Lecture 28:

Course Wrap Up & Project Presentation Tips

Parallel Computer Architecture and Programming
CMU 15-418/15-618, Spring 2017

Tunes

OneRepublic

Counting Stars

(Native)

“Lately, I've been, I've been losing sleep

Dreaming about the things that we could be…”

- everyone

Today

- Exam 2 discussion

- Parallelism competition hints/guidelines

- Wrap up: a few final comments about class topics

Announcements

- <u>Please</u> fill out course evaluations, it helps make the course better

- Final project presentations: Friday May 12th
  - 2pm on Wed May 10th staff + judges start looking at your project pages to pick finalists
    - 20 finalist teams announced at 2pm on Thursday
  - Parallelism Competition:
    - 8:30-11:30am, Rashid Auditorium (with breakfast and snacks)
    - Everyone is expected to attend to support their classmates.
  - Additional project presentations:
    - Presentations the rest of the day in Smith Hall to course staff

- Project reports due at 11:59pm that evening, no late days

Exam 2 discussion (no slides)

Final projects

Presentation day

- Yes, presentations are considered when determining your overall project grade, but in general Friday is a celebration of finishing the course: the "Parallelism Competition" and all the project presentations are chance to talk about the neat work you have done.

Presentation format

- Each group has six minutes to talk
  - Plus 1-2 minutes for questions from judges
- Presentation format is up to you: slides, live demo, etc.
  - Start with name(s) of students, title of project on a title slide
  - Both students in a 2-person group should speak
- Present off your own laptop, or make arrangements with staff

Project presentation tips

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 come up to you after the talk

- Clarity is highly prized: the audience remembers you
  - "Hey man, that was a great talk... are you looking for a job anytime soon?"

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 parallel programming
  - CS terminology/concepts need not defined

- 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

- The judges (and course staff) are trying to figure out 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?

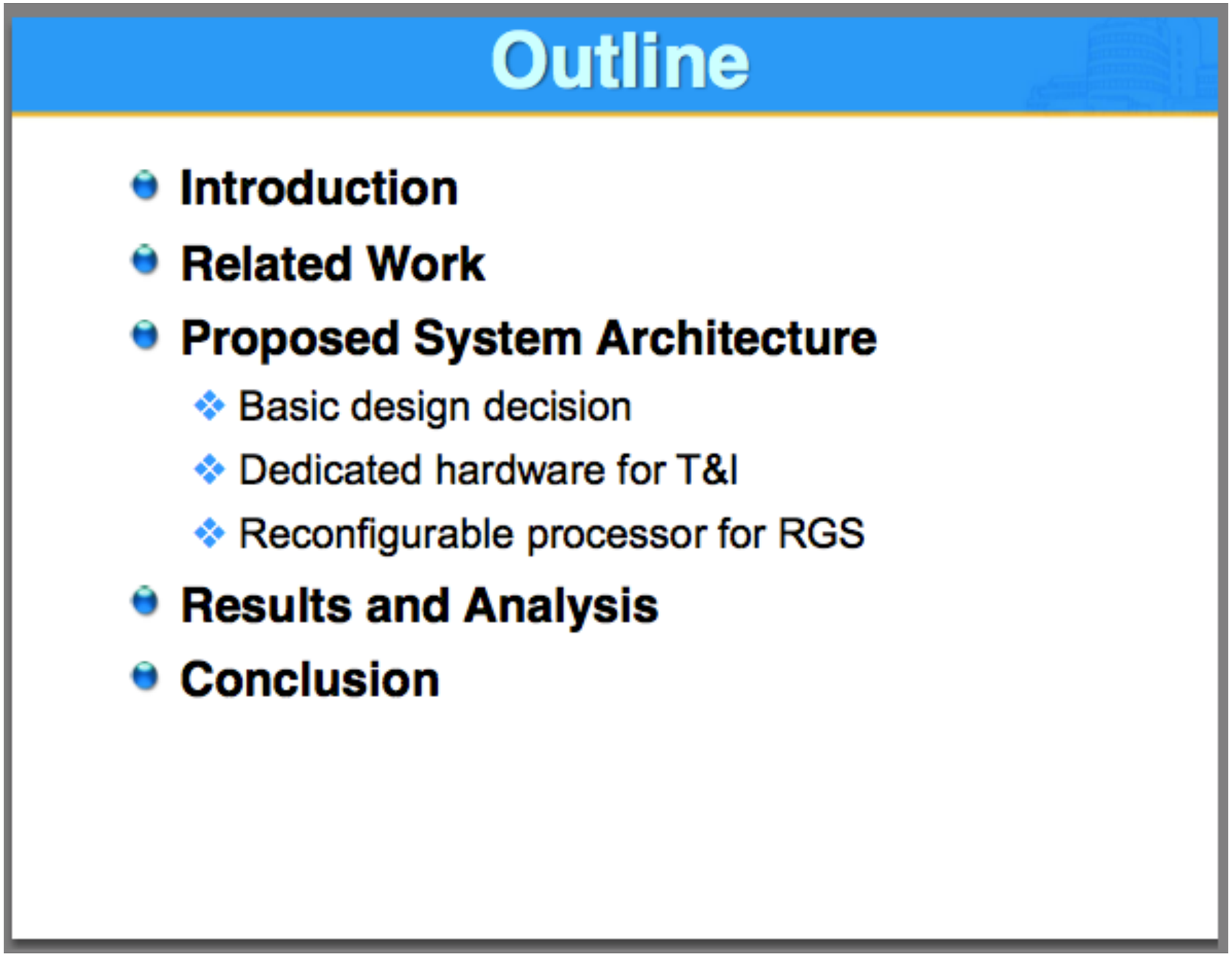
Pick a focus

- 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

Ignoring every sentence matters

Never ever, ever, ever do this!



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).

- 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)

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 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)
- 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

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)

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 state 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

6.

Surprises* are almost always bad:

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

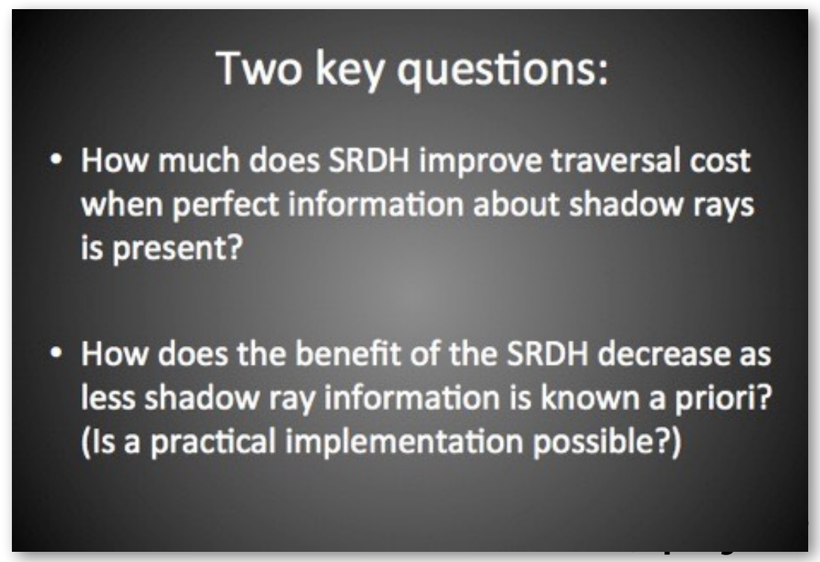
* I am referring to surprises in talk narrative and/or exposition. A surprising result is great.

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 approach?")

- **In the algorithm/system description section:**
  - "We need to first establish some terminology…"
  - "Even though I just told you our solution to 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 section:**
  - Speaker: "Key questions to ask about our approach are…"
  - Listener: "Thanks! I agree, those are good questions. Let's see what the results say!"



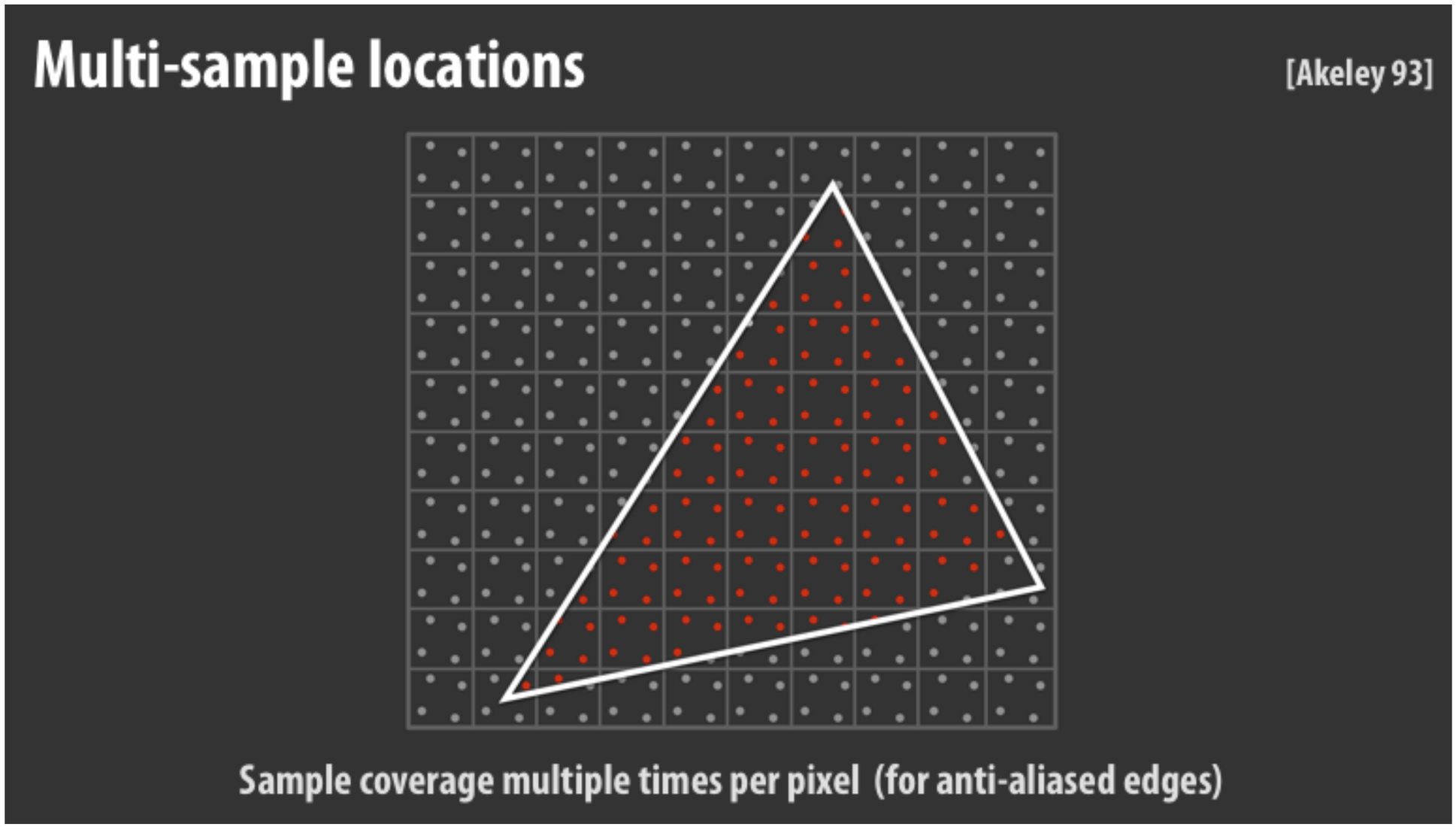
7.

Always, always, always explain any figure or graph

(remember, the audience does not want to think)

Explain every figure

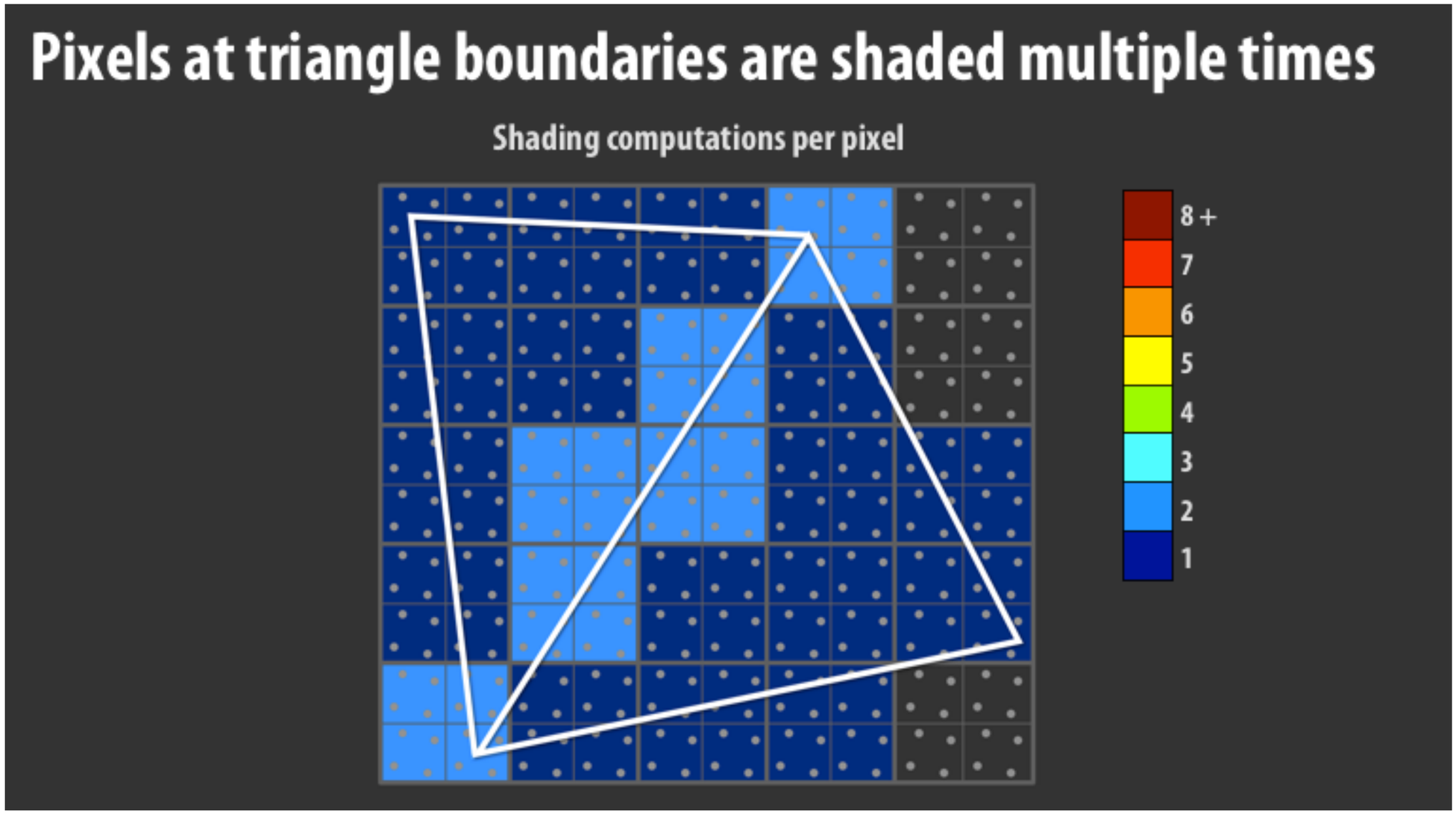
- Explain every visual element used in the figure (never make the audience decode a figure)
- Refer to highlight colors explicitly (explain why the visual element is highlighted)



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

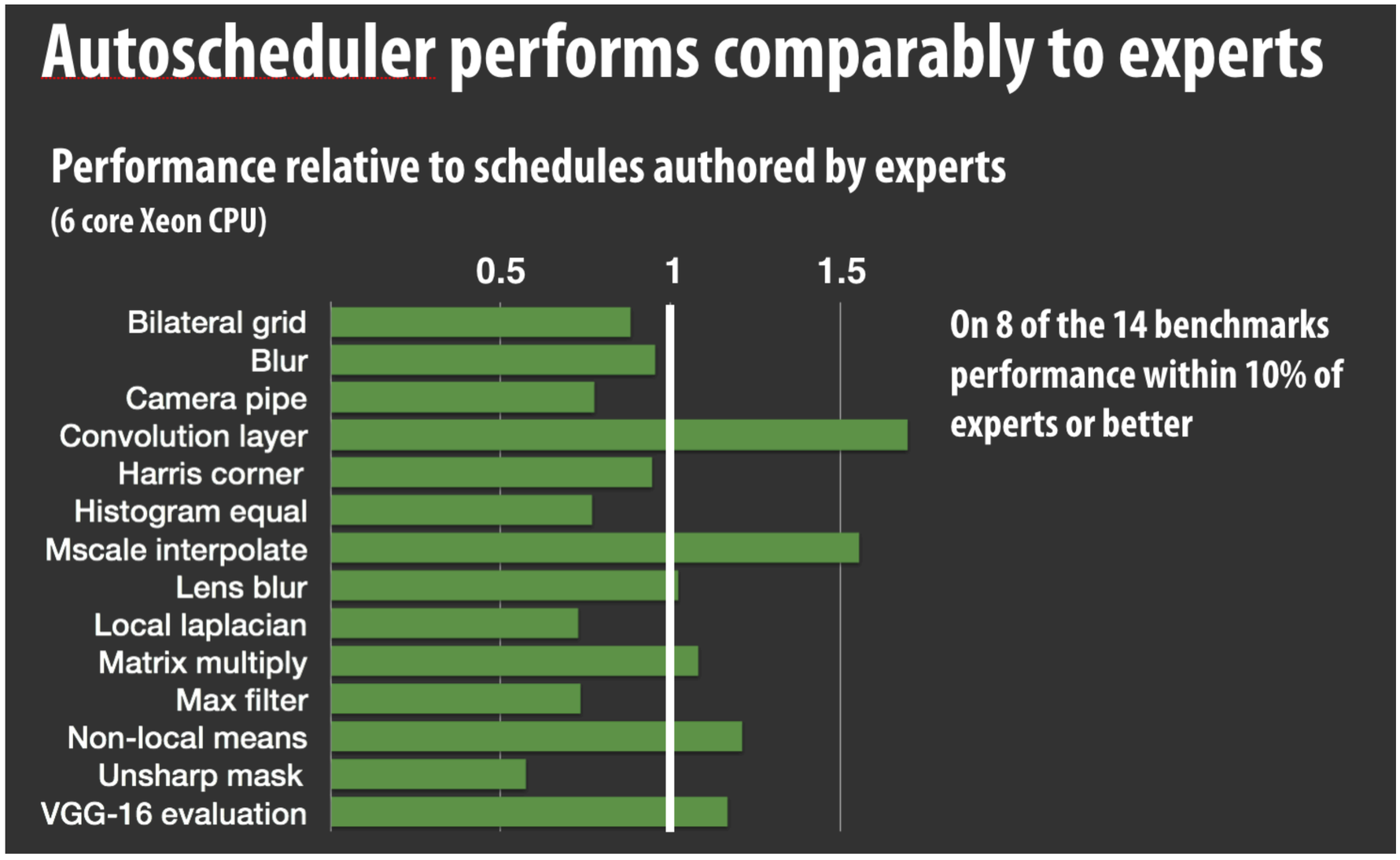
- Lead the listener through the key points of the figure
- Useful phrase: "As you can see…"
  - It's like verbal eye contact. It keeps the listener engaged and makes the listener happy… "Oh yeah, I can see that! I am following this talk! Yippee!"



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. As you can see 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 one point 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.

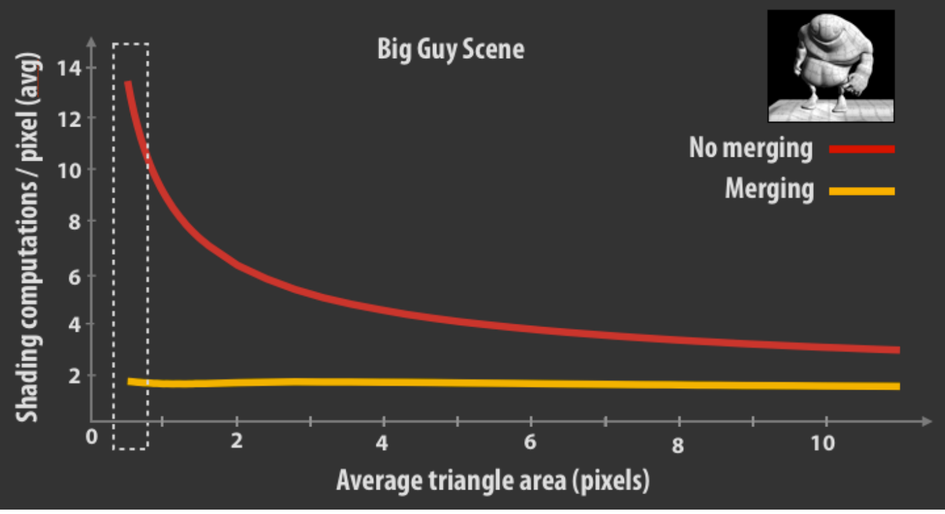
8.

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

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

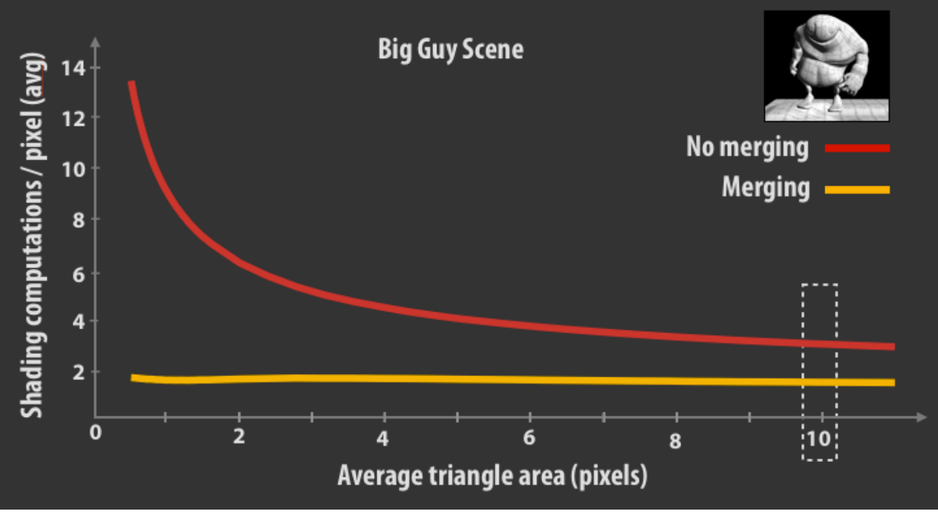
Merging reduces total shaded quad fragments

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

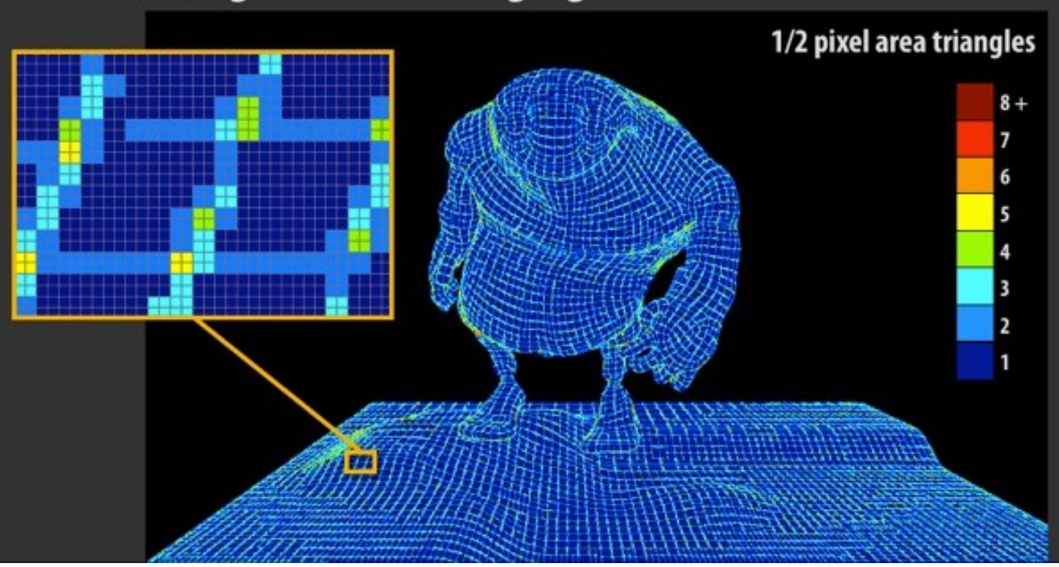


Merging reduces total shaded quad fragments

Ten-pixel-area triangles: 2x reduction

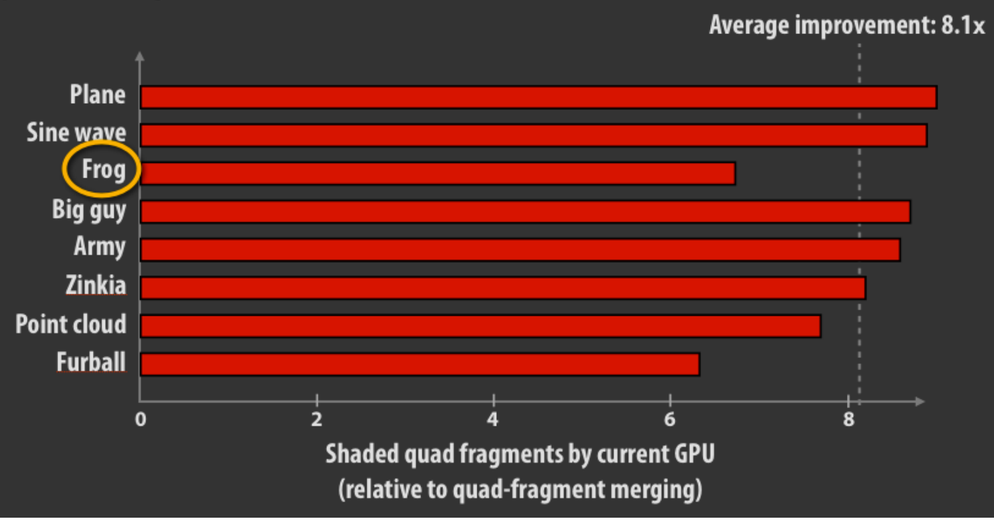


Extra shading occurs at merging window boundaries

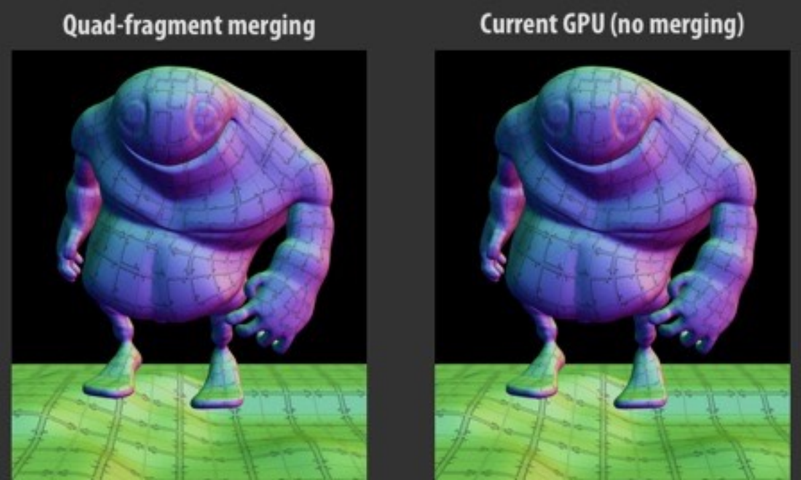


For micropolygons: factor of eight across scenes

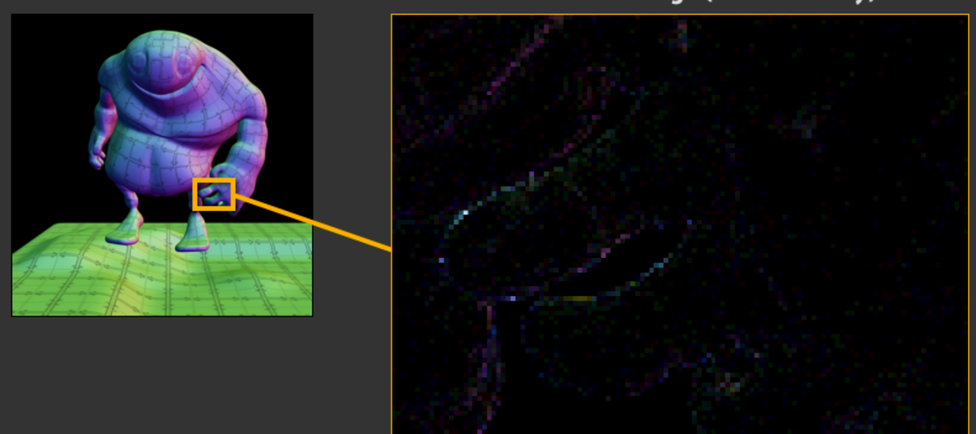
1/2 pixel area triangles

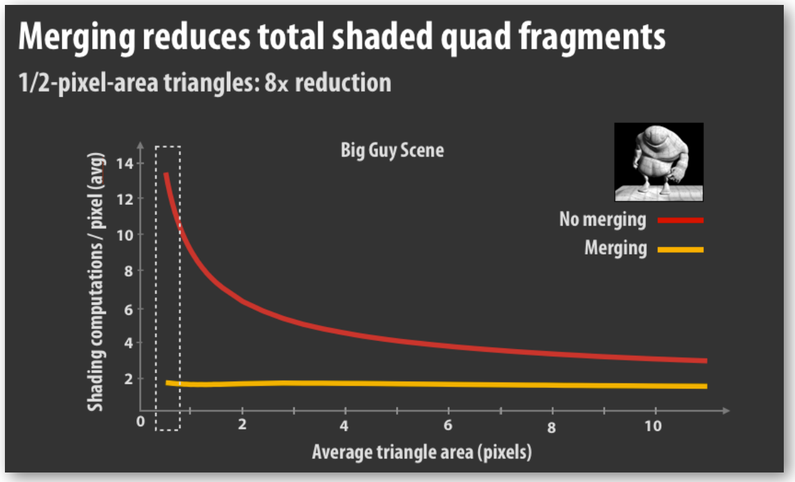


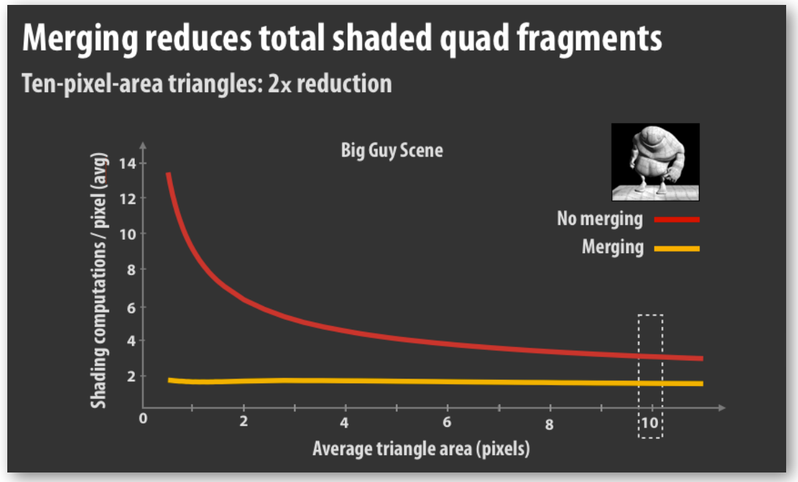
Nearly identical visual quality

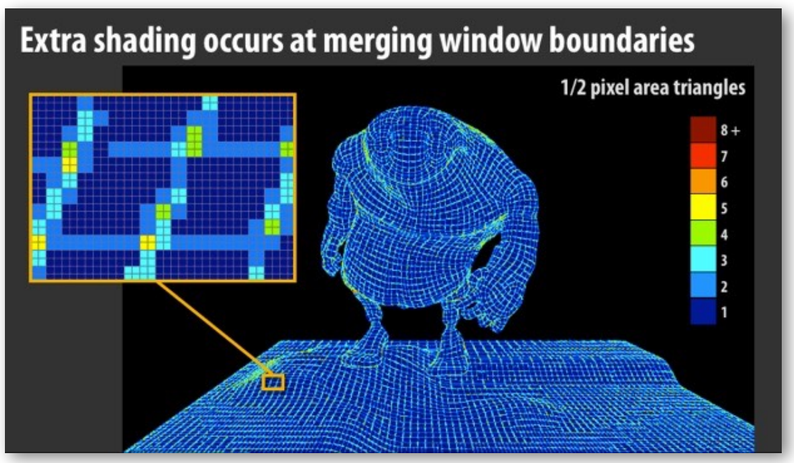


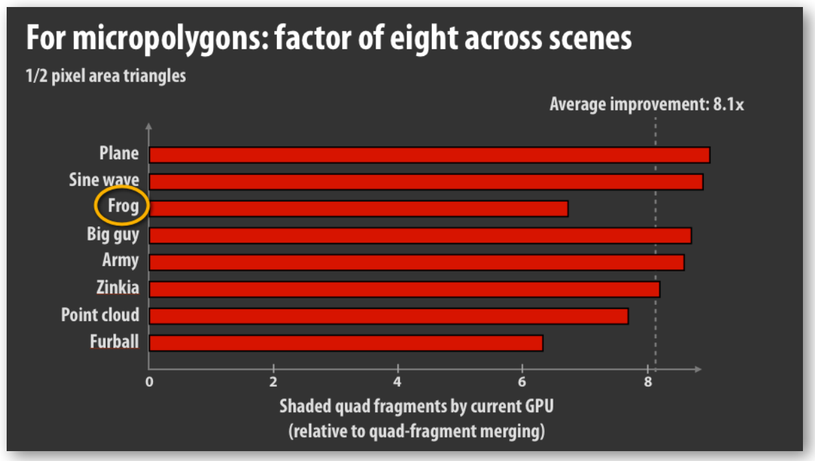
Differences exist near silhouettes

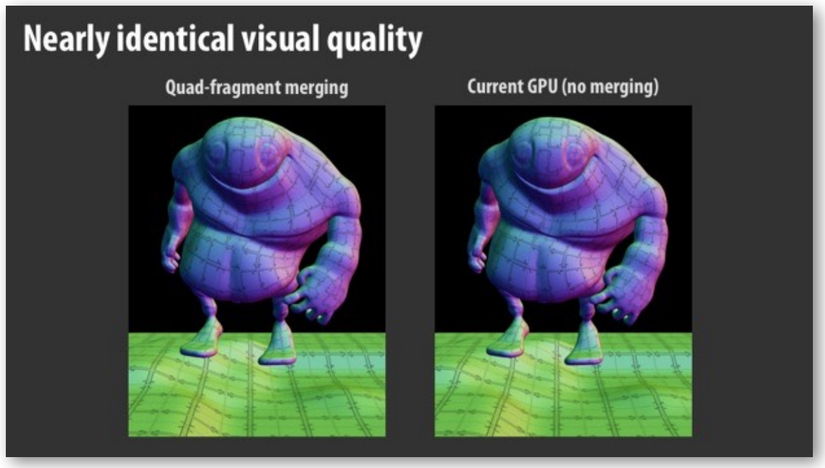


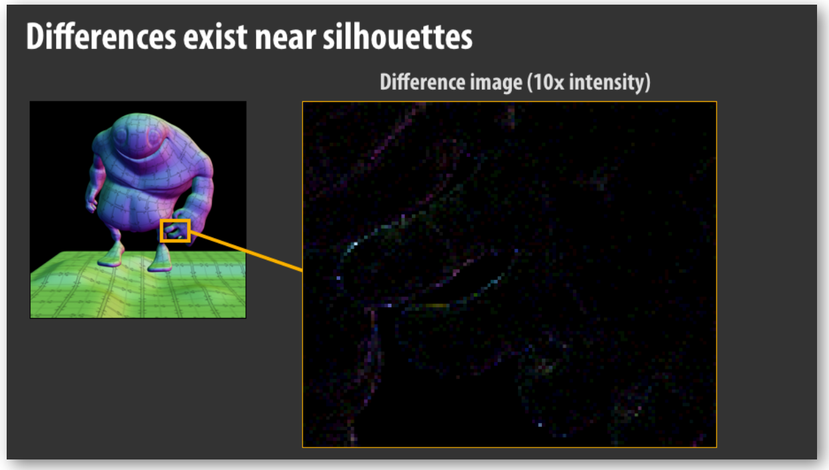












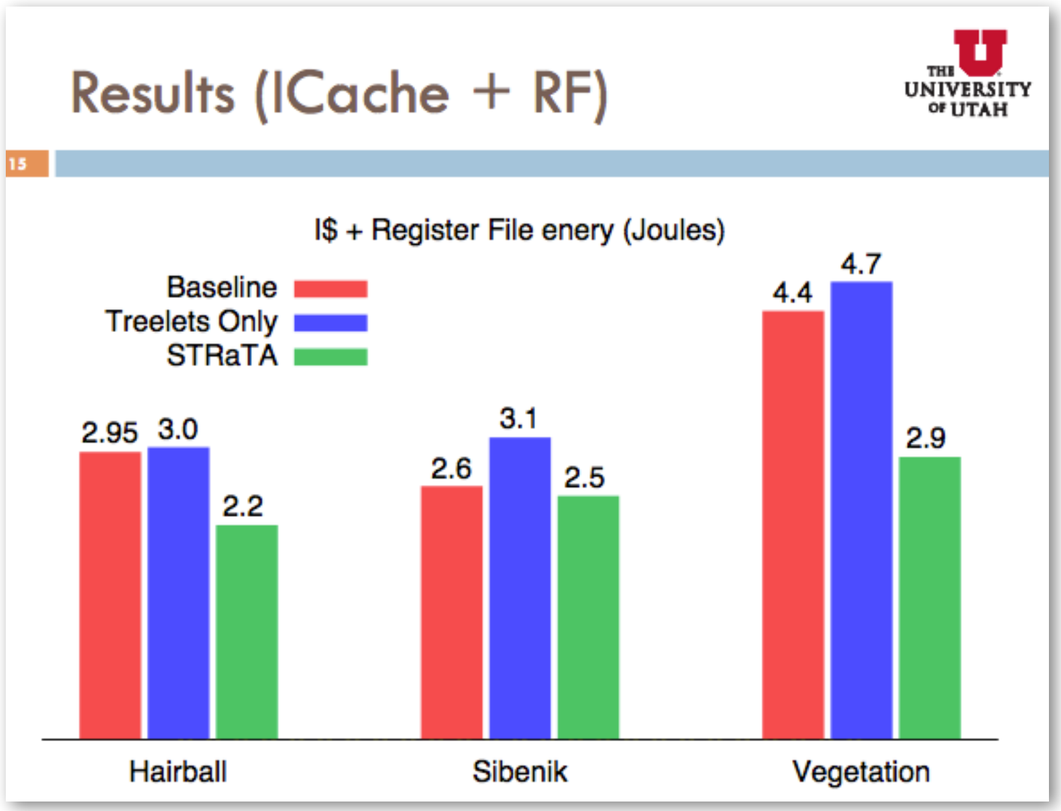
Place the point of the slide in the title:

It provides context for interpreting the graph

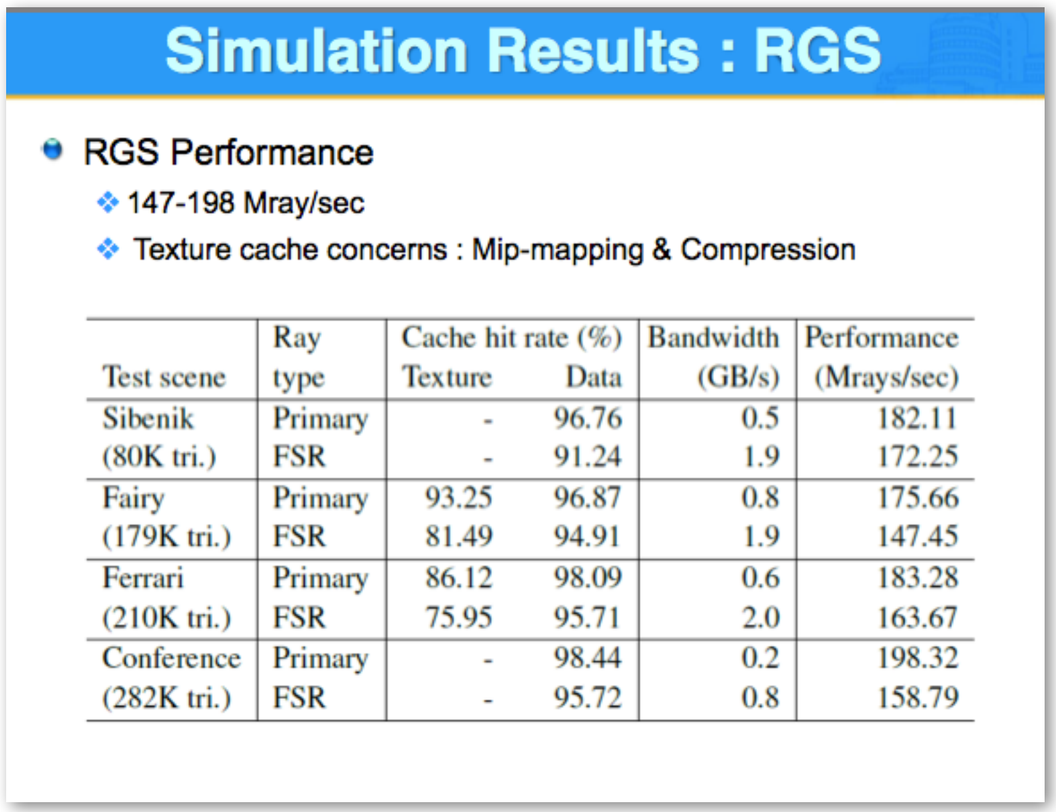
(Listener: "Let me see if I can verify the point in the graph to check my understanding")

Another example of the "audience prefers not to think" principle

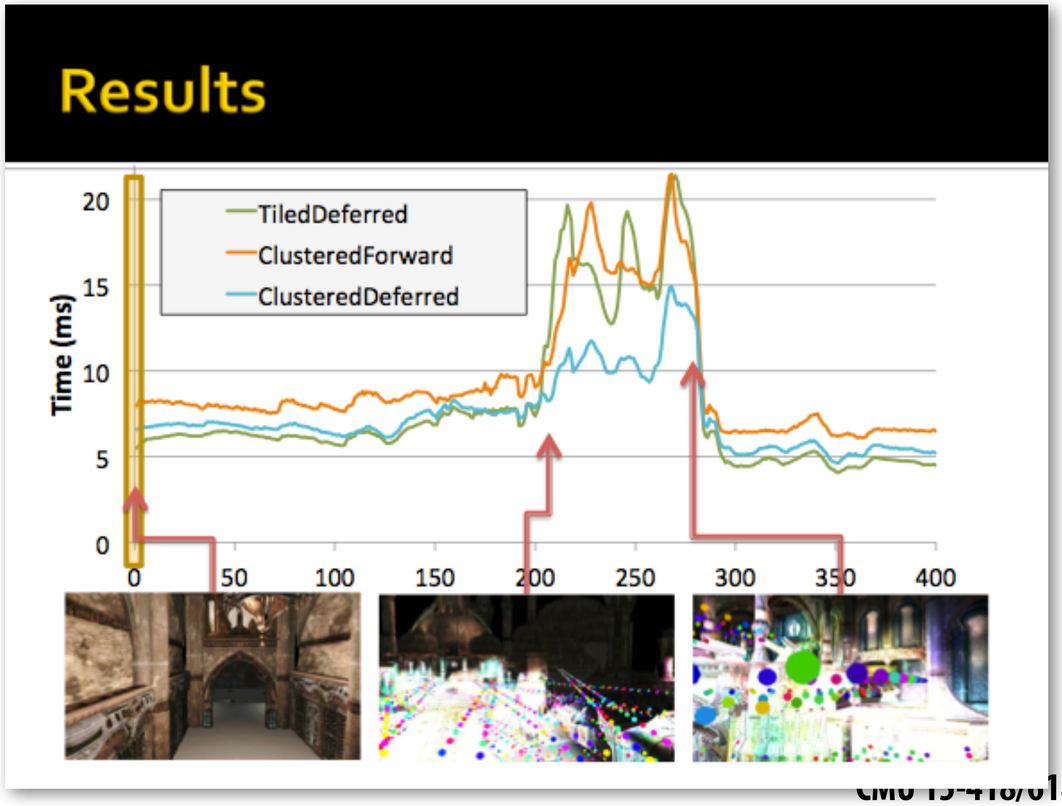
Bad examples of results slides



- Notice how you (as an audience member) are working 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



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



Titles matter.

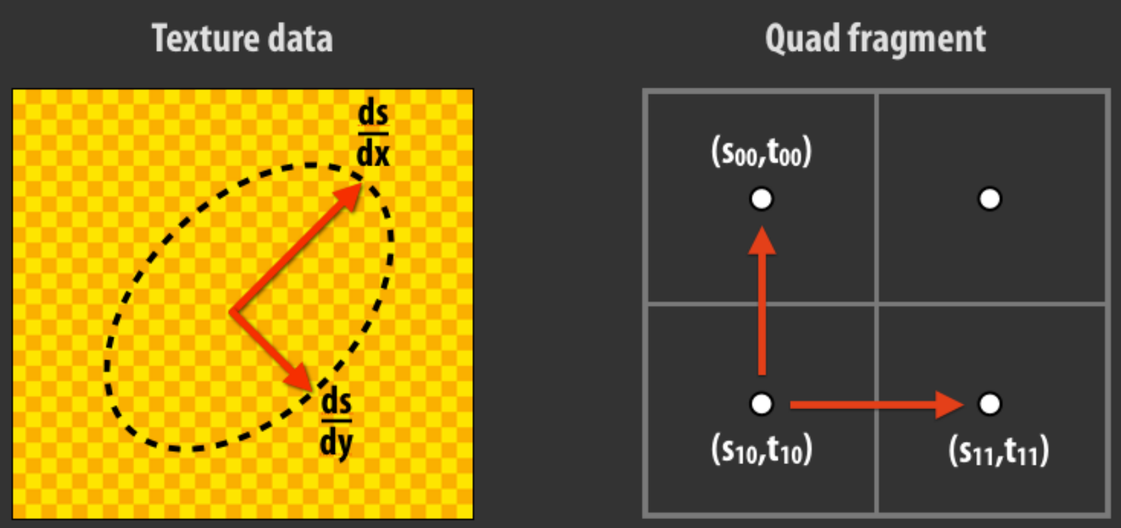
9.

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 rest of the 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…
```

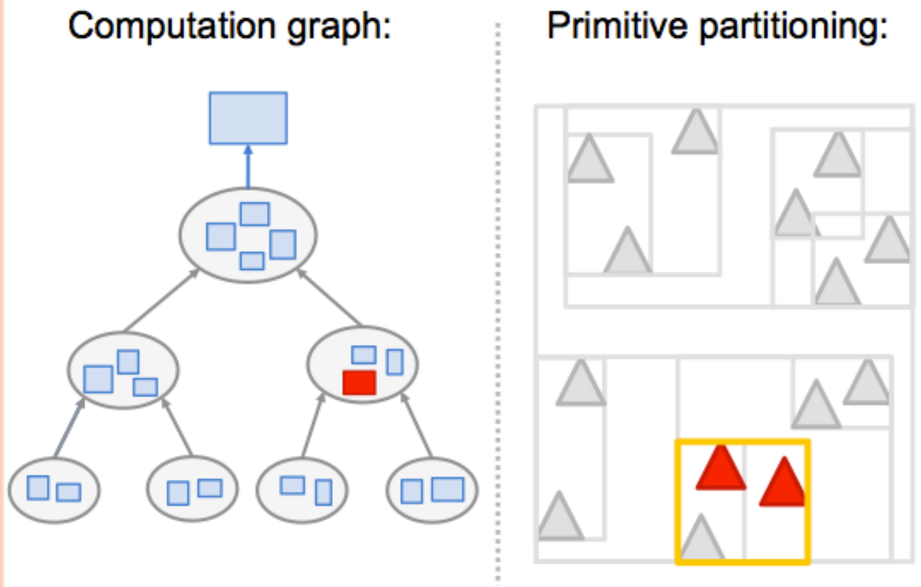
SRDH:

```
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

Audience: "Why is the speaker telling me this again?"

(Why before what.)

10.

Provide evidence that you've done a good job (You have successfully applied principles learned in 418/618 to a problem of your choosing)

(This is a 418/618 specific tip, but it certainly generalizes to other contexts as well.)

Evaluating results

(Show your system is fast, or efficient, or as good as it can be given what you've learned in this class … in other words, that you did a good job)

- **Compare against published results**
  - **"Our code is 10% faster than this publication (or this well known system)"**

- **Determine a fraction of peak**
  - **"We achieve 80% of peak performance on this machine"**

- **Be truthful about comparisons between a CUDA implementation utilizing an entire GPU and single threaded, non-SIMD C program on a CPU (I'd rather not hear this conclusion: "GPU is 300x faster than the CPU".)**

11.

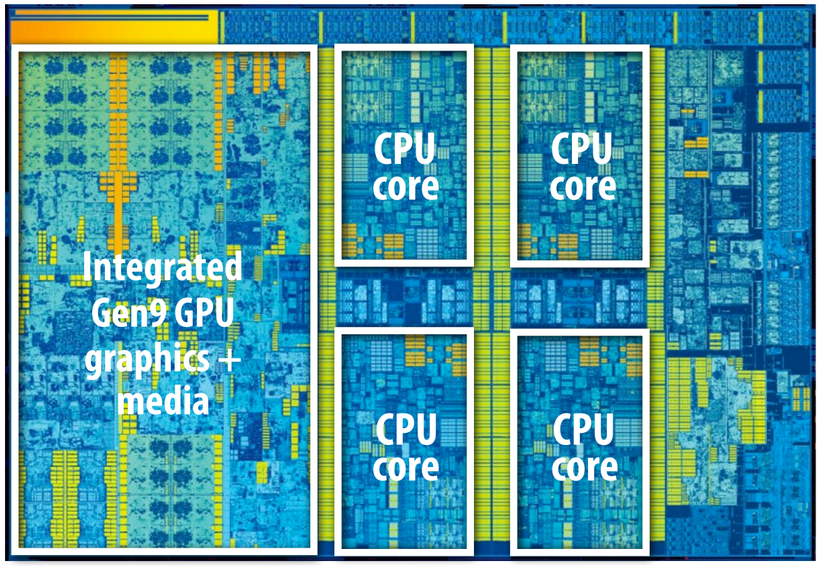
Practice the presentation

Practice the presentation

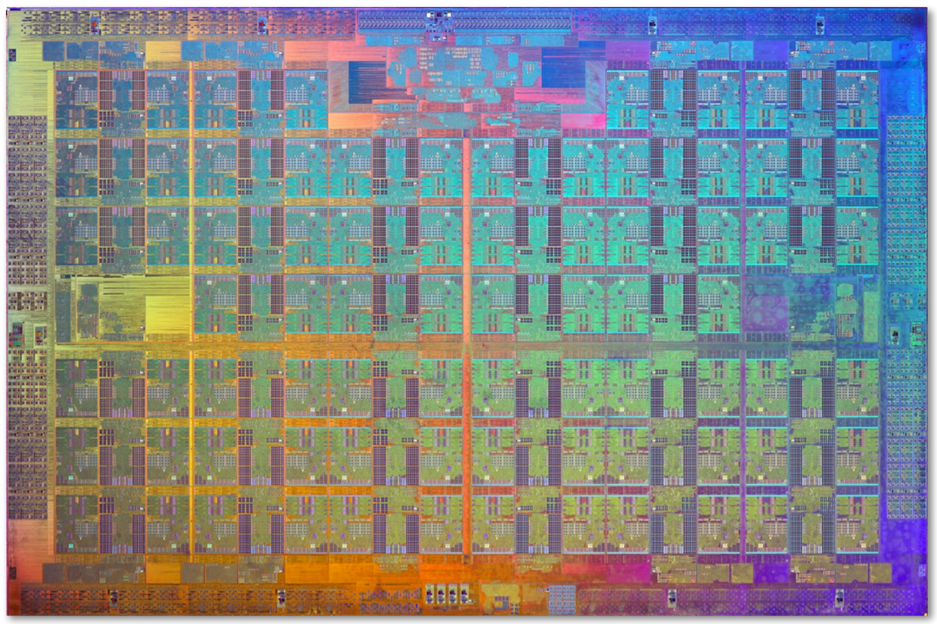
- Given the time constraints, you'll need to be smooth to say everything you want to say
- To be smooth you'll have to practice
- I hope you rehearse your demo or presentation several times the night before (in front of a friend or two that's not in 418)
  - It's only a 6 minute talk. So a couple of practice runs are possible in a small amount of time

Course wrap up

For the foreseeable future, the primary way to obtain higher performance computing hardware is through a combination of increased parallelism and hardware specialization.



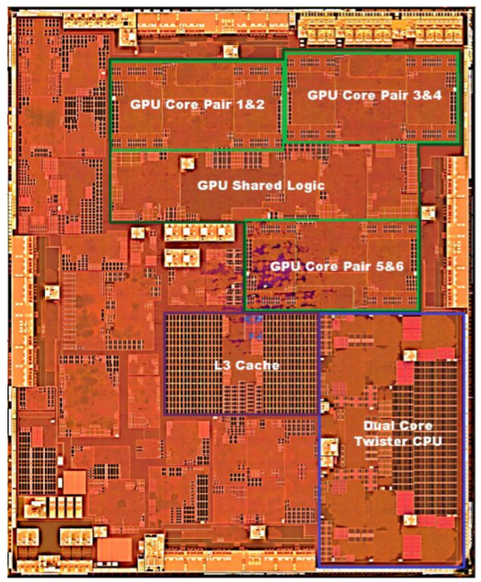
Intel Core i7 CPU + integrated GPU and media



Intel Xeon Phi
72 cores, 16-wide SIMD, 4-way multi-threading

NVIDIA Maxwell GPU
(single SMM core)
32 wide SIMD
2048 CUDA/core threads per SMM

FPGA
(reconfigurable logic)



Apple A9
Heterogeneous SoC
multi-core CPU + multi-core GPU + media ASICs

Today's software is surprisingly inefficient compared to the capability of modern machines

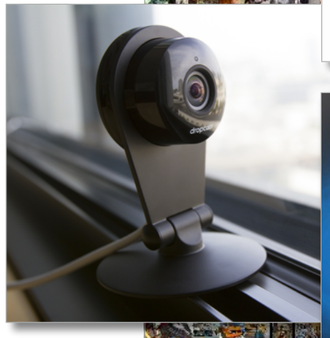
A lot of performance is currently left on the table (increasingly so as machines get more complex, and parallel processing capability grows)

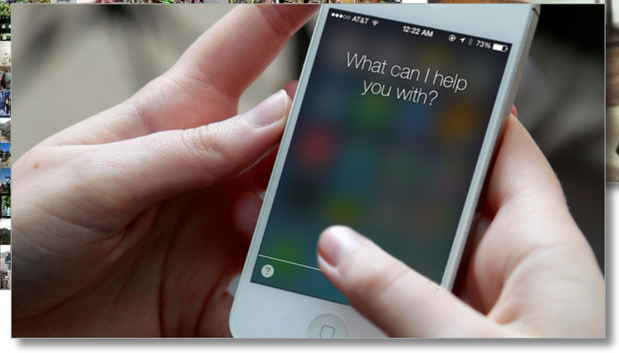
Extracting this performance stands to provide a notable impact on many compute-intensive fields (or, more importantly enable new applications of computing!)

Given current software programming systems and tools, understanding how a parallel machine works is important to achieving high performance.

A major challenge going forward is making it simpler for programmers to extract performance on these complex machines.

This is particularly important given how exciting (and efficiency-critical) the next generation of computing applications are likely to be.





Key issues we have addressed this semester

Identifying parallelism

(or conversely, identifying dependencies)

Efficiently scheduling parallelism

1. Achieving good workload balance

2. Overcoming communication constraints:

Bandwidth limits, dealing with latency, synchronization

Exploiting data/computation locality = <u>efficiently managing state</u>!

3. Scheduling under heterogeneity (using the right processor for the job)

We addressed these issues at many scales and in many contexts

Heterogeneous mobile SoC

Single chip, multi-core CPU

Multi-core GPU

CPU+GPU connected via bus

Clusters of machines

Large scale, multi-node supercomputers

Thank you to our TAs!

Alex

Ravi

Teguh

Junhong

Yicheng

Tao

Anant

Riya

Call for TAs!

- 15-418/618 is offered in both the Fall (by Mowry, Railing) and in the Spring (by Randy and I)

- We will be looking for TAs for future offerings of the course!
  - This is important, since we <u>need</u> great TAs in this course to be successful

Other classes with overlapping topics

- Visual Computing Systems (15-769)
- Computer architecture (18-447)
- Compilers (15-441)
- Distributed computing (15-447)
- Operating systems (15-410)
- Database systems (15-445) (new course next semester Pavlo)

Beyond assignments and exams

What I see a lot of…

Amazing CMU CS student

Works really hard to maximize grades in CS classes

High GPA looks good on resume

Good resume handed out at CS job fair

Resume gets student first-round interview

Student knows their stuff in interview
(aces fine-grained linked list locking question)

GOOD JOB
Woot!

But let's be honest, this is what really happens…

Amazing CMU CS student

Works DOES NOT SLEEP in order to maximize grades in MANY CS classes
or
A DOUBLE MAJOR
or
A TRIPLE MAJOR

(Most waking hours spent on coursework, tired all the time)

Even more impressive resume handed out at CS job fair

Resume gets student first-round interview

Student knows their stuff in interview
(aces fine-grained linked list locking question)

GOOD JOB
Woot!
("but man, CMU is a brutal place")

Discussion:

Why?

RULE:

To be really good* at something, you have to be really talented (you are), AND you have to work really hard at it.

You have to struggle/agonize over it.
You have to immerse yourself in it.
You have to think about it all the time.

There are very, very few exceptions to this rule.
(And they are really, really lucky people.)

So this is not a talk urging you to work less. (Sorry.)

*** Note: good != successful. Success also requires fortunate circumstances and luck.**

HYPOTHESIS:

For some of you (but not all): challenging yourself to ace as many classes as possible may not be the most effective way to maximize your efforts at CMU and opportunities afterward.

It may not be the best way to get a competitive job.

It may not be the best way to get the coolest jobs.

It may not be the best way to prepare yourself have the most impact in a future job.

There are other ways to demonstrate and prepare yourself for future excellence.
(these other ways are often more challenging than taking extra classes)

Idea 1: wisely manage yourself in classes in your later years at CMU.

(yes, this is much easier said than done)

Imagine this situation

You are signed up for a normal load of four classes.

One of them is my class, 15-418: Parallel Computer Architecture and Programming. Woot!

You are considering loading up with a fifth class…

say 15-410, or 15-440…

Lots of options!

- You could do what it takes to get A's in both classes (probably middle-of-the-road work due to lack of time)

- What if you gave reasonable effort in my class, resulting in a B (you took my class because you anticipate exposure to the material might be useful in the future, although you don't intend to make a living in parallel programming)? But… this gave you time to do <u>outstanding</u> work on the assignments and final project in another class!

The “ivory-tower” advice

You should find ways to immerse yourself in the projects and ideas you find most interesting. It is the best way to learn deeply.
(and that will show up in an interview. “Tell me about your project… wait, you implemented what?”)

The more practical advice

The really unique opportunities (a.k.a., coolest jobs) in the world tend to come through people that know you, not by submitting resumes.

You better believe colleagues in industry are asking us about the best students all the time. (finding good people is hard, and frustrating, for employers)

The best bosses are looking for people that have done special things.

Idea 2: try undergraduate research

The conventional path I was talking about…

Amazing CMU CS student

Works DOES NOT SLEEP in order to maximize grades in MANY CS classes
or
A DOUBLE MAJOR
or
A TRIPLE MAJOR

(Most waking hours spent on coursework, tired all the time)

Even more impressive resume handed out at CS job fair

Resume gets student first-round interview

Student knows their stuff in interview
(aces linked list reversal question, recalls what a mantissa is)

GOOD JOB
Woot!
(“but man, CMU is a brutal place”)

An alternative path…

Amazing CMU CS student

Takes fewer classes, but DOESN'T SLEEP because he/she does an amazing project in 15-418. (really interested in parallel programming)

Student: "Hey Kayvon, I liked your class, is there anything I can help with in your research group next semester?"

Kayvon: "Yo! You did the coolest work in 418 in YEARS, you should totally come help with this project in my group."

Student gets awesome experience working side-by-side with CMU Ph.D. students and professors. Learns way more than in class. (BUT STILL PROBABLY DOESN'T SLEEP… SO IT GOES)

Kayvon, to super-awesome friend in industry: "Hey, you've got to hire this kid, they know more about parallel architecture than any undergrad in the country. They've been doing publishable research on it."

WICKED GOOD JOB

Woot!

Kayvon, circa 2002 (junior year at CMU)

My TA in Professor Hodgins' computer animation class (Ph.D. student Kiran Bhat) pulled me aside on the last day of class and told me I should come join the Graphics Lab

Kiran

Why research (or independent study)?

- You will learn way more about a topic than in any class.
- You think your undergrad peers are amazingly smart? Come see our Ph.D. students! (you get to work side-by-side with them and with faculty). Imagine what level you might rise to.
- It's way more fun to be on the cutting edge. Industry might not even know about what you are working on. (imagine how much more valuable you are if you can teach them)
- It widens your mind as to what is possible.

What my Ph.D. students are working on these days…

- Generating efficient code from image processing or deep learning DSLs (Halide Autoscheduler)
- Designing a new shading language for future real-time graphics pipelines
- Parallel computing platforms for analyzing large video collections at scale (Scanner: "Spark for video")
- Designing more efficient DNNs to accelerate evaluation
- Computer graphics tools for theatrical lighting design

And maybe you might like it and want to go to grad school?

Remember my comment about people…

Without question, the number one way to get into a top grad school is to receive a strong letter of recommendation from a CMU faculty member. You get that letter from participating in a research team.

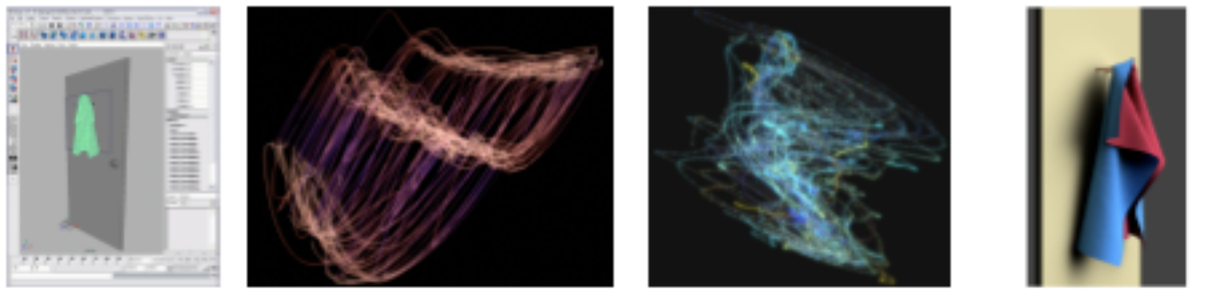
DWIC letter: (“did well In class” letter) What you get when you ask for a letter from a faculty member who you didn’t do research with, but got an ‘A‘ in their class. This letter is essentially thrown out by a Ph.D. admissions committee.

I'm no exception: got gentle hints from my professors

(Note: this was also true in deciding to be a professor)

Precomputing Interactive Dynamic Deformable Scenes

Doug L. James and Kayvon Fatahalian
Carnegie Mellon University



(a) Precomputation (b) Reduced dynamics model (c) Reduced illumination model (d) Real-time simulation

Figure 1: *Overview of our approach:* (a) Given a deformable scene, such as cloth on a user-movable door, we precompute (impulsive)
reduction on observed dynamic
c) Deformed state geometries are
(d) The final simulation responds
ffects, and runs in real time.

ral Phenomena Animation, Phys-
y Based Modeling

f our everyday world, and a key
ures, clothing, fractured materi-
alistic natural environments. It
ge for real-time interactive envi-
vironments may wish to incorpo-
onents for increased realism, but
of secondary importance so very
vailable. Unfortunately, many re-
ll notoriously expensive to simu-
le nonlinear deformable systems
mentally expensive [Bridson et al.
me constraints can be onerous.
few (if any) major video games
deformable physics is a substan-
ollisions complicate both runtime
of interesting deformable scenes,
zing physical models in real-time
tic real-time animation of global
expensive for deformable scenes,
computed as easily as for rigid

ke a balance between complexity
in types of interactive deformable
interactions, to be simulated at
od tabulates state space models of
s in a way that effectively allows
runtime. To limit storage costs
t the state space models into very
st-squares (Karhunen-Loève) ap-
analysis. One might note that the

Subject: Re: [redacted] recommendations
From: "Jessica Hodgins" [redacted]
Date: Fri, November 22, 2002 8:08 am
To: "Doug L. James" [redacted] (more)
Priority: Normal
Options: View Full Header | View Printable Version | Download this as a file

I can do it but I REALLY think that you should be applying for
PhD programs, not master's programs.

Jessica

On Nov 22, 11:08am, Kayvon Fatahalian wrote:
> Subject: [redacted] recommendations
>
> I have applied to the CS masters program at [redacted], and am soliciting
> recommendations to accompany my application. could I ask either of
you to
> send a copy of letters you have drafted for me, either for NSF or (in
Doug's
> case for CMU 5th year) over to [redacted] as well.
>

Research is just one option…

(Despite what many of us biased faculty tell you,
there are many other equally good ones)

Why not start your own project?

Start your own project

Interested in applying technology to a problem that excites you? Give it a shot!

There are plenty of independent study opportunities at CMU.
(and there's funding available)

Like it enough to be your own boss?

Consider starting your own company.
(Project Olympus might give you some money.)

Why go work for Zuckerberg when you can start a company that kicks his ass?
(or he buys for $1B like Instagram)

TartanHacks

ScottyLabs

SURG

StuCo:

OLYMPUS

My Big Point

There are many ways to be EXCELLENT in CS at CMU

There are many ways to be excellent in CS at CMU

Take more classes

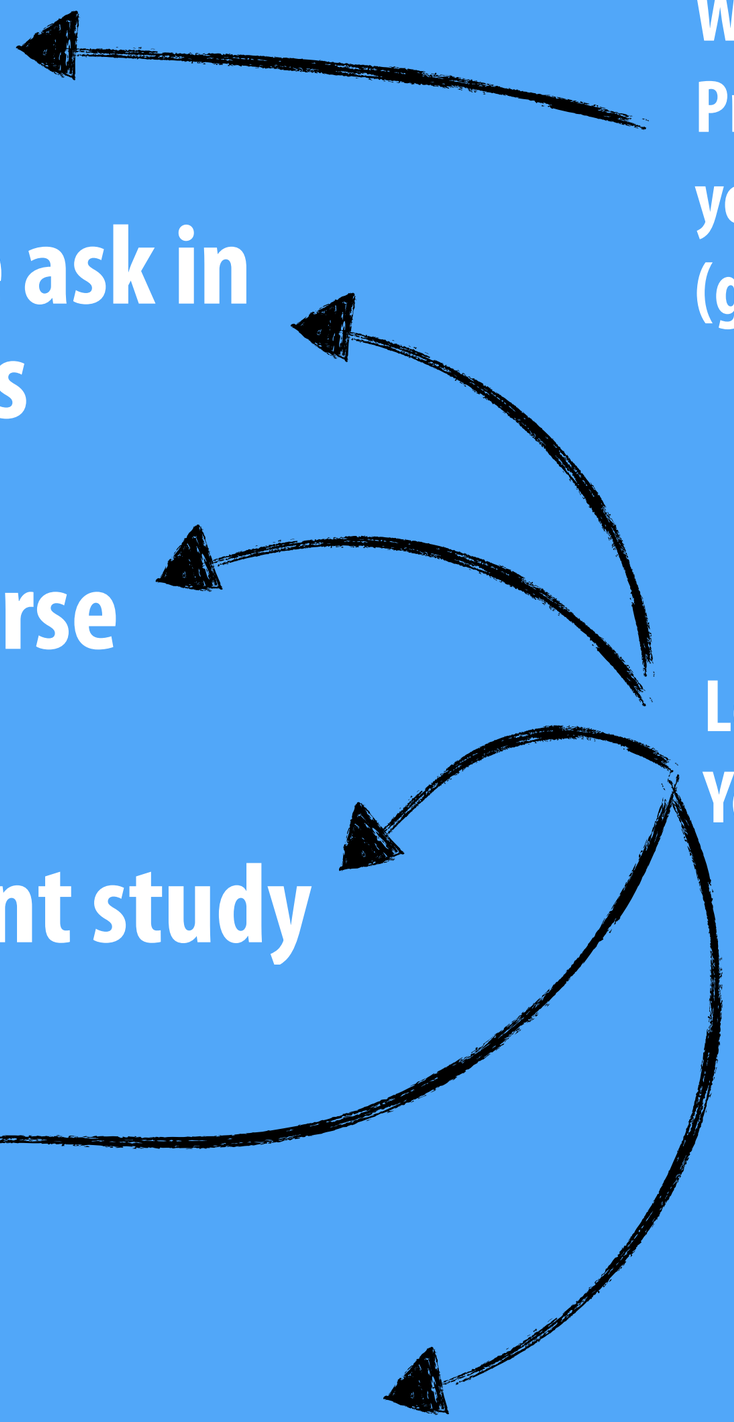
Go beyond what we ask in your favorite classes

Help redesign a course

Lead an independent study

Do research

Create a startup / non-profit



Well-structured effort:
Professors give you the problems, you have to solve them (great problem solvers get A's)

Less-structured effort:
You pick the problems to work on.

But...

Failure.

Taking on harder (and more open ended challenges) means you are more likely to have a hiccup.

The cost of failure?

Go beyond what we ask in class
Create a startup

Do research
Lead an independent study

Yes, there is a higher likelihood of having a setbacks in these activities than in a class.

I encourage you to practice some risk taking while at CMU

Getting over fear of temporary* failure to embrace measured risk is extremely powerful life skill to learn now.

* Let's be honest here: Failure is not a good thing, but if it leads to changes/learning/improvement that ultimately lead to success then it's a positive.

The cost of failure?

You are lucky because you are extremely talented. The cost of "failure" for (many of you) you is actually much less than for others because your backup plan is amazingly good.

Take the shot. If it doesn't work out, you'll try something else and, you'll probably succeed… or just go get that pretty darn good job you would have gotten anyway.

Think bigger, think broader

You are fortunate.

You are smart, talented, and hard-working.

You are in an amazing environment at CMU.

(think about the people and projects going on around you)

How can you maximize that opportunity while you are here?

The mechanisms are in place, if they aren't, we'll help you create them:

Course projects

Research

Independent study

Entrepreneurship

The biggest sign you are in the "real-world" isn't when you are paying your own bills, showing up to work on time, or ensuring your code passes regressions... it is asking your own questions and making your own decisions.

And there's a lot more to decide on at CMU than classes.

Or in other words*…
there are "grades" you can get at CMU
that are much higher than A's.

* More precisely, Dave Eckhardt's words

Kayvon's

Final

Claim

It is far more difficult (and creative) to deliver on a "once in a few years" final project in 15-418/618 than it is to take an extra class.

Ditto for pursuits like senior theses, independent study, starting a company…

The world rewards initiative.

The world rewards risk takers.

(Force yourself to get comfortable with risk taking and the possibility of occasional failure while at CMU.)

Many people are really smart.

Many people can work really hard.

Far fewer people can pick the right problems to work on and develop the confidence and creativity to lead.

Thanks for being a great class!

Good luck on projects. Expectations are high.

See you a week from Friday!