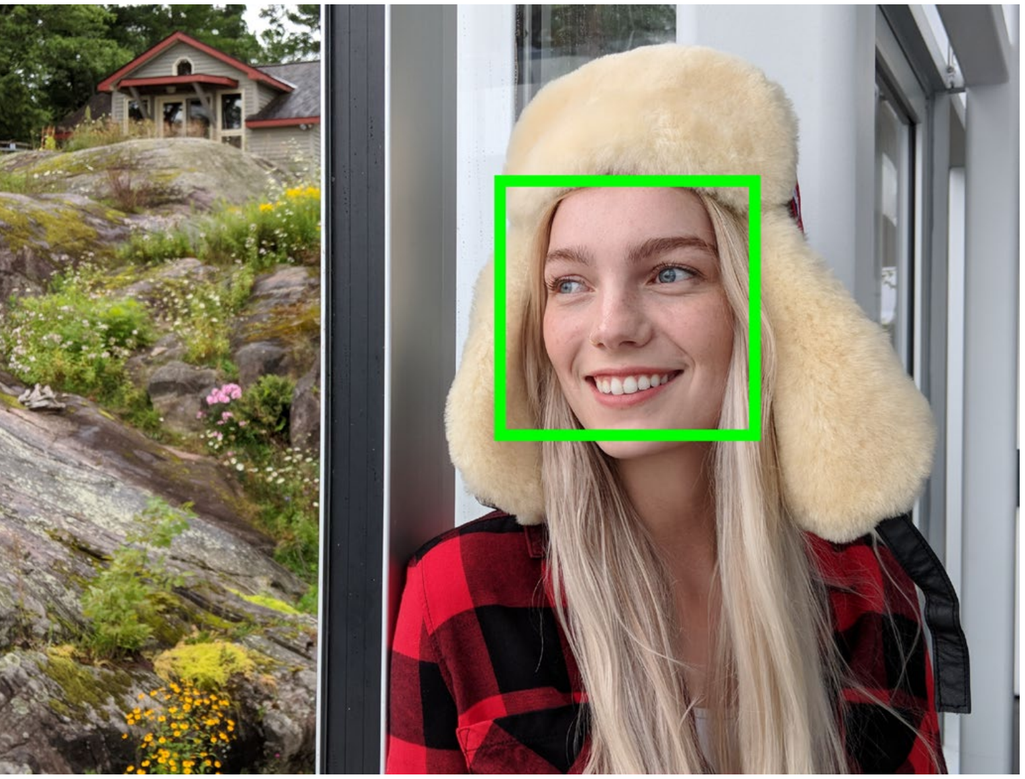
Lecture 17:

Rendering for Learning + Press Tips + Class Recap

Visual Computing Systems
Stanford CS348K, Spring 2020

One theme in class so far

- Employing machine learning techniques to improve photographs
  - Auto-focus / auto-exposure
  - Denoising
  - Person segmentation for portrait mode



Input image /w detected face

Segmentation

Scene Depth Estimate

Generated image (note blurred background. Blur increases with depth)

Image credit: [Wadha 2018]

One theme in class so far

- Machine learning as a way to improve rendering (denoising)

Image credit: Intel Open Image Denoise : https://openimagedenoise.github.io/

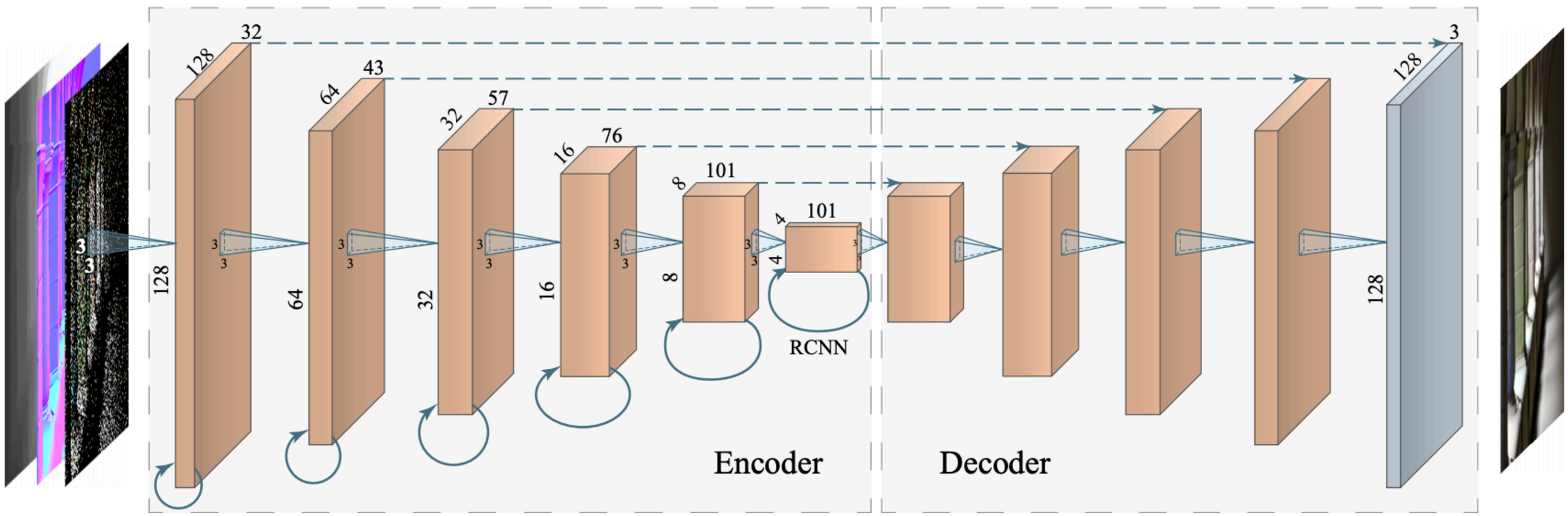
One theme in class so far

- Machine learning as a way to improve rendering (denoising)

Image credit: Intel Open Image Denoise : https://openimagedenoise.github.io/

Example: NVIDIA interactive denoiser

- Fairly standard encoder-decoder architecture (but with recurrent links for frame-to-frame coherence)
- Input: depth, normal, roughness noisy RGB
- Output: denoised RGB



[Chakravarty et al. 2016]

Rendering to Support Model Training

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.stanfor

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

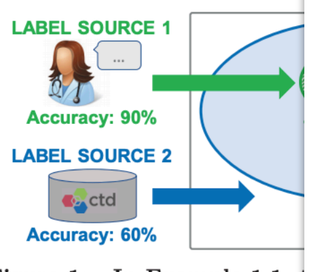


Figure 1: In Example 1.1, t by sources of differing accura key challenges arise in using effectively. First, we need a known source accuracies to Second, we need to pass on formation to the end model

advent of *deep learning* techniq specific representations of input to be the most time-consuming engineering. These learned repre effective for tasks like natural lang

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

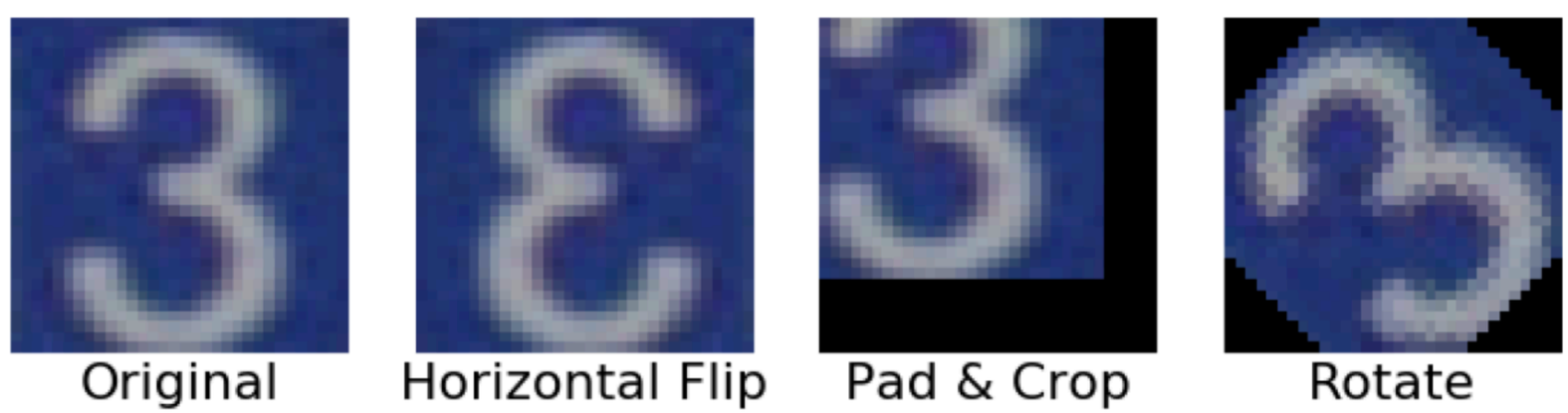
In the life cycle of many production machine-learning applications, maintaining and improving deployed models is the dominant factor in their total cost and effectiveness–much greater than the cost of *de novo* model construction. Yet, there is little tooling for model life-cycle support. For such applications, 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

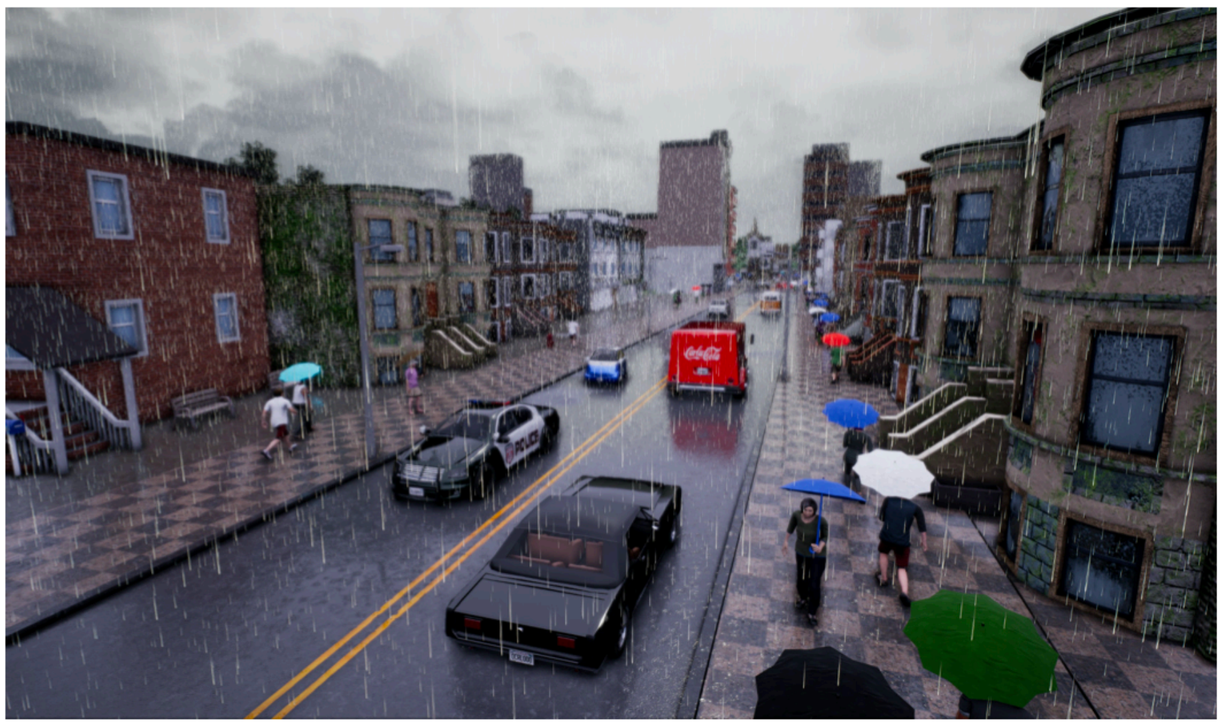
A common strategy for automatically generating new labeled training data from a small number of labeled examples (as long as augmentations don't change classification result)

[Credit: Zhu et al. 2017]



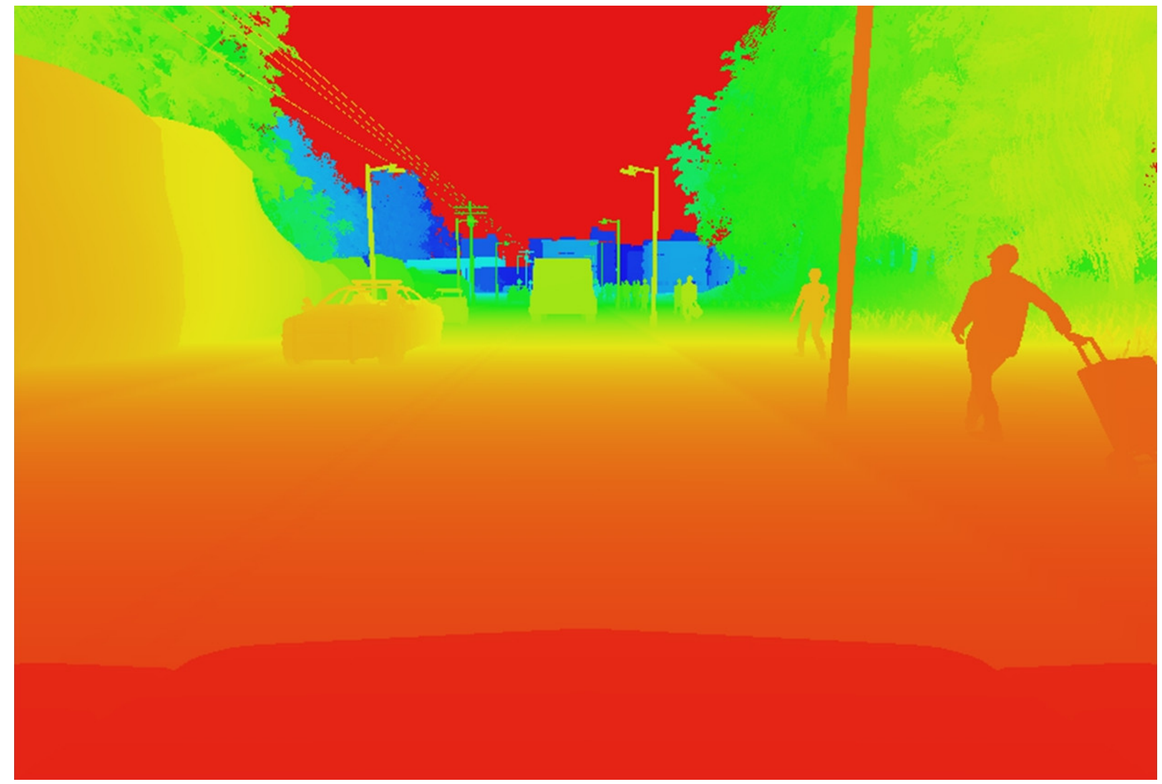
[Credit: Ho et al. 2019]

Carla: urban driving simulator based on Unreal Engine

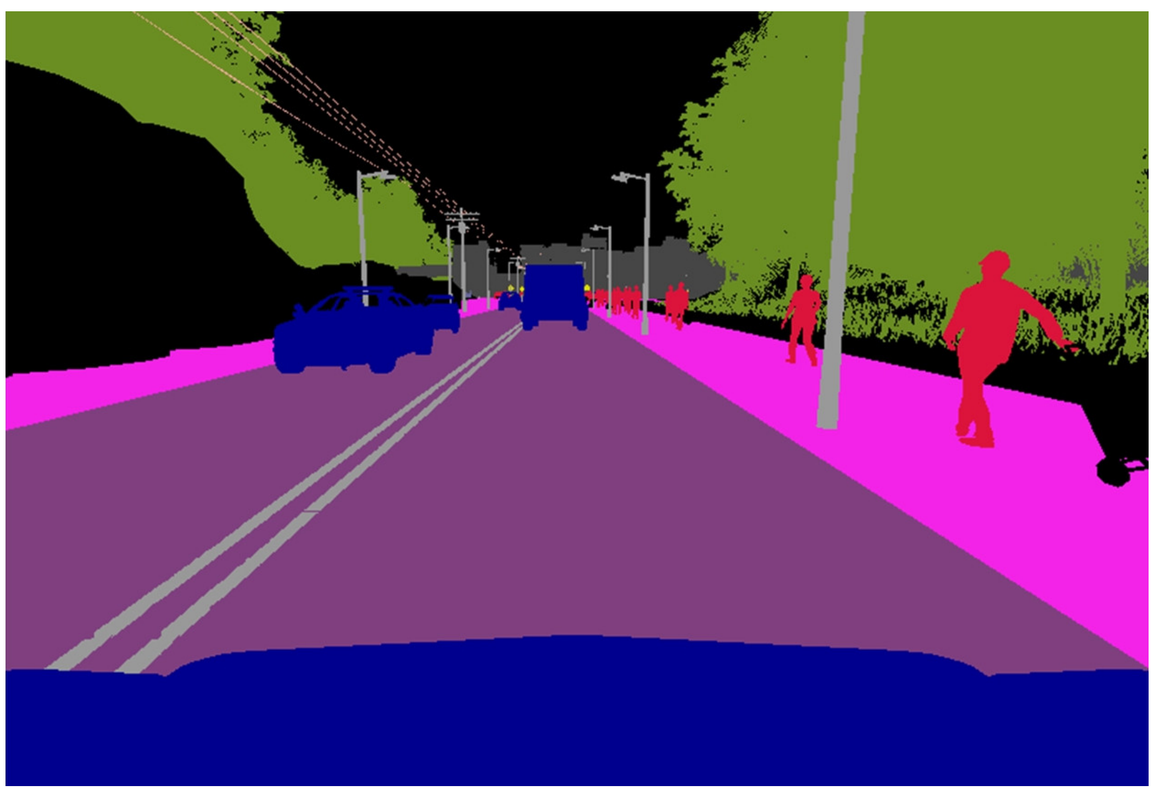



Example Carla outputs

RGB



Depth



Object type

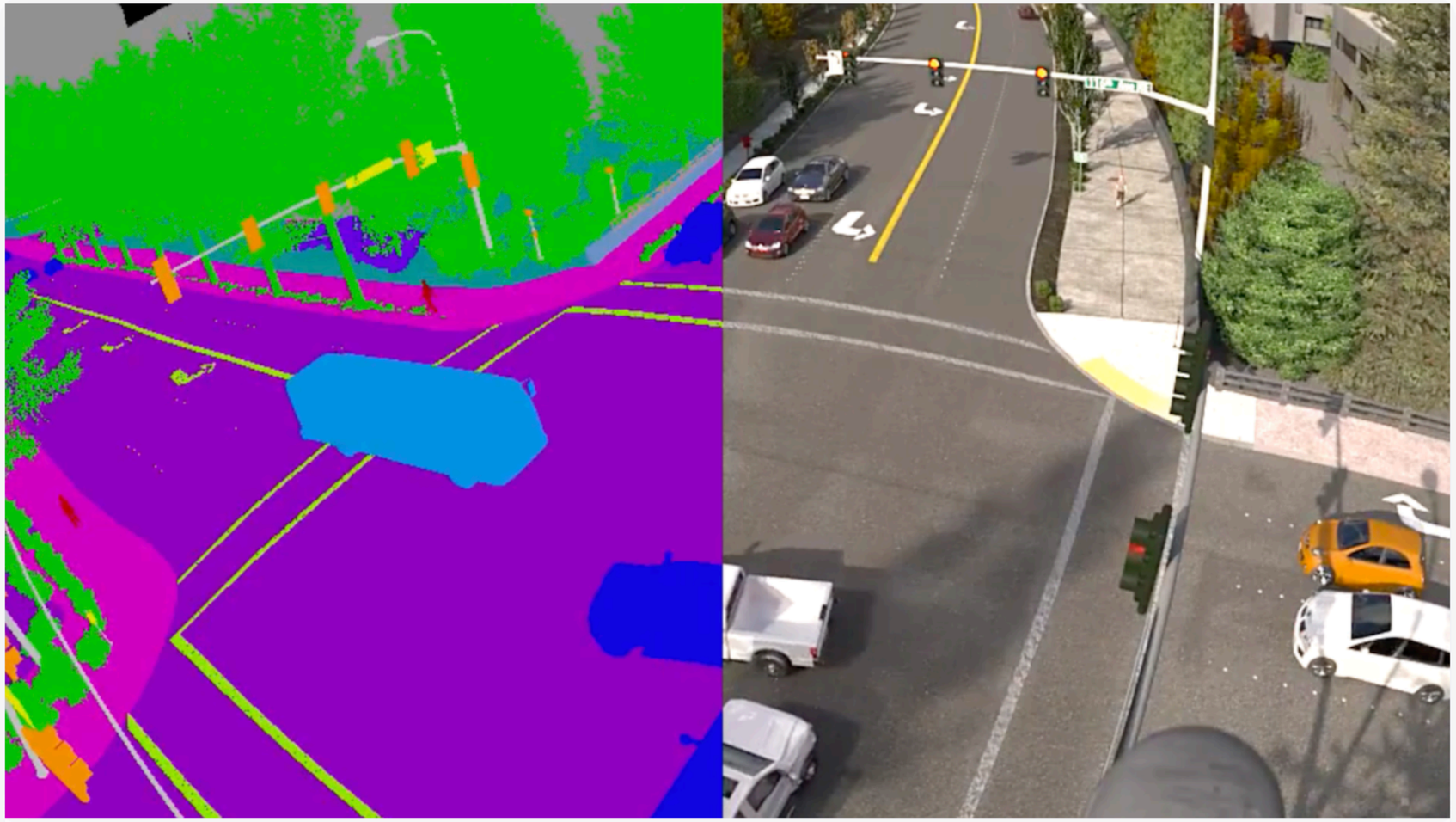
Since renderer has complete description of scene, it can output detailed, fine-grained labels as well as RGB image.

(would be laborious to annotate)

Unity **Blog**

Subscribe to RSS

Search the blog...



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

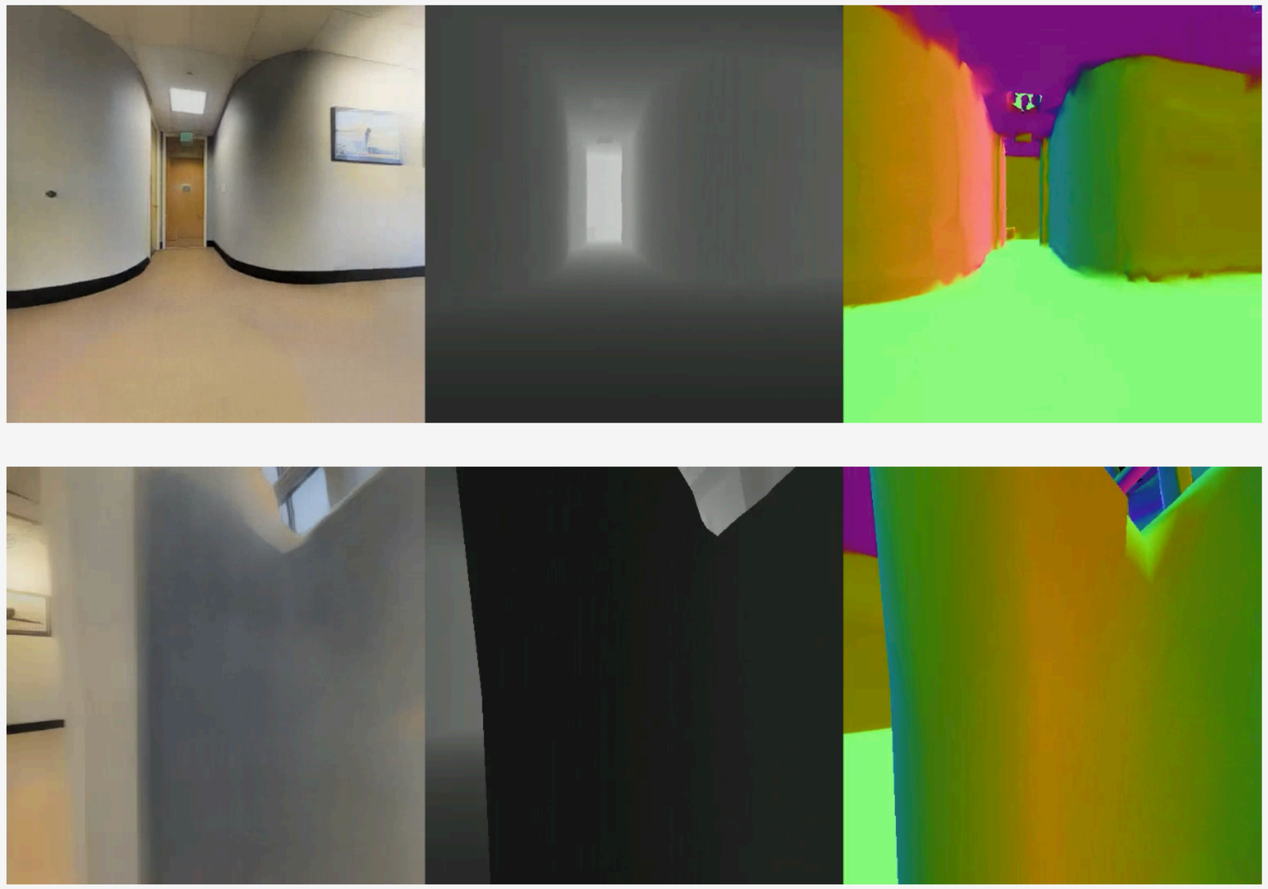
Introducing the new Input System
October 14, 2019

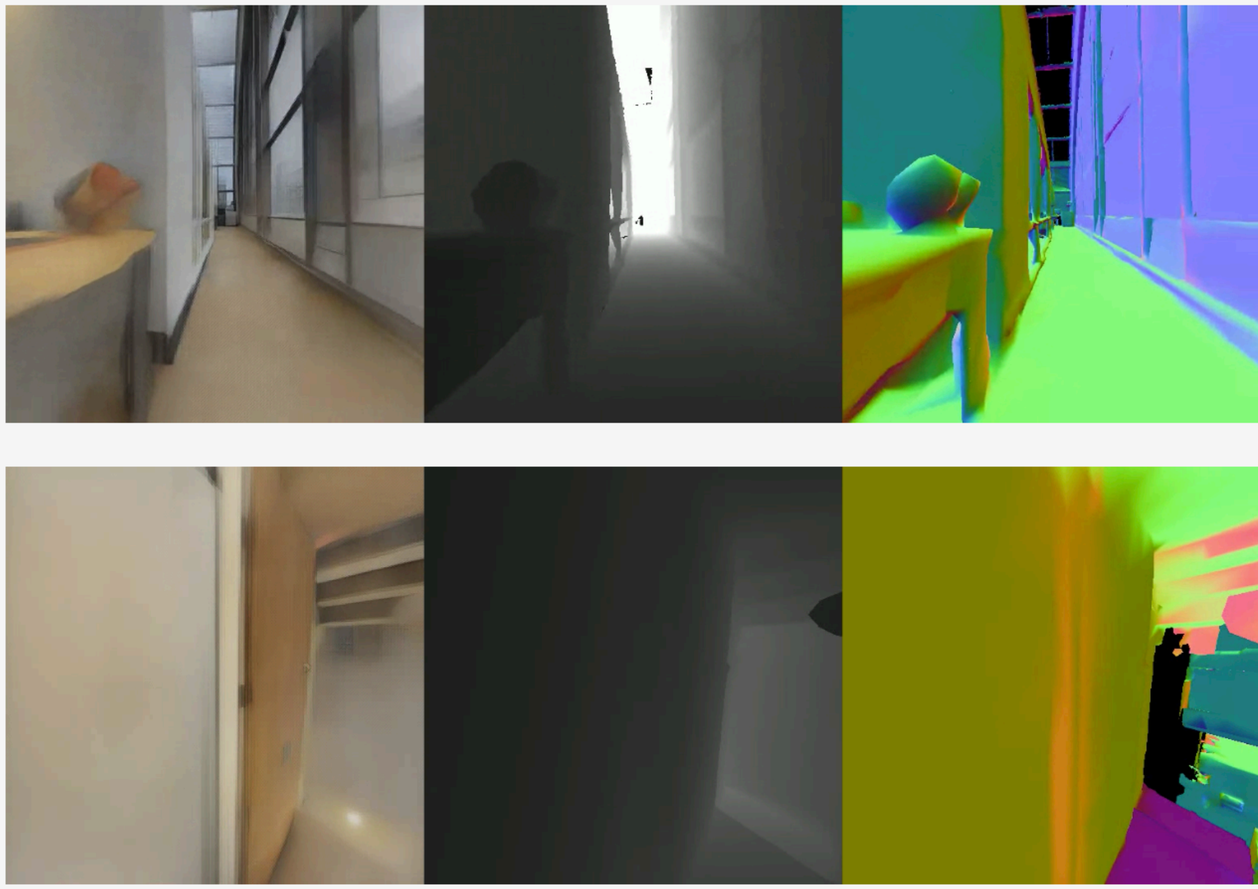
2D Pixel Perfect: How to set up your Unity project for retro 8-bit games
March 13, 2019

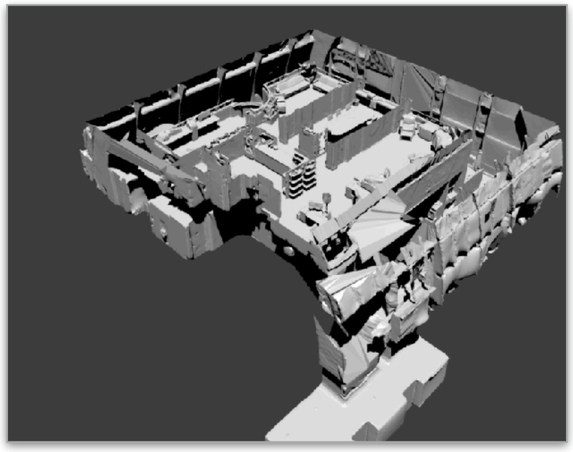
The High Definition Render Pipeline: Getting Started Guide for Artists
September 24, 2018

Gibson: acquire/render real world data

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

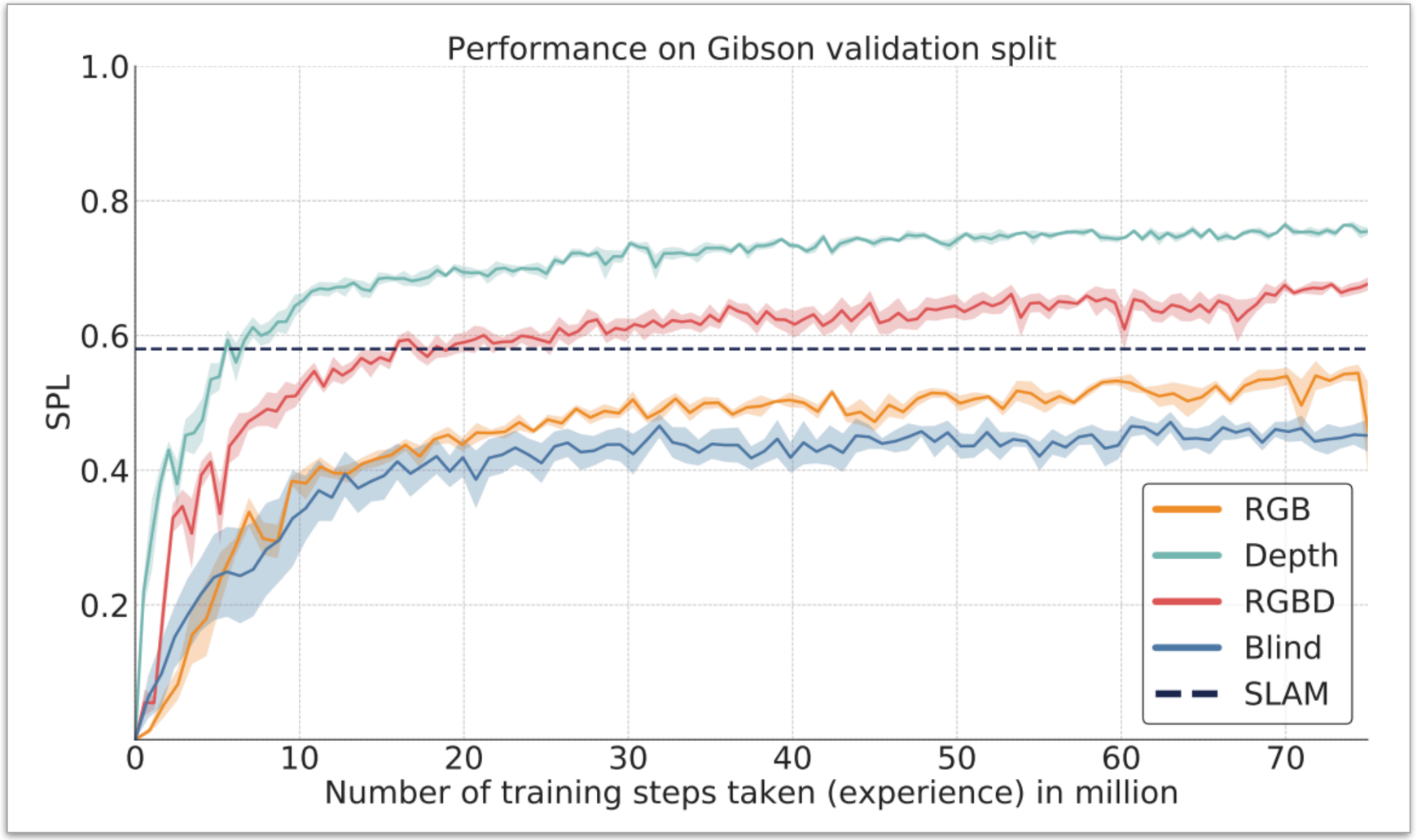






AI Habitat

- **Focus on high-performance rendering to enable longer (more training steps) RL training runs**



AI Habitat

- **Focus on high-performance rendering 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.

Interesting (open) rendering systems research questions

- If you had to design a rendering system "from the ground up" to support 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?

New Methods for Image Synthesis

Traditional image synthesis

- Input: model of a scene
  - Triangles, materials, lights, camera
- Renderer simulates physics of light-surface interactions to estimate appearance of scene from camera view

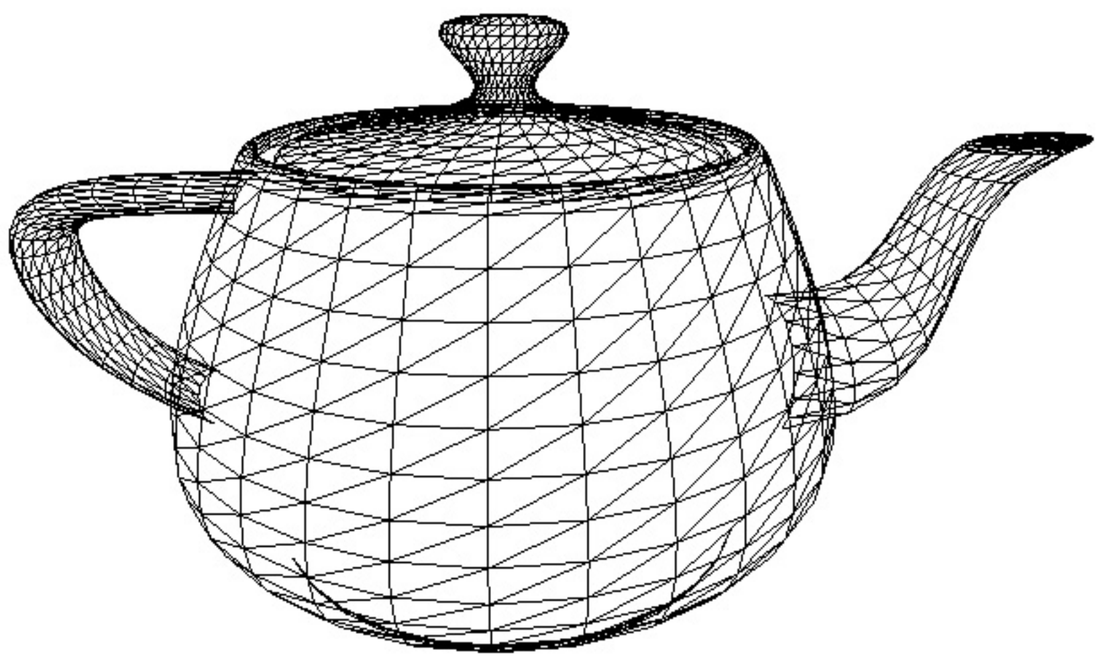


Image credit: Henrik Wann Jensen

Input: description of a scene

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

Output: image

Screenshot: Far Cry 5

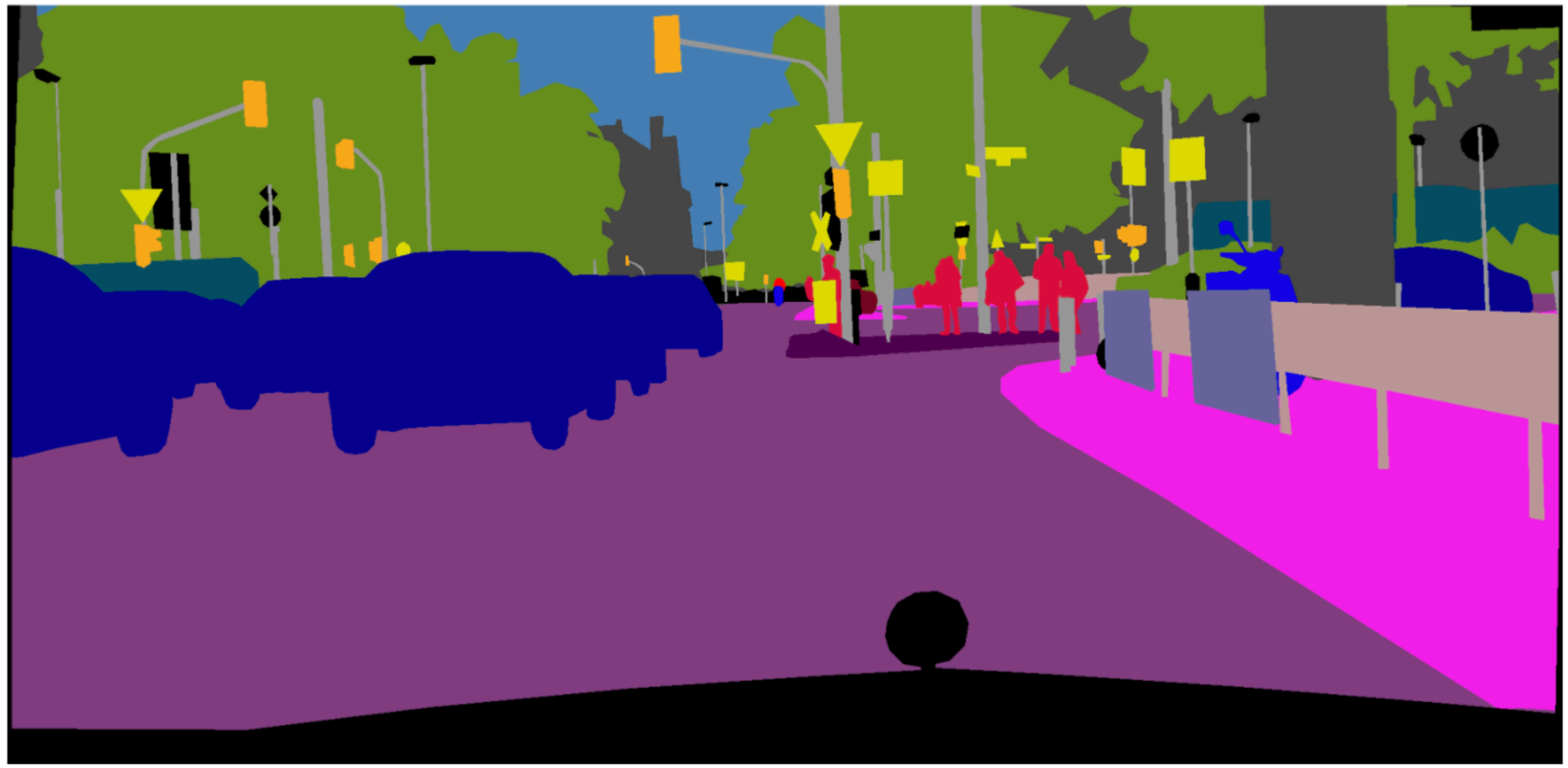
Screenshot: Assasin's Creed (Odyssey)

Screenshot: Assasin's Creed (Odyssey)

Screenshot: Forza Motorsport 7

GAN-based image synthesis

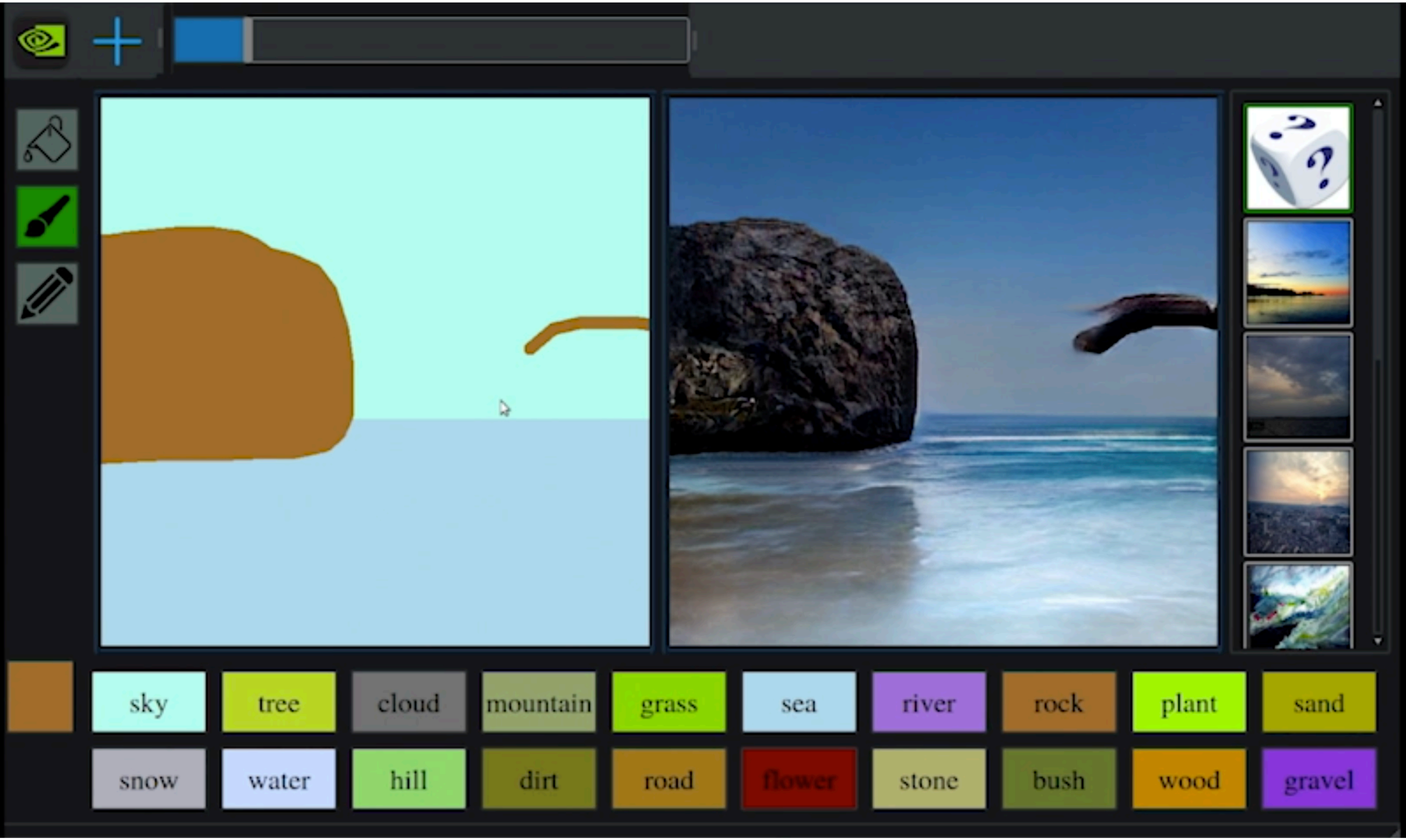
Input Image



Output Image

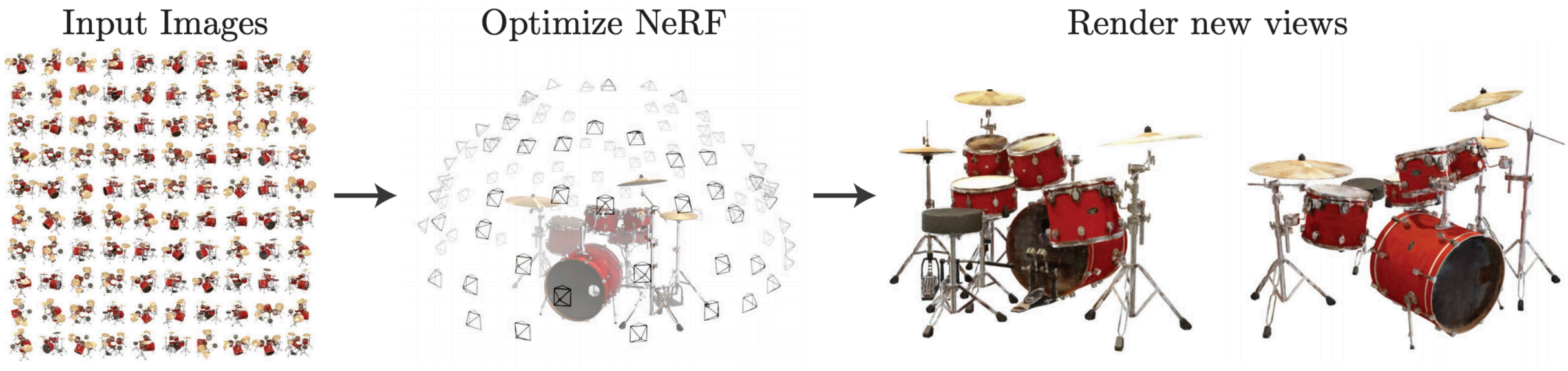
Pix2Pix HD, Wang 2018

GAN-based image synthesis



GuaGAN, Park et al. 2019

Neural radiance fields (NeRF)



Given a sparse set of images…

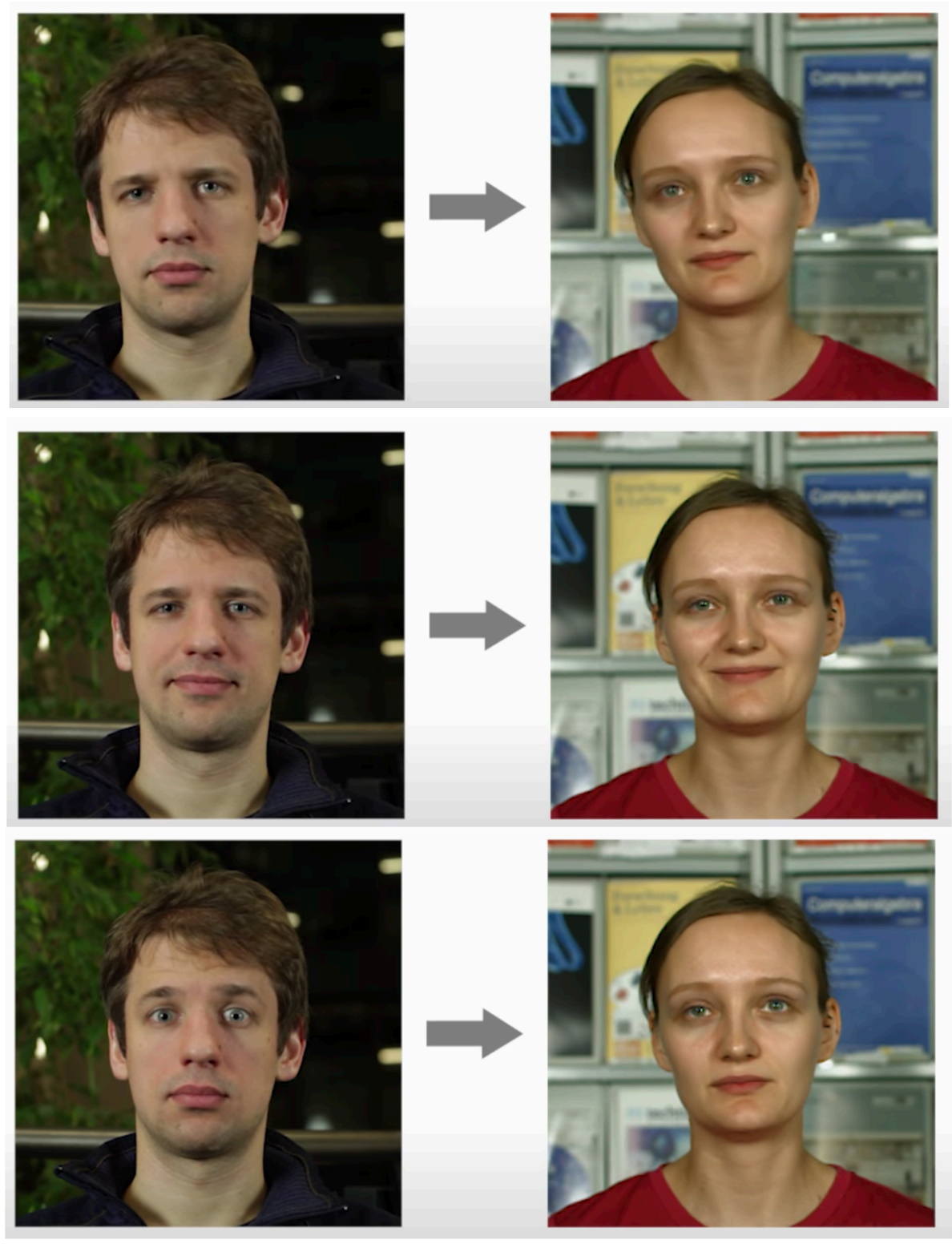
Train DNN that outputs light traveling from scene point (x,y,z) in direction D

In other words, what would a ray tracer compute if it shot a ray from (x,y,z) in direction *d*….

[Mildenhall 2020]

Deep video portraits

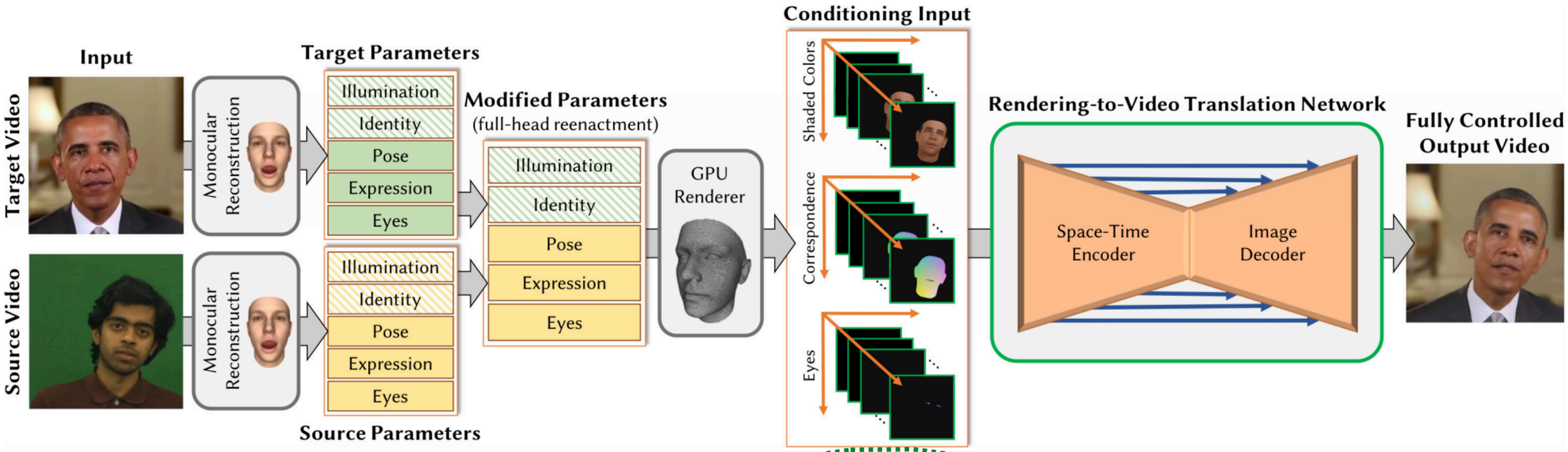
[Kim et al. 2018]

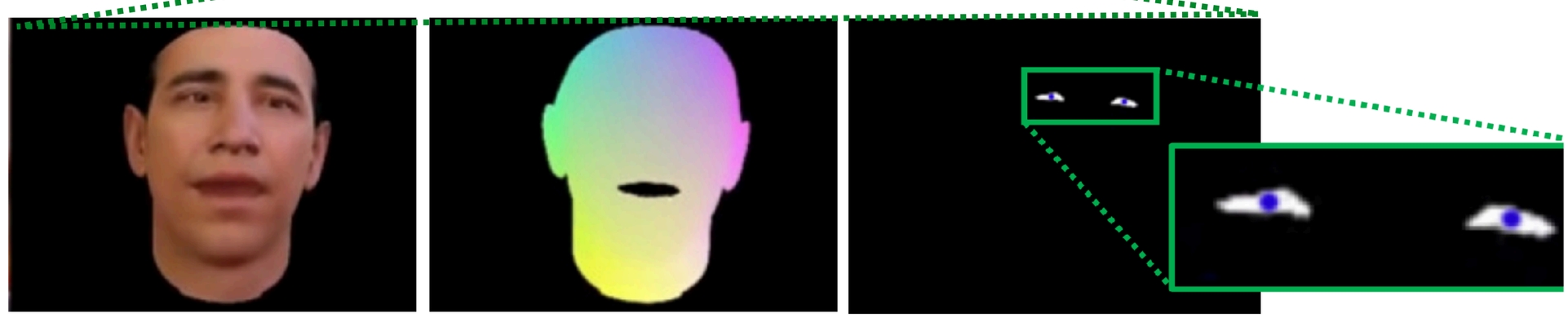


Input Output

Deep video portraits

[Kim et al. 2018]





Diffuse Rendering Correspondence Eye and Gaze Map

Main idea:

- **Extract face model parameters from source video**
- **Render target face using those parameters (using traditional rendering techniques)**
- **Use neural image-to-image techniques to turn rendered face into photocell face**

Open question: what should be modeled vs. what should be learned from data?

- What is the right scene representation for rendering?
  - Traditional: triangles, textures, material models, etc.
  - Extreme data-driven: learn model that embodies knowledge for transforming one source scene image into desired target scene image
  - Many hybrids: learn aspects of traditional information
    - NeRF: learn to return result of ray query
    - Neural textures/neural volumes (encode aspects of surfaces, materials and textures in learned latent space)
    - Render image with Cg, then add detail with neural techniques

Course Recap

A few course themes

We've read a number of industry papers

- *Thinking like an architect*
  - What are inputs/outputs, constraints, and goals?
  - What are the "services" the system should perform (what is hard for a user to do in a world without the system)
  - Once you can establish answers to these questions, you can consider your solution options

- Knowledge of applications and systems is necessary to choose efficient solutions
  - HW designers use their hammer (custom accelerators, new interconnects)
  - SW systems folks use their hammer (distributed computing, workload specialization)
  - Algorithms folks use their hammer (smaller DNN models, change optimization hyperparameters, model noise in weak labels)
  - Best solutions pick the right hammers for the job)

A few course themes

- Knowledge of applications and systems is necessary to do meaningful evaluation
  - Is this the right workload?
    - Speedup on fully connected layers vs. conv layers?
    - Does it have the right data distribution?
  - Am I measuring cost in FLOPs, but increasing data movement?
  - Am I optimizing an algorithm that is not the right choice for this problem?

Communicating like an architect

Who painted this painting?

Salvador Dali (age 22)

My point: learn the basic principles before you consciously choose to break them

Philosophy: why give talks?

Benefit TO YOU of a good (clear) talk

- Non-linear increase in the <u>impact</u> of your work
  - Others are more likely to remember and build upon your work
  - Others are more likely to come up to you after the talk
  - Others are most likely to adopt your ideas

- Clarity is highly prized in the world: the audience remembers you
  - "Hey, that was a great talk... are you looking for a job anytime soon?"
  - "Hey, that was a great talk, I'm working on something that you might find helpful."

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.

Here are some tips to help.

1.

Put yourself in your audience's shoes

This is a major challenge for most technical speakers. (including professors)

(Tip: recite a sentence out loud to yourself. Do you really expect someone who has not been working with you everyday on the project to understand what you just said?)

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 <u>not</u> to explain what you did, but to explain what you think <u>we should know</u>

Ignoring every sentence matters

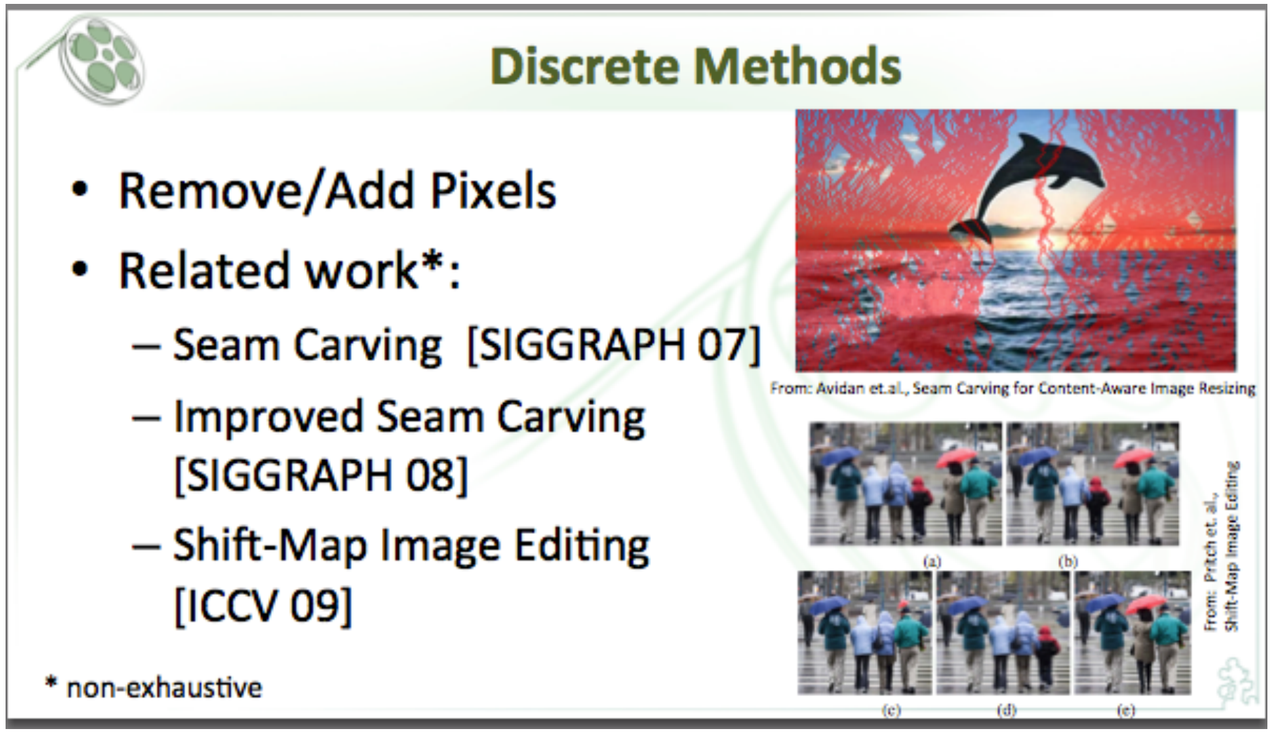
Never ever, ever, ever do this!

Outline

- Introduction
- Related Work
- Proposed System Architecture
  - Basic design decision
  - Dedicated hardware for T&I
  - Reconfigurable processor for RGS
- Results and Analysis
- Conclusion

Bad example 2

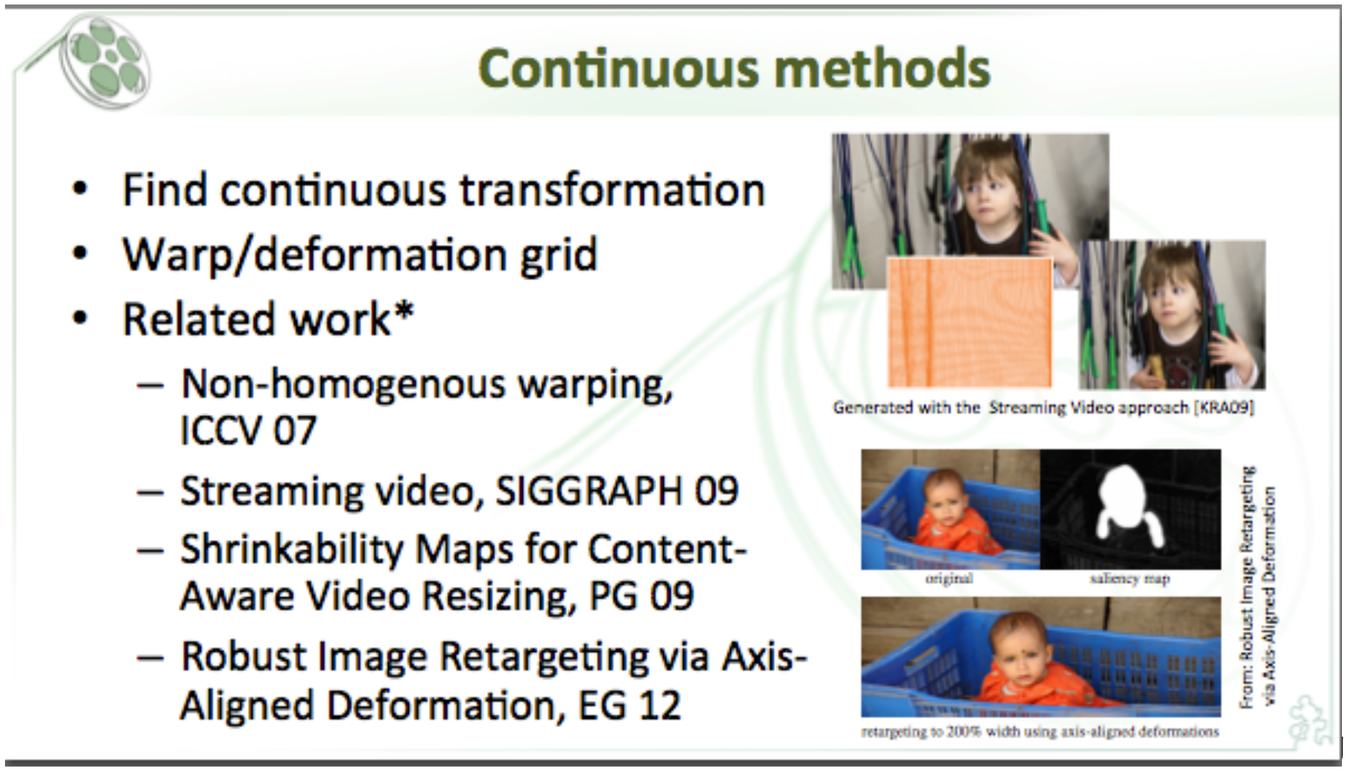
- **Who is the audience for this? (how does this benefit them?)**



- **Experts?**
  - **They likely know these papers exist. These slides don't tell them what about these papers is most relevant to this talk**
- **Non-experts?**
  - **They won't learn the related work from these two slides**

This type of related work section says little more than "others have worked in this area before".

- **I suspect your audience assumes this is the case.**
- **Every sentence matters: if it doesn't provide value, take it out (or replace it with comments that do provide value)**



3.

The audience prefers not to think (much)

The audience has a finite supply of mental effort

- The audience does not want to burn mental effort about things <u>you know</u> 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 with what you did (did you consider all edge cases? Is your evaluation methodology sound? Is this a good platform for this workload?)
  - Implications of your approach to other things
  - Connections to their own work or project

4.

Set up the problem.
Establish inputs, outputs, and constraints
(goals and assumptions)

Basics of problem setup

- **What is the computation performed (or system built)?**
  - What are the inputs? What are the outputs?

- **Why does this problem stand to benefit from optimization?**
  - "Real-time performance could be achieved"
  - "Researchers could run many more trials, changing how science is done"
  - "It is 90% of the execution time in this particular system"

- **Why is it hard? (What made your project interesting? What should we reward you for?)**
  - What turned out to be the hardest part of the problem?
  - This may involve describing a few key characteristics of the workload (e.g., overcoming divergence, increasing arithmetic intensity)

Example: 3D rendering problem

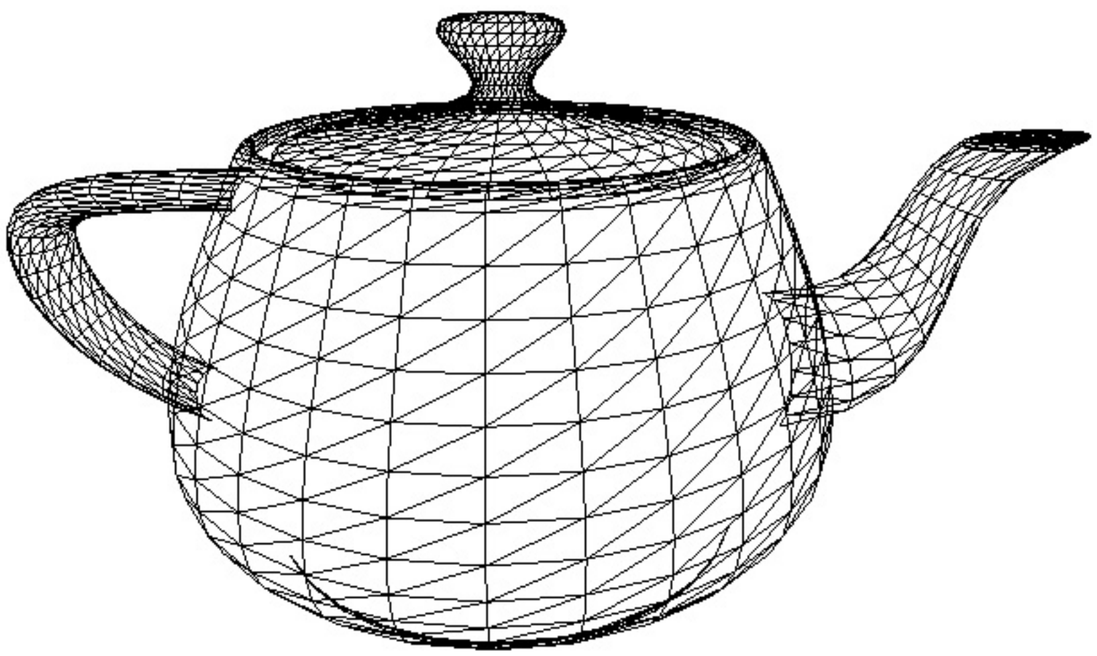


Image credit: Henrik Wann Jensen

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?

Establish goals and assumptions early

- We are working under the following constraints:
  - Example: the outputs should have these properties
  - Example: the algorithm...
    - Should be real time
    - Must interoperate with this existing library which we cannot change
    - Must ascribe to this interface (because it's widely used)
    - Must be able to work without human annotations
  - Example: the system…
    - Need not compile all of Python, only this subset… (because for my domain that's good enough)
    - Should realize about 90% of the performance of hand-tuned code, with much lower development time

5.

How to describe a system

How to describe a system

- Start with the nouns (the key boxes in a diagram)
  - Major components (processors, memories, interconnects, etc.)
  - Major entities (particles, neighbor lists, pixels, pixel tiles, features, etc.)
  - What is <u>state</u> in the system?

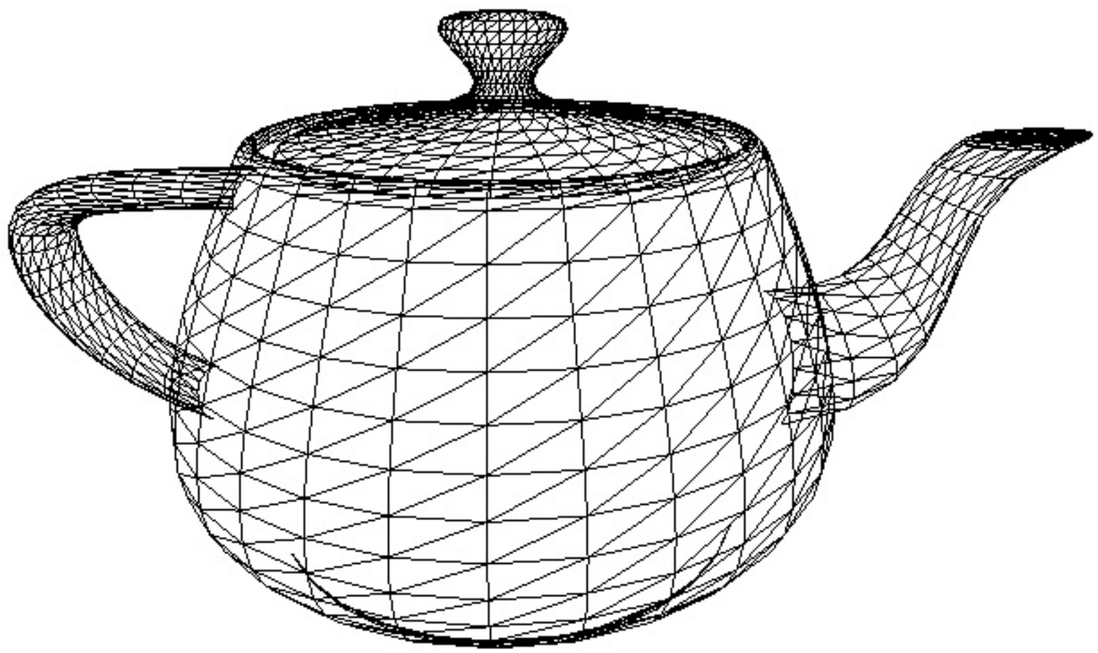
- Then describe the verbs
  - Operations that can be performed on the state (update particle positions, compute gradient of pixels, traverse graph, etc.)
  - Operations produce, consume, or transform entities

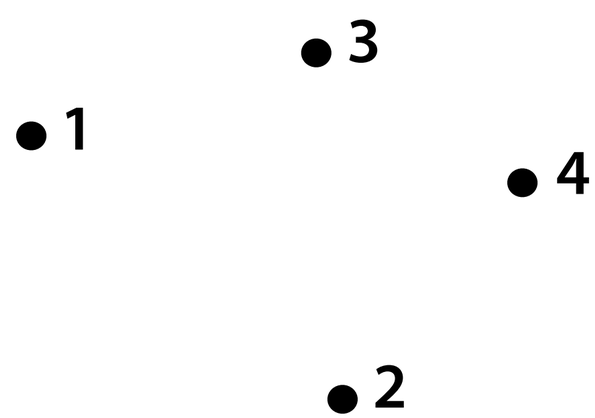
Tip: how to explain "a system"

- Step 1: describe the <u>things</u> (key entities) that are manipulated
  - The nouns

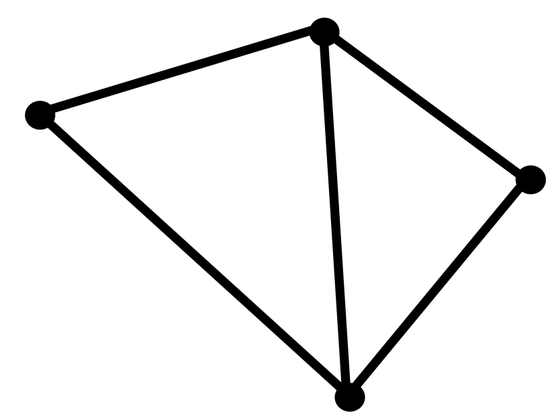
Real-time graphics primitives (entities)

Represent surface as a 3D triangle mesh



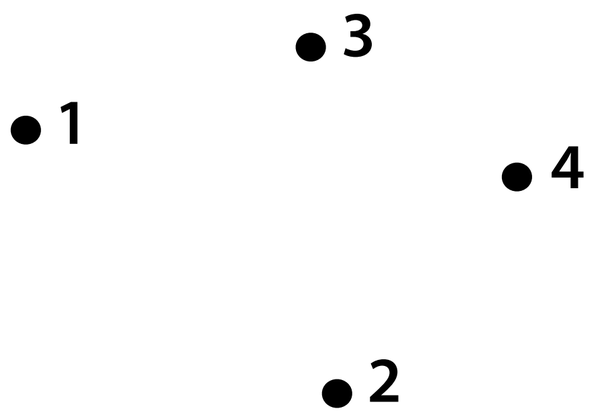


Vertices
(points in space)

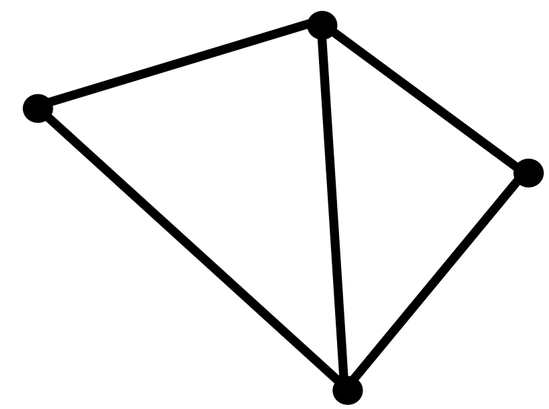


Primitives
(e.g., triangles, points, lines)

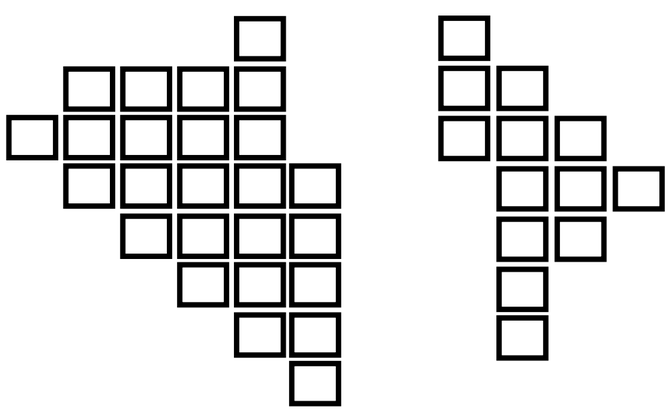
Real-time graphics primitives (entities)



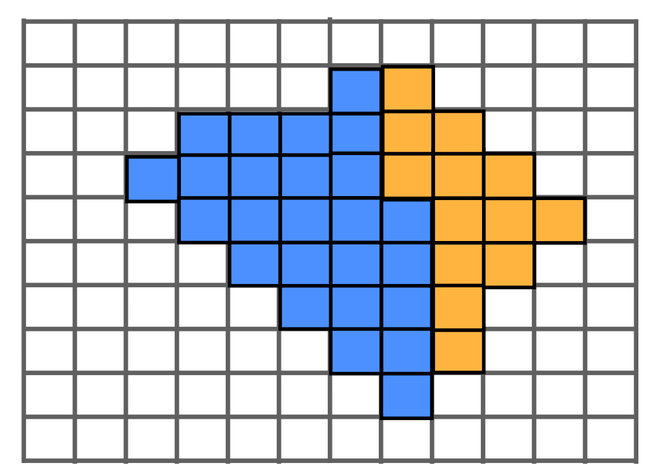
Vertices
(points in space)



Primitives
(e.g., triangles, points, lines)



Fragments



Pixels (in an image)

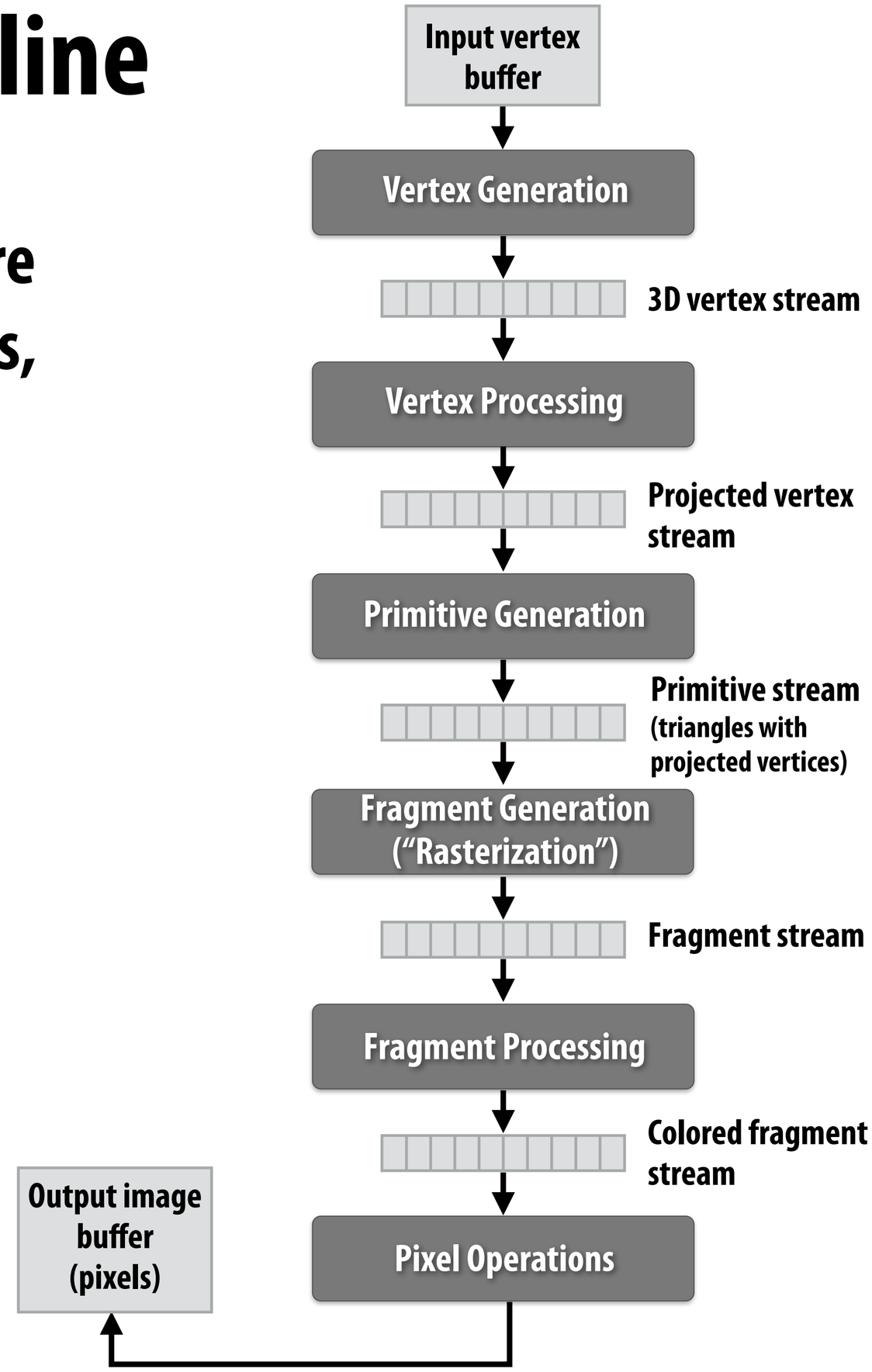
How to explain "a system"

- Step 1: describe the <u>things</u> (key entities) that are manipulated
  - The nouns

- Step 2: describe the operations the system performs on these entities
  - The verbs

Real-time graphics pipeline

Abstracts process of rendering a picture as a sequence of operations on vertices, primitives, fragments, and pixels.



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.

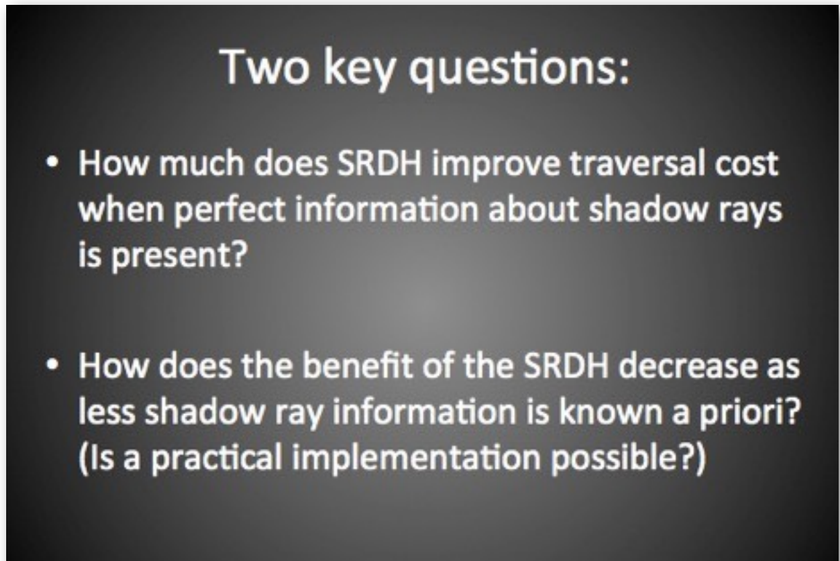
* I am referring to surprises in talk narrative and/or exposition. A surprising result is great.

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

- **<u>Why</u> 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:**
  - Speaker: "Key questions to ask about our approach are..."
  - Listener: "Thanks! I agree, those are good questions. Let's see what the results say!"



* This slide is an example of "audience does not want to waste mental effort on things you can tell them"

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

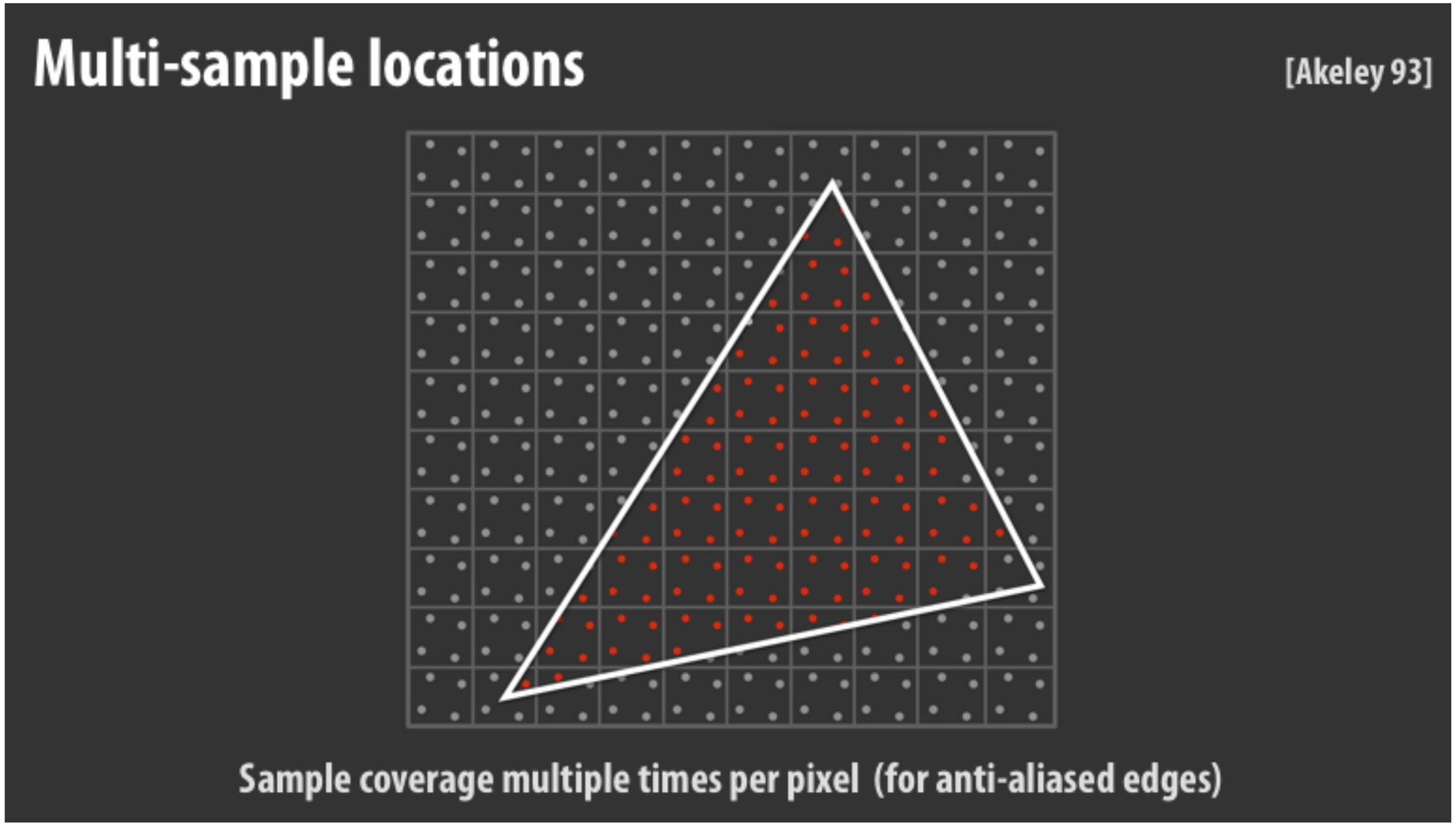
7.

Always, always, always explain any figure or graph

(remember, the audience does not want to think)

Explain every figure

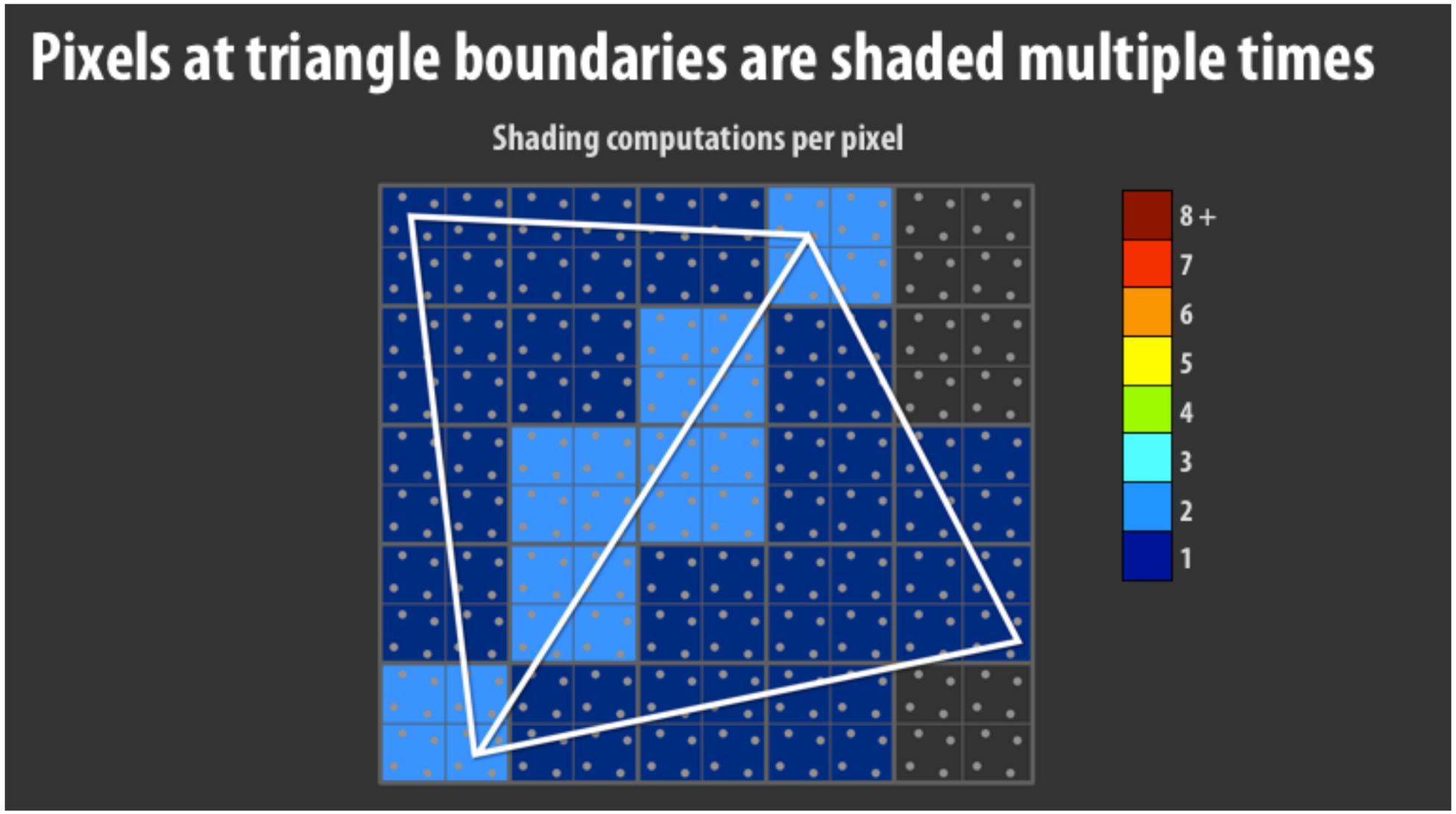
- Explain <u>every visual element</u> used in the figure (never make the audience decode a figure)
- Refer to highlight colors explicitly (explain why the visual element is highlighted)



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

Explain every figure

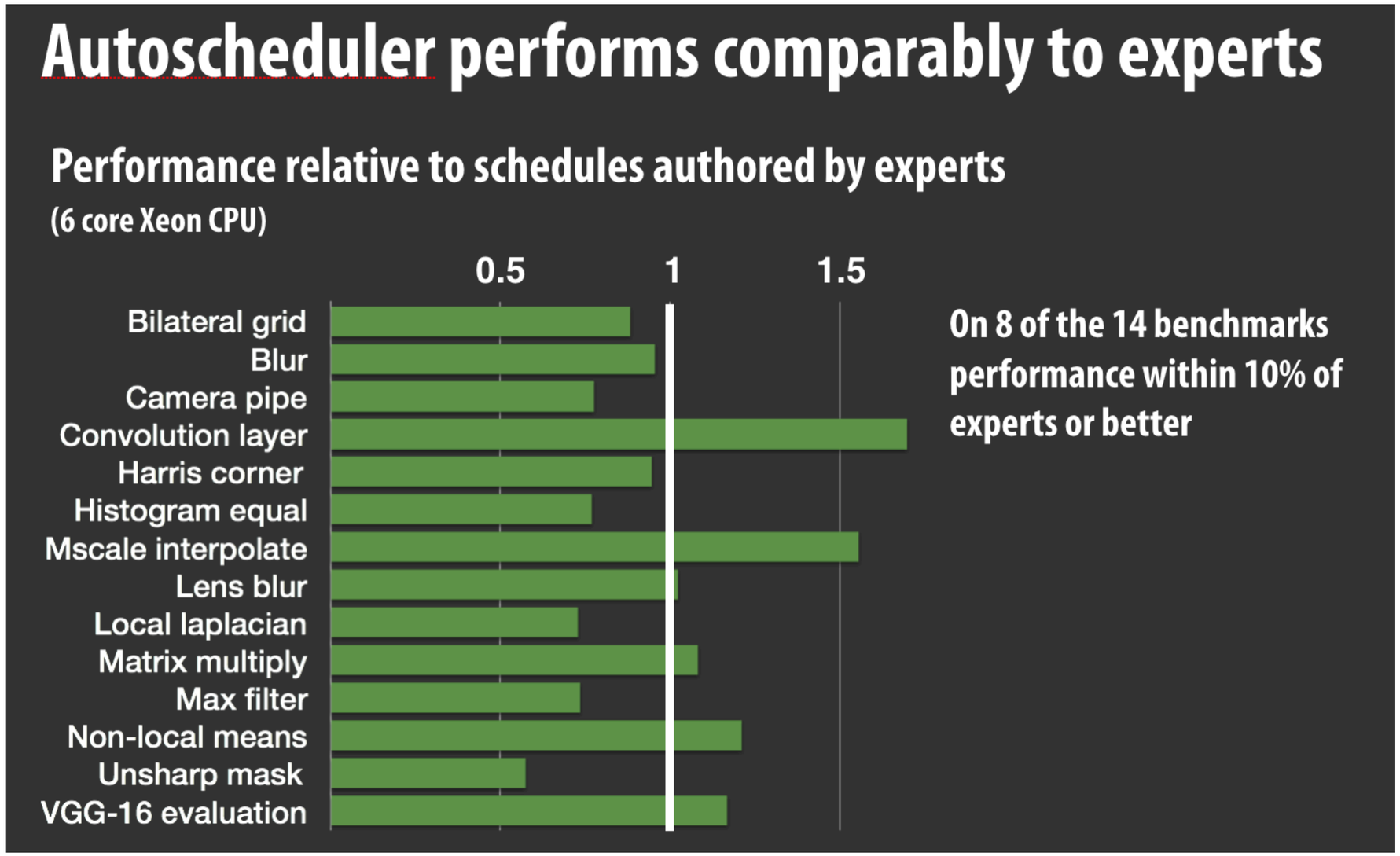
- Lead the listener through the key points of the figure
- Useful phrase: "As you can see..."
  - It's like verbal eye contact. It keeps the listener engaged and makes the listener happy... "Oh yeah, I can see that! I am following this talk! Yippee!"



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. <u>As you can see</u> in 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.
- 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 (more on this later!)



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.

8.

In the results section:

One point per slide!

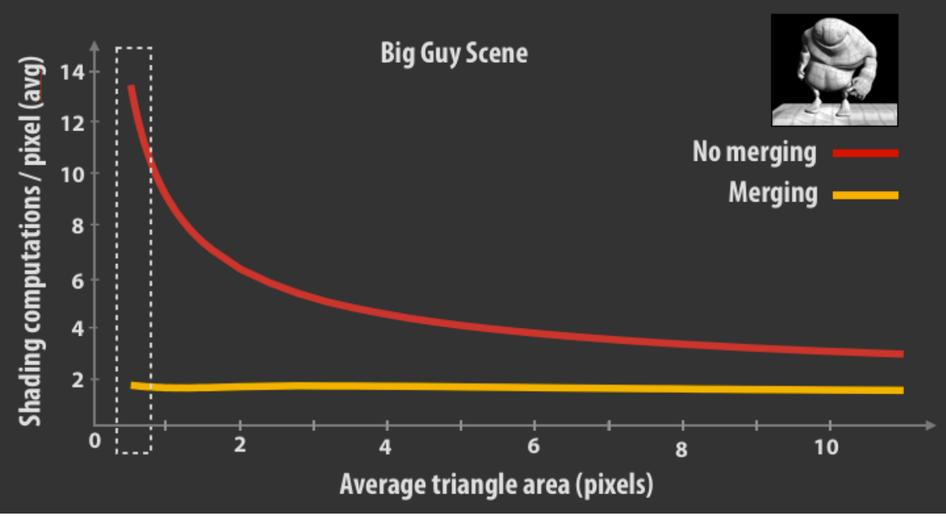
One point per slide!

One point per slide!

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

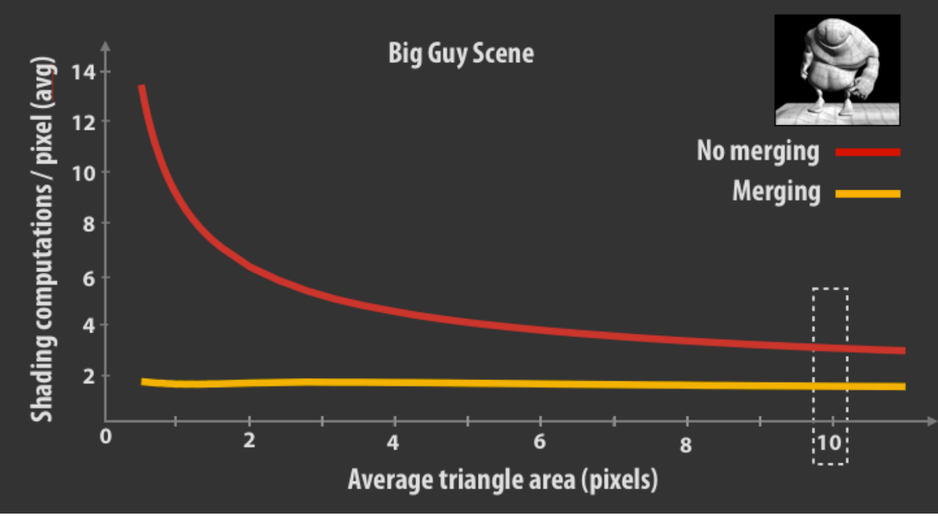
Merging reduces total shaded quad fragments

1/2-pixel-area triangles: 8x reduction

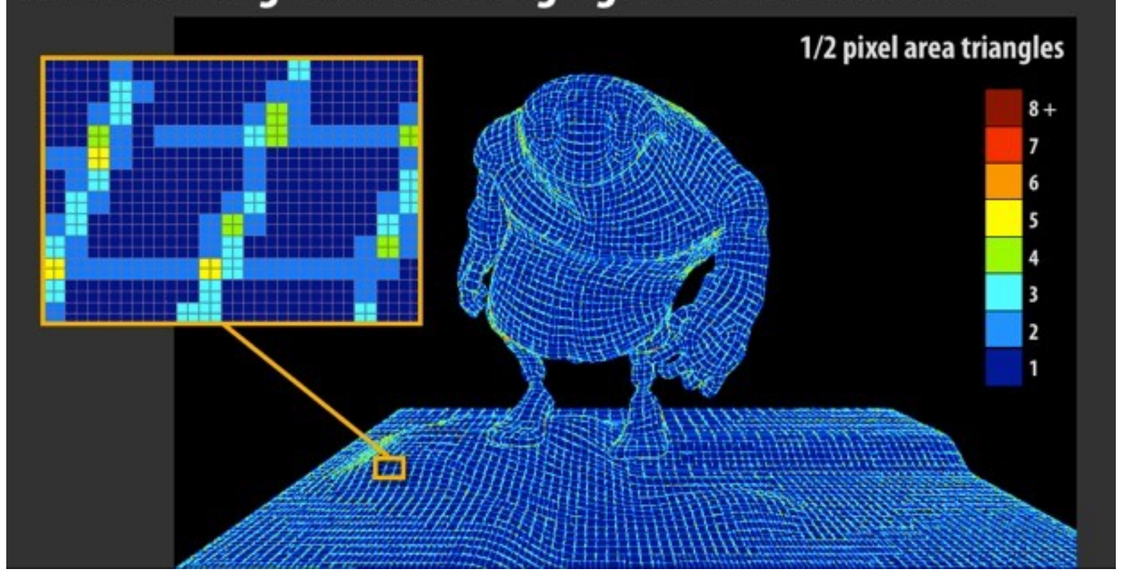


Merging reduces total shaded quad fragments

Ten-pixel-area triangles: 2x reduction

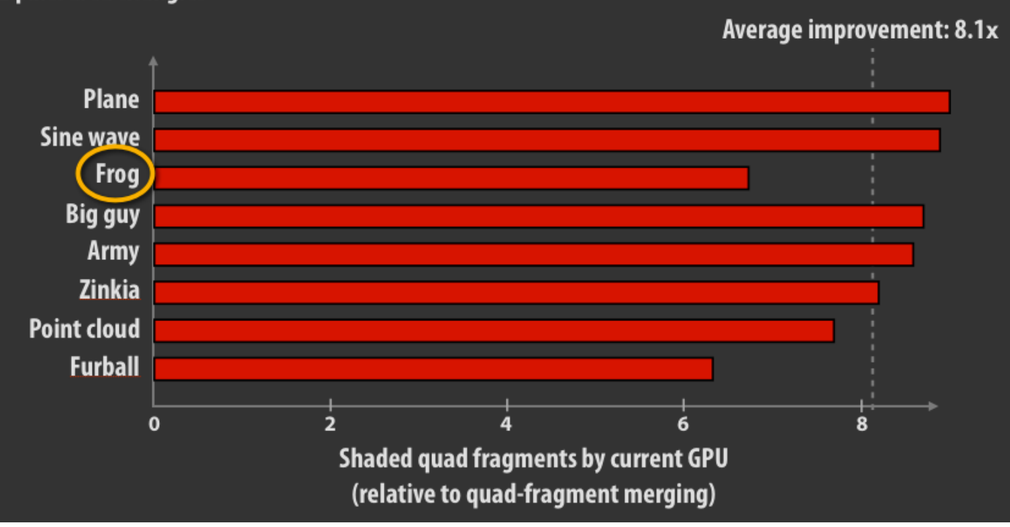


Extra shading occurs at merging window boundaries

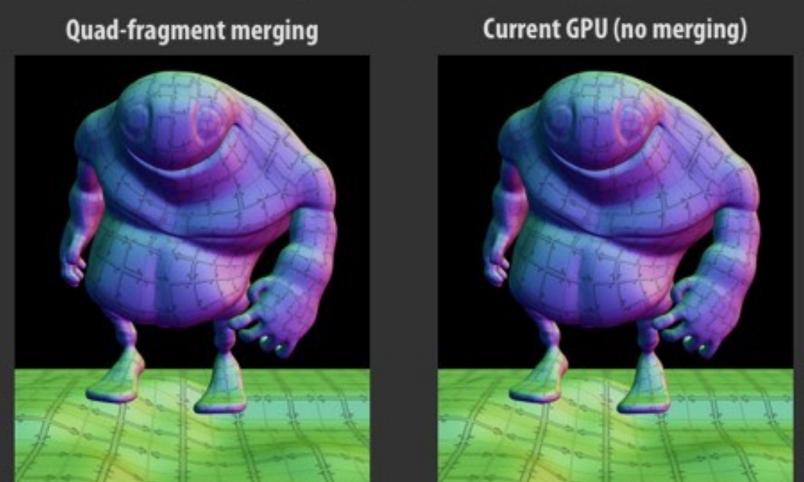


For micropolygons: factor of eight across scenes

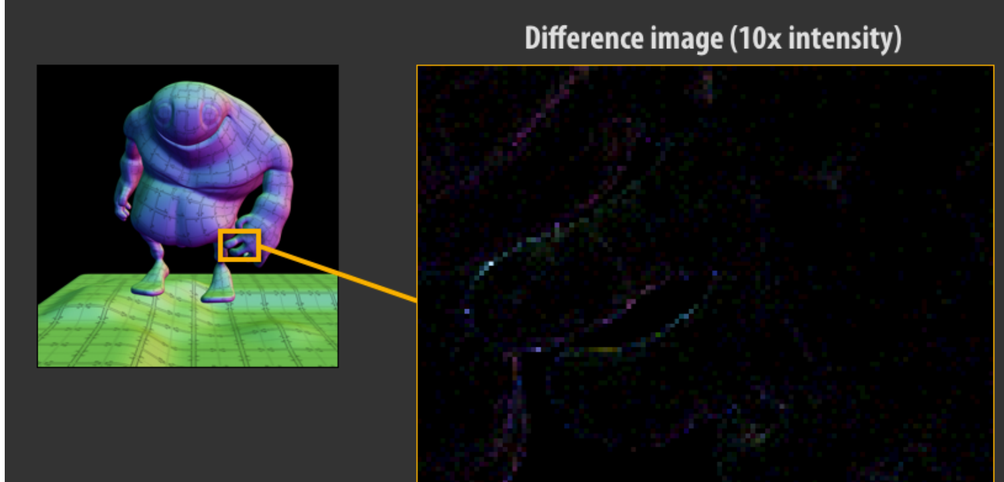
1/2 pixel area triangles



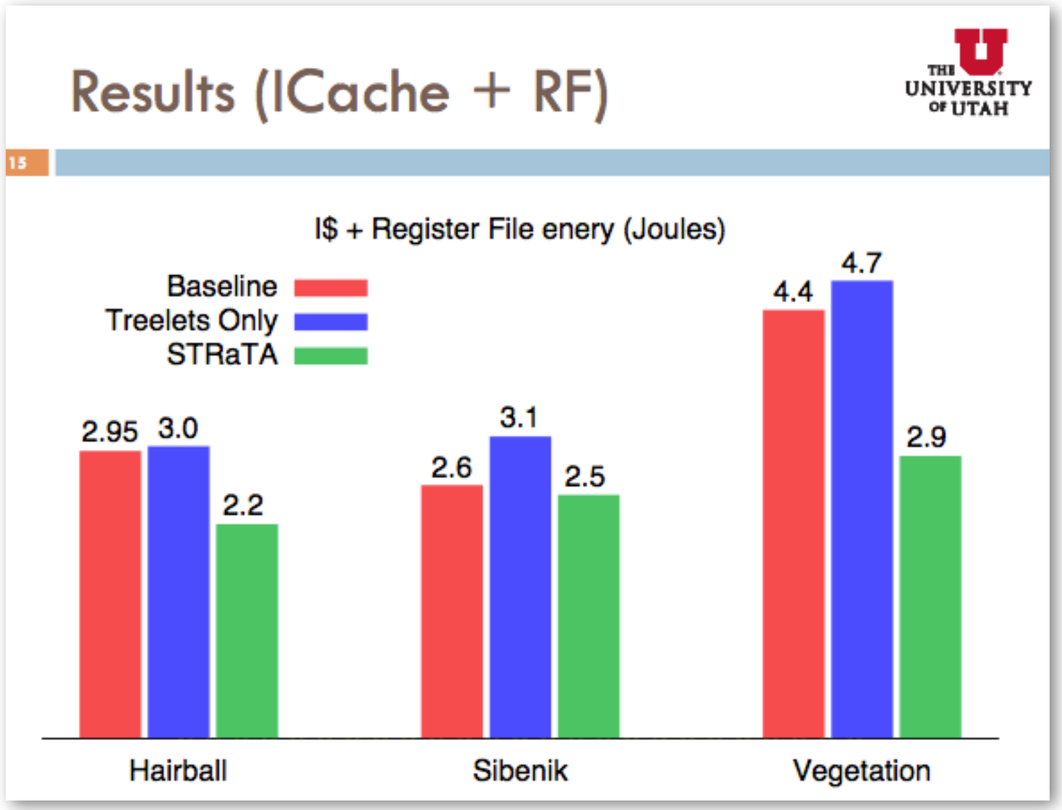
Nearly identical visual quality



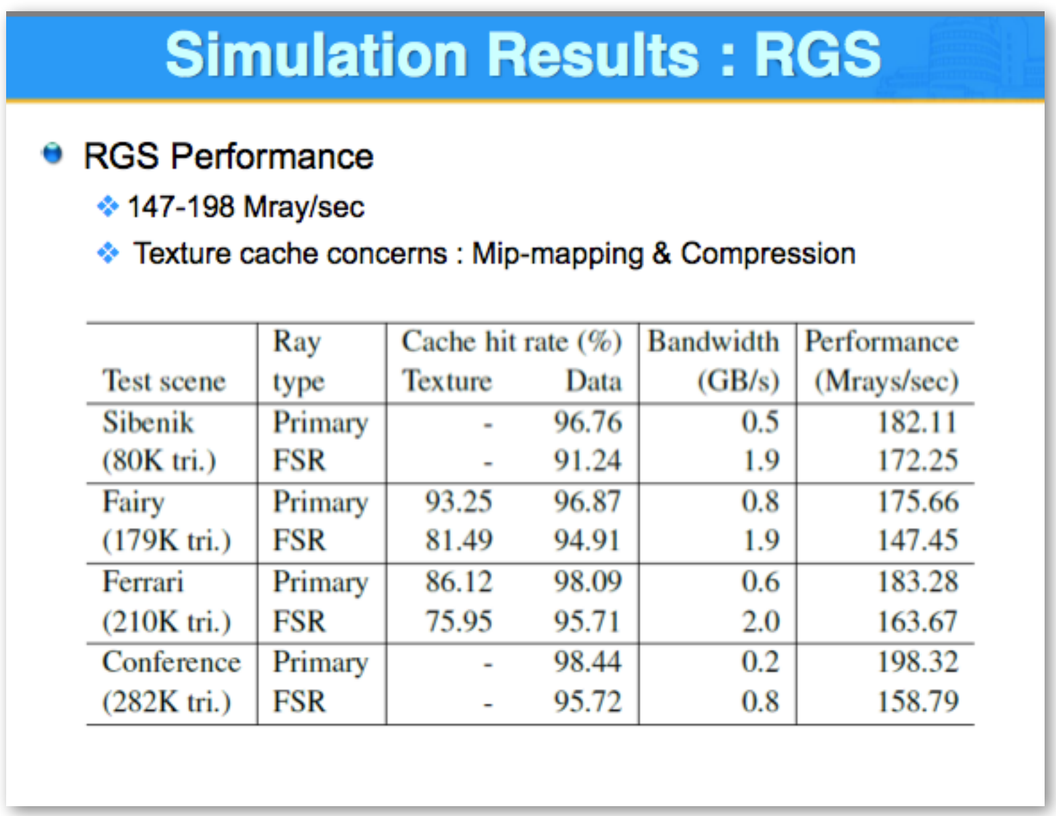
Differences exist near silhouettes



Bad examples of results slides



- Notice how you (as an audience member) are working hard to interpret the trends in these graphs
  - You are asking: what do these results say?
- You just want to be told what to look for



Simulation Results : RGS

- RGS Performance
  - 147-198 Mray/sec
  - Texture cache concerns : Mip-mapping & Compression

Test scene	Ray type	Cache hit rate (%) Texture	Data	Bandwidth (GB/s)	Performance (Mrays/sec)
Sibenik	Primary	-	96.76	0.5	182.11
(80K tri.)	FSR	-	91.24	1.9	172.25
Fairy	Primary	93.25	96.87	0.8	175.66
(179K tri.)	FSR	81.49	94.91	1.9	147.45
Ferrari	Primary	86.12	98.09	0.6	183.28
(210K tri.)	FSR	75.95	95.71	2.0	163.67
Conference	Primary	-	98.44	0.2	198.32
(282K tri.)	FSR	-	95.72	0.8	158.79

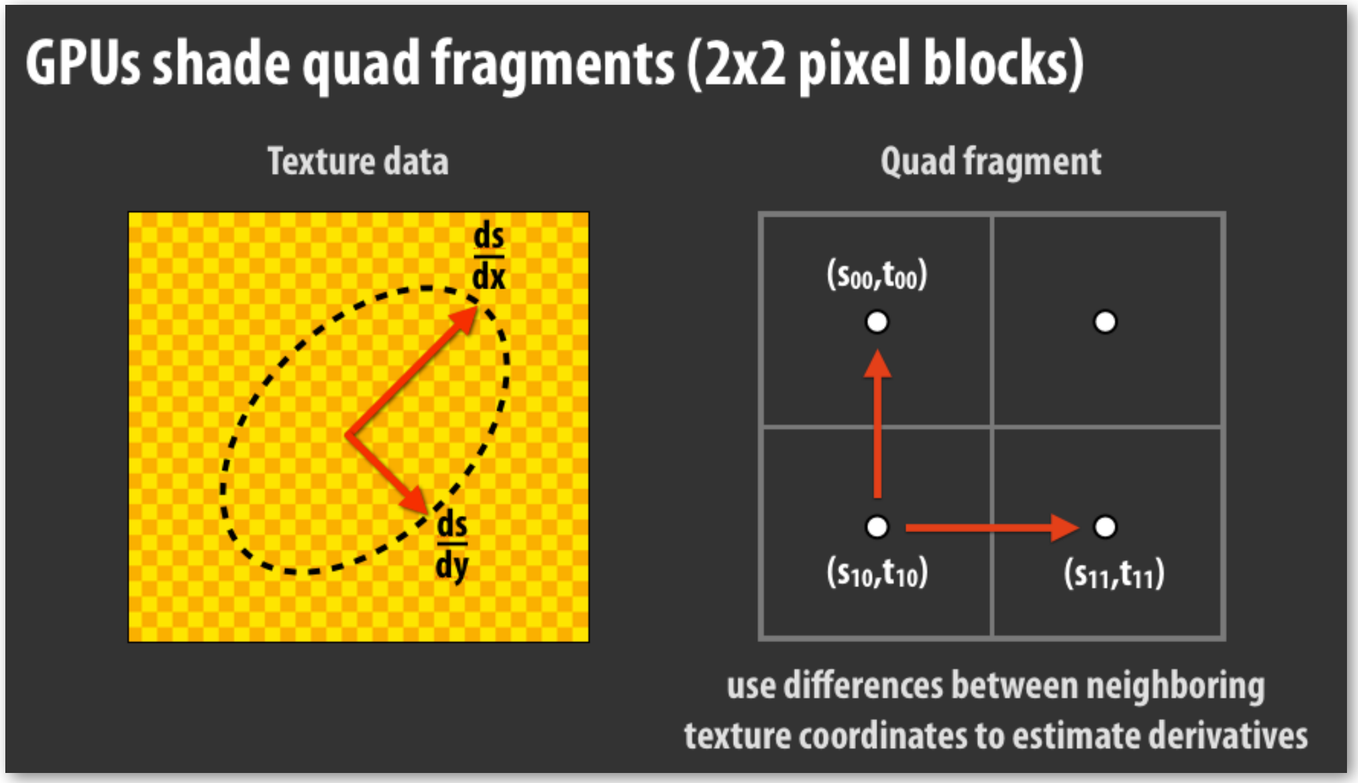
9.

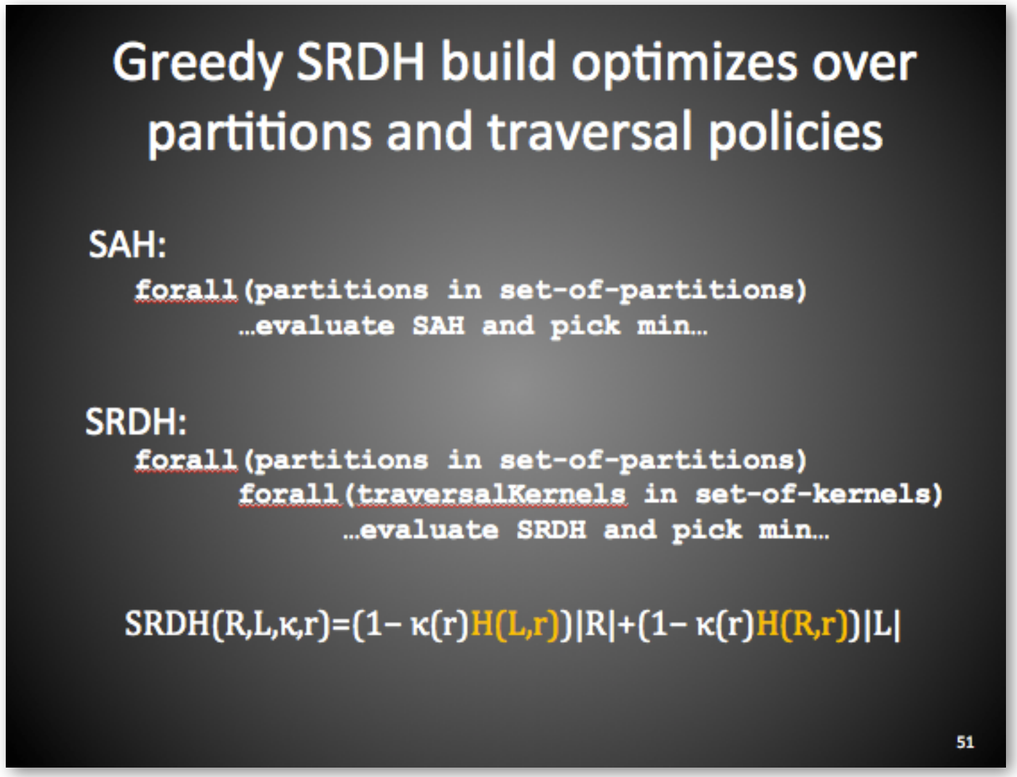
Titles matter.

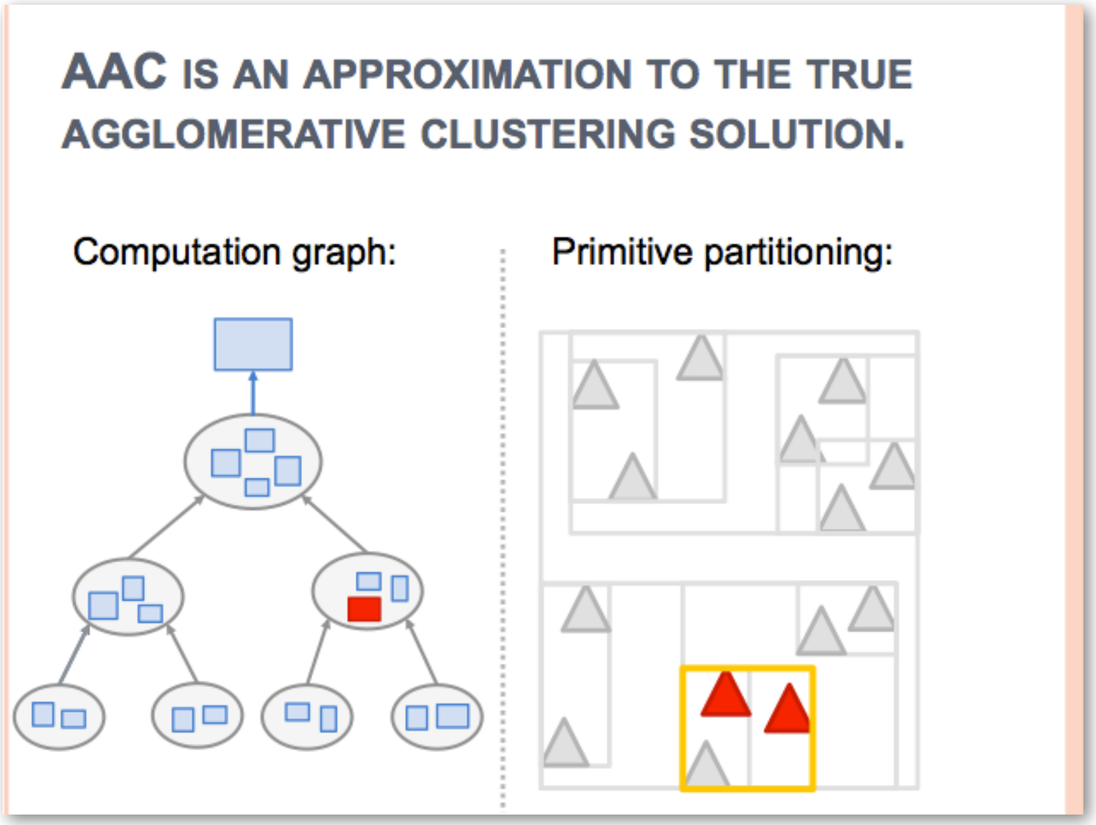
If you read the titles of your talk all the way through, it should be a great summary of the talk.

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

Examples of good slide titles







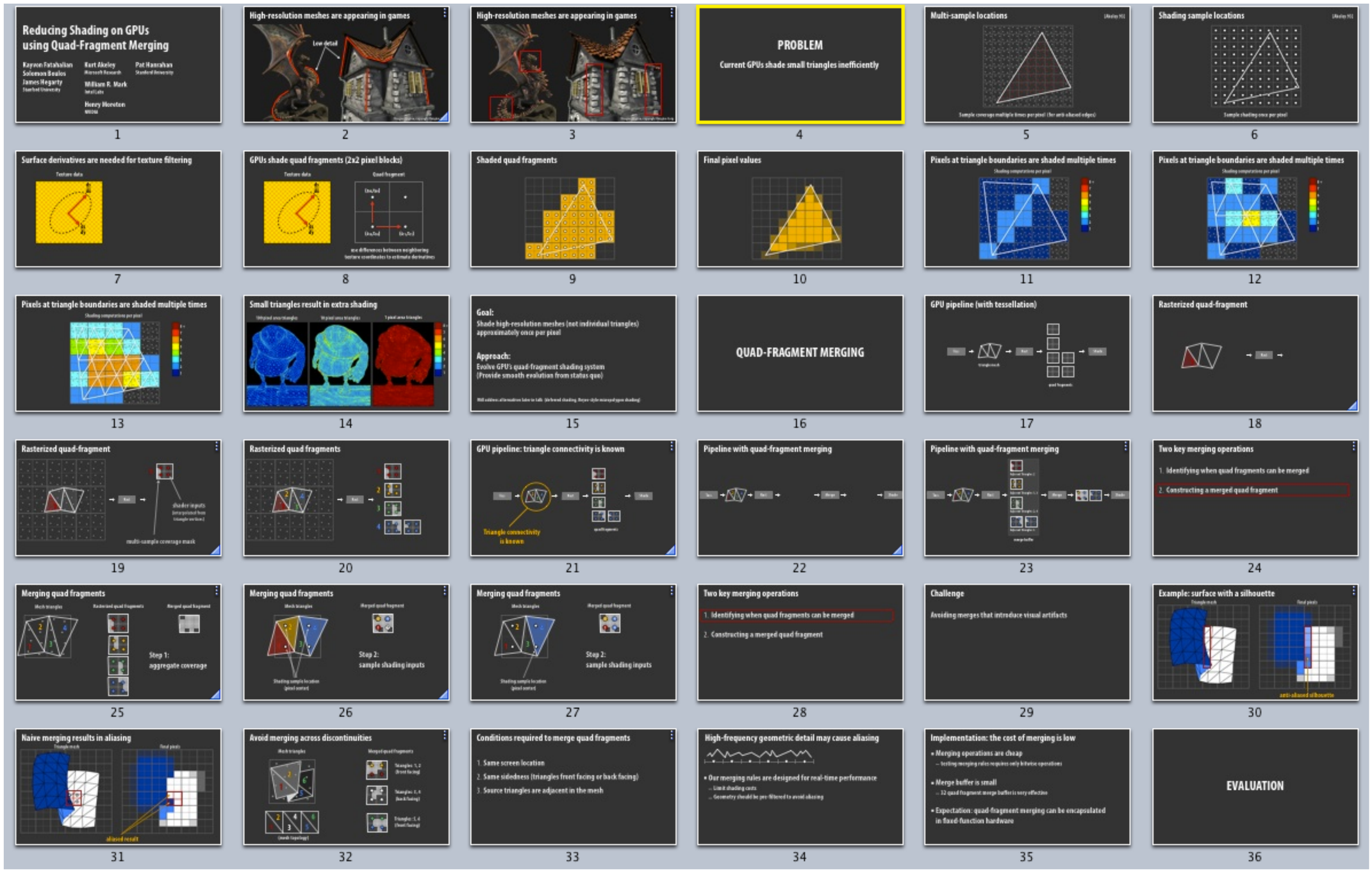
The reason for meaningful slide titles is convenience and clarity for the audience

"Why is the speaker telling me this again?"

(Why before what.)

Read your slide titles in thumbnail view

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



10.

Practice the presentation

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