

CCU
2022

HOW I TEACH MODERN C++ ONE PIXEL AT A TIME

MIKE SHAH

Abstract: The C++ language has a reputation of being a very powerful, fast, and expressive, but hard to learn language for beginners. My opinion though, is that we can always be improving our learning materials for beginners--that's why I teach the C++ programming language one pixel at a time. What does a pixel have to do with the language? It's not so much the graphics (though that is a motivating domain to use the C++ language), but in the ability for beginning programmers to visualize and see what they are programming. In this talk I reflect on how to not just teach C++, but how to motivate, inspire, and get beginners to build cool graphical projects while learning the Modern C++ language.

Abstract: The C++ language has a reputation of being a very powerful, fast, and expressive, but hard to learn language for beginners. My opinion though,

is that
beginners
time.
graph
in the
prog
motiv
learn

If I'm successful, both you and I will have a system for course construction that's effective for our students.

(At the very least, you'll have some neat assignment ideas to build off of)

at a
ne
out
re
to
e

Abstract: The C++ language has a reputation of being a very powerful, fast, and expressive, but hard to learn language for beginners. My opinion though,

is that
beginners
time.
graph
in the
prog
motiv
learn

For folks attending online, please feel free to use the Q&A, I'll answer questions at the end.

For folks in the room, ask questions at any time, and I'll repeat the question.

at a
ne
out
re
to
e

Who Am I?

by Mike Shah

- **Assistant Teaching Professor** at Northeastern University in Boston, Massachusetts.
 - I teach courses in computer systems, computer graphics, and game engine development.
 - My **research** in program analysis is related to **performance** building static/dynamic analysis and software visualization tools.
- I do **consulting** and technical training on modern C++, Concurrency, OpenGL, and Vulkan projects
 - (**Usually graphics or games related**)
- I like teaching, guitar, running, weight training, and anything in computer science under the domain of **computer graphics**, visualization, concurrency, and parallelism.
- Contact information and more on: www.mshah.io



BCU
2022

HOW I TEACH MODERN C++ ONE PIXEL AT A TIME

April 7, 2022 | 14:00 - 15:30

Mike Shah, Ph.D. | [@MichaelShah](https://twitter.com/MichaelShah)

www.youtube.com/c/MikeShah

www.mshah.io

3CCW
2022

HOW I TEACH MODERN C++ ONE PIXEL AT A TIME

April 7, 2022 | 14:00 - 15:30

Mike Shah, Ph.D. | [@MichaelShah](https://twitter.com/MichaelShah)

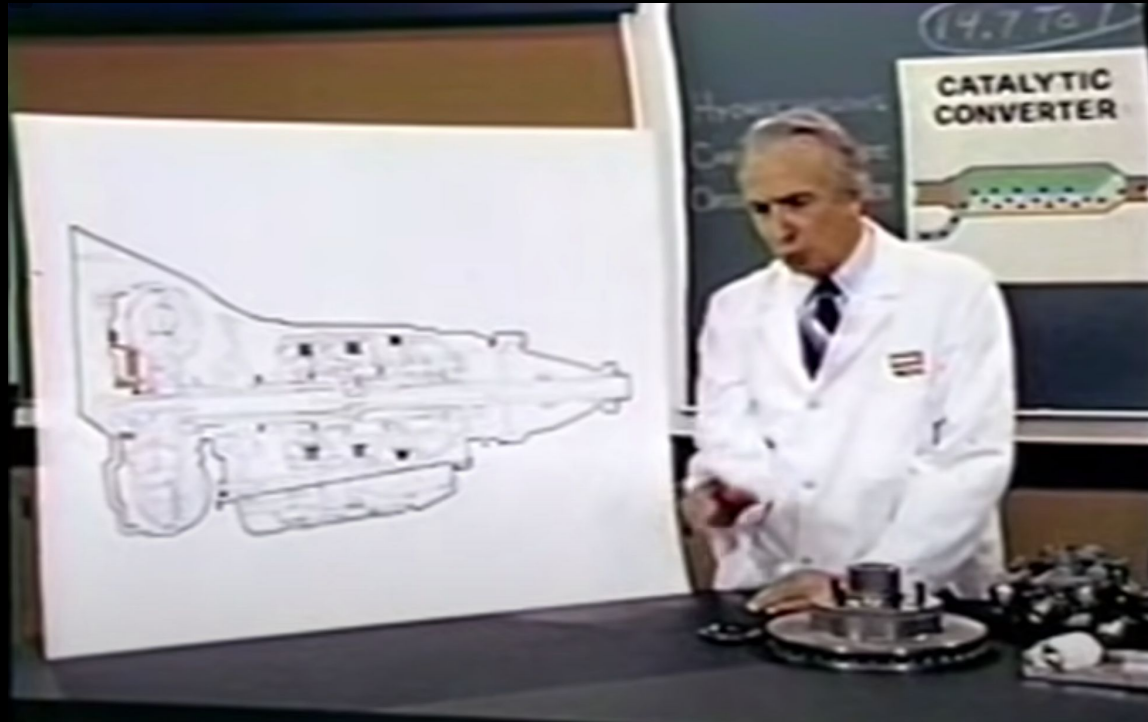
www.youtube.com/c/MikeShah

www.mshah.io

This talk is about
teaching C++ in a
semester long
University Course

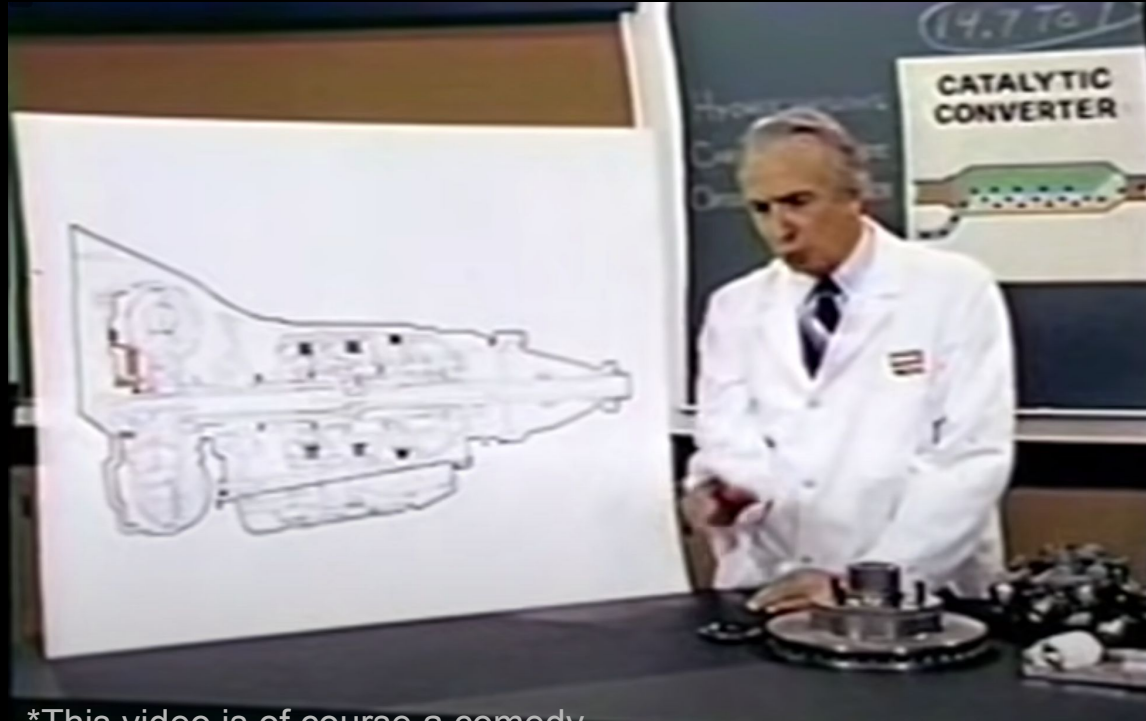
Before we begin...a video

"Turbo Encabulator" the Original (1/3)



"Turbo Encabulator" the Original (2/3)

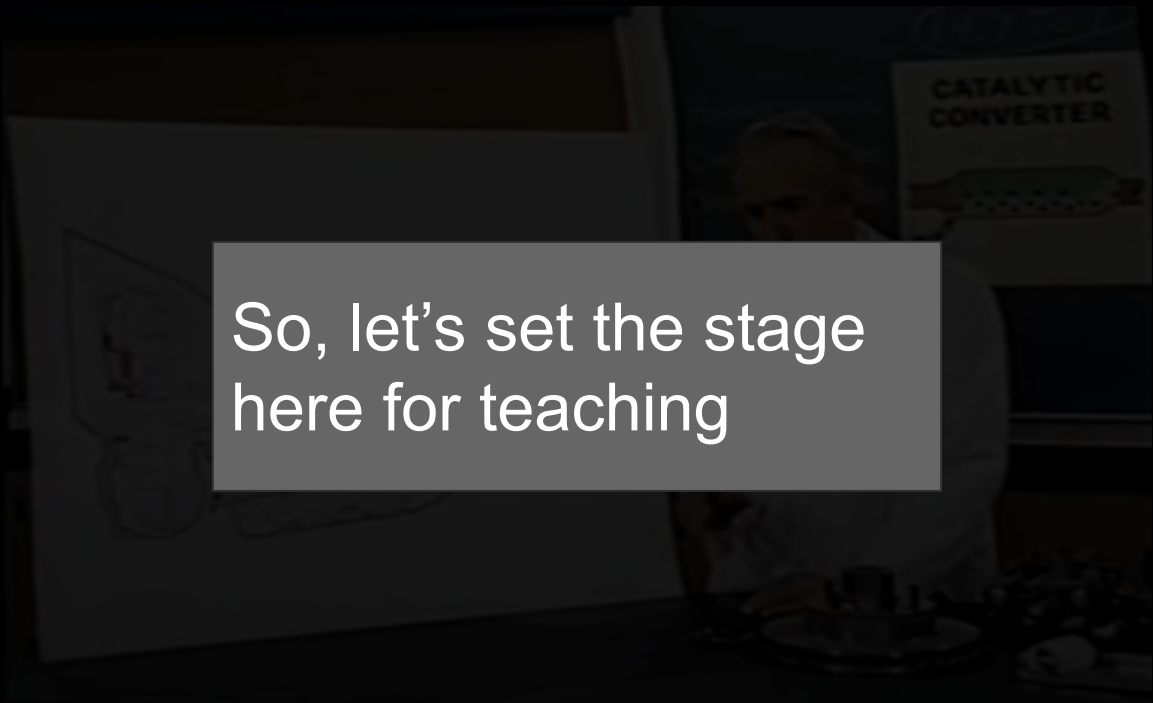
This should not be your style of teaching...almost anything*



*This video is of course a comedy

"Turbo Encabulator" the Original (3/3)

This should not be your style of teaching...almost anything*



So, let's set the stage
here for teaching

*This video is of course a comedy

Some Context - Who is being taught?

My Pre-Course Survey to understand my students
(Pro Tip: Always survey your students)



My Programming in C++ Course

- This is a course that I teach to students usually in their 2nd year at university
 - It is a semester long course
 - Caveat: This particular iteration is crammed into a 7 week summer session moving twice as fast
- Most students have taken at least 2 semesters of programming prior to taking this course
 - Usually with experience in Python, Java or a functional language like Scheme
 - Most students have not seen manual memory management (malloc/free or new/delete)
 - A good subset of students are part of data science or game programming based degrees

CS 3520 Programming in C++

"Programming in C++, one pixel at a time" -- Your Instructor -- Your Instructor



Instructor

- Instructor: Mike Shah
- E-mail: mikeshah@ Northeastern.edu (Read [How to send an e-mail](#))
- Office: Virtual Nighthinge-432A
- Student Hours: In Location TBD
 - Review course material with me and ask questions
 - Date/Time: By appointment
 - Sign up for a 15-30 minute appointments arranged by e-mail on [Google Hangouts here](#)
- Forum: [Plazza Forum Board](#)
- Microsoft Teams Link for Live Video: [Teams Link](#)



General Purpose, Multi-paradigm



Powerful, Modern



...and fun language

Schedule/Road Map

The following is our tentative syllabus for the course, some changes should be expected throughout the semester. I will announce in class lecture, piazza, or through e-mail any major changes.

- To get all of the assignments/activities for the course, you must first click the following link: [Course Monospace](#) Do not do a "git pull" until class starts (Occasionally I make changes/spelling corrections)

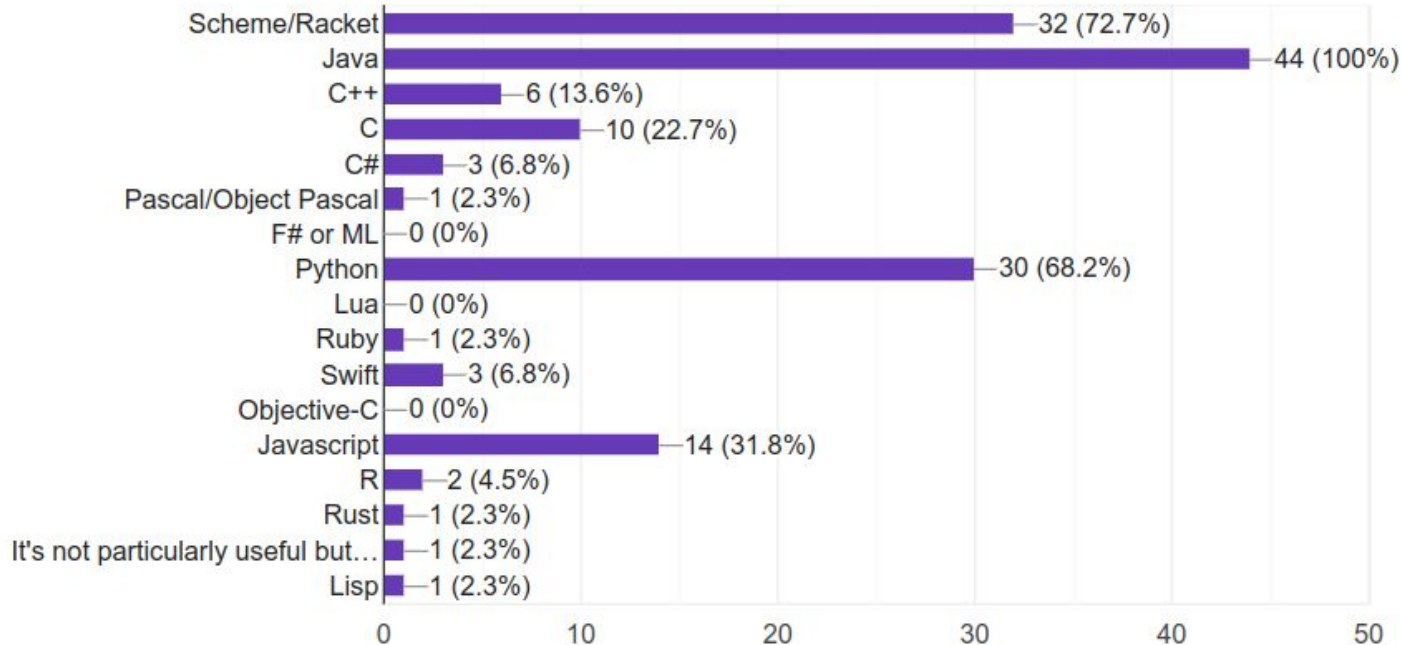
Week	Date	Lecture and Readings	Assignments	Note(s)
1	Monday - May 10, 2021	Module 1 - C++ is not the C Language -- Course Introduction Video	A1 Released -- Guessing Game (Due May 14 Anywhere on Earth)	Welcome back to class!
1	Tuesday - May 11, 2021	Module 2 - C++ Batteries Included - STL and GDB Debugging with the STL Video	--	--
1	Wednesday - May 12, 2021	Module 3 - Functions 1: Making our code more modular GDB and Functions Video	--	--
1	Thursday - May 13, 2021	Module 4 - Input and Output Video	A2 Released -- Debugger (Due May 20 Anywhere on Earth)	--

My Course - Pre-Course Survey Data (1/6)

What programming languages do you know?

 Copy

44 responses

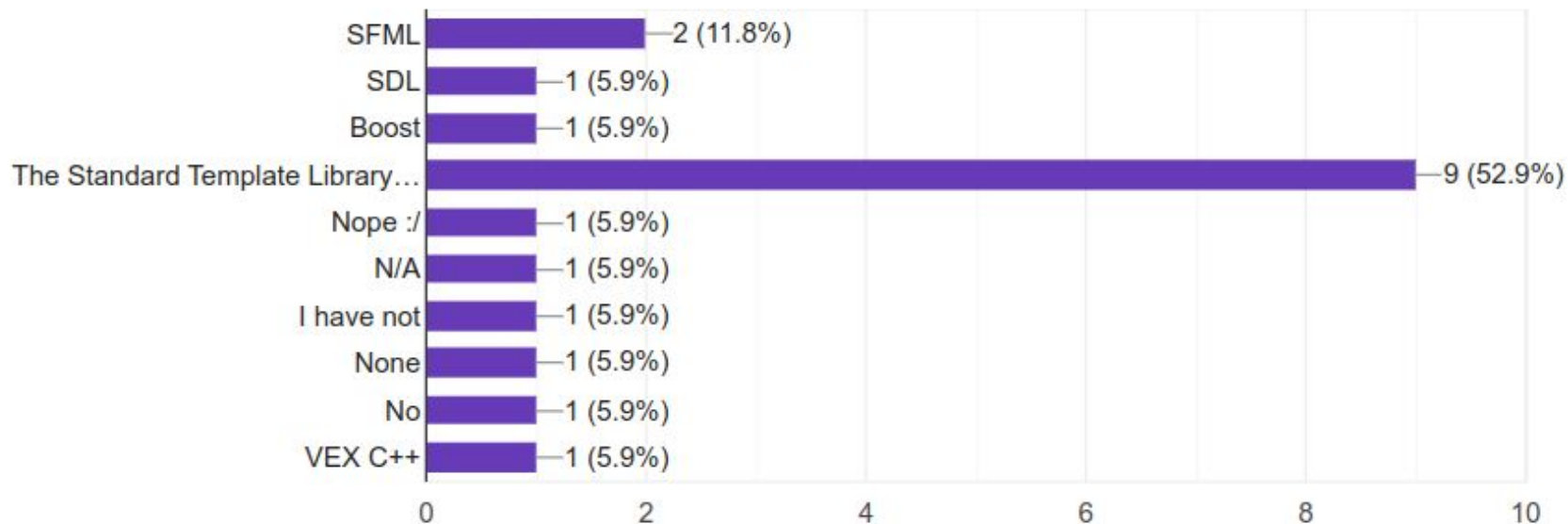


My Course - Pre-Course Survey Data (2/6)

Have you used any of the following libraries or APIs?



17 responses

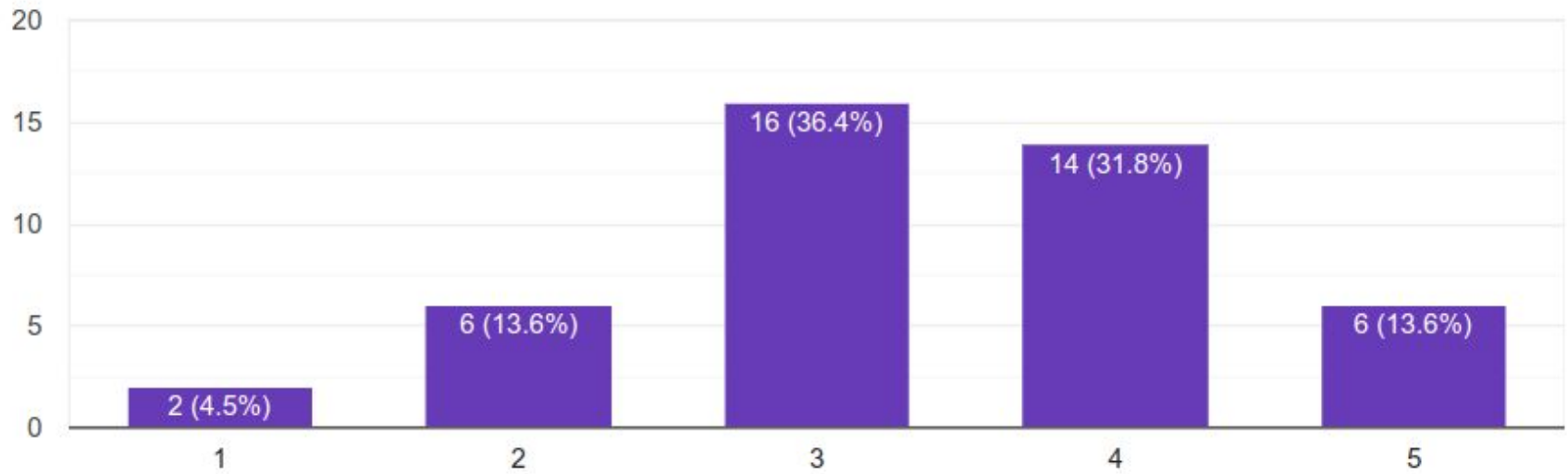


My Course - Pre-Course Survey Data (3/6)

How comfortable are you with Object Oriented Programming?



44 responses



Not very comfortable

Average

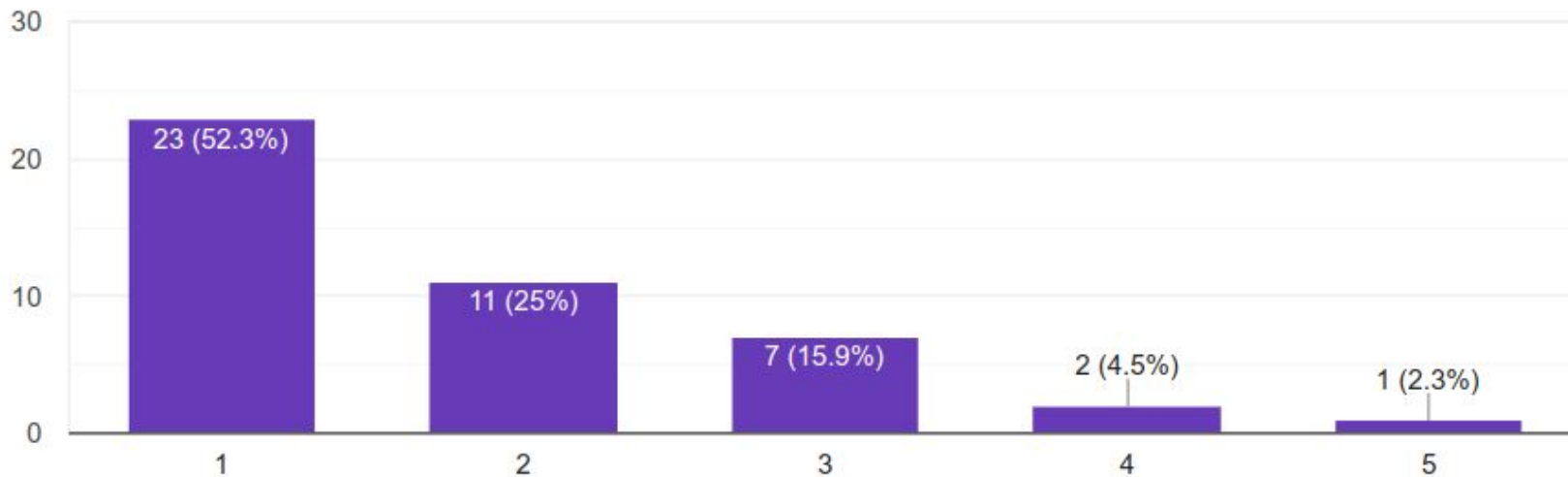
Very comfortable!

My Course - Pre-Course Survey Data (4/6)

How comfortable are you with C++?



44 responses



Not very comfortable

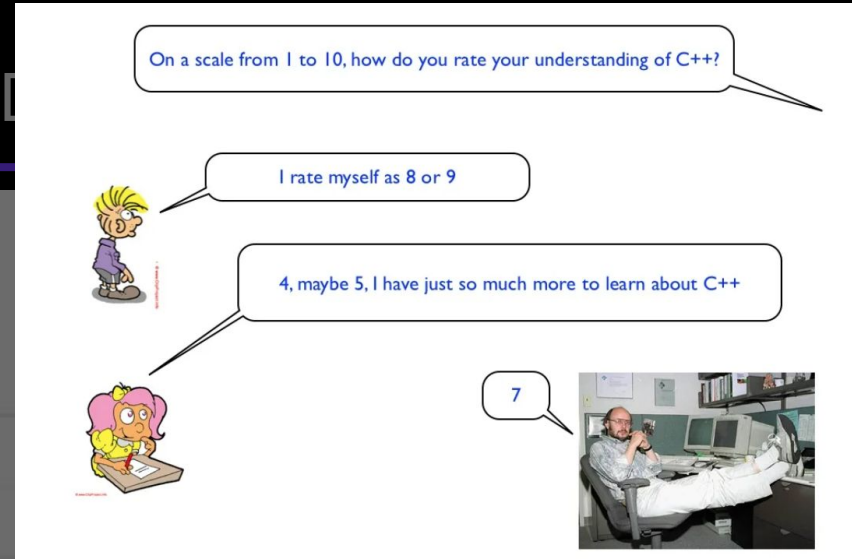
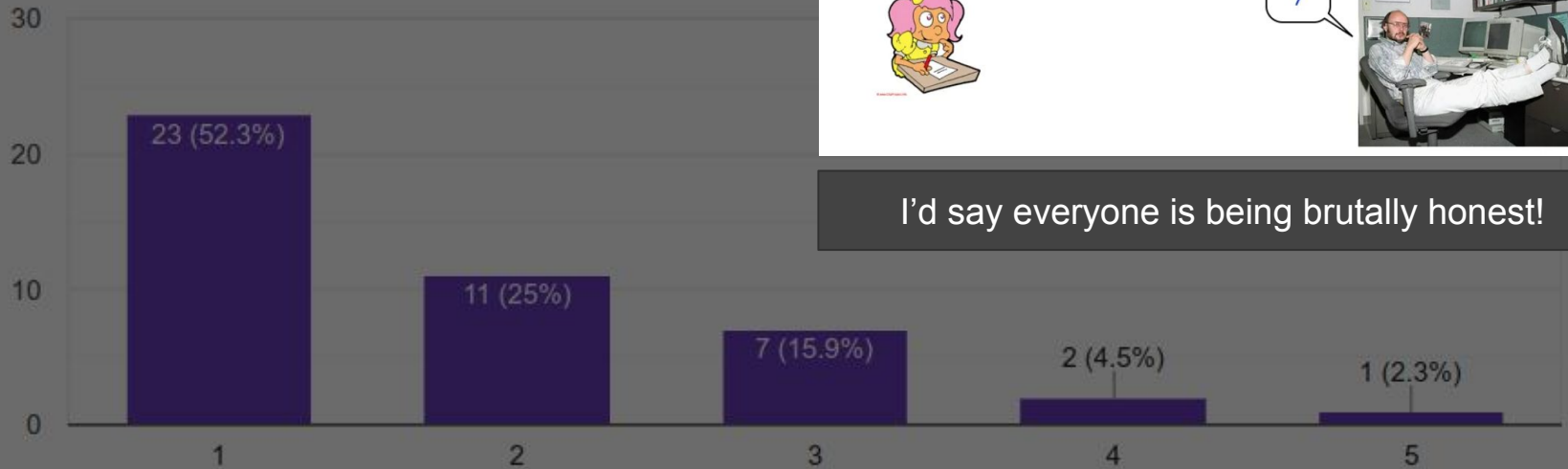
Average

Very comfortable!

My Course - Pre-Course Survey

How comfortable are you with C++?

44 responses



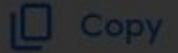
I'd say everyone is being brutally honest!

My Course - Pre-Course Survey Data (6/6)

How comfortable are you with C++?

44 responses

So here's my audience. Folks who know some programming, but haven't been exposed to C++



30

20

10

0

23 (52.3%)

11 (25%)

7 (15.9%)

2 (4.5%)

1 (2.3%)

1

2

3

4

5

Not very comfortable

Average

Very comfortable!

SCC
2022

HOW I TEACH MODERN C++ ONE PIXEL AT A TIME

April 7, 2022 | 14:00 - 15:30
Mike Shah, Ph.D. | [@MichaelShah](https://twitter.com/MichaelShah)
www.youtube.com/c/MikeShah
www.mshah.io

Now, I know how I
teach. So I hope that
will be interesting, but...

SCC
2022

HOW I TEACH MODERN C++ ONE PIXEL AT A TIME

April 7, 2022 | 14:00 - 15:30
Mike Shah, Ph.D. | [@MichaelShah](#)
www.youtube.com/c/MikeShah
www.mshah.io

Now, I know how I
teach. So I hope that
will be interesting, but...
I want **you** to think
about how you teach

3CC4
2022

HOW [YOU MIGHT] TEACH MODERN C++ [NEXT] TIME

April 7, 2022 | 14:00 - 15:30
Mike Shah, Ph.D. | [@MichaelShah](https://twitter.com/MichaelShah)
www.youtube.com/c/MikeShah
www.mshah.io

Now, I know how I
teach. So I hope that
will be interesting, but...
I want **you** to think
about how **you** teach

3CCW
2022

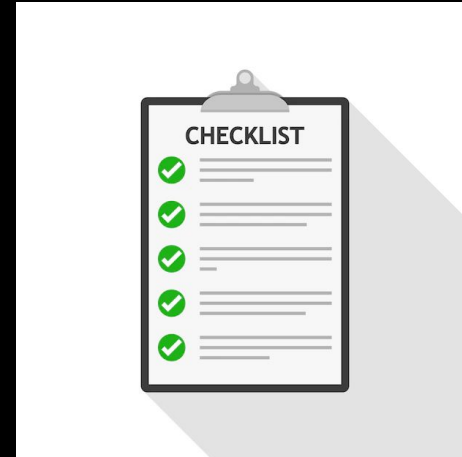
HOW [YOU MIGHT] TEACH MODERN C++ [NEXT] TIME

April 7, 2022 | 14:00 - 15:30
Mike Shah, Ph.D. | [@MichaelShah](https://twitter.com/MichaelShah)
www.youtube.com/c/MikeShah
www.mshah.io

The talk will be more applicable if you reflect on your teaching--whether a teacher, professor, C++ trainer, manager, youtuber, etc.

Some Homework for the Audience

(Whether present, virtual, or in the future!)



Homework (Yes...I am a professor)

(Take a screenshot or take a few notes -- we'll revisit this list today)

- Will you (or are you currently) teaching with a common theme?
- Are you willing to sacrifice 20% of your course so students have a chance to understand 99.9% of the remaining 80% of your course?
- Do you discuss debugging early in the course?
- Will you teach C++ without telling them the language is scary, and they must also know the 'C' language?
- Your other notes:

Homework (Yes...I am a professor)

(Take a screenshot or take a few notes -- we'll revisit this list today)

- Will you (or any other professor) have a common theme?
- Are you willing to give students a chance to understand how students have a of your course?
- Do you discuss the importance of your course?
- Will you teach a course that is scary, and they must also take it?
- Your other notes

Let's begin!

Why do we need to think about how we teach
C++?

(In my opinion) C++ Greatest **Strength** for a learner (1/2)

- The #1 strength of C++ is that as a language it scales with you as you learn more computer science.
 - You can use it as a simple procedural language to start
 - You can use it as you learn different paradigms (i.e. functional, object-oriented, etc.) or different design patterns
 - You can leverage the language to work with the system as you learn about computer architecture
 - You can use C++ with large and small software
 - You learn over time the pros/cons of C++'s design which helps you understand other language design decisions
 - ...and many more skills you can scale into (parallel programming, networking, etc.)

C++ reference
C++98, C++03, C++11, C++14, C++17, C++20, C++23 | Compiler support C++11, C++14, C++17, C++20, C++23

Free-standing implementations	Diagnostics library basic_stacktrace (C++23)	Iterators library Ranges library (C++20)
Language	Memory management library unique_ptr (C++11) shared_ptr (C++11)	Algorithms library Constrained algorithms (C++20)
Basic concepts	General utilities library Function objects — hash (C++11) Utility functions pair — tuple (C++11) optional (C++17) — any (C++17) variant (C++17)	Numerics library Common math functions Mathematical special functions (C++17) Mathematical constants (C++20) Numeric algorithms Pseudo-random number generation Floating-point environment (C++11) complex — valarray
Keywords	Strings library basic_string basic_string_view (C++17) Null-terminated strings: byte — multibyte — wide	Date and time library Localizations library
Preprocessor	Containers library array (C++11) — vector — deque list — forward_list (C++11) map — multimap set — multiset unordered_map (C++11) unordered_multimap (C++11) unordered_set (C++11) unordered_multiset (C++11) stack — queue — priority_queue span (C++20)	Input/output library Stream-based I/O Synchronized output (C++20) I/O manipulators
Expressions	Concepts library (C++20)	Filesystem library (C++17)
Declaration	Metaprogramming library (C++11) Type traits — ratio Integer sequence (C++14)	Regular expressions library (C++11) basic_regex — algorithms
Functions	Technical specifications	Concurrency support library (C++11) atomic — atomic_flag atomic_ref (C++20) thread — jthread (C++20) mutex condition variable future — promise
Statements	Standard library extensions (library fundamentals TS) resource_adaptor — invocation_type	
Classes	Standard library extensions v2 (library fundamentals TS v2) propagate_const — ostream_jolner — randint observer_ptr — detection_idiom	
Overloading	Standard library extensions v3 (library fundamentals TS v3) scope_exit — scope_fall — scope_success — unique_resource	
Templates	Concurrency library extensions (concurrency TS) — Transactional Memory (TM TS)	
Exceptions	Reflection (reflection TS)	

External Links — Non-ANSI/ISO Libraries — Index — std Symbol Index

(In my opinion) C++ Greatest Strength for a learner (2/2)

- The #1 strength of C++ is that as a language it scales with you as you learn more computer science.
 - You can use it as a simple procedural language to start
 - You can use it as you learn different paradigms (i.e. functional, object-oriented, etc.) or different design patterns
 - You can leverage the language to work with the system as you learn about computer architecture
 - You can use C++ with large and small software
 - You learn over time the pros/cons of C++'s design which helps you understand other language design decisions
 - ...and many more skills you can scale in (parallel programming, network...

Look at all of this cool stuff to learn!

C++ reference	
C++98, C++03, C++11, C++14, C++17, C++20, C++23 Compiler support C++11, C++14, C++17, C++20, C++23	
Freestanding implementations	Diagnostics library basic_stacktrace (C++23)
Language	Memory management library unique_ptr (C++11) shared_ptr (C++11)
Basic concepts	General utilities library Function objects – hash (C++11) Utility functions pair – tuple (C++11) optional (C++17) – any (C++17) variant (C++17) String conversions (C++17) Formatting (C++20) Bit manipulation (C++20)
Keywords	Strings library basic_string basic_string_view (C++17) Null-terminated strings: byte – multibyte – wide
Preprocessor	Containers library array (C++11) – vector – deque list – forward_list (C++11) map – multimap set – multiset unordered_map (C++11) unordered_multimap (C++11) unordered_set (C++11) unordered_multiset (C++11) stack – queue – priority_queue span (C++20)
Expressions	Iterators library
Declaration	Ranges library (C++20)
Initialization	Algorithms library Constrained algorithms (C++20)
Functions	Numerics library Common math functions Mathematical special functions (C++17) Mathematical constants (C++20) Numeric algorithms Pseudo-random number generation Floating-point environment (C++11) complex – valarray
Statements	Date and time library
Classes	Localizations library
Overloading	Input/output library Stream-based I/O Synchronized output (C++20) I/O manipulators
Templates	Filesystem library (C++17)
Exceptions	Regular expressions library (C++11) basic_regex – algorithms
Headers	Concurrency support library (C++11) atomic – atomic_flag atomic_ref (C++20) thread – jthread (C++20) mutex condition_variable future – promise
Named requirements	
Feature test macros (C++20)	
Language support library	
Source code information (C++20)	
Type support	
Program utilities	
Coroutine support (C++20)	
Three-way comparison (C++20)	
numeric_limits – type_info	
initializer_list (C++11)	
Concepts library (C++20)	
Metaprogramming library (C++11)	
Type traits – ratio integer_sequence (C++14)	
Technical specifications	
Standard library extensions (library fundamentals TS) resource_adaptor – invocation_type	
Standard library extensions v2 (library fundamentals TS v2) propagate_const – ostream_joiner – randint observer_ptr – detection_idiom	
Standard library extensions v3 (library fundamentals TS v3) scope_exit – scope_fail – scope_success – unique_resource	
Concurrency library extensions (concurrency TS) – Transactional Memory (TM TS)	
Reflection (reflection TS)	
External Links – Non-ANSI/ISO Libraries – Index – std Symbol Index	

(In my opinion) C++ Greatest **Weakness** for a learner

- “Hmm, the language is so big--I’m overwhelmed! Where do I begin?”

C++ reference
C++98, C++03, C++11, C++14, C++17, C++20, C++23 | Compiler support C++11, C++14, C++17, C++20, C++23

Freestanding implementations	Diagnostics library basic_stacktrace (C++23)	Iterators library
Language	Memory management library unique_ptr (C++11) shared_ptr (C++11)	Ranges library (C++20)
Keywords	General utilities library Function objects – hash (C++11) Utility functions pair – tuple (C++11) optional (C++17) – any (C++17) variant (C++17) String conversions (C++17) Formatting (C++20) Bit manipulation (C++20)	Algorithms library Constrained algorithms (C++20)
Preprocessor	Strings library basic_string basic_string_view (C++17) Null-terminated strings: byte – multibyte – wide	Numerics library Common math functions Mathematical special functions (C++17) Mathematical constants (C++20) Numeric algorithms Pseudo-random number generation Floating-point environment (C++11) complex – valarray
Expressions	Containers library array (C++11) – vector – deque list – forward_list (C++11) map – multimap set – multiset unordered_map (C++11) unordered_multimap (C++11) unordered_set (C++11) unordered_multiset (C++11) stack – queue – priority_queue span (C++20)	Date and time library
Declaration	Concepts library (C++20)	Localizations library
Functions	Metaprogramming library (C++11) Type traits – ratio integer_sequence (C++14)	Input/output library Stream-based I/O Synchronized output (C++20) I/O manipulators
Statements	Technical specifications	Filesystem library (C++17)
Classes	Standard library extensions (library fundamentals TS) resource_adaptor – invocation_type	Regular expressions library (C++11) basic_regex – algorithms
Overloading	Standard library extensions v2 (library fundamentals TS v2) propagate_const – ostream_joiner – randint observer_ptr – detection_idiom	Concurrency support library (C++11) atomic – atomic_flag atomic_ref (C++20) thread – jthread (C++20) mutex condition_variable future – promise
Templates	Standard library extensions v3 (library fundamentals TS v3) scope_exit – scope_fail – scope_success – unique_resource	
Exceptions	Concurrency library extensions (concurrency TS) – Transactional Memory (TM TS)	
Headers	Reflection (reflection TS)	
Named requirements		
Feature test macros (C++20)		
Language support library		
Source code information (C++20)		
Type support		
Program utilities		
Coroutine support (C++20)		
Three-way comparison (C++20)		
numeric_limits – type_info		
initializer_list (C++11)		

External Links – Non-ANSI/ISO Libraries – Index – std Symbol Index

C++ Pre-Course Survey

Tell me a little bit about who you are, and what you want to get out of this course!

What is your primary motivation for taking this course? (i.e. What would you feel sad about if you did not learn it? What are you most excited about learning? I do shift contents based off of your feedback)

I do not know enough about c++ to know what I am most excited to learn. The class simply seemed interesting and I enjoyed the short experience I had with c++ years ago.

[Three-way comparison \(C++20\)](#)

[map - multimap](#)

[cat - multiset](#)

[Concurrency support library \(C++11\)](#)

[propagate_const](#) — [ostream_joiner](#) — [randint](#)
[observer_ptr](#) — [detection idiom](#)

Standard library extensions v3 ([library fundamentals TS v3](#))

[scope_exit](#) — [scope_fail](#) — [scope_success](#) — [unique_resource](#)

Concurrency library extensions ([concurrency TS](#)) — **Transactional Memory** ([TM TS](#))

Reflection ([reflection TS](#))

[External Links](#) — [Non-ANSI/ISO Libraries](#) — [Index](#) — [std Symbol Index](#)

C++ Pre-Course Survey

Tell me a little bit about who you are, and what you want to get out of this course!

What is your primary motivation for taking this course? (i.e. What would you feel sad about if you did not learn it? What are you most excited about learning? I do shift contents based off of your feedback)

I do not know enough about c++ to know what I am most excited to learn. The class simply seemed interesting and I enjoyed the short experience I had with c++ years ago.

[propagate_const](#) — [ostream_joiner](#) — [randint](#)
[observer_ptr](#) — [detection idiom](#)
Standard library extensions v3 ([library fundamentals TS v3](#))
[scope_exit](#) — [scope_fail](#) — [scope_success](#) — [unique_resource](#)
Concurrency library extensions ([concurrency TS](#)) — **Transactional Memory** ([TM TS](#))
Reflection ([reflection TS](#))

[External Links](#) — [Non-ANSI/ISO Libraries](#) — [Index](#) — [std Symbol Index](#)

(Bringing back the original question (1/2))

- Why do we need to think about how we teach C++?
^ This is our job as educators (whether a professor, teacher, trainer, colleague giving a tech talk, etc.) to guide our learners on a journey for learning how to use a programming language to solve problems.

C++ reference	
C++98, C++03, C++11, C++14, C++17, C++20, C++23 Compiler support C++11, C++14, C++17, C++20, C++23	
Free-standing implementations	Diagnostics library basic_stacktrace (C++23)
Language	Memory management library unique_ptr (C++11) shared_ptr (C++11)
Basic concepts	General utilities library Function objects – hash (C++11) Utility functions pair – tuple (C++11) optional (C++17) – any (C++17) variant (C++17) String conversions (C++17) Formatting (C++20) Bit manipulation (C++20)
Keywords	Strings library basic_string basic_string_view (C++17) Null-terminated strings: byte – multibyte – wide
Preprocessor	Containers library array (C++11) – vector – deque list – forward_list (C++11) map – multimap set – multiset unordered_map (C++11) unordered_multimap (C++11) unordered_set (C++11) unordered_multiset (C++11) stack – queue – priority_queue span (C++20)
Expressions	Iterators library
Declaration	Ranges library (C++20)
Initialization	Algorithms library Constrained algorithms (C++20)
Functions	Numerics library Common math functions Mathematical special functions (C++17) Mathematical constants (C++20) Numeric algorithms Pseudo-random number generation Floating-point environment (C++11) complex – valarray
Statements	Date and time library
Classes	Localizations library
Overloading	Input/output library Stream-based I/O Synchronized output (C++20) I/O manipulators
Templates	Filesystem library (C++17)
Exceptions	Regular expressions library (C++11) basic_regex – algorithms
Headers	Concurrency support library (C++11) atomic – atomic_flag atomic_ref (C++20) thread – jthread (C++20) mutex condition_variable future – promise
Named requirements	
Feature test macros (C++20)	
Language support library	
Source code information (C++20)	
Type support	
Program utilities	
Coroutine support (C++20)	
Three-way comparison (C++20)	
numeric_limits – type_info	
initializer_list (C++11)	
Concepts library (C++20)	
Metaprogramming library (C++11)	
Type traits – ratio	
integer_sequence (C++14)	
Technical specifications	
Standard library extensions (library fundamentals TS) resource_adaptor – invocation_type	
Standard library extensions v2 (library fundamentals TS v2) propagate_const – ostream_joiner – randint observer_ptr – detection idiom	
Standard library extensions v3 (library fundamentals TS v3) scope_exit – scope_fail – scope_success – unique_resource	
Concurrency library extensions (concurrency TS) – Transactional Memory (TM TS)	
Reflection (reflection TS)	
External Links – Non-ANSI/ISO Libraries – Index – std Symbol Index	

(Bringing back the original question (2/2))

- Why do we need to think about a good course, seminar, tech talk, etc. should guide the learner. Our job, is to provide a 'map' that can be easily read and followed. to use a programming language to solve problems.



External Links – Non-ANSI/ISO Libraries – Index – std Symbol Index

<https://www.fluentcpp.com/getthemap/>
(Just for fun, a map of C++ STL Algorithms)

[Course Construction]

Where do we start?

What will students build?

What skills will students learn?

Here's where I start (1/3)

- First, I need a strong course mantra or theme
 - This gives you something to center a whole course around
 - Whenever I have a decision to make, I ask myself if it fits the course theme.
 - This constraint--helps me, and it helps keep students in the same domain
 - (Some folks will start with a series of learning objectives, but I think the theme comes first, then the learning objectives)
- So as you know, my theme is....

Here's where I start (2/3)

- First, I need a strong course mantra or theme
 - This gives you something to center a whole course around
 - Whenever I have a decision to make, I ask myself if it fits the course theme.
 - This constraint--helps me, and it helps keep students in the same domain
 - (Some folks will start with a series of learning objectives, but I think the theme comes first, then the learning objectives)
- So as you know, my theme is...

CS 3520 Programming in C++

"Programming in C++, one pixel at a time" -- Your Instructor

Here's where I start (3/3)

- First, I need a strong course mantra or theme
 - This gives you something to center a whole course around
 - Whenever I have a decision to make, I ask myself if it fits the course theme.
 - This constraint--helps me, and it helps keep students in the same domain
 - (Some folks will start with a series of learning objectives, but I think the theme comes first, then the learning objectives)
- So as you know, my theme is...

CS 3520 Programming in C++

"Programming in C++, one pixel at a time" -- Your Instructor

Backstory

- Now, I had this idea to do a course with this theme for some time.
 - 1.) Could possibly be cool to students
 - (I'll get to why 'cool' and 'inspiration' are important to students)
 - 2.) Could help students better be able to *think* and *see* a real world problem they are solving
 - (The assignments could involve an algorithmic part, and have them solve real problems)
- A single article inspired this idea of focusing on 'pixel effects'



Here's the article! (1/2)

https://fabiensanglard.net/doom_fire_psx/

- The original inspiration for an early assignment and the theme of this course
- Recreating the DOOM fire effect from the opening scene.
- And I kept seeing this effect implemented over and over again on twitter!

FABIEN SANGLARD'S WEBSITE

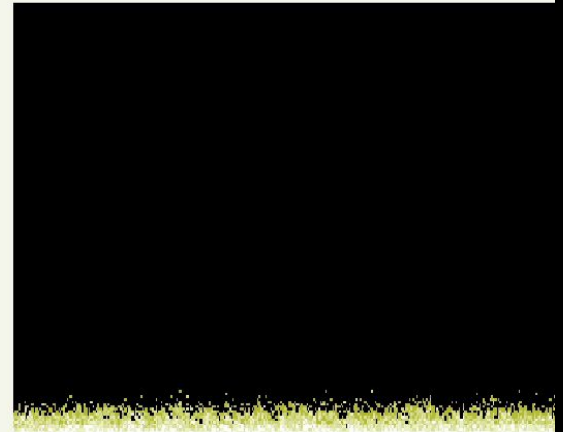
ABOUT EMAIL RSS DONATE

Dec 28, 2018

HOW DOOM FIRE WAS DONE

The [Game Engine Black Book: DOOM](#) features a whole chapter about DOOM console ports and the challenges they encountered. The utter failure of the 3DO, the difficulties of the Saturn due to its affine texture mapping, and the amazing "reverse-engineering-from-scratch" by Randy Linden on Super Nintendo all have rich stories to tell.

Once heading towards disaster^[1], the Playstation 1 (PSX) devteam managed to rectify course to produce a critically and commercially acclaimed conversion. [Final DOOM](#) was the most faithful port when compared to the PC version. The alpha blended colored sectors not only improved visual quality, they also made gameplay better by indicating the required key color. Sound was also improved via reverberation effects taking advantage of the PSX's Audio Processing Unit.



Here's the article! (2/2)

<https://f>

- The original inspiration for an early assignment and the theme of this course
- Recreating the DOOM fire effect from the opening scene.
- And I kept seeing this effect implemented over and over again on twitter!
 - (And added my own)

FABIEN SANGLARD'S
ABOUT EMAIL RSS

Dec 28, 2018

HOW DOOM FIRE WAS DONE

The Game Engine Black Book: DOOM features a whole chapter about DOOM console ports and the challenges they encountered. The utter failure of the 3DO, the difficulties of the Saturn due to its affine texture mapping, and the amazing "reverse-engineering-from-scratch" by Randy Linden on Super Nintendo all have rich stories to tell.

Once heading towards disaster^[1], the Playstation (PSX) devteam managed to rectify course to produce a critically and commercially acclaimed conversion. Final DOOM was the most faithful port when compared to the PC version. The alpha blended colored sectors not only improved visual quality, they also made gameplay better by indicating the required key color. Sound was also improved via reverberation effects, taking advantage of the PSX's Audio Processing Unit.

Mike Shah, Ph.D. @MichaelShah · Apr 29, 2021
Replying to @MichaelShah
The above effect is based off of the Doom effect as documented above. Using circles for every pixel I found more appealing, but rendering circles is more expensive.

1

Mike Shah, Ph.D. @MichaelShah · Apr 29, 2021
Here's the same effect with one pixel sized squares. See if you can tell a difference. Probably not by much.



1

Mike Shah, Ph.D. @MichaelShah · Apr 29, 2021
Here's the effect without the 'sway' back and forth of the fire. Looks fine for 'fireplaces' or relatively static fire, but probably less convincing..



1

Mike Shah, Ph.D. @MichaelShah · Apr 29, 2021
Here's the same fire as above. The difference here is the smaller color palette. Also, likely less convincing of a fire, but a cheaper effect.



1

Mike Shah, Ph.D. @MichaelShah · Apr 29, 2021
And here's a less down sampled, higher framerate gif of the original fire in the first post if you made it this far as a reward :) This time you can see the actual circles which look nice for the smoke, perhaps 'gray' colors should be rendered as circles, everything else rects.



Let there be fire!

https://fabiansanglard.net/doom_fire_psx/

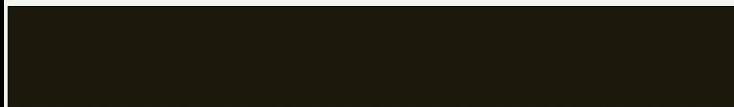
- Now from looking at the actual effect, it is really only a few lines of code.
 - A small enough problem for students to see the core idea in JavaScript
 - But I could see how to implement this, students would have to understand:
 - 1D-arrays and/or 2D-arrays
 - iteration
 - function calls
 - and a color palette (A 1D-array, but storing color data in a lookup table)

CORE IDEA

At its core, the fire effect relies on a simple array (a.k.a framebuffer) covering the whole screen. Each value in the framebuffer is within $[0, 36]$. These values are associated with a palette where colors range from white to black, using yellow, orange, and red along the way. The idea is to model the fire particle's temperature as it elevates and cools down.



The framebuffer is initialized full black (with zeros) with a single line of white pixels at the bottom (36) which will be the "source" of the fire.



The array is drawn on screen such that the top-left pixel is at array index zero and the bottom-right pixel is at index $\text{FIRE_HEIGHT} * \text{FIRE_WIDTH} - 1$. In other words, screen-space origin is at the top-left corner.

For each pixel, the "heat" is propagated to the pixel directly above.

```
function doFire() {
  for(x = 0 ; x < FIRE_WIDTH; x++) {
    for (y = 1; y < FIRE_HEIGHT ; y++) {
      spreadFire(y * FIRE_WIDTH + x);
    }
  }
}

function spreadFire(from) {
  var to = from - FIRE_WIDTH;
  firePixels[to] = firePixels[from] - 1;
}
```

Notice that the bottom screen line is never updated. This line, populated with zeros, is the constant "generator" of fire. This simple version with linear cooling (-1) yields a boring uniform output.



CS 3520 Programming in C++

"Programming in C++, one pixel at a time" -- Your Instructor

- So I have a theme for my course-- graphical programs manipulating pixels
 - Students will write graphical programs to help them 'see' as they learn
 - Algorithmically interesting problems could be solved.
 - Theme feels inclusive (interesting to a wide variety of students)

CS 3520 Programming in C++

"Programming in C++, one pixel at a time" -- Your Instructor

- So I have a theme for my course-- graphical programs manipulating pixels
 - Students will write graphical programs to help them 'see' as they learn
 - Algorithmically interesting problems could be solved.
 - Theme feels inclusive (potentially interesting to a wide variety of students)

Let's do a quick check on the 'homework' I gave everyone (myself included)

CS 3520 Programming in C++

"Programming in C++, one pixel at a time" -- Your Instructor

- So I have a theme for my course-- graphical programs manipulating pixels
 - Students will write graphical programs to help them 'see' as they learn
 - Algorithmically interesting problems could be solved.
 - Theme feels inclusive (potentially interesting to a wide variety of students)
- Will you (or are you currently) teaching with a common theme?
 - Are you willing to sacrifice 20% of your course so students have a chance to understand 99.9% of the remaining 80% of your course?
 - Will you teach C++ without telling them the language is scary, and they must also know the 'C' language?
 - Do you discuss debugging early in the course?

CS 3520 Programming in C++

"Programming in C++, one pixel at a time" -- Your Instructor

- So I have a theme for my course-- graphical programs manipulating pixels
 - Students will write graphical programs to help them 'see' as they learn
 - Algorithmically interesting problems could be solved.
 - Theme feels inclusive (potentially interesting to a wide variety of students)

- Will you (or are you currently) teaching with a common theme?
- Are you willing to sacrifice 20% of your course so students have a chance to understand 99.9% of the remaining 80% of your course?
- Will you teach C++ without telling them the language is scary, and they must also know the 'C' language?
- Do you discuss debugging early in the course?

CS 3520 Programming in C++

"Programming in C++, one pixel at a time" -- Your Instructor

- So I have a theme for my course-- graphical programs manipulating pixels
 - Students will write graphical programs to help them 'see' as they learn
 - Algorithmically interesting problems could be solved.
 - Theme feels inclusive (potentially interesting to a wide variety of students)

- Will you (or are you currently) teaching with a common theme?
- Are you willing to sacrifice 20% of your course so students have a chance to understand 99.9% of the remaining 80% of your course?
- Will you teach C++ without telling them the language is scary, and they must also know the 'C' language?
- Do you discuss debugging early in the course?

- **Okay...what is the rest of the course going to look like?**

[Course Construction]

✓ Where do we start?

What will students build?

What skills will students learn?

What will students build? (1/2)

- So I have at least one 'neat' pixel effect, but what about the rest of the course?
 - Maybe I could search some of the contents of C++ books table of contents?
 - Maybe I could retrofit some of the exercises to involve graphics.
 - But that doesn't seem quite right--so I'm focusing on what students will build--cool 2D pixel graphics effects.
 - Again, guided my my 'mantra' or 'theme'

What will students build? (2/2)

So I have at least one 'neat' pixel effect, but what about the rest of the course?

Maybe I could search some of the contents of C++ books table of contents?

Maybe I could retrofit some of the exercises to involve graphics.

But that doesn't seem quite right--so I'm focusing on what students will build--cool 2D pixel graphics effects.

Again, guided my my 'mantra' or 'theme'

So now, began my search for cool 2D effects (beyond the DOOM Fire) that would inspire and engage students while learning C++. Some inspired from my real world experience, others examples that look neat.

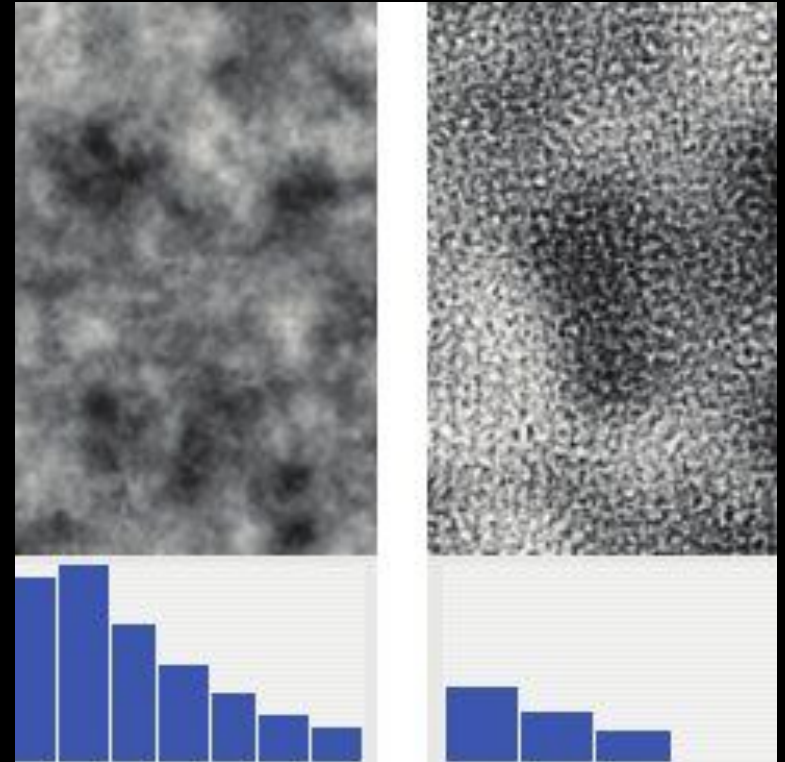
Edge detection and other Image Processing Techniques

- So it started simple--image processing
 - Convert images to grayscale
 - Create filters
 - Edge detection
 - Blurring of images
 - (And eventually do these with `std::thread`)
- And I liked that students could verify the output--keep their confidence up knowing when they've reached a solution!



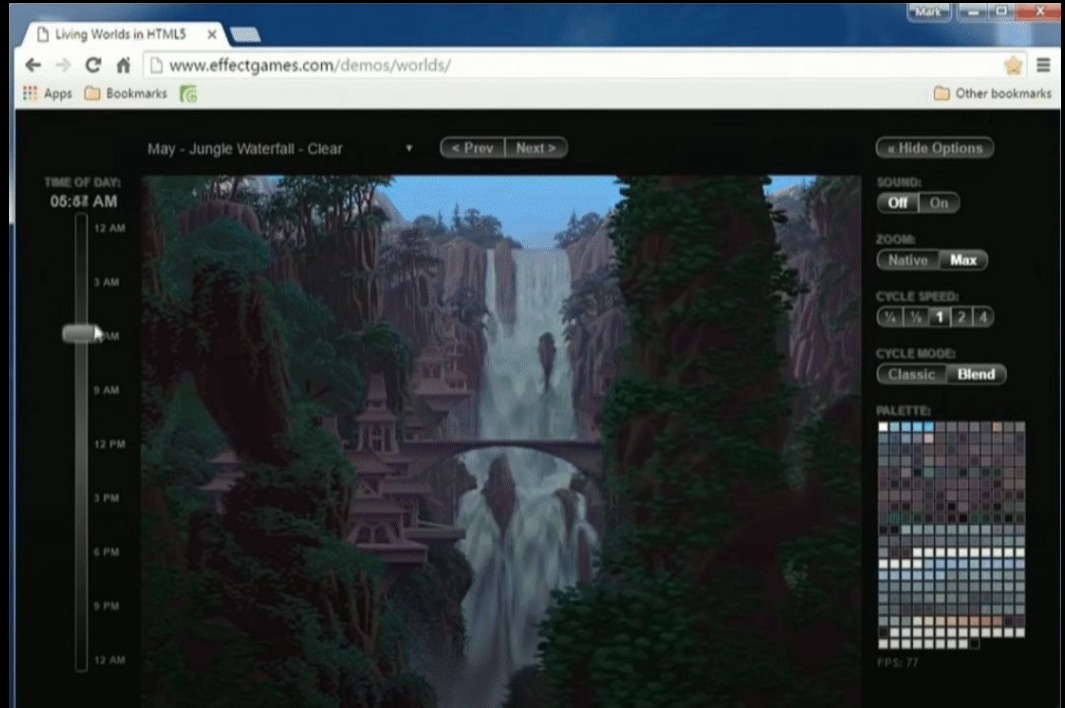
Procedural Textures and Noise

- If students are going to do image processing, then they should learn about 'noise' functions
 - Force students to further think algorithmically
 - Procedural textures are also very creative and may appeal to students with combined degrees in art taking the class.



8 Bit & '8 Bitish' Graphics-Outside the Box

- Mark Ferrari's Game Developer Conference 2016 Talk [\[link\]](#)
 - Various 8 bit graphics tricks
 - Palette Shifting shown on the right
 - (That's one image--no frames of animation, just pixels in the palette rotating))
- Again--another inspiring, and self-contained example of doing a cool 2D trick.
 - Something students could show off to students outside of the class!



Parallax Scrolling

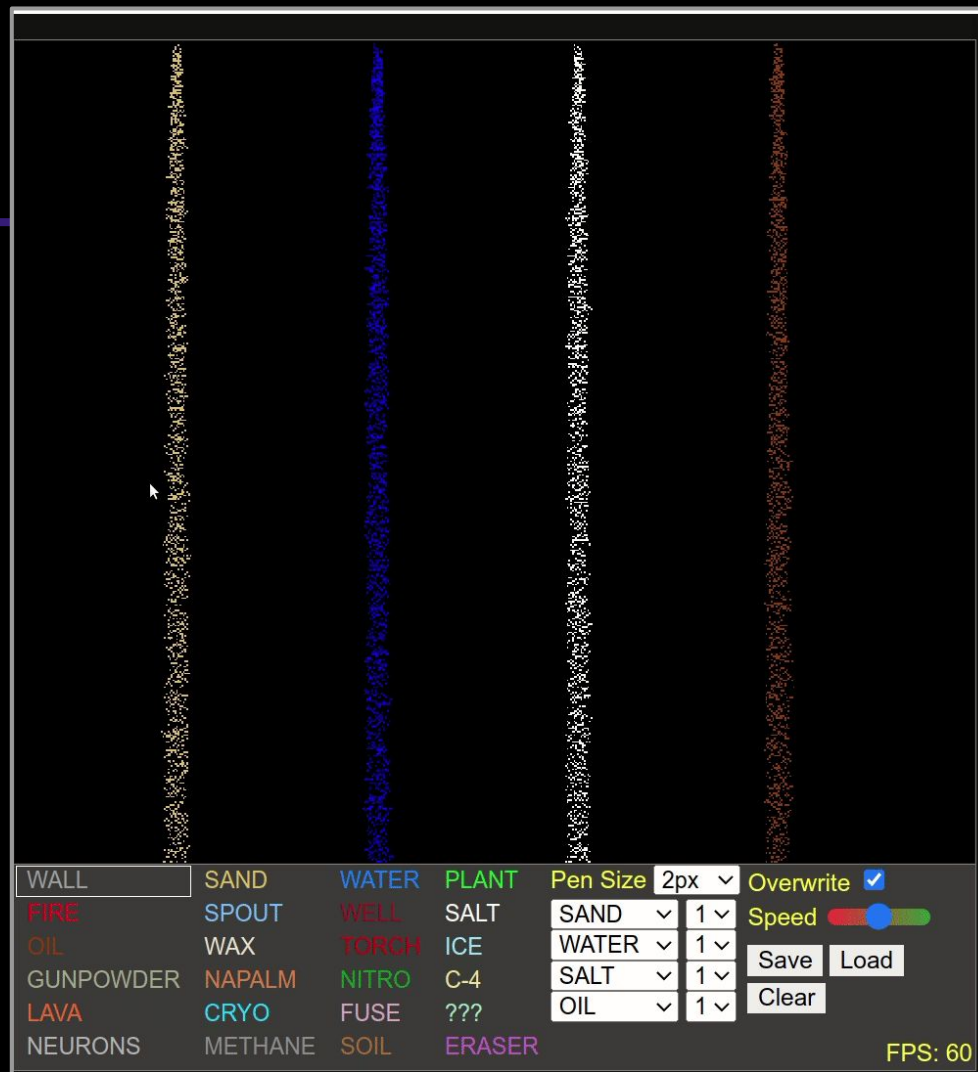
- Create the illusion of depth in a 2D scene
 - Another assignment students could be creative with, importing their own images (or using a default)
 - Lots of students interested in games!



Falling Sands Game

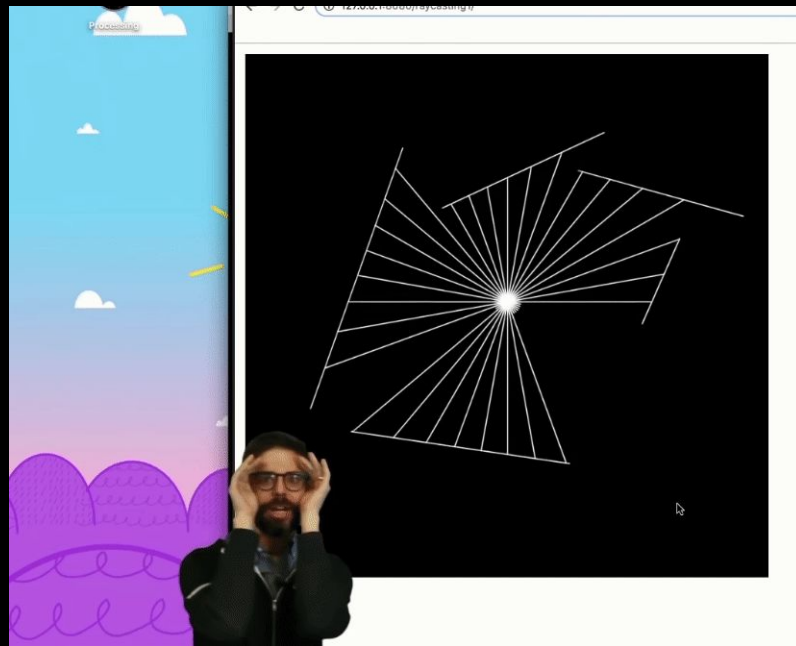
- More gaming examples come to mind
 - I started thinking more about interacting with pixels
- A falling sands game may be particularly interesting to students,
 - Eventually forces students to build a bigger project and think about software architecture
 - (A falling sands game has different materials follow different rules)

https://boredhumans.com/falling_sand.php



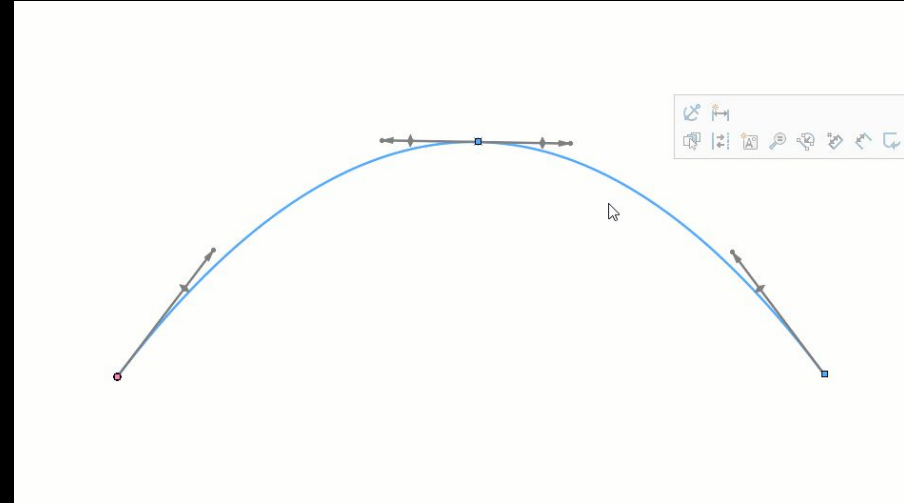
2D Lighting (Raycasting)

- Now I'm thinking more about interaction
- It's fun to play with your program once you write something
- Here's a cool lighting techniques
 - Maybe this would be a fun way of creating the classic 'implement a vector2 class so I can force operator overloading on you'



2D Splines

- Implementing Splines may be another cool custom data type.
 - May be interesting to incorporate STL data structures like `std::list` or `std::vector` to store each of the control points
 - Great, students can see how the STL can save them time



Quad Trees

- What happens when a data structure does not exist in the STL?
 - C++ programmers have to build their own data structures, here come the Quad trees!
- A data structure for optimizing the interaction of lots of objects
 - Felt like something that could be built on after having students implement a binary tree.

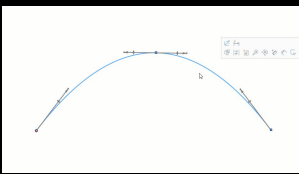
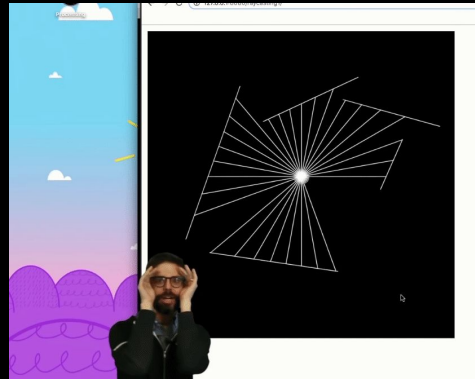
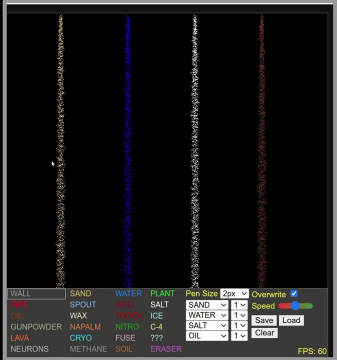
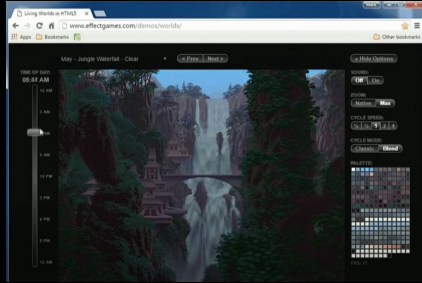
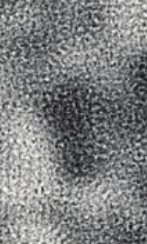
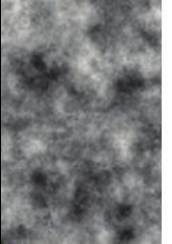


2D Game with lighting effects

- Maybe I can have students start putting all of these effects together
 - This time creating a larger 'game like' environment with some cool effects.



Enough Ideas--Time to think about the execution



[Course Construction]

- ✓ Where do we start?
- ✓ What will students build?
What skills will students learn?

Wait...not so fast on that checkmark

[Course Construction]

- ✓ Where do we start?
- ✓ What will students build?

What skills will students learn?

Question to Audience (1/2):

- Have you ever been part of a class where the instructor says:
 - “I’ll just quickly go through these last slides as time is running out”
 - “I’ll let you read the rest of the material after class”
 - “I won’t cover this in class, but here are some resources...”

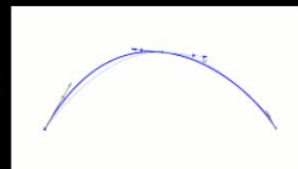
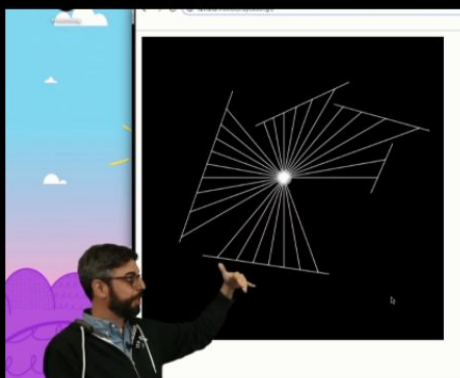
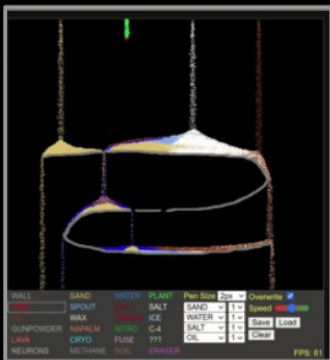
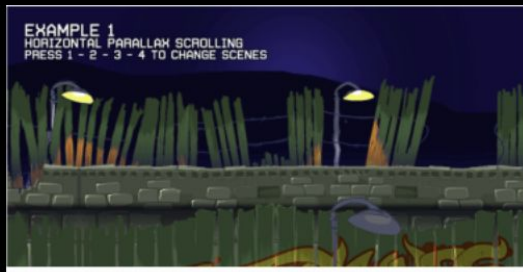
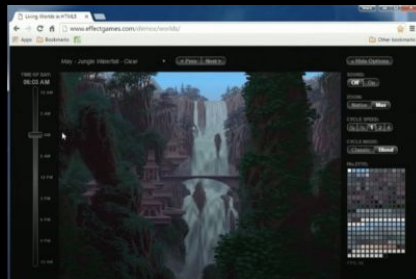
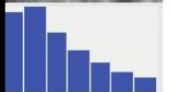
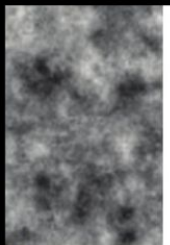
Question to Audience (2/2):

- Have you ever been part of a class where the instructor says:
 - “I’ll just quickly go through these last slides as time is running out”
 - “I’ll let you read the rest of the material after class”
 - “I won’t cover this in class, but here are some resources...”

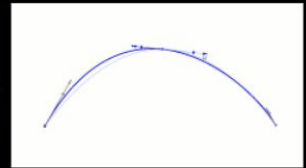
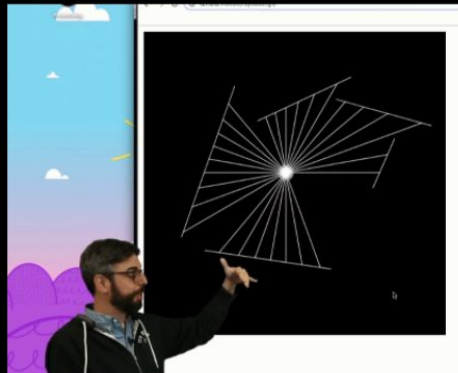
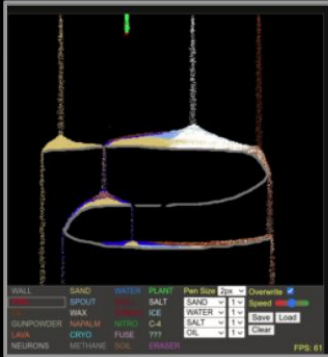
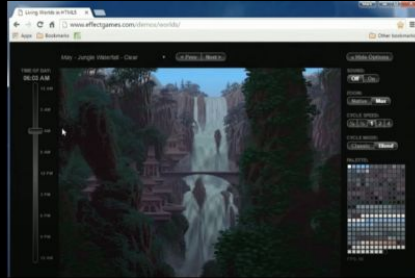
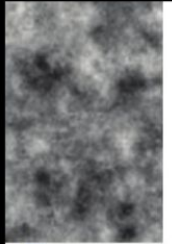
^ Yeah, all of the above break my second item here.

- Will you (or are you currently) teaching with a common theme?
- Are you willing to sacrifice 20% of your course so students have a chance to understand 99.9% of the remaining 80% of your course?
- Will you teach C++ without telling them the language is scary, and they must also know the ‘C’ language?
- Do you discuss debugging early in the course?

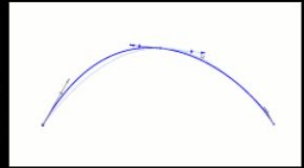
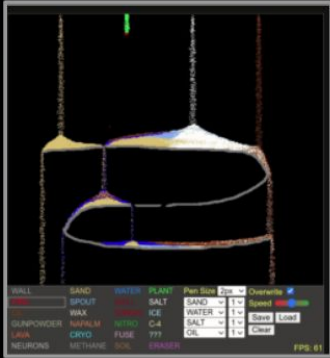
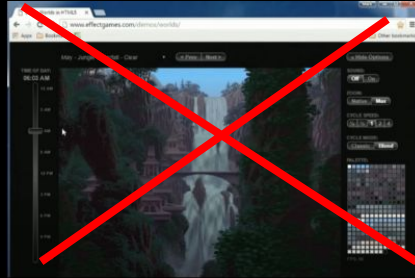
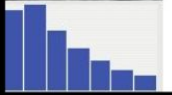
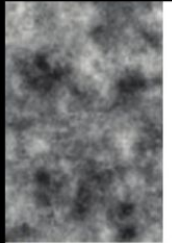
Each idea here could be split to multiple assignments...



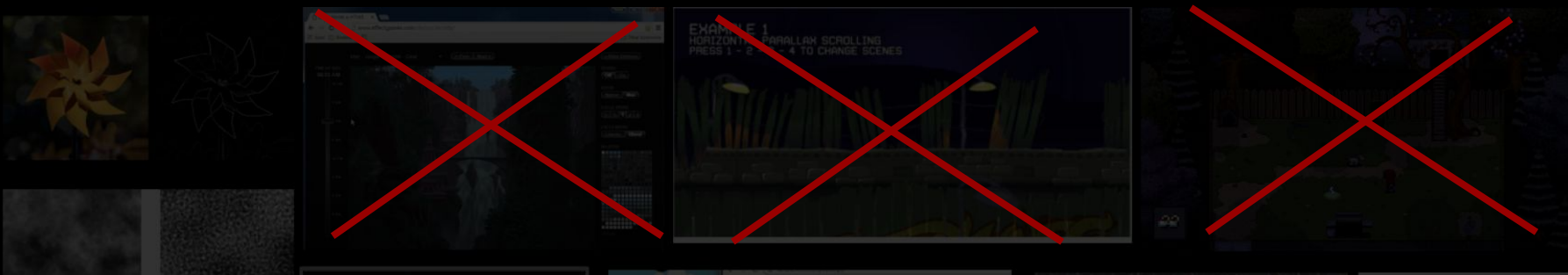
And I want the students to **own** each project (i.e. not give them starter code)



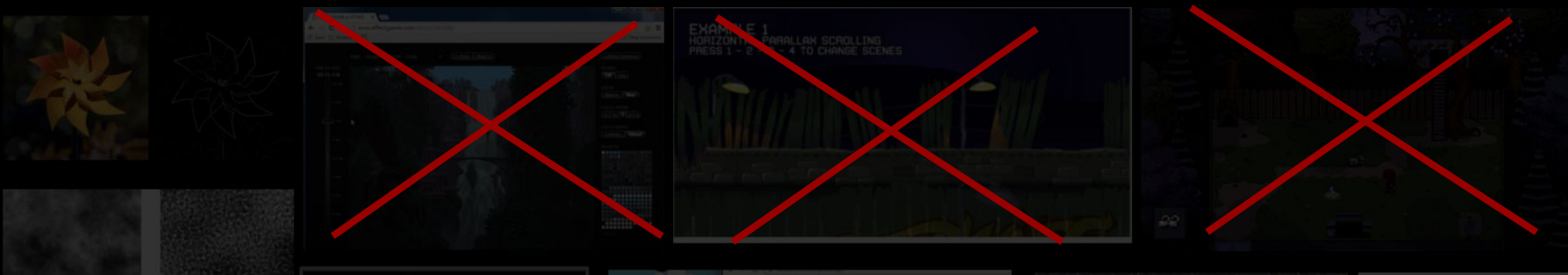
About half of these are going to go (they'll be future assignments for a game programming class or rotated into this class in the future)



Now our course is more appropriately scoped

- 
- Will you (or are you currently) teaching with a common theme?
 - Are you willing to sacrifice 20% of your course so students have a chance to understand 99.9% of the remaining 80% of your course?
 - Will you teach C++ without telling them the language is scary, and they must also know the 'C' language?
 - Do you discuss debugging early in the course?

Now our course is more appropriately scoped

- 
- Will you (or are you currently) teaching with a common theme?
 - Are you willing to sacrifice 20% of your course so students have a chance to understand 99.9% of the remaining 80% of your course?
 - Will you teach C++ without telling them the language is scary, and they must also know the 'C' language?
 - Do you discuss debugging early in the course?

Okay, now I get my checkmark--now let's think about the skills students are going to learn building everything

[Course Construction]

- ✓ Where do we start?
- ✓ What will students build?

What skills will students learn?

What skills will students learn? (1/2)

- Now that we've got the assignments, I need to think really carefully about what skills students will learn in each assignment.
 - I need to figure out some specific skills
 - “Hmm, what have I actually used in the real world”
 - I need to figure out which assignments exercise those skills in a reasonable way
 - I need to figure out a reasonable progression
 - (both in terms of difficulty, and revisiting the same skills in different contexts)

What skills will students learn? (2/2)

- Now that we've got the assignments, I need to think really carefully about what skills students will learn in each assignment.
 - I need to figure out some specific skills
 - “Hmm, what have I actually used in the real world”
 - I need to figure out which assignments exercise those skills in a reasonable way
 - I need to figure out a reasonable progression
 - (both in terms of difficulty, and revisiting the same skills in different contexts)



So I spent some time thinking, and inspiration would arrive...from Twitter!

this is worth reading and is probably mostly true for most individual programming environments
otoh, I use printf debugging because I use a lot of different programming languages / environments and I don't have time or energy for learning that many interactive debuggers...

 **Robert O'Callahan** @rocallahan · Apr 27, 2021

Print Debugging Should Go Away
robert.ocallahan.org/2021/04/print-...

 21

 5

 100



<https://robert.ocallahan.org/2021/04/print-debugging-should-go-away.html>

this is worth reading and is probably mostly true about programming environments. I use printf debugging because I use a lot of languages / environments and I don't have time for many interactive debuggers...

Robert O'Callahan @rocallahan · Apr 27, 2021
Print Debugging Should Go Away
robert.ocallahan.org/2021/04/print-...

21 5 100



Mike Shah, Ph.D. @MichaelShah · Apr 27, 2021

I'm doing an experiment summer semester for my C++ class where I teach them how to print out values in gdb before std::out to get output. Will report back findings :p

1 2



Mike Shah, Ph.D. @MichaelShah

Replying to @tekknolagi and @johnregehr

They're going to have to learn how to use a debugger eventually :D (Idea is not originally mine however, but I do want to try it: youtu.be/YnWhqhNdYyk?t=...)

And realistically--they'll see me using streams, but the first say two lectures I'm going to resist.

 [youtube.com](https://www.youtube.com/watch?v=YnWhqhNdYyk)
CppCon 2015: Kate Gregory "Stop Teaching C"
<http://www.Cppcon.org>—Presentation Slides, PDFs, Source Code and other presenter materials are available at: ...

- This idea I recalled from Kate Gregory's 2015 CppCon talk
- And to be honest...I don't recall a single lecture in my undergrad dedicated to debugging
 - (I dedicate a lecture in my software engineering course--why not offer more immersion here!)



Mike Shah, Ph.D. @MichaelShah · Apr 27, 2021

I'm doing an experiment summer semester for my C++ class where I teach them how to print out values in gdb before std::out to get output. Will report back findings :p



1



2



Mike Shah, Ph.D.

@MichaelShah

Replying to @tekknolagi and @johnregehr

They're going to have to learn how to use a debugger eventually :D (Idea is not originally mine however, but I do want to try it: youtu.be/YnWhqhNdYyk?t=...)

And realistically--they'll see me using streams, but the first say two lectures I'm going to resist.



youtube.com

CppCon 2015: Kate Gregory "Stop Teaching C"

<http://www.cppcon.org>—Presentation Slides, PDFs, Source Code and other presenter materials are available at: ...

Kate Gregory's talk in 2015 -- inspires me to teach debugging (1/3)

- I thought about this for a few minutes
 - What do I do in the real world? I'm debugging half the time.
 - Debugging is how I get myself out of trouble.
 - If I don't teach my students how to get out of trouble
 - They don't learn a very critical real world skill
 - They come to office hours and I teach them one at a time--and that's not efficient at scale.

cppcon

Instead: Use the debugger

- Let learners see the value of their variables while the program runs
- Simplifies sample code by removing multiple output statements
- Keeps learners connected to their code in one environment rather than flipping between code and output

KATE GREGORY
@gregcons

Stop Teaching C

Kate Gregory's talk in 2015 -- inspires me to teach debugging (2/3)



Mike Shah, Ph.D. @MichaelShah · Apr 27, 2021



I'm doing an experiment summer semester for my C++ class where I teach them how to print out values in gdb before `std::out` to get output. Will report back findings :p

- **(Spoiler Alert)** So here's the findings--
 - (next slide)

Kate Gregory's talk in 2015 -- inspires me to teach debugging (3/3)



Mike Shah, Ph.D. @MichaelShah · Apr 27, 2021



I'm doing an experiment summer semester for my C++ class where I teach them how to print out values in gdb before `std::out` to get output. Will report back findings :p

- **(Spoiler Alert)** So here's the findings--
 - I did teach at least 1 debugging skill at the end of every lecture (often relevant to the topic)
 - Most students figure out `std::cout` on week 1 and use that anyway
 - Were my efforts a waste of time?
 - (next slide)

Kate Gregory's talk in 2015 -- inspires me to teach debugging (3/3)



Mike Shah, Ph.D. @MichaelShah · Apr 27, 2021



I'm doing an experiment summer semester for my C++ class where I teach them how to print out values in gdb before `std::out` to get output. Will report back findings :p

- **(Spoiler Alert)** So here's the findings--
 - I did teach at least 1 debugging skill at the end of every lecture (often relevant to the topic)
 - Most students figure out `std::cout` on week 1 and use that anyway
 - Were my efforts a waste of time?
 - No -- students did use GDB or LLDB
 - No -- I was able to help them in office hours without 'GDB' being a mysterious or magical tool (They'd often ask -- 'how did you do that again' and try themselves).
 - No -- several students email me 6-8 months later while at their first C++ internship thanking me for teaching them GDB.

So...what skills will students learn (1/5)

- Debugging (in theory)
 - Every lecture they're exposed.
 - Some fully embrace it, some don't, and that's fine.
 - When the projects get larger, they'll embrace debugging more.

So...what skills will students learn (2/5)

- Debugging (in theory)
 - Every lecture they're exposed.
 - Some fully embrace it, some don't, and that's fine.
 - When the projects get larger, they'll embrace debugging more.

- Will you (or are you currently) teaching with a common theme?
- Are you willing to sacrifice 20% of your course so students have a chance to understand 99.9% of the remaining 80% of your course?
- Will you teach C++ without telling them the language is scary, and they must also know the 'C' language?
- Do you discuss debugging early in the course?

So...what skills will students learn (3/5)

- Debugging (in theory)
 - Every lecture they're exposed.
 - Some fully embrace it, some don't, and that's fine.
 - When the projects get larger, they'll embrace debugging more.

- Will you (or are you currently) teaching with a common theme?
- Are you willing to sacrifice 20% of your course so students have a chance to understand 99.9% of the remaining 80% of your course?
- Will you teach C++ without telling them the language is scary, and they must also know the 'C' language?
- Do you discuss debugging early in the course?

And not just 'discuss', but actually practice!

So...what skills will students learn (4/5)

- Debugging (in theory)
 - Every lecture they're exposed.
 - Some fully embrace it, some don't, and that's fine.
 - When the projects get larger, they'll embrace debugging more.
- (next skill)

So...what skills will students learn (5/5)

- **Debugging (in theory)**
 - Every lecture they're exposed.
 - Some fully embrace it, some don't, and that's fine.
 - When the projects get larger, they'll embrace debugging more.
- **The Standard Template Library**
 - (why? Next slide)

Embracing the STL - Containers

- I tell students C++ is a **'batteries included'** language.
 - I tell students we have data structures available in a library--similar to the way Python has a built-in List and Dictionary data structures.
- No need to implement these from scratch
 - That may be an interesting exercise, but for now I want students building software.
 - Again, in the real world (and from my experience), students will use libraries like the STL

Strings library

```
basic_string  
basic_string_view (C++17)  
Null-terminated strings:  
byte – multibyte – wide
```

Containers library

```
array (C++11) – vector – deque  
list – forward_list (C++11)  
map – multimap  
set – multiset  
unordered_map (C++11)  
unordered_multimap (C++11)  
unordered_set (C++11)  
unordered_multiset (C++11)  
stack – queue – priority_queue  
span (C++20)
```

Embracing the STL - Algorithms

- Embracing the STL also gives me an advantage as a teacher!
- I can also show students quick examples like this
 - “Here’s a collection of data, and we can sort it”
 - This makes C++ ***so beautiful*** as a language and not something so scary.
- So students are exposed to both STL containers and STL Algorithms libraries

Algorithms library

The algorithms library defines functions for a variety of purposes that operate on ranges of elements. Note that a range is defined as a pair of iterators, with the last element to inspect or modify.

Constrained algorithms

C++20 provides **constrained** versions of most algorithms in the STL. A range can be specified as either an **iterator-sentinel** pair or a **range-compare** pair. Additionally, **range-compare** pointer-to-member callables are supported. Additionally, the **ranges** namespace provides functions to return all potentially useful information computed during the execution of an algorithm.

```
std::vector<int> v = {7, 1, 4, 0, -1};  
std::ranges::sort(v); // constrained algorithm
```

pssst -- and sometimes I show them in Godbolt!

- I want them to learn how to ‘play’ and try out new things.
- That it’s okay to prototype small snippets of code to learn
 - Pro tip: It’s also nice to save the compiler explorer snippets for students to quickly run

```
1 #include <iostream>
2 #include <algorithm>
3
4 int main(){
5
6     std::vector<int> v = {5,4,3,2,1};
7     std::ranges::sort(v);
8
9     std::ranges::for_each( v.begin(),
10                          v.end(),
11                          [](const int& i){ std::cout << i << ","; }
12                          );
13
14     return 0;
15 }
```

x86-64 gcc 11.2



-std=c++23

Program returned: 0

Program stdout

1,2,3,4,5,

So...what skills will students learn (1/2)

- **Debugging (in theory)**
 - Every lecture they're exposed.
 - Some fully embrace it, some don't, and that's fine.
 - When the projects get larger, they'll embrace debugging more.
- **Embrace the STL**
 - C++ is a 'batteries included' language.
- **Play and Prototype with Code**
 - Godbolt is an example tool that you can quickly prototype in

So...what skills will students learn (2/2)

- **Debugging (in theory)**
 - Every lecture they're exposed.
 - Some fully embrace it, some don't, and that's fine.
 - When the projects get larger, they'll embrace debugging more.
- **Embrace the STL**
 - C++ is a 'batteries included' language.
- **Play and Prototype with Code**
 - Godbolt is an example tool that you can quickly prototype in
- **(Next Skill -- compiling and linking)**

SFML Library - Compiling and Linking

- A subtle bonus of doing a graphics themed course means we depend on an external graphics library
 - Students thus early on get experience with compiling and linking
 - Always starting on the command line
 - Then eventually they get a build script (and eventually make)



So...what skills will students learn (1/2)

- **Debugging (in theory)**
 - Every lecture they're exposed.
 - Some fully embrace it, some don't, and that's fine.
 - When the projects get larger, they'll embrace debugging more.
- **Embrace the STL**
 - C++ is a 'batteries included' language.
- **Play and Prototype with Code**
 - Godbolt is an example tool that you can quickly prototype in
- **Compiling and Linking**
 - Learn how to work in C++

So...what skills will students learn (2/2)

- **Debugging (in theory)**
 - Every lecture they're exposed.
 - Some fully embrace it, some don't, and that's fine.
 - When the projects get larger, they'll embrace debugging more.
- **Embrace the STL**
 - C++ is a 'batteries included' language.
- **Play and Prototype with Code**
 - Godbolt is an example tool that you can quickly prototype in
- **Compiling and Linking**
 - Learn how to work in C++
- **(next skill -- Modern constructs versus old constructs)**

Modern C++ versus old constructs

- So throughout this process, where do I stand on teaching C++11 and beyond (i.e. C++23) versus C++98
 - I **still** see universities teaching old school C++ courses.
- The modern stuff is what makes the language more approachable in my opinion
 - Teach them C++20 <algorithm> using ranges
 - Show them lambda functions (explain the captures as needed)
 - Teach them new containers (e.g. string_view)
 - Show them how nice it is to use std::function
 - Show them smart pointers
- **BUT...**

I do choose a few 'old' constructs first

1. raw pointers before smart pointers

- Students first need to understand that a pointer holds an address
 - Students need to experience the pitfalls of pointers first
 - Then, I can shift students mental model of a pointer, to think more about 'ownership' of memory.
- (Further justification: Students can't avoid learning raw pointers forever, as they'll eventually need to interface with C libraries)

2. raw arrays before `std::array`

- Same reason as above--a little too much abstraction.
 - Students are okay understanding raw 1D and 2D arrays
 - Much less abstraction in the debugger as well
 - (Initially easier to examine a 1D array)
 - Note: Students see `std::vector` well before they see a raw array however!

Summary - So...what skills will students learn

1. Debugging (in theory)

- a. Every lecture they're exposed.
 - i. Some fully embrace it, some don't, and that's fine.
 - ii. When the projects get larger, they'll embrace debugging more.

2. Embrace the STL

- a. C++ is a 'batteries included' language.

3. Play and Prototype with Code

- a. Godbolt is an example tool that you can quickly prototype in

4. Compiling and Linking

- a. Learn how to work in C++

5. Modern C++

- a. Teach the 'new' stuff--mostly

So we know where to start, what to build, and the skills students should acquire. Let's see the course

[Course Constructed]

- ✓ Where do we start?
- ✓ What will students build?
- ✓ What skills will students learn?

Course Modules

CS 3520 Programming in C++

"Programming in C++, one pixel at a time" -- Your Instructor -- Your Instructor



Instructor

- Instructor: Mike Shah
- E-mail: [mkeshah\(a\)t\(Northeastern\(dot\)jedu\)\(Read How to send an email\)](mailto:mkeshah(a)t(Northeastern(dot)jedu)(Read How to send an email))
- Office: Virtual Nightingale-432A
- Student Hours: In Location TBD
 - Review course material with me and ask questions
 - Date/Time: By appointment
 - Sign up for a 15-30 minute appointments arranged by e-mail on Google Hangouts [here](#)
- Forum: [Piazza Forum Board](#)
- Microsoft Teams Link for Live Video: [Teams Link](#)



General Purpose,
Multi-paradigm



Powerful, Modern



, and fun language



The full [Course Youtube Playlist](#)

Course Information

- **Course Number:** 3520
- **Semester:** Summer 21 - Summer 1
- **Piazza:** [Piazza Forum Board](#)
 - Use piazza instead of e-mail for all course related matters
 - This way all course staff can answer questions
 - Class as a whole can participate and learn in discussions
 - Code also formats better nicely pasted in
 - (Linking your github is also easy for private posts)
- **Section 1:** 3520
 - Schedule: Modules released Monday, Tuesday, Wednesday, and Thursday by 6pm EST.
 - Location: Online
 - Teams Link for Live Classes: [Teams Link](#)

Schedule/Road Map

The following is our tentative syllabus for the course, some changes should be expected throughout the semester. I will announce in class lecture, piazza, or through e-mail any major changes.

- To get all of the assignments/activities for the course, you must first click the following link: [Course Monorepo](#). Do not do a 'git pull' until class starts (Occasionally I make changes/spelling corrections)

Week	Date	Lecture and Readings	Assignments	Note(s)
1	Monday - May 10, 2021	Module 1 - C++ is not the C Language -- Course Introduction Video	A1 Released -- Guessing Game (Due May 14 Anywhere on Earth)	Welcome back to class!
1	Tuesday - May 11, 2021	Module 2 - C++ Batteries Included - STL and GDB Debugging with the STL Video	--	--

Teaching Assistant(s)	E-mail	Hours
Colan Biemer	biemer.c	◦ Hours/Location here
Madison Kang	kang.mad	◦ Hours/Location here
Jennifer Ribeiro	ribeiro.je	◦ Hours/Location here

What is being taught

(The curriculum and assignments)

Module 1 - C++ is not the C Language -- Course Introduction | [Video](#)

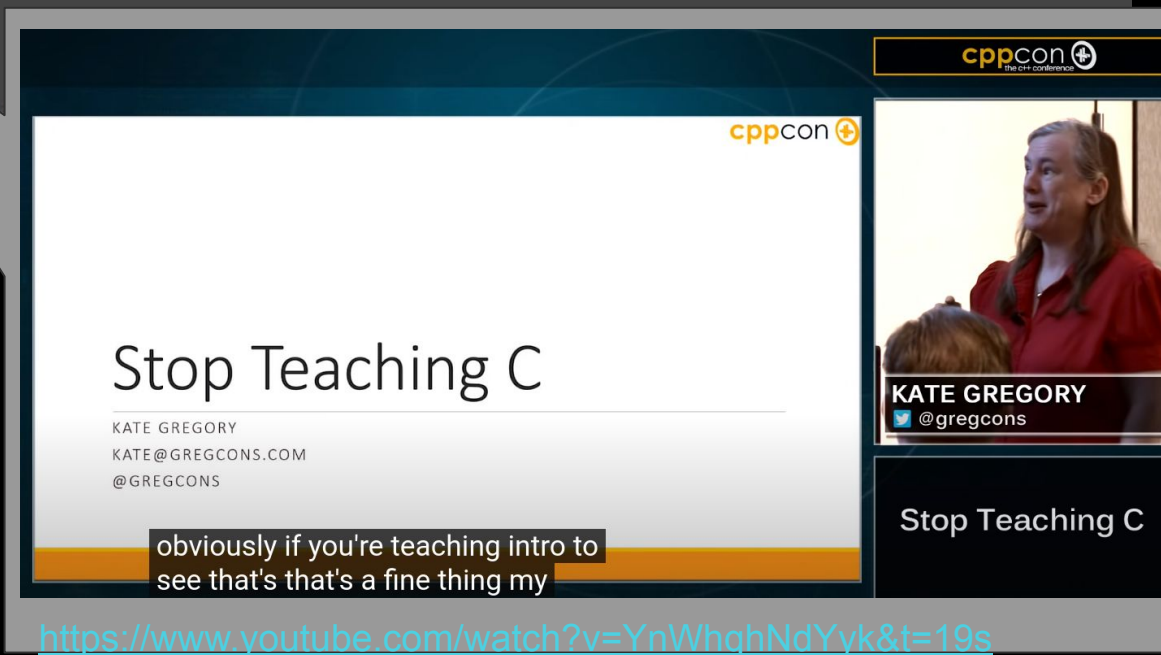
- Lecture outline
 - Course Structure
 - Quick Examples in C++
 - [Begin Module!](#)

- A quick tour of the C++ language showing some features of the language
- Goal is to get students to be excited!

Module 1 - C++ is not the C Language -- Course Introduction | [Video](#)

- Lecture outline
 - Course Structure
 - Quick Examples
 - [Begin Module!](#)

The first lecture is 100% inspired by *CppCon 2015: Kate Gregory "Stop Teaching C" [in a C++ class]*



The screenshot shows a video player interface. The main content is a presentation slide with the following text:

- Top right: **cppcon** logo with a plus sign icon.
- Center: **Stop Teaching C**
- Below title: KATE GREGORY, KATE@GREGCONS.COM, @GREGCONS
- Bottom: A subtitle box containing the text "obviously if you're teaching intro to see that's that's a fine thing my"

On the right side of the video player, there is a small inset video of a woman (Kate Gregory) speaking. Below the video is a name tag: **KATE GREGORY** with a Twitter icon and **@gregcons**. At the bottom right of the video player, the text **Stop Teaching C** is displayed.

At the bottom of the entire image, there is a URL: <https://www.youtube.com/watch?v=YnWhqhNdYyk&t=19s>

Module 2 - C++ Batteries Included - STL and GDB Debugging with the STL | [Video](#)

- Lecture outline
 - C++ Standard Template Library
 - Using GDB to Debug and Investigate the STL
 - [Begin Module!](#)
- Again, stolen from Kate Gregory -- I lean on the C++ STL.
 - I make the C++ language feel like a language 'with batteries included'
 - Students coming from a Python/Java background are used to this.
 - Just give your students `std::vector` on day 2--explain arrays later.

Module 3 - Functions 1: Making our code more modular | GDB and Functions | [Video](#)

- Lecture outline
 - Pass by Value
 - Pass by Reference
 - const parameters
 - constexpr parameters
 - Function polymorphism
 - Small math library
 - Using GDB to debug functions
 - [Begin Module!](#)

- Now we get into functions
 - Still no real graphical 'output' yet (coming soon!)
- Students have to use 'GDB' if they want to inspect any values
 - I have them implement some math functions to do this
 - It's neat to show students pass-by-value and copies being made with GDB. Also neat to simply show breakpoints on functions

Module 4 - Input and Output | [Video](#)

- Lecture outline
 - Streams
 - Input Output functions
 - File Reading and Writing
 - Text Parsing
 - `std::filesystem`
 - [Begin Module!](#)

- Finally we can output and read in text
 - Students can also start working with files, to read/write data.
- Just enough for them to do some their first real graphical exercise.

Module 4 - In

- Lecture
- St
- In
- Fi
- Te
- st
- B



- First real graphical assignment
 - An image to ascii image convertor.

Module 5 - C++: Thinking about memory | [Video](#)

- Lecture outline
 - Revisiting arrays and raw arrays
 - arrays
 - Raw Pointers
 - Pointer Arithmetic
 - Segmentation fault in GDB
 - [Begin Module!](#)

- Now we start learning a bit more about memory
 - Again, I'm teaching raw arrays and pointers here first.
- Notice the GDB exercise on debugging.
 - Usually I'm targeting GDB exercises for where I think students will run into trouble.

Module 6 - Memory Allocation and Layout | [Video](#)

- Lecture outline
 - Stack Memory
 - Heap Memory
 - Static Memory
 - 2D Arrays
 - Structs
 - [Begin Module!](#)

- I continue with more on memory allocation and memory layout
 - Starting to prepare students for 2D image processing tasks.
- (Explicit GDB exercise not listed on outline anymore, it's assumed)
 - GDB exercise examining arrays, and addresses of heap vs stack memory

Module 7 - Object-Oriented Programming 1 | [Video](#)

- Lecture outline
 - Structs
 - RAI
 - Classes
 - Encapsulation
 - [Begin Module!](#)

- Foundational ideas about creating new data types
- Teach RAI emphasizing it's "nice to clean up your memory/resources after you are finished with them"
- Using GDB to observe callstack with constructors/destructors

Module 8 - OO Continued and SFML Library Introduction | [Video](#)

- Lecture outline
 - Interface versus Implementation
 - The Three Amigos (or Three musketeers) Copy constructor
 - The Four musketeers! (Copy Assignment)
 - Shallow and Deep Copy
 - SFML Library and Virtual Machine
 - [Begin Module!](#)

- More topics on building custom data types
- At this point we're diving a little more into the SFML library
 - Can actually show students some code in a library
 - Goal is to make reading others code and documentation less scary early on
- GDB exercise finding copies of objects passed into functions

Module 8 - OO C

- Lecture outli
 - Interfa
 - The T
 - The F
 - Shallo
 - SFML
 - Begin



- Students start doing some image processing examples

- Students start focusing more on object oriented programming
- Along the way implementing Fire



- And the Falling Sands game for OOP experience



Module 9 - Scope, Building libraries and more on C++ tooling | [Video](#)

- Lecture outline
 - Implementation and Interface
 - Scope Operator
 - Static and Dynamic Linking of Libraries
 - Linking SFML (Idd, and GDB)
 - [Begin Module!](#)

Module 10 - Object-Oriented Programming 2 | [Video](#)

- Lecture outline
 - Single Inheritance
 - Multiple Inheritance
 - virtual
 - pure virtual
 - vtable
 - [Begin Module!](#)

Module 11 - Putting it Together -- Building Data Structures | [Video](#)

- Lecture outline
 - Review: Linked Data Structures with Pointers
 - Trees
 - Binary Trees
 - Algorithm Class - Binary Search Tree
 - #include binary_search
 - [Begin Module!](#)

Module 12 - Falling Sands | [Video](#)

- Lecture outline
 - Application Design
 - Structuring a program
 - Code Walk
 - [Begin Module!](#)

Module 13 - No Class Memorial Day | [Video](#)

Module 14 - Pointers 2 - Getting Smart | [Video](#)

- Lecture outline
 - Building Data Structures
 - Different Types of Smart Pointers
 - [Begin Module!](#)

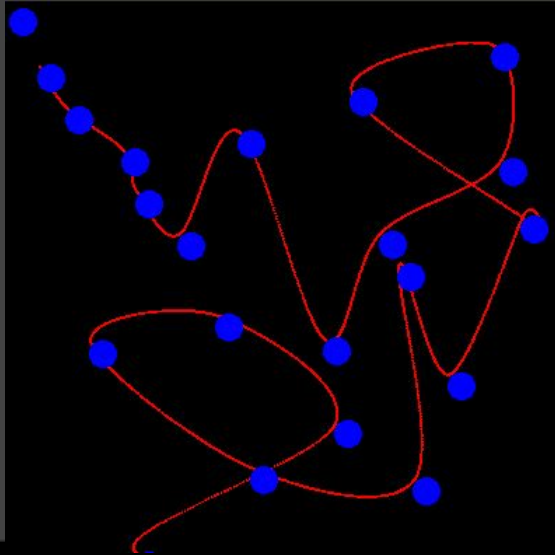
Module 15 - C++ Templates | [Video](#)

- Lecture outline
 - Function Templates
 - Class Templates
 - [Begin Module!](#)

Module 16 - Review (Possibly Quad Tree Data Structure Overview) | [Video](#)

- Lecture outline
 - Object Oriented Programming Review
 - Review on Scope
 - Compilation Process
 - Quad Tree Overview
 - [Begin Module!](#)

- Students begin working on splines and eventually quad trees



- Color Cycling and working with palettes
- Assignment descriptions start allowing more more freedom in how task is achieved.



Module 17 - Graphs | [Video](#)

- Lecture outline
 - Introduction to Graphs
 - Adjacency List
 - Adjacency Matrix
 - Graph Interface/Adjacency List implementation
 - [Begin Module!](#)

Module 18 - No Class | [Video](#)

Module 19 - Splines Data Structure | [Video](#)

- Lecture outline
 - More Graphs
 - Spline Data Structure Explanation
 - Spline Data Structure implementation
 - [Begin Module!](#)

Module 20 - Color Cycling, Sprite Animation, and Resource Manager(Singleton) | [Video](#)

- Lecture outline
 - Introduction to Sprite Animation
 - Introduction to Color Cycling
 - Mesh Warping Idea
 - Mesh Warping Algorithm
 - Mesh Warping Start of Implementation
 - [Begin Module!](#)

- Revisiting old assignments using concurrency and asynchronous I/O
- More tools on iterators and algorithms



Module 21 - Concurrency and Threading | [Video](#)

- Lecture outline
 - Architecture History
 - What is a thread?
 - C++ Threading Examples
 - [Begin Module!](#)

Module 22 - C++ Memory Model and threading continued | [Video](#)

- Lecture outline
 - Asynchronous I/O
 - C++ Memory Model
 - Tracking Memory Allocations
 - [Begin Module!](#)

Module 23 - Iterators and Algorithm | [Video](#)

- Lecture outline
 - Iterators
 - More on #include algorithm
 - [Begin Module!](#)

Module 24 - Move Semantics | [Video](#)

- Lecture outline
 - Move Semantics
 - [Begin Module!](#)

- Course starts to wrap up
- Auxiliary topics that are of interest to students, what I think is important, and what will get students for next courses that use C++ (e.g. computer graphics)
 - This is where future assignments may get incorporated

Module 25 - Advanced Topic 1 - Pybind11 | [Video](#)

- Lecture outline
 - Scripting vs compiled languages
 - Pybind11
 - TRACE Reminder
 - [Begin Module!](#)

Module 26 - Advanced Topic 2 - Graphical User Interface(GUI) | [Video](#)

- Lecture outline
 - Graphical User Interfaces
 - Building from source
 - wxWidgets
 - [Begin Module!](#)

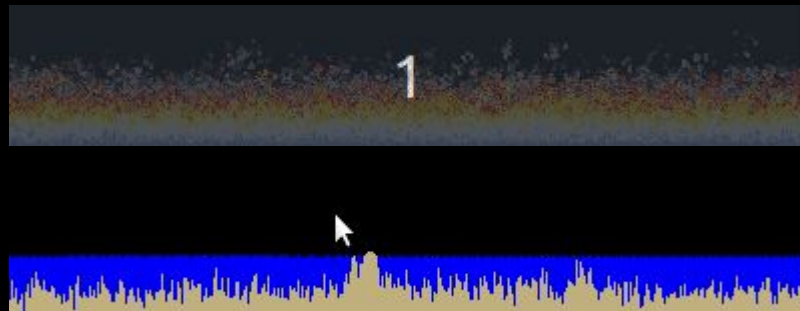
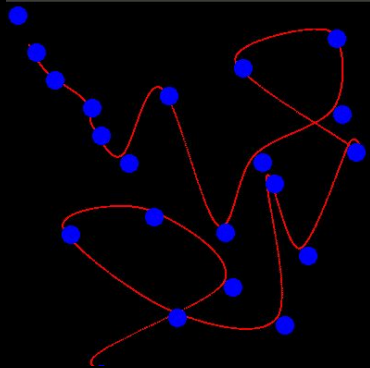
Module 27 - Flex Day - Topic at Instructors Discretion | [Video](#)

- Lecture outline
 - Tools for Making your C++ Experience Better!
 - Tooling - Using Make
 - Tooling - Using CMake
 - cppcheck - Static Analysis
 - [Begin Module!](#)

Module 28 - Course Wrap Up | [Video](#)

- Lecture outline
 - TRACE
 - Summary of the course
 - [Begin Module!](#)

So here's a subset of what students end up with!



Summary so far

So far in this talk... (1/2)

- I've tried to capture to you my process to course creation
 - How I try to provide a solid foundation for building a class
 - (From idea to implementation)
- Some concrete examples of what skills my learners will develop
 - A subset of the assignments students learn from
 - (Several smaller assignments/in-class exercises not shown)

But there's something missing, something very important, a key ingredient I believe to teaching...

So far in this talk... (2/2)

- I've tried to capture to you my process to course creation
 - How I try to provide a solid foundation for building a class
 - (From idea to implementation)
- Some concrete examples of what skills my learners will develop
 - A subset of the assignments students learn from
 - (Several smaller assignments/in-class exercises not shown)

But there's something missing, something very important, a key ingredient I believe to teaching...

Maintaining the excitement and enthusiasm of your students!

Excitement!

Here's was Last Pre-Course Survey Question I asked (1/2)

What else should I know about you? Please let me know if you need any additional accommodations

Excited to learn more from you!

I'm excited to learn!

I am very excited to finally learn c++ and hopefully meet some other students through this class.

I am excited to take this class and learn about C++ (finally)!

Nothing much besides the fact that I'm going to face a bit of a learning curve coming in. This is my first time using github and I don't have any experience or prior knowledge about C++, but I'm sure with some time and practice I'll get the hang of it. Definitely looking forward to this course!

Here's was Last Pre-Course Survey Question I asked (2/2)

What else should I know about you? Please let me know if you need any additional accommodations

Excited to learn more from you!

I'm excited to learn!

I am very excited to finally learn c++ and hopefully meet some other students through this class.

I am excited to take this class and learn about C++ (finally)!

Nothing much besides the fact that I'm going to face a bit of a learning curve coming in. This is my first time using github and I don't have any experience or prior knowledge about C++, but I'm sure with some time and practice I'll get the hang of it. Definitely looking forward to this course!

Teaching Wisdom #1

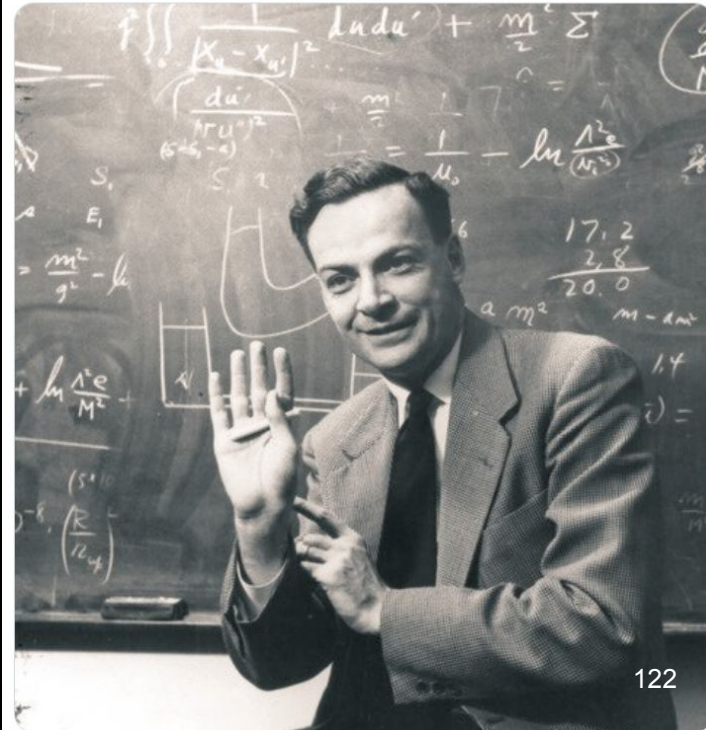
- I think this wisdom from some other great teachers is useful!
 - See the late [Richard Feynman](#) to the right
----->



Prof. Feynman
@ProfFeynman

The goal of teaching should not be to help the students learn how to memorize and spit out information under academic pressure. 🧠

The purpose of teaching is to inspire the desire for learning in them and make them able to think, understand, and question.



Teaching Wisdom #2

- (Or if you'd prefer the words of a computer scientist)



Morgan McGuire @CasualEffects · Feb 16

The first responsibility of a teacher is to unconditionally convince the student that they can succeed. Teachers are mentors and cheerleaders first and repositories of knowledge second. This holds at any level and subject, for academics and athletics, with every student.

Teaching Wisdom #3

- You can't fake passion and enthusiasm when teaching
 - That's why I picked examples I was excited about (and you'll have to replicate this for your own use case)
 - My honest enthusiasm helps keep my students excited when for 28 modules of C++ they see how excited I am getting.
 - My job is to teach C++ in a fun and encouraging way so students will pursue further self-study and courses in C++



A Few Teaching Notes as we Wrap Up

A Few Strategies for Teaching (1/2)

1. First identify your audience

- Is this for an undergraduate, a professor, a professional, or someone who has never programmed?

2. Speak in plain terms

- Explain as if a ten year old could understand.
 - I put Resource Acquisition is Initialization (RAII) every single time to avoid the additional cognitive load it takes for students to avoid jargon.
- Then, progressively discuss a topic deeper eventually hitting your target audiences skill level
- Note; If you use metaphors or analogies to the real world, think globally (e.g. ask yourself if everyone will understand an American Football references?)

A Few Strategies for Teaching (2/2)

1. Tell a Story

- This is the part that will make folks care
- Why are they learning this subject matter? Why does it matter? Can you motivate with history? Can you motivate to students in another way?

2. Identify your own knowledge gaps, and keep refining

- Sometimes your own knowledge gaps will be points you'll want to take note of--probably a good idea to break down those gaps into smaller pieces (for yourself and your student)

**A Pixel Is Not A Little Square,
A Pixel Is Not A Little Square,
A Pixel Is Not A Little Square!
(And a Voxel is Not a Little Cube)¹**

Technical Memo 6

Alvy Ray Smith
July 17, 1995

Abstract

My purpose here is to, once and for all, rid the world of the misconception that a pixel is a little geometric square. This is not a religious issue. This is an issue that strikes right at the root of correct image (sprite) computing and the ability to correctly integrate (converge) the discrete and the continuous. The little square model is simply incorrect. It harms. It gets in the way. If you find yourself thinking that a pixel is a little square, please read this paper. I will have succeeded if you at least understand that you are using the model and why it is permissible in your case to do so (is it?).

http://alvyray.com/Memos/CG/Microsoft/6_pixel.pdf

A Few Strategies for Teaching (3/3)

- Anyone hate how many words I put on my slides? :)
- I stand by breaking that powerpoint rule
 - Students want to review the slides later on and not guess what I was trying to say.
 - (Note: You do need to have thoughtful transitions to guide a student in your slides however!)

1. First identify your audience
 - Is this for an undergraduate, a professor, a professional, or someone who has never programmed?
2. Speak in plain terms
 - Explain as if a ten year old could understand.
 - I put Resource Acquisition is Initialization (RAII) every single time to avoid the additional cognitive load it takes for students to avoid jargon.
 - Then, progressively discuss a topic deeper eventually hitting your target audiences skill level
 - Note; If you use metaphors or analogies to the real world, think globally (e.g. ask yourself if everyone will understand an American Football references?)

Live Coding

- Delivery -- I do live coding about 40% of the time in my class
 - In my personal University experience as a student this was very rare.
 - At my current University many instructors spend a portion of class doing this or sharing recordings.
 - I think this is a good thing
 - I keep my mistakes in my lectures or lecture videos for others to learn from
 - It also gives students a chance to ask 'what if we change this'

Couple more ground rules when you're teaching students

- Students want to know you're competent (i.e. the expert)
- Students do want you to be organized
- Students do want to feel supported.
- Students **are not** impressed by technical jargon
- Students **should not** be given 'the language standard'
 - It's find to point them to the core guidelines later on--some students just like to know a more definitive answer.
 - But I would not teach directly 'from the guidelines text', teach instead from the inspiring projects you want them to learn from.

What is Next?

(Next iteration of the course)

What's next

- Next iteration of course will be supported by lots of supplemental youtube videos
 - *Key word* 'supplemental'.
 - Not required for students to watch, not even encouraged, but there as a definitive resource so students don't have to hunt.
 - Overloading students on resources (especially video) can be detrimental -- especially during pandemic.
- 1 or 2 new assignments
 - Rotating in new assignments that did not make the cut.
 - A 14 week semester versus Summer half semesters allow students to soak in material at a more reasonable pace.
 - (Remember, don't keep adding content if students can't absorb the current contents!)

Things I want more of in the next iteration

- Reading more good (by my standard) C++ code
 - From my own samples, or C++ open source projects)
 - I will have to think carefully about this though--honestly this may belong in a follow up course (e.g. I teach a software engineering course in C++)

Final Check

- Will you (or are you currently) teaching with a common theme?
- Are you willing to sacrifice 20% of your course so students have a chance to understand 99.9% of the remaining 80% of your course?
- Will you teach C++ without telling them the language is scary, and they must also know the 'C' language?
- Do you discuss debugging early in the course?

Final Check on our Homework

- Did I tell my students C++ was scary or hard anywhere?
 - (A quick search with ctrl+F tells me I did not)

- Will you (or are you currently) teaching with a common theme?
- Are you willing to sacrifice 20% of your course so students have a chance to understand 99.9% of the remaining 80% of your course?
- Will you teach C++ without telling them the language is scary, and they must also know the 'C' language?
- Do you discuss debugging early in the course?

Final Check on our Homework

- Did I tell my students C++ was scary or hard anywhere?
 - (A quick search with ctrl+F tells me I did not)

- ✓ Will you (or are you currently) teaching with a common theme?
- ✓ Are you willing to sacrifice 20% of your course so students have a chance to understand 99.9% of the remaining 80% of your course?
- ✓ Will you teach C++ without telling them the language is scary, and they must also know the 'C' language?
- ✓ Do you discuss debugging early in the course?

Post Class Survey

(Optional) It has been a true pleasure teaching you this Summer 1 semester! Please leave any feedback here that you have not otherwise left on TRACE :)

12 responses

Thanks for the great class. I enjoyed it a lot.

Thank you for a great semester! I've really appreciated all your help and enthusiasm for c++!! I hope to take more of your classes soon :)

Already did!

Thank you for the great semester!!

Loved the assignments focusing on images! Made it more fun

Pleasure taking your course!

Thank you!!

As a non-CS major I thought this class was super well structured and easy to follow. Having a lot of the assignments deal with graphics also made the assignments fun since there was a visual aspect. Can happily say I am ending this class not hating C++! Thanks for a great semester :)

Post Class Survey

(Optional) It has been a true pleasure teaching you this Summer 1 semester! Please leave any feedback here that you have not otherwise left on TRACE :)

12 responses

Thanks for the great class! I enjoyed it a lot

Thank you for a great semester! I've really appreciated all your help and enthusiasm for c++!! I hope to take more of your classes soon :)

Already did!

Thank you for the great semester!!

Loved the assignments focusing on images! Made it more fun

Pleasure taking your course!

Thank you!!

As a non-CS major I thought this class was super well structured and easy to follow. Having a lot of the assignments deal with graphics also made the assignments fun since there was a visual aspect. Can happily say I am ending this class not hating C++! Thanks for a great semester :)

**ACCW
2022**

Thank you for listening to my talk!

HOW I TEACH MODERN C++ ONE PIXEL AT A TIME

April 7, 2022 | 14:00 - 15:30

Mike Shah, Ph.D. | [@MichaelShah](https://twitter.com/MichaelShah)

www.youtube.com/c/MikeShah

www.mshah.io

