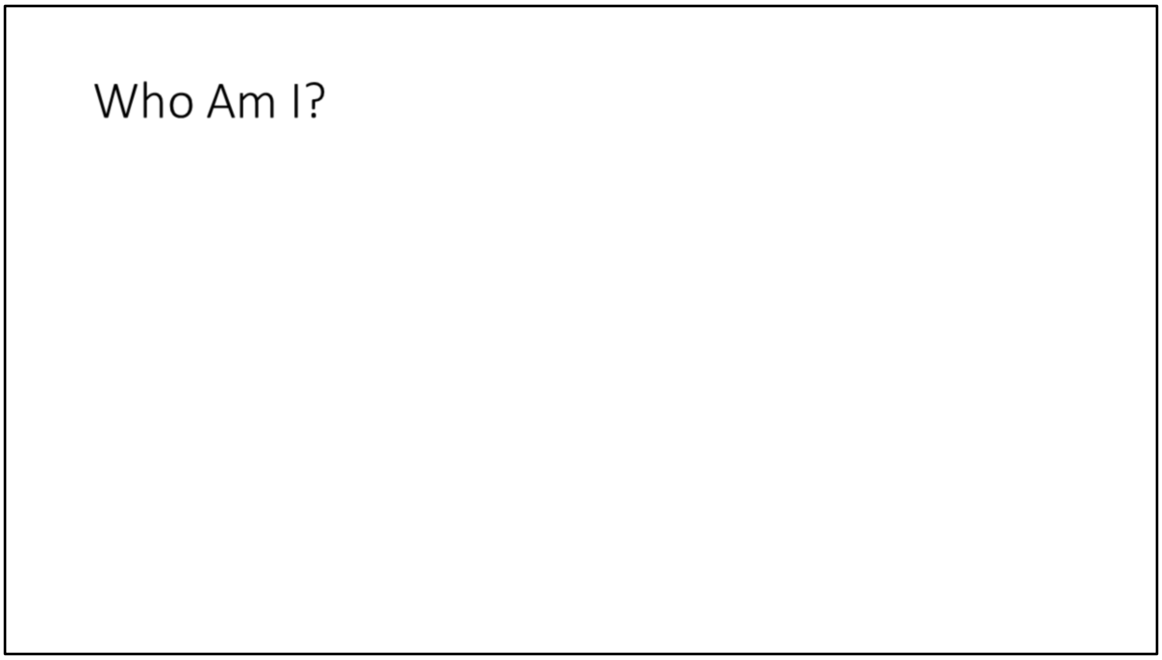
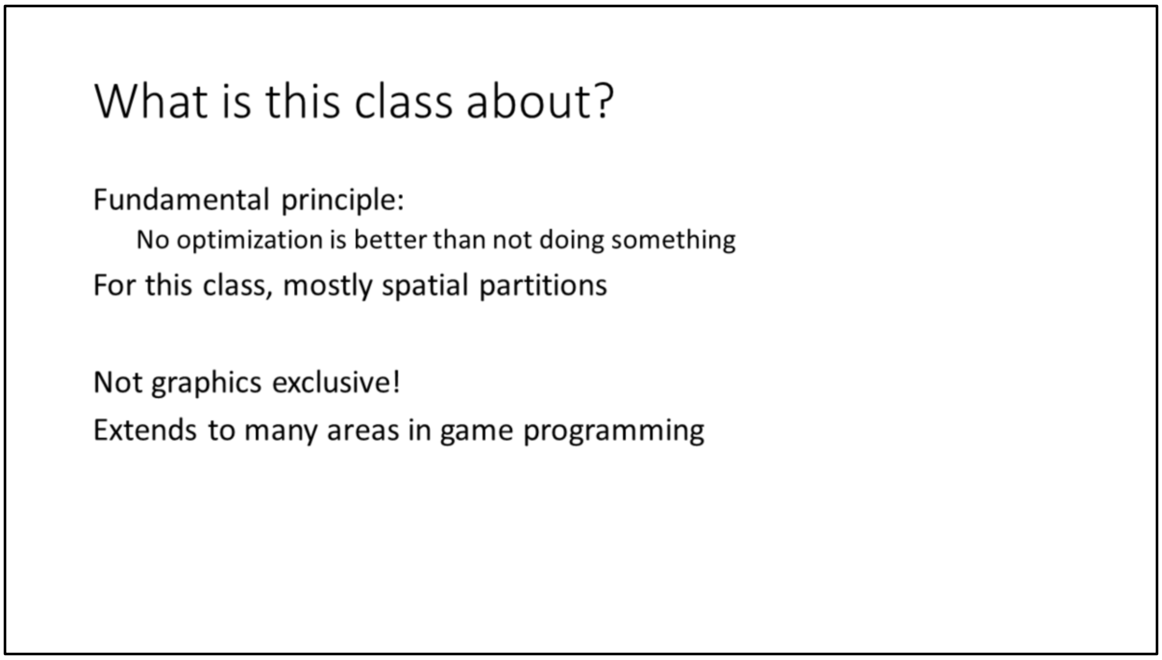
CS350

jodavis42@gmail.com



Who Am I?

I graduated from DigiPen RTIS winter of 2010. My senior year I started work on the Zero Engine as one of the founding members. Afterwards I got my masters spring 2014 here. I've worked on many things over the years, from build systems, crash dumps, ui, cross platform porting, and so on, however my main focus has always been physics. I was a leading member in physics club for a while and if you ever check out the slides you'll probably come across my name. I do love talking about physics so if you ever want to run something by from your physics engines let me know!



What is this class about?

Fundamental principle:

No optimization is better than not doing something

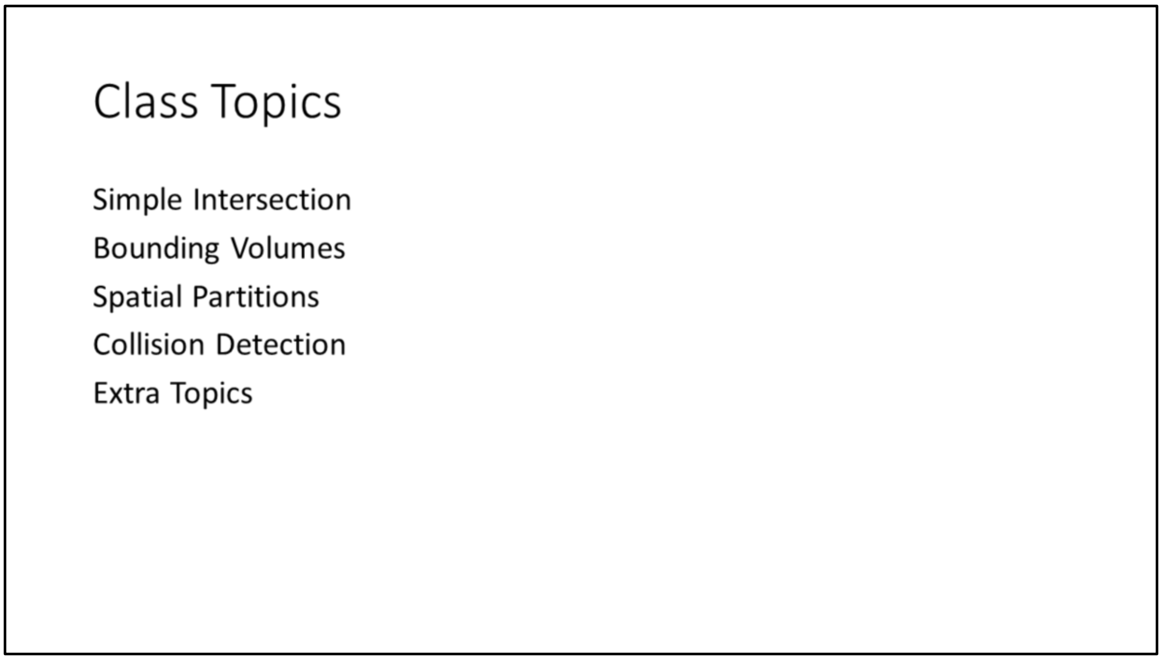
For this class, mostly spatial partitions

Not graphics exclusive!

Extends to many areas in game programming

Despite what the course title may be, this class is not actually a graphics class. This is fundamentally a computational geometry class. The primary point of this class is that no matter how efficient of a technique you use (for rendering, collision detection, etc…) you can always make it faster by not doing it when you don't need to. In fact, no optimization can ever be better than not doing something. Ideally we want efficient ways to determine to not do something (typically through spatial partitions).

Previous teachers have emphasized the graphics side of this, however I want to take a bit of a different approach. Many of the techniques that will be covered in this class originated from graphics, however a lot of them are no longer relevant in rendering but many of them are still incredibly useful in other fields. For instance, physics extensively uses bounding volumes and spatial partitions to perform broadphasing. Graphics needs these techniques for frustum culling. Even simpler, an editor needs to be able to raycast efficiently. Because of these reasons I'm going to mostly stay away from graphics topics and try to show the broader use of these topics.



Class Topics

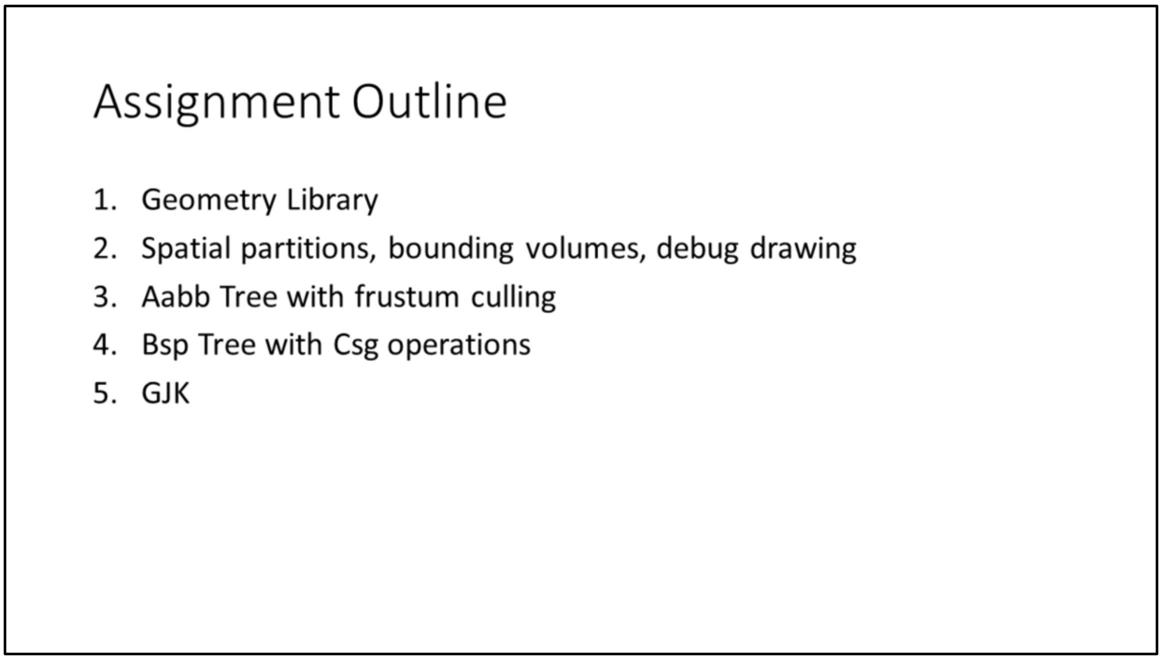
Simple Intersection

Bounding Volumes

Spatial Partitions

Collision Detection

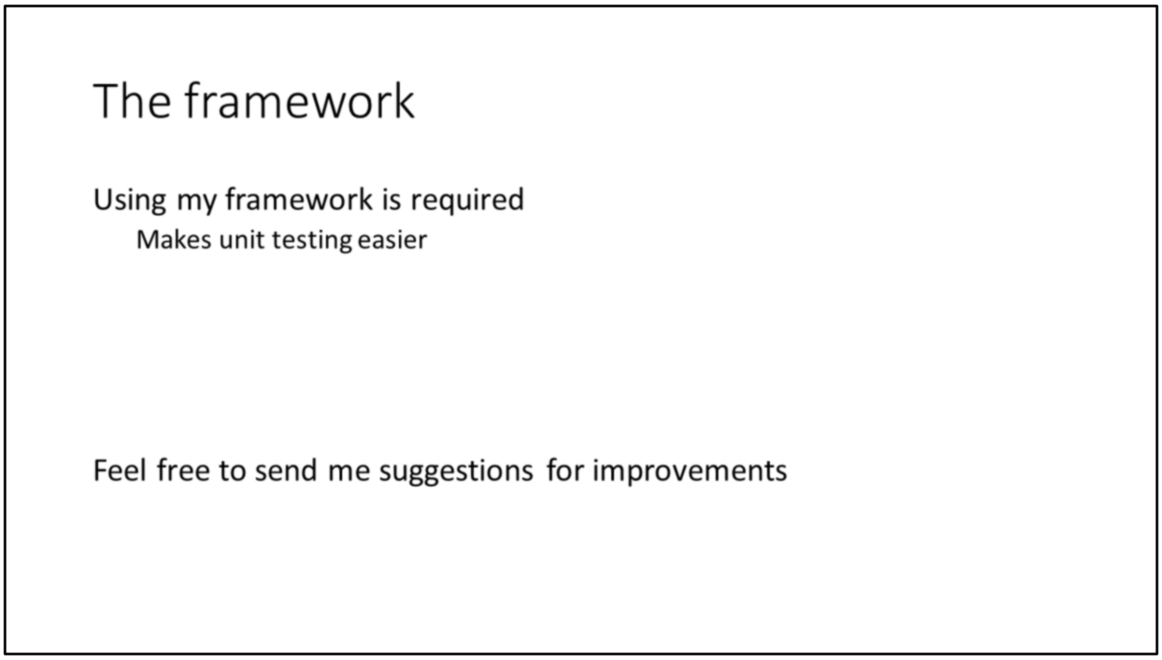
Extra Topics



Assignment Outline

1. Geometry Library
2. Spatial partitions, bounding volumes, debug drawing
3. Aabb Tree with frustum culling
4. Bsp Tree with Csg operations
5. GJK

The assignments for this class build upon each other to a certain extent. The first assignment is the geometry library that you will need for the rest of the assignments. This includes a bunch of tests between rays, triangles, planes, frustums, aabbs, spheres and so on.
The seconds assignment you must compute various bounding volumes and then use them to create some very simple spatial partitions. Also in this assignment basic debug drawing is required.
The third assignment is the first major spatial partition: an aabb tree. This is a very versatile and performant spatial partition that should serve you well.
The fourth assignment is a bsp-tree which you'll use to perform constructive solid geometry operations (subtraction, intersection, etc…)
Finally you'll implement gjk as a generic collision detection algorithm.



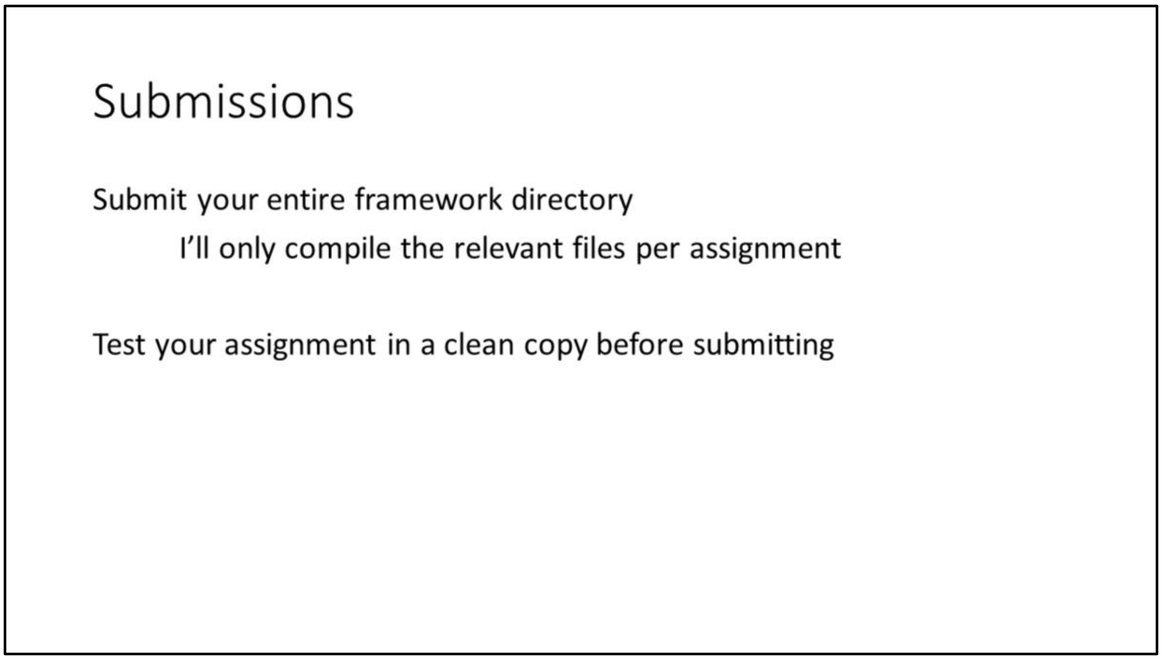
The framework

Using my framework is required

- Makes unit testing easier

Feel free to send me suggestions for improvements

I have provided a framework for all of the assignments. This framework is required, primarily to make grading easier. However, I've put a fair amount of work in to try to make sure it's a nice framework and it's easy to implement your assignments in (I implemented them all in this framework). While I doubt I can make any major improvements mid-semester, please talk/email me if you have any suggestions for improvement.



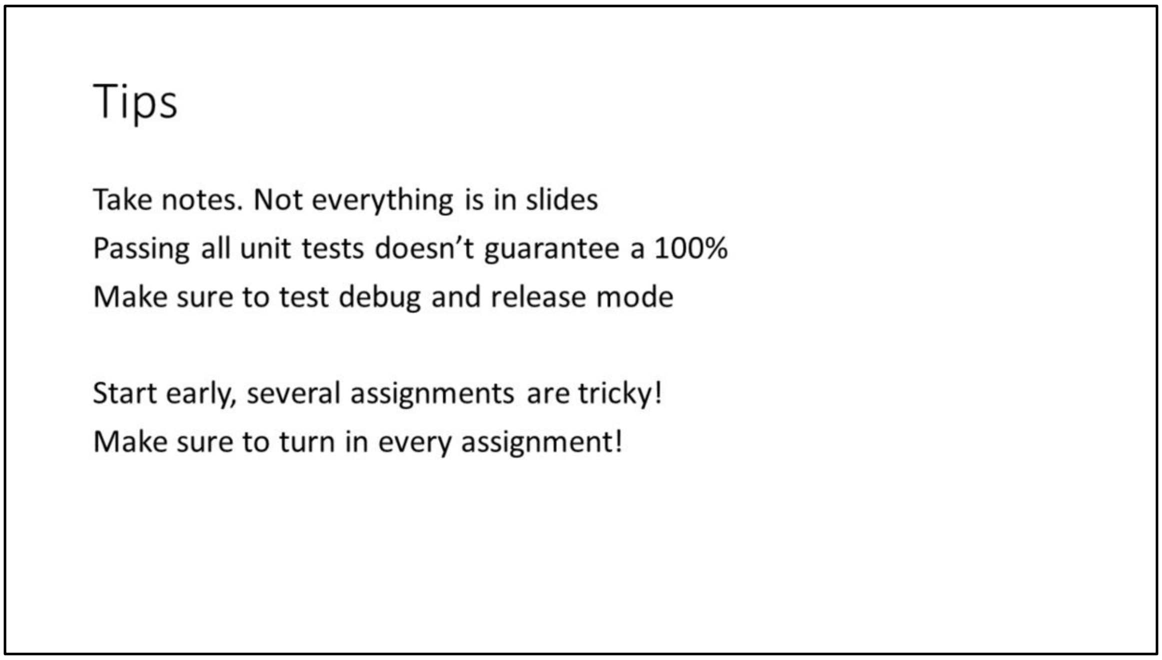
Submissions

Submit your entire framework directory

I'll only compile the relevant files per assignment

Test your assignment in a clean copy before submitting

When submitting for an assignment I only need the relevant files per assignment, however I want you to submit your entire project. This is in-case I need to verify your assignment works in your project if something goes wrong. Do your best to avoid adding code in non-designated areas so that I can compile them in my own project.



Tips

Take notes. Not everything is in slides

Passing all unit tests doesn't guarantee a 100%

Make sure to test debug and release mode

Start early, several assignments are tricky!

Make sure to turn in every assignment!

Now a few tips for passing this class.
1. Take notes! While I try to put a lot of information into my slides, not everything makes it in. Sometimes this is to avoid just giving you the answer and sometimes this is because making slides is hard.
2. While I give you a lot of unit tests to help you determine if something is wrong, passing all of them doesn't guarantee getting full points. Sometimes there are tests I haven't thought to write and sometimes there are more tests I will run behind the scenes. Make sure to think through the problem and anything that can go wrong.
3. Make sure to test debug and release mode. Sometimes people forget to test release which I run tests in. Also when it comes to debugging your assignments I may be forced to run debug mode.
4. This should go without saying but I have to say it anyways, start early! Many students have waited till the last minute thinking they'd easily get an assignment done to only realize they didn't understand the assignment. Starting early allows you to understand what you don't understand and ask questions. I even reward you for turning in an assignment 1 week early!
5. And finally, make sure to turn every assignment in. While my late policy isn't super harsh I will be firm about the 1 week deadline. Missing 1 assignment means -18% which makes passing the class a lot harder.