of your approach to their work
  - Connections to their own work

Which leads me to…

The audience does not want to think about "why" you are telling them something.

Tip 6

Surprises* are almost always bad:

Say <u>where</u> you are going and <u>why</u> you must go there before you say <u>what</u> you did.

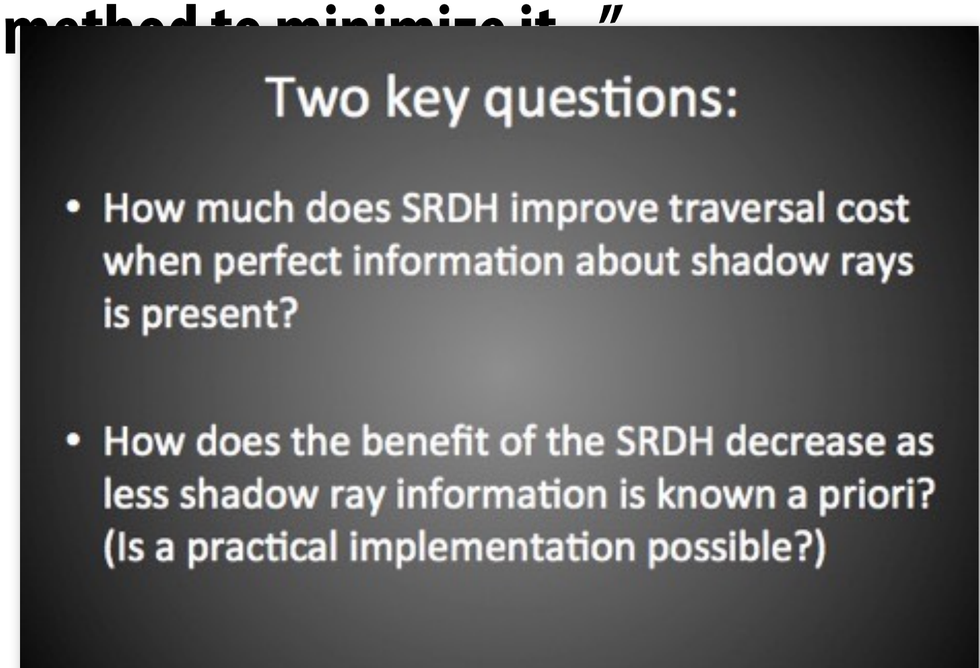
* I am referring to surprises in talk narrative and/or exposition. A surprising result is great.

Give the <u>why</u> before the <u>what</u>

- **Why provides the listener context for…**
  - Compartmentalizing: assessing how hard they should pay attention (is this a critical idea, or just an implementation detail?). Especially useful if they are getting lost.
  - Understanding how parts of the talk relate ("Why is the speaker now introducing a new optimization framework?")

- **In the algorithm description:**
  - "We need to first establish some terminology"
  - "Even given X, the problem we still haven't solved is…"
  - "Now that we have defined a cost metric we need a method to minimize it…"

- **In the results/evaluation:**
  - Speaker: "Key questions to ask about our approach are…"
  - Audience: "Thanks! I agree, those are good questions. Let's see what the results say!"



Big surprises in a narrative are a bad sign

- Ideally, you want the audience to always be able to <u>anticipate</u>* what you are about to say
  - This means: your story is so clear it's obvious!
  - It also means the talk is really easy to present without notes or text on slides (it just flows)

- If you are practicing your talk, and you keep forgetting what's coming on the next slide (that is, you can't anticipate it)…
  - This means: you probably need to restructure your talk because a clear narrative is not there.
  - It's not even obvious to you! Ouch!

* Credit to Abhinav Gupta for suggesting the term anticipation, and for the example on this slide

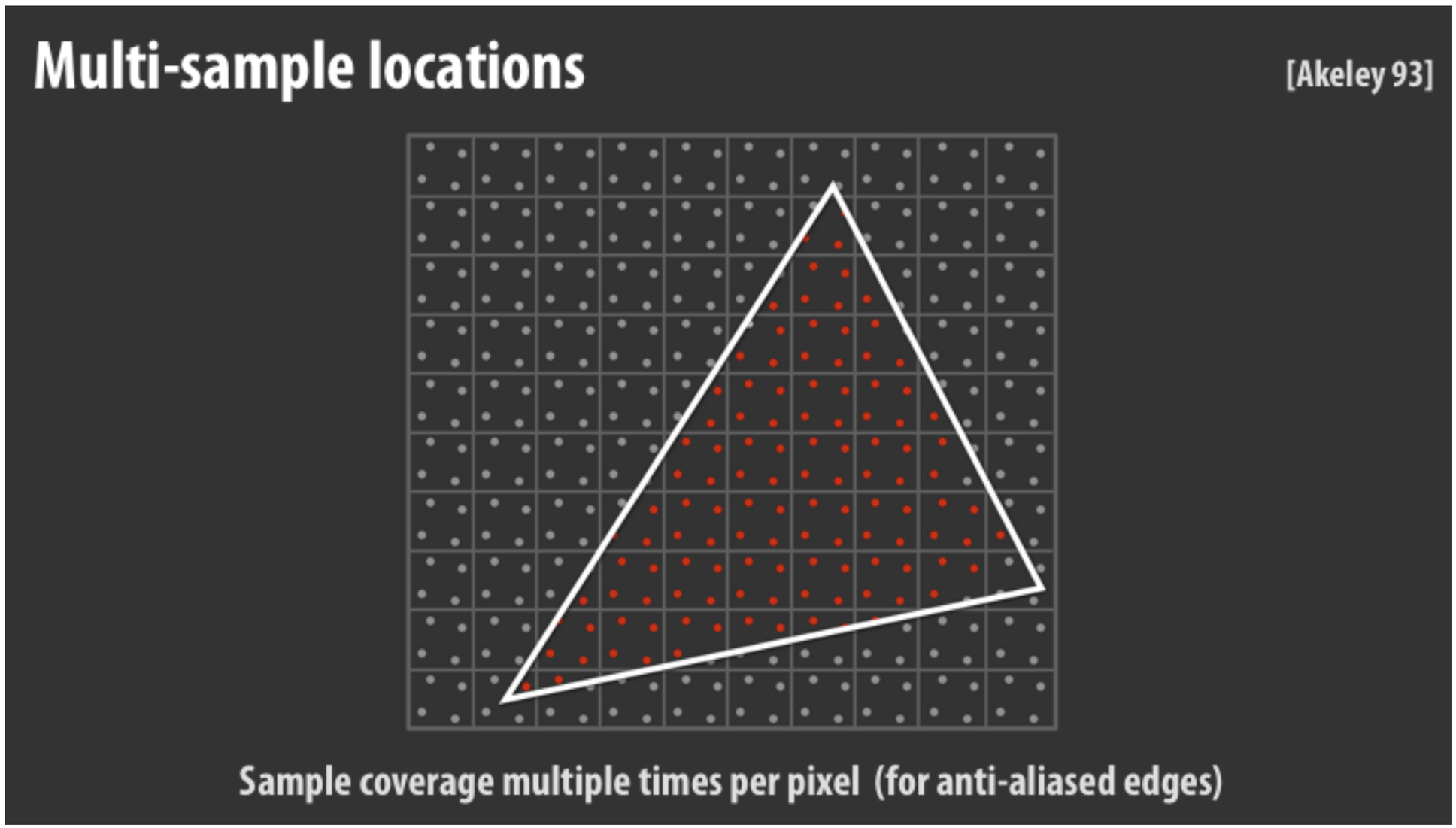
Tip 7

Always, always, always explain any figure or graph

(remember, the audience does not want to think about things you can tell them)

Explain every figure

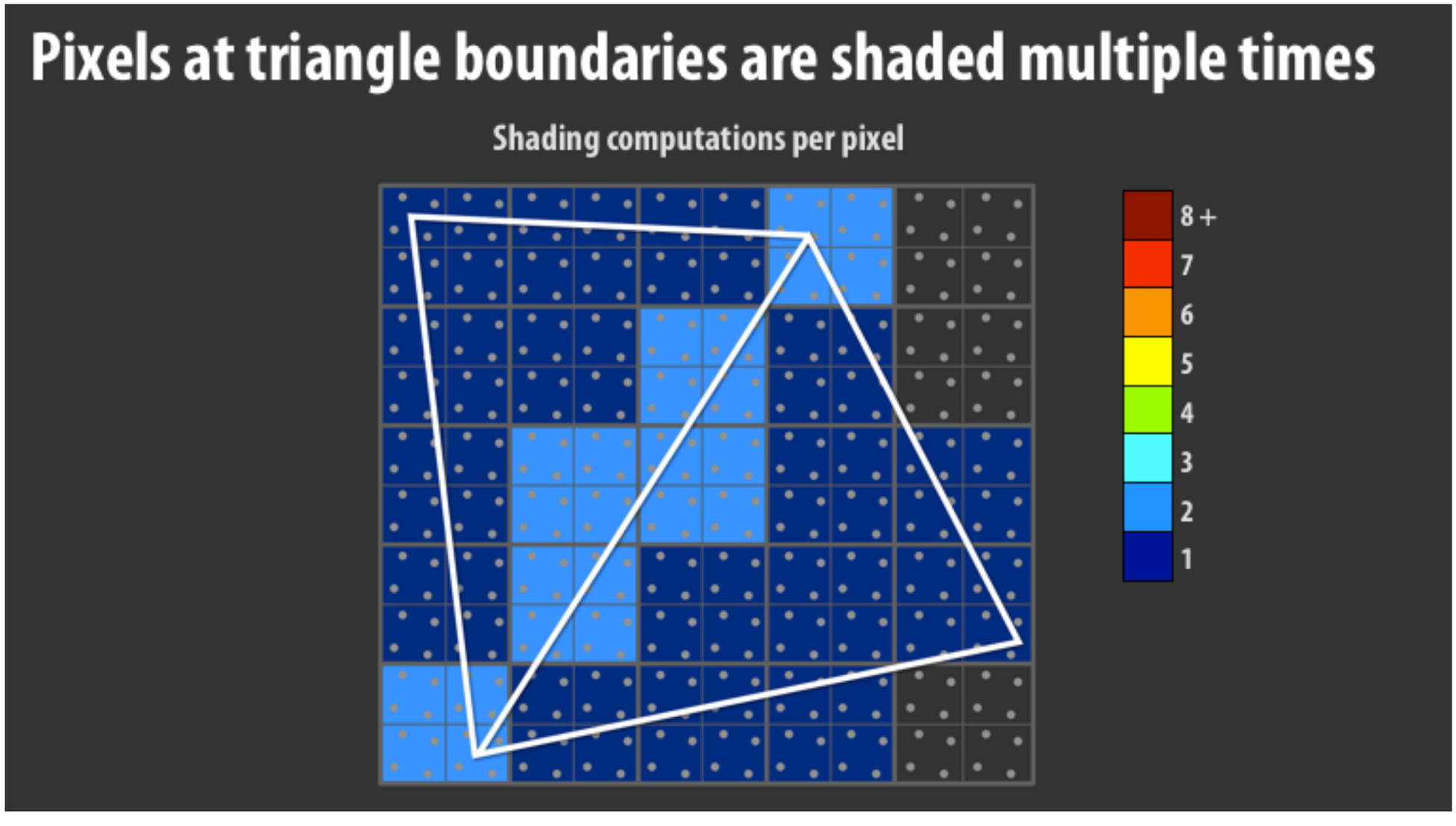
- Explain every visual element in the figure (never make the audience decode a figure)
- Refer to highlight colors explicitly (explain why the visual element is highlighted)



Example voice over: "Here I'm showing you a pixel grid, a projected triangle, and the location of four sample points at each pixel. Sample points falling within the triangle are colored red.

Explain every figure

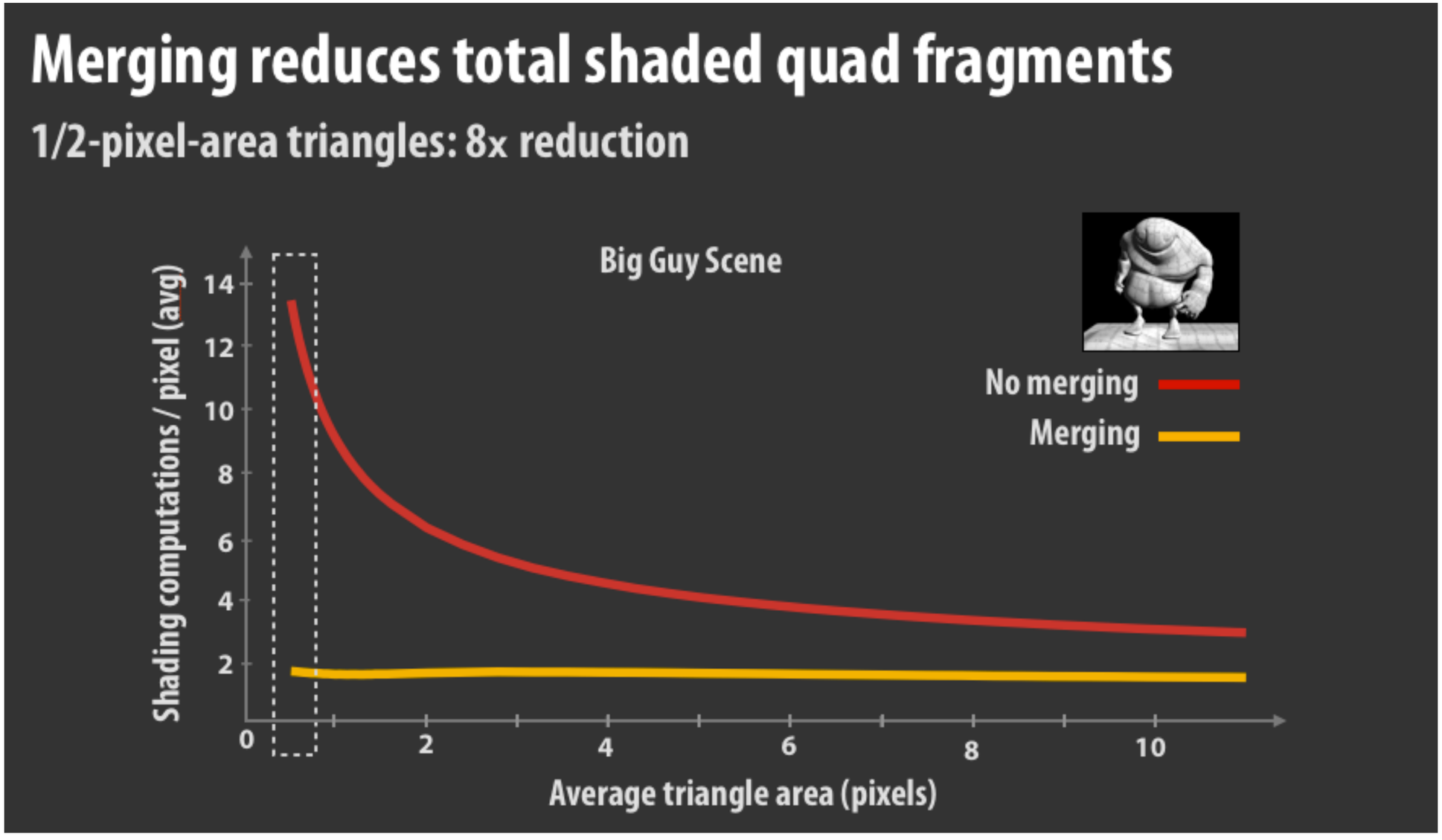
- Lead the listener through the key points of the figure
- Useful phrase: "As you can see…"
  - It's like verbal eye contact. It keeps the listener engaged and makes the listener happy… "Oh yeah, I can see that! I am following this talk!"



Example voice over: "Now I'm showing you two adjacent triangles, and I'm coloring pixels according to the number of shading computations that occur at each pixel as a result of rendering these two triangles. As you can see from the light blue region, pixels near the boundary of the two triangles get shaded twice.

Explain every results graph

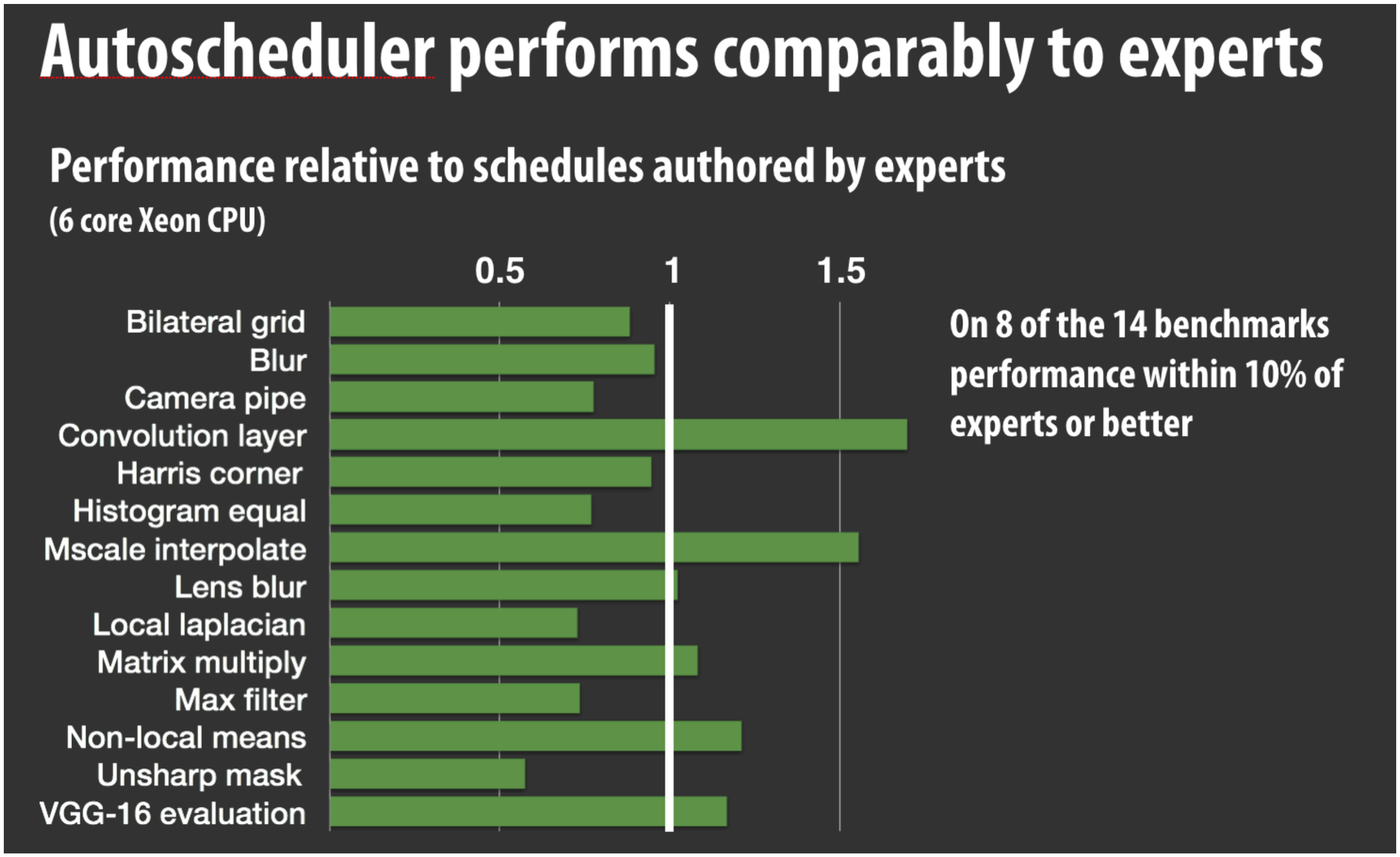
- May start with a general intro of what the graph will address ("anticipate" the result)
- Then describe the axes (and your axes better have labels!)
- Then describe the <u>one point</u> that you wish to make with this results slide



Example voice over: "Our first questions were about performance: how much did the algorithm reduce the number of the shading computations? And we found out that the answer is a lot. This figure plots the number of shading computations per pixel when rendering different tessellations of the big guy scene. X-axis gives triangle size. If you look at the left side of the graph, which corresponds to a high-resolution micropolygon mesh, <u>you can see</u> that merging, shown by yellow line, shades over eight times less than the convention pipeline.

Explain every results graph

- May start with a general intro of what the graph will address.
- Then describe the axes (your axes better have labels!)
- Then describe the <u>one point</u> that you wish to make with this results slide



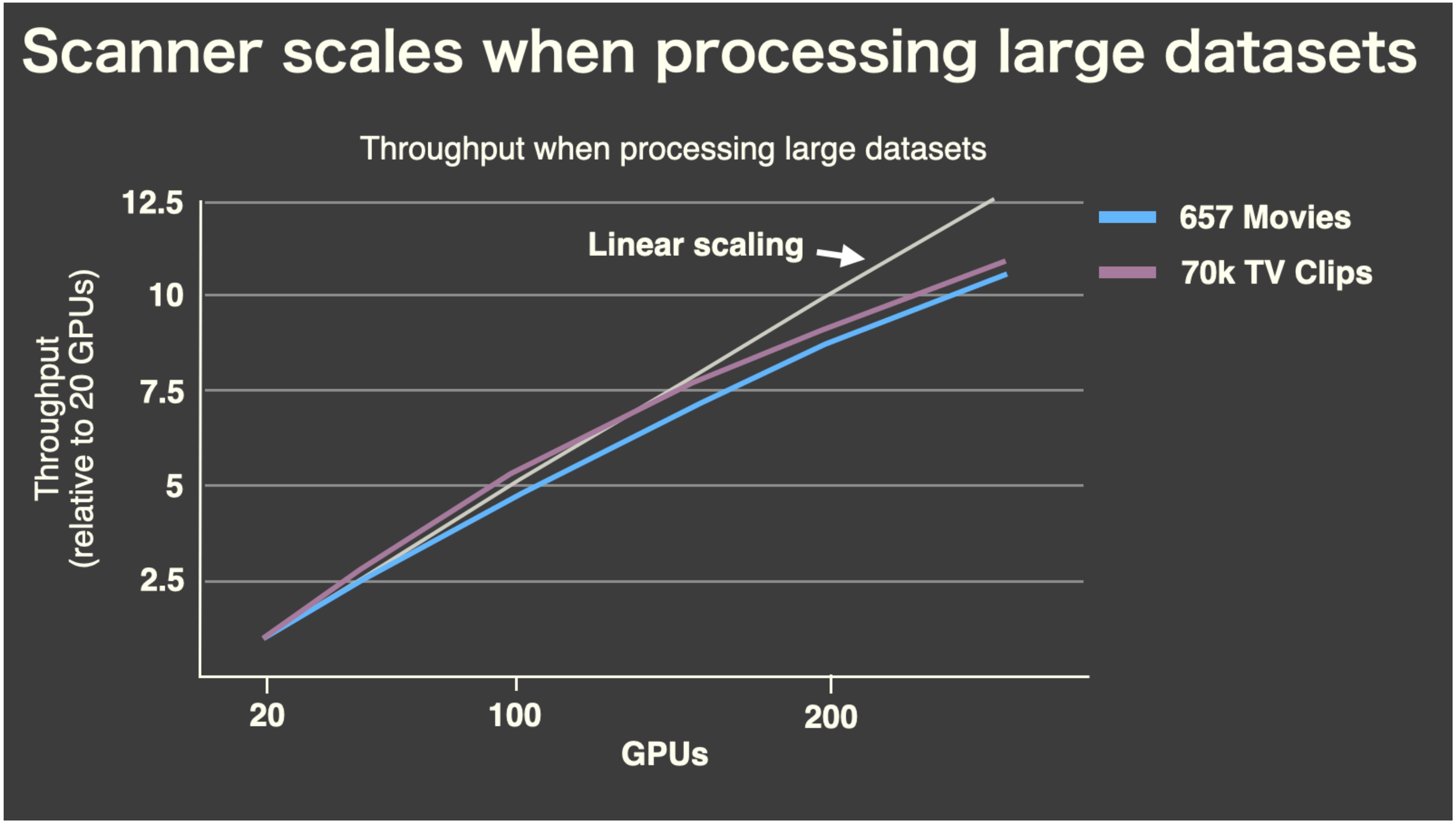
Example voice over: "Our first question was about performance: how fast is the auto scheduler compared to experts? And we found out that it's quite good. This figure plots the performance of the autoscheduler compared to that of expert code. So expert code is 1. Faster code is to the right. As you can see, the auto scheduler is within 10% of the performance of the experts in many cases, and always within a factor of 2.

Tip 8

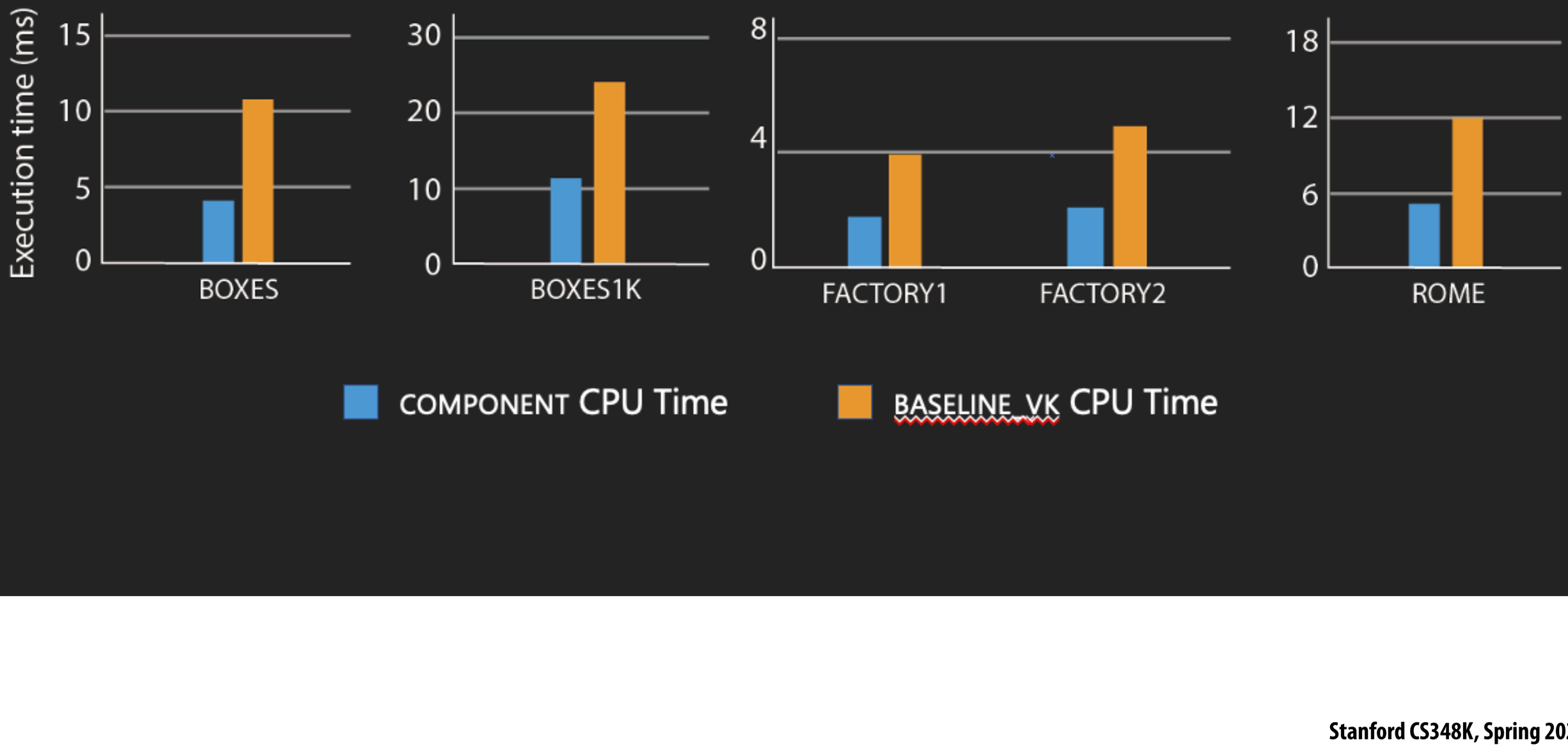
In the results section:
One point per slide!
One point per slide!

(and the point is the title of the slide!!!)

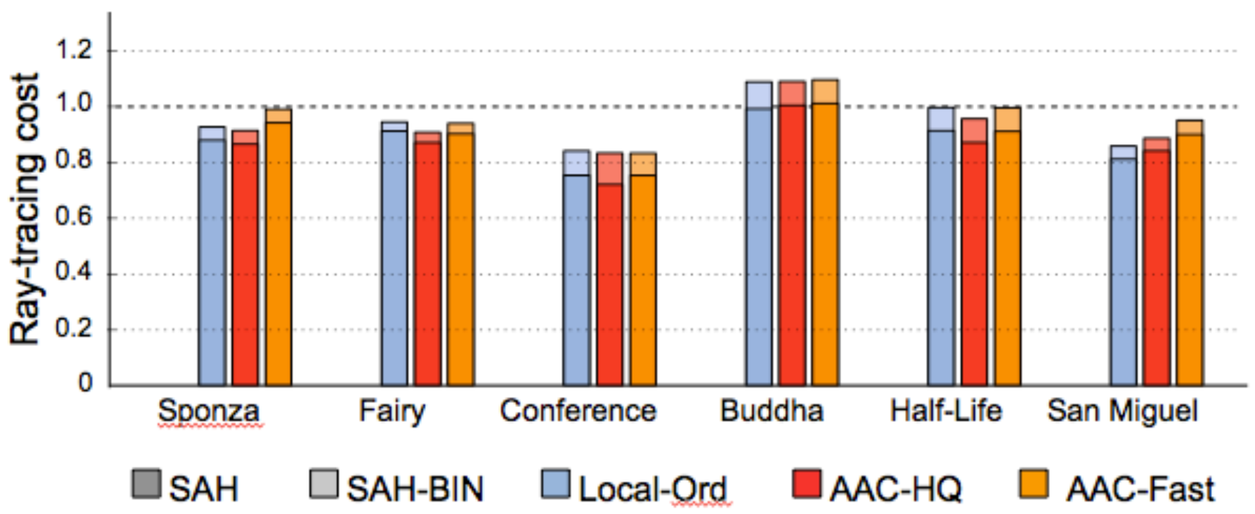
- **Make the point of the graph the slide's title:**
  - **It provides audience context for interpreting the graph ("Let me see if I can verify that point in the graph to check my understanding")**
  - **Another example of the "audience prefers not to think" principle**



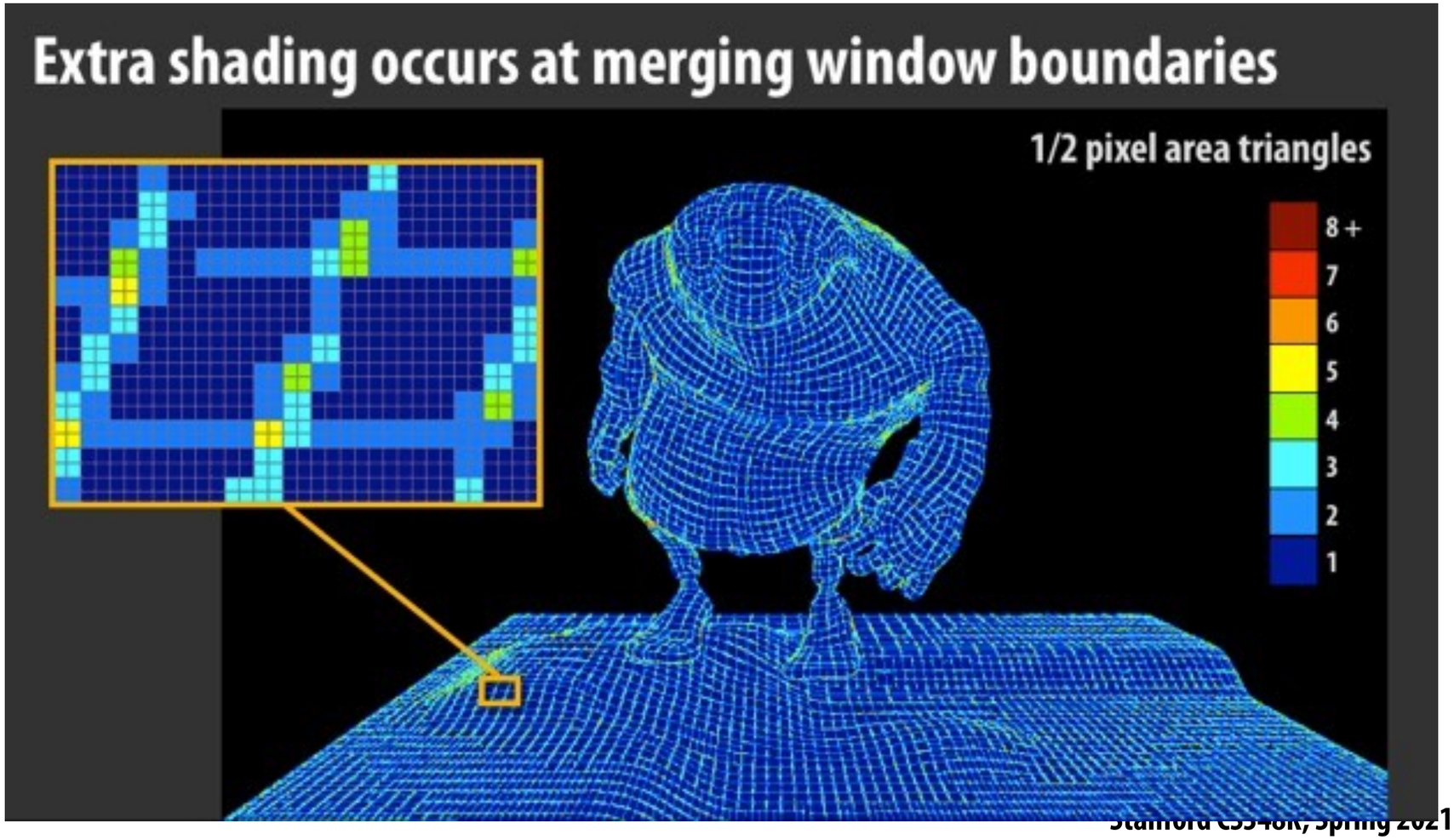
The COMPONENTS renderer uses 2x less CPU time than BASELINE_VK



AAC-Fast produces BVHs with equal or lower cost than the **full sweep build** in all cases except Buddha.



Extra shading occurs at merging window boundaries



Tip 9

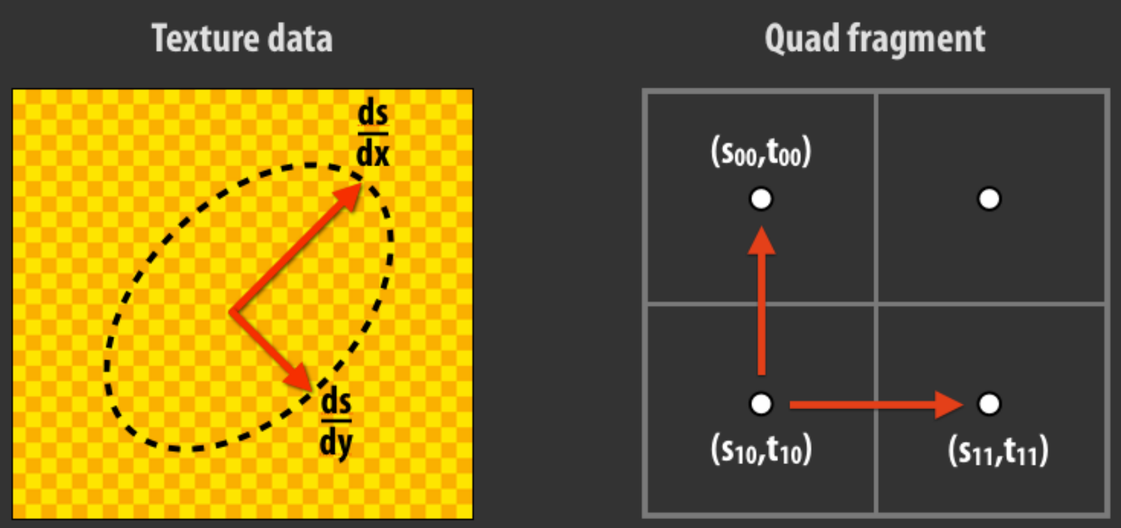
Titles matter

If you read the titles of your talk all the way through, it should be a great summary of the talk.

(basically, this is "one-point-per-slide" for the whole talk)

Examples of good slide titles

GPUs shade quad fragments (2x2 pixel blocks)



use differences between neighboring texture coordinates to estimate derivatives

Greedy SRDH build optimizes over partitions and traversal policies

SAH:

```
forall (partitions in set-of-partitions)
    …evaluate SAH and pick min…
```

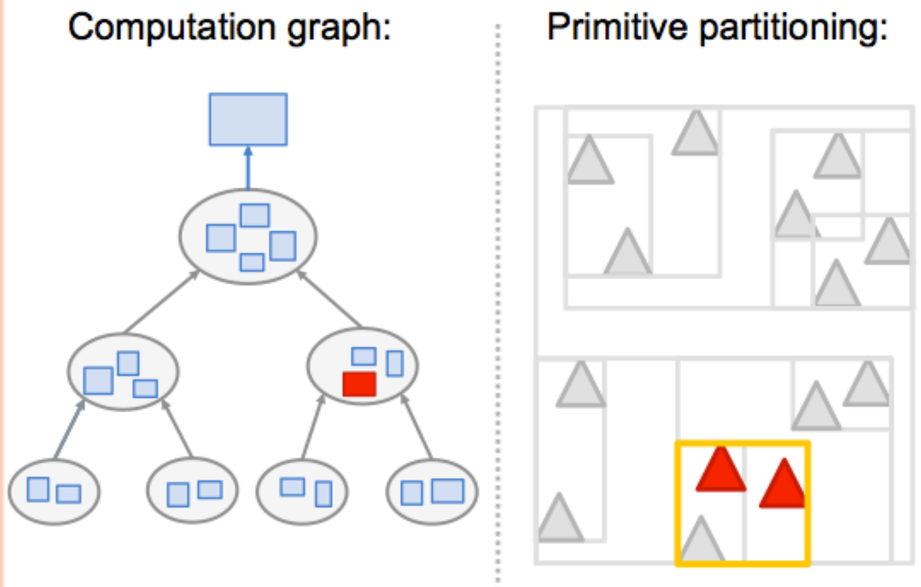
SRDH:

```
forall (partitions in set-of-partitions)
    forall (traversalKernels in set-of-kernels)
        …evaluate SRDH and pick min…
```

$$SRDH(R,L,\kappa,r)=(1-\kappa(r)H(L,r))|R|+(1-\kappa(r)H(R,r))|L|$$

51

AAC IS AN APPROXIMATION TO THE TRUE AGGLOMERATIVE CLUSTERING SOLUTION.



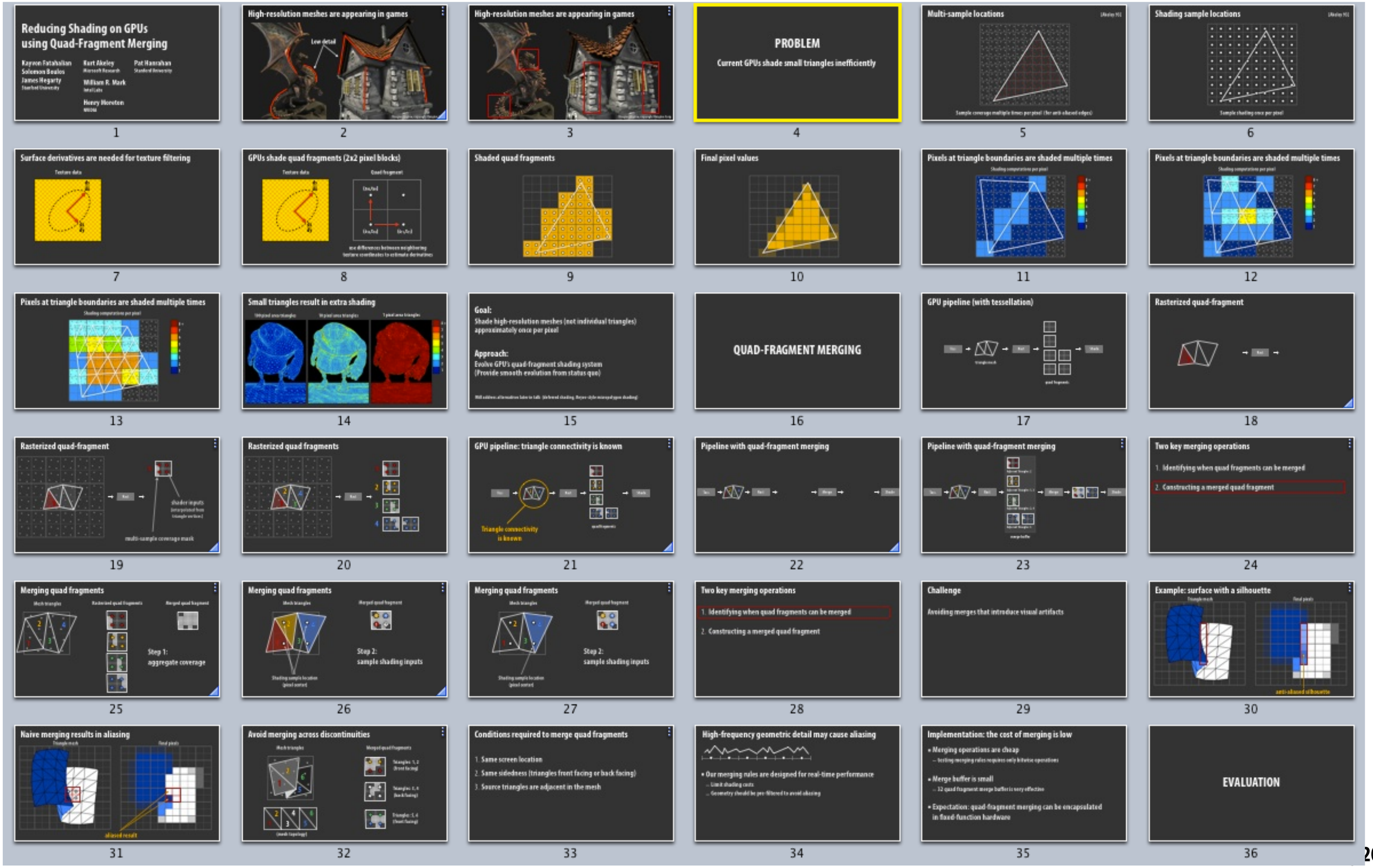
The reason for meaningful slide titles is convenience and clarity for the audience

"Why is the speaker telling me this again?"

(Recall "why before what")

Read your slide titles in thumbnail view

Do they make all the points of the story you are trying to tell?



2021

Tip 10

Practice the presentation

- Given the time constraints, you'll need to be smooth to say everything you want to say
- To be smooth you'll have to practice
- Rehearse your presentation several times the night before (in front of a partner or friend)
  - It's only a short presentation, so a couple of practice runs are possible in a small amount of time

General principles to keep in mind

Identify your audience (us), and strive for perfect clarity for them.

“Every sentence matters.”

“Show, don’t tell.”

“The audience prefers not to think” (about things you can just tell them)

“Surprises are bad”: say <u>why</u> before what

(indicate why you are saying something before you say it)

Explain every figure, graph, or equation