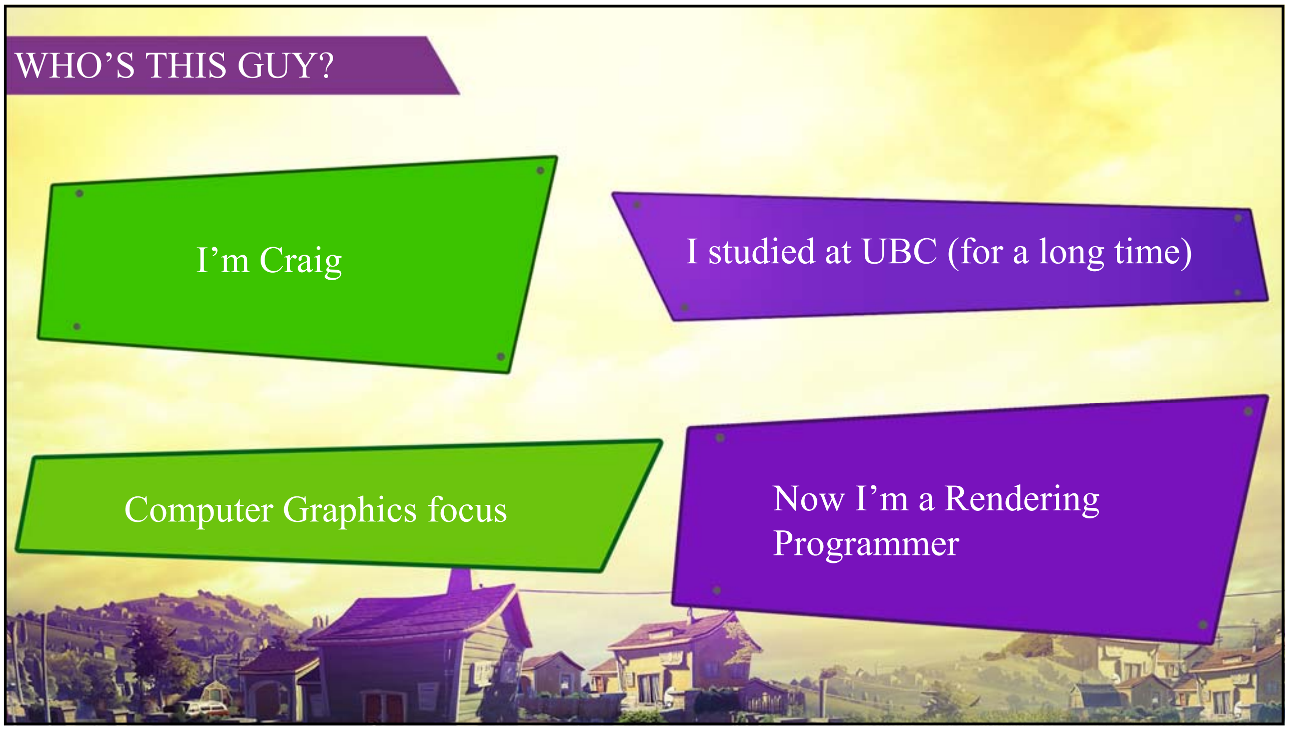
Videogame programming!

Craig Peters



WHO'S THIS GUY?

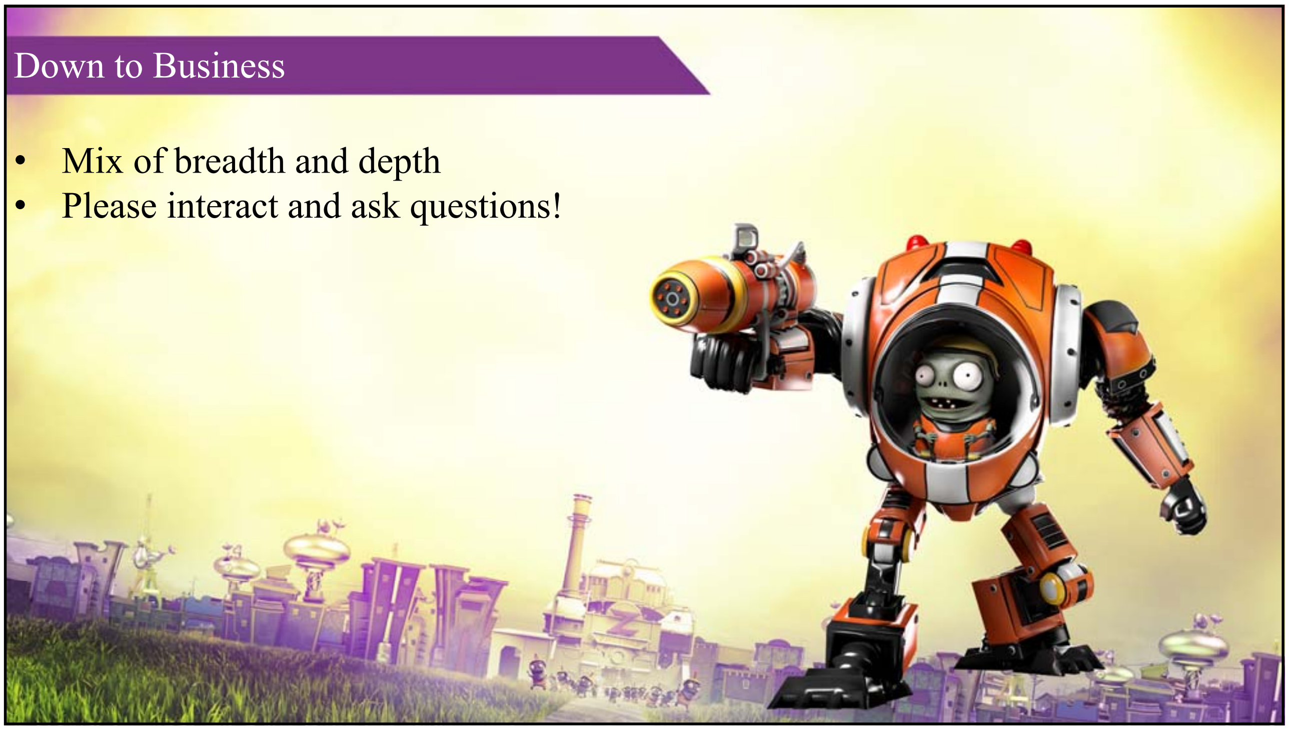
I'm Craig

I studied at UBC (for a long time)

Computer Graphics focus

Now I'm a Rendering Programmer

What Games Have You Made? [illegible]

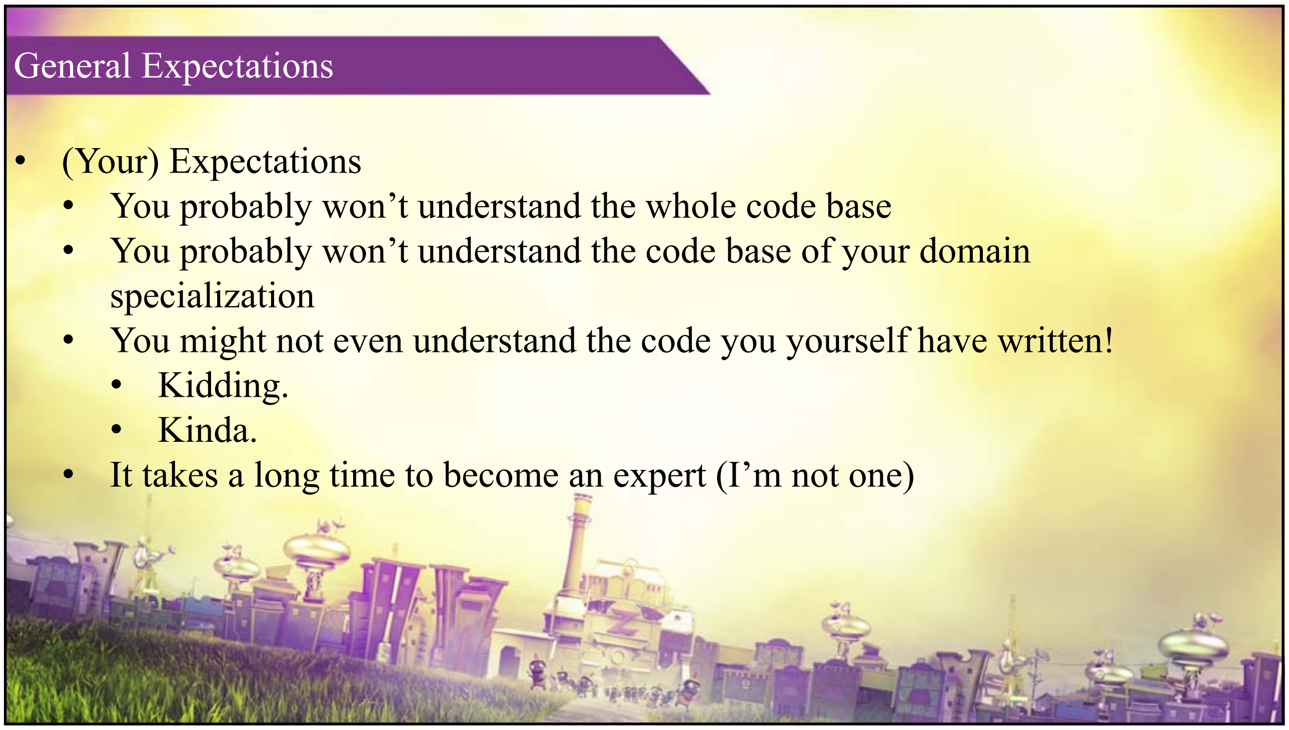


Down to Business

- Mix of breadth and depth
- Please interact and ask questions!

Noticeable Differences from Uni to Industry

- (Their) Expectations
- People!
- Priorities / Balance
- Scope, Scale, Complexity

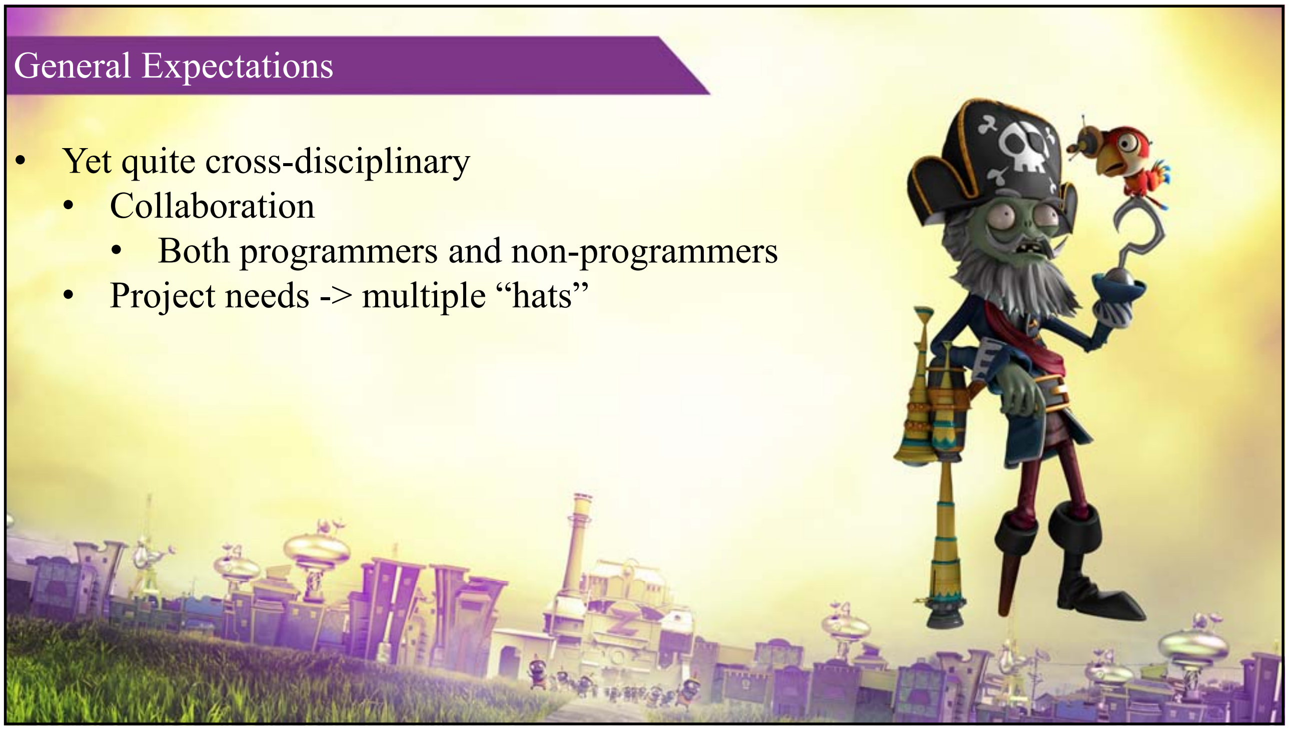


General Expectations

- (Your) Expectations
  - You probably won’t understand the whole code base
  - You probably won’t understand the code base of your domain specialization
  - You might not even understand the code you yourself have written!
    - Kidding.
    - Kinda.
  - It takes a long time to become an expert (I’m not one)

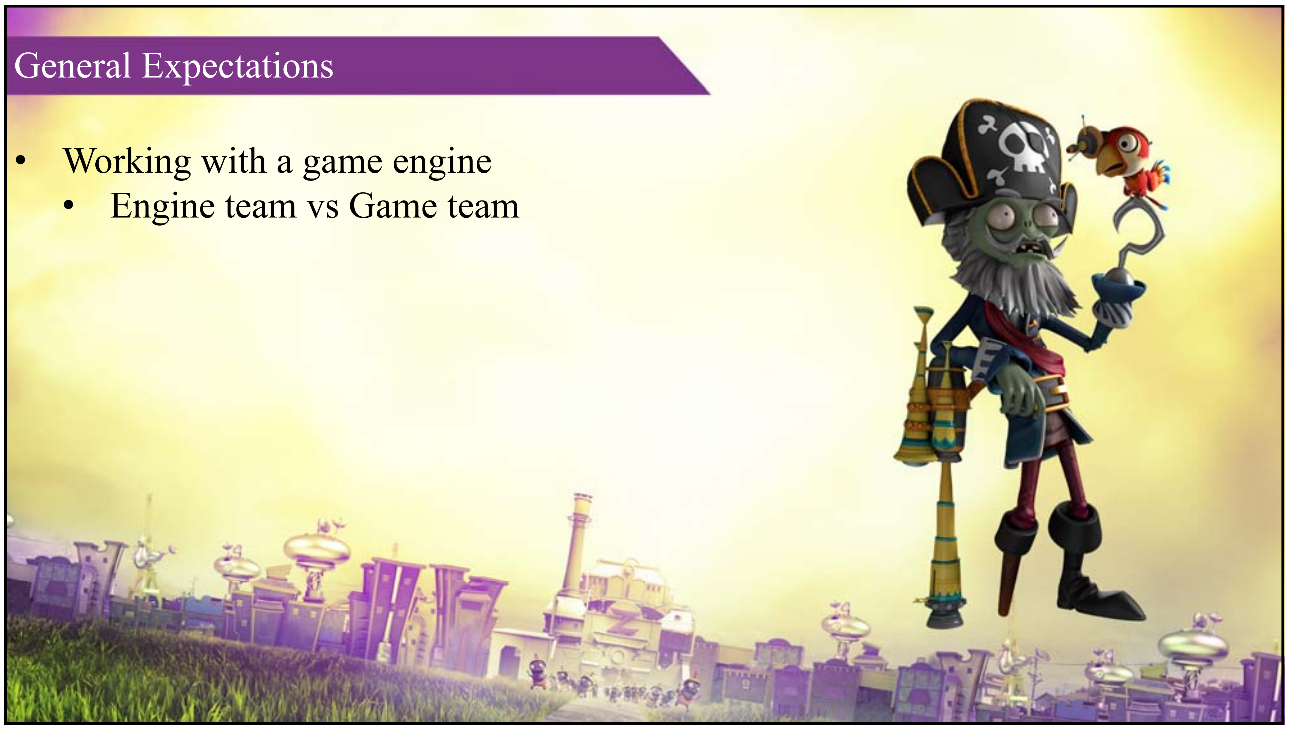
General Expectations

- Specialization Heavy
  - Animation
  - Rendering
  - AI
  - Systems
  - UI
  - Gameplay
  - Audio
  - Physics
  - Online
  - Quality Engineering



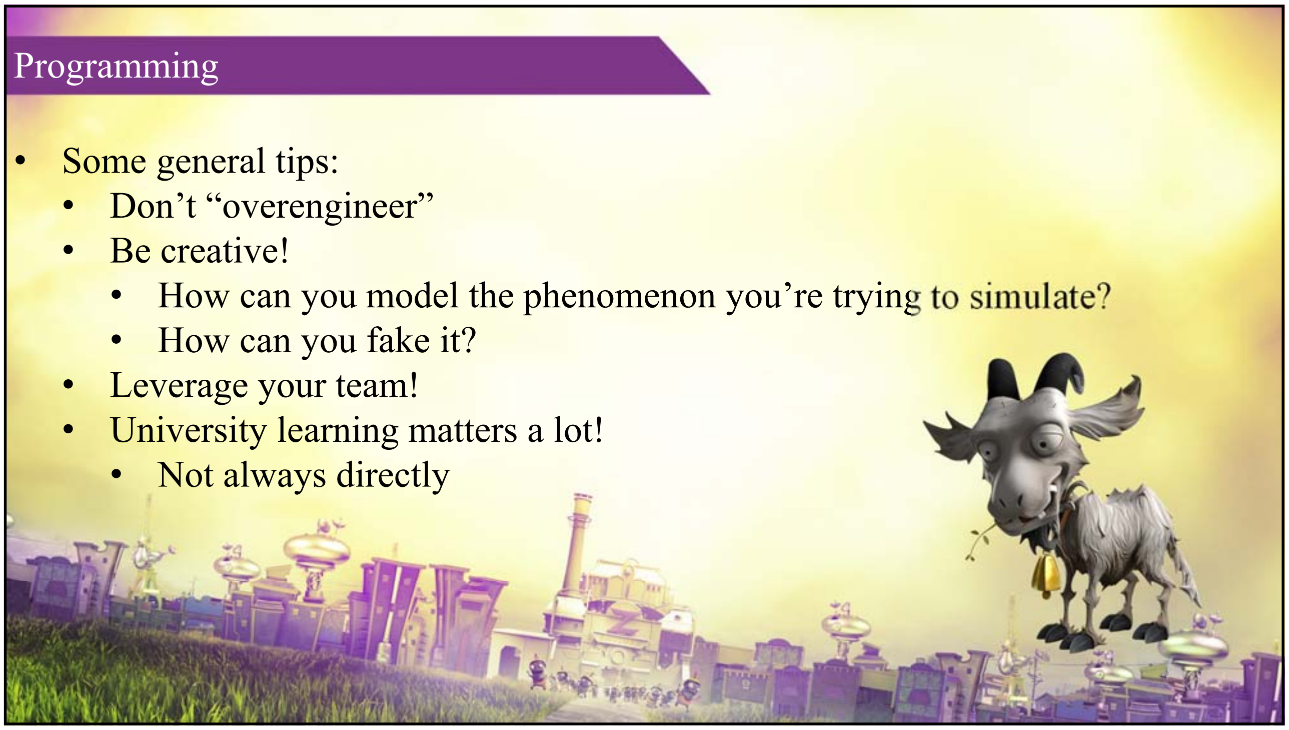
General Expectations

- Yet quite cross-disciplinary
  - Collaboration
    - Both programmers and non-programmers
  - Project needs -> multiple "hats"



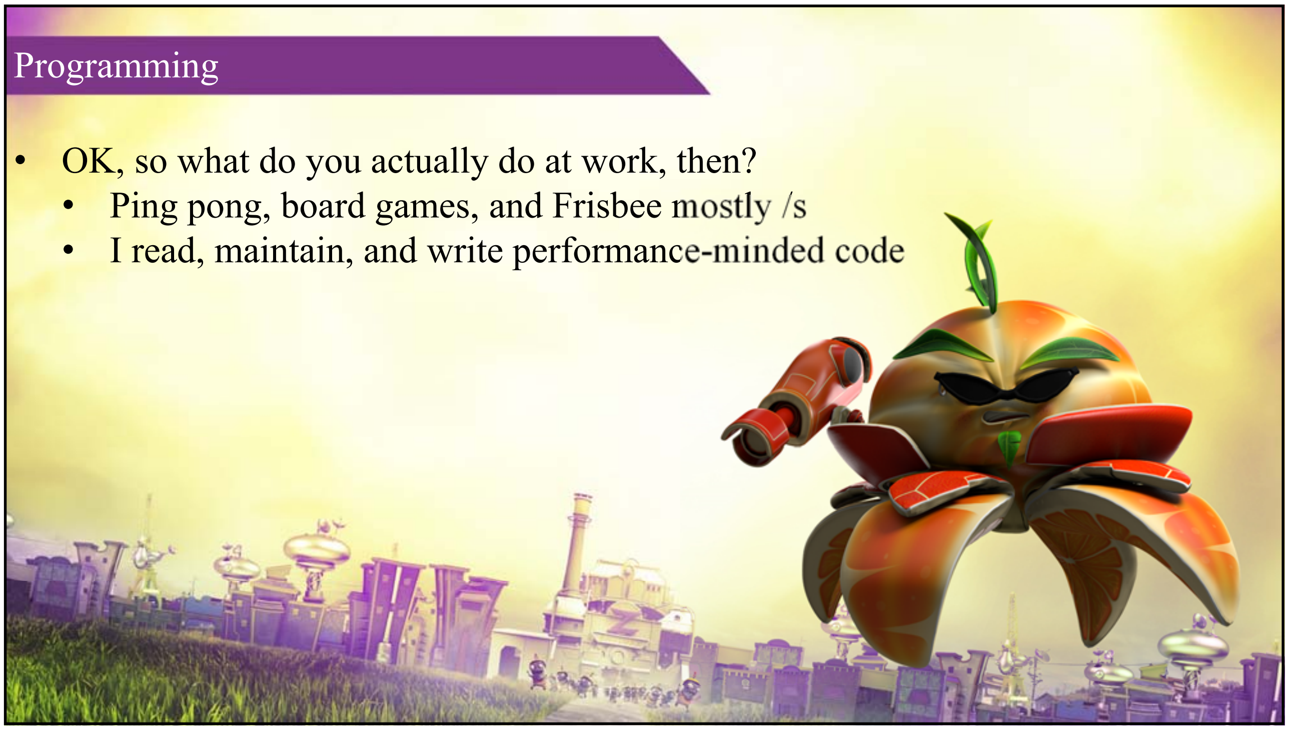
General Expectations

- Working with a game engine
  - Engine team vs Game team



Programming

- Some general tips:
  - Don't "overengineer"
  - Be creative!
    - How can you model the phenomenon you're trying to simulate?
    - How can you fake it?
  - Leverage your team!
  - University learning matters a lot!
    - Not always directly

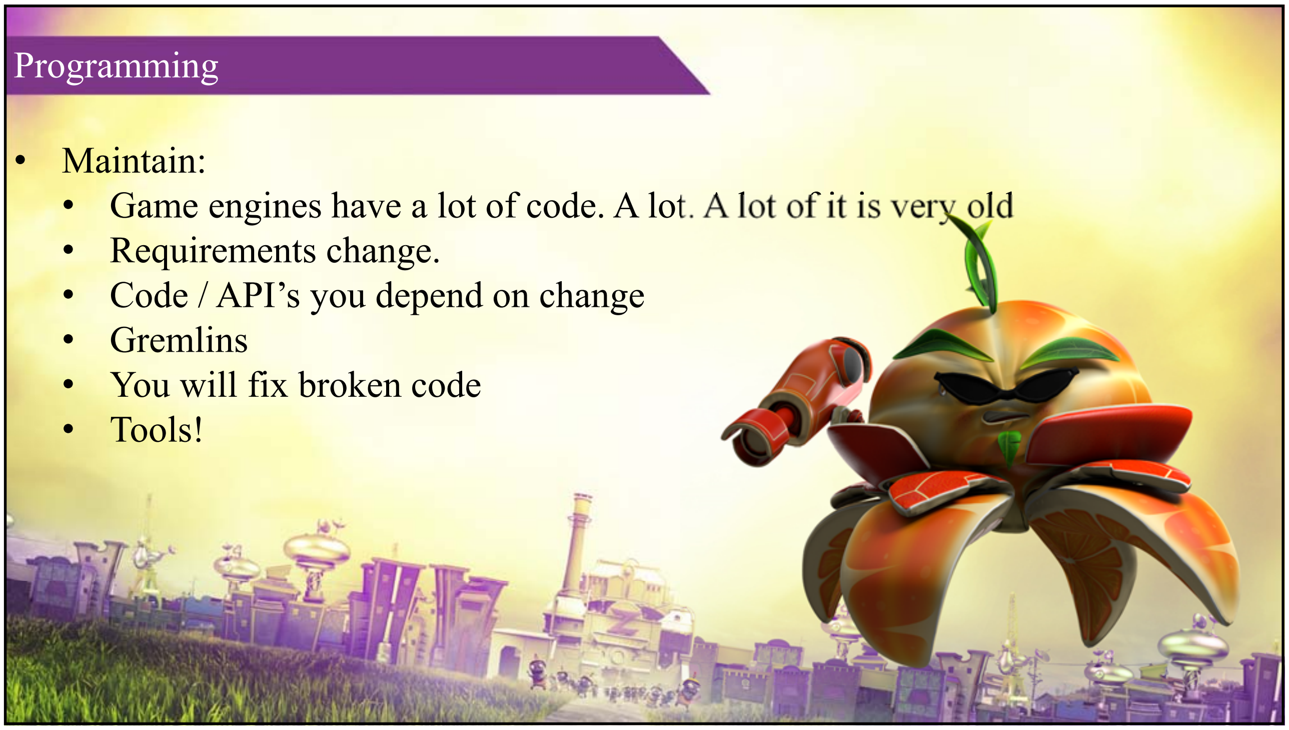


Programming

- OK, so what do you actually do at work, then?
  - Ping pong, board games, and Frisbee mostly /s
  - I read, maintain, and write performance-minded code

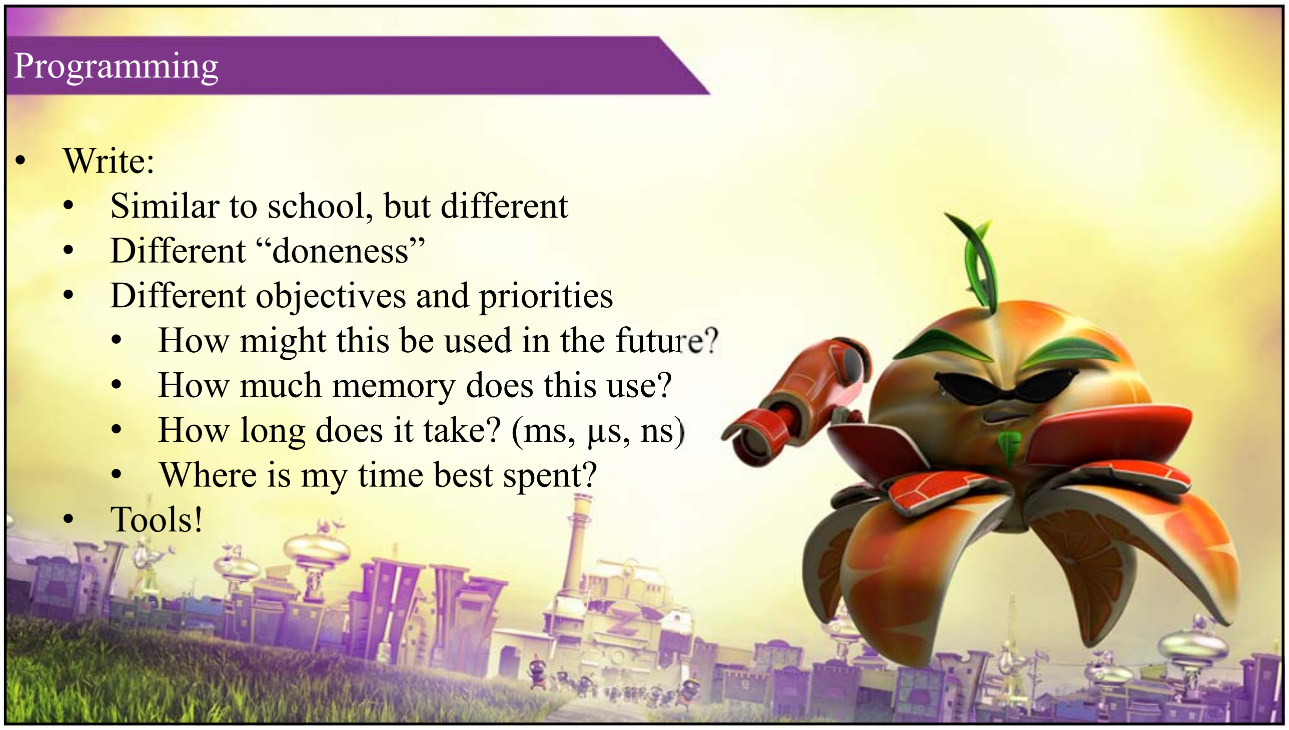
Programming

- Read:
  - Peer and Personal Review
  - Understand it before you change it



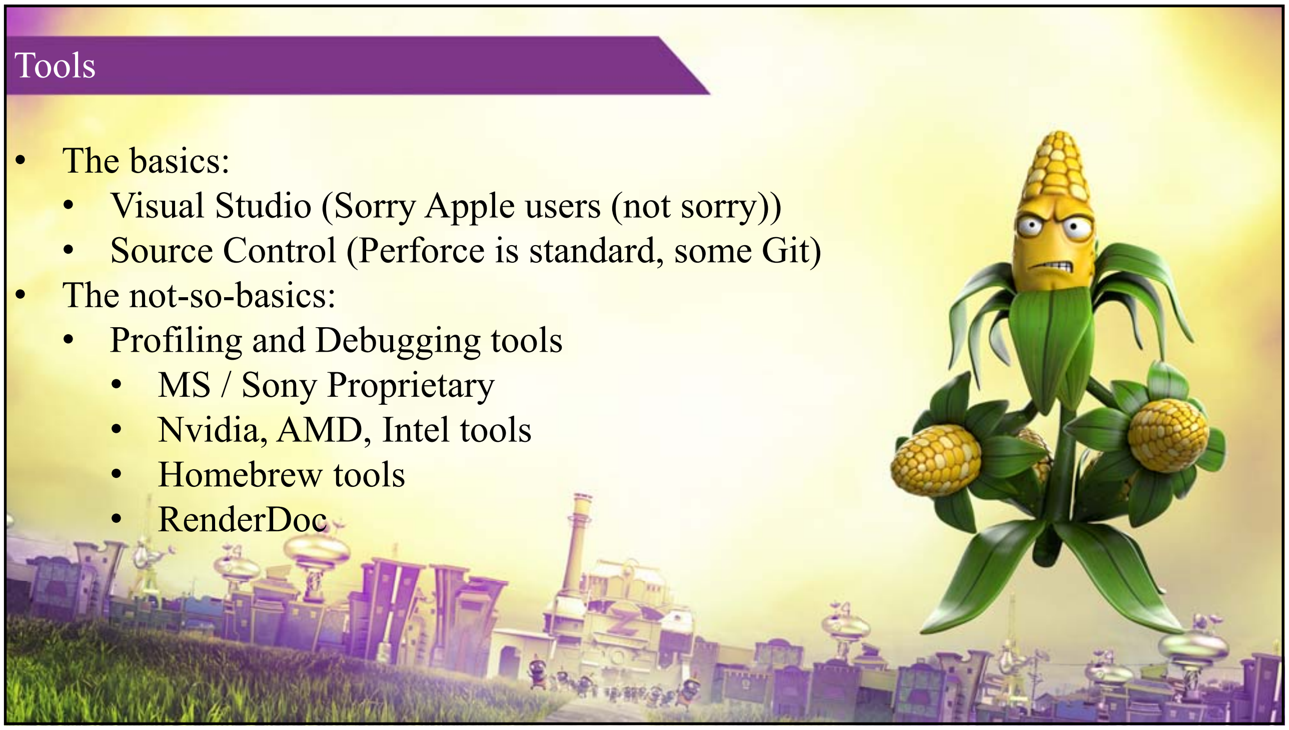
Programming

- Maintain:
  - Game engines have a lot of code. A lot. A lot of it is very old
  - Requirements change.
  - Code / API's you depend on change
  - Gremlins
  - You will fix broken code
  - Tools!



Programming

- Write:
  - Similar to school, but different
  - Different "doneness"
  - Different objectives and priorities
    - How might this be used in the future?
    - How much memory does this use?
    - How long does it take? (ms, µs, ns)
    - Where is my time best spent?
  - Tools!



Tools

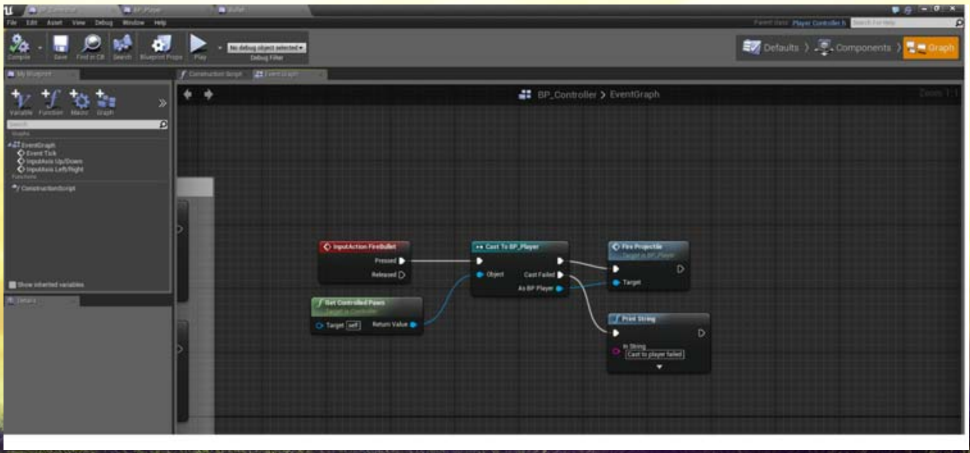
- The basics:
  - Visual Studio (Sorry Apple users (not sorry))
  - Source Control (Perforce is standard, some Git)
- The not-so-basics:
  - Profiling and Debugging tools
    - MS / Sony Proprietary
    - Nvidia, AMD, Intel tools
    - Homebrew tools
    - RenderDoc

RenderDoc

PIX

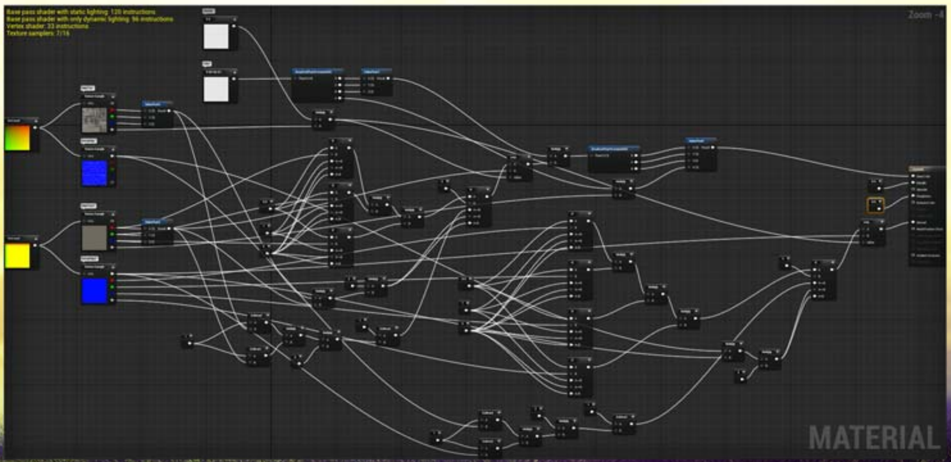
Cool Stuff

- Data Driven Design:
  - My team strives to put as much as possible into data – why?



Cool Stuff

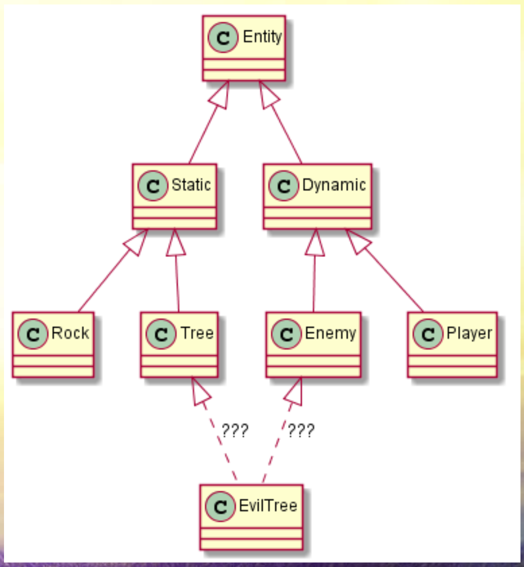
- Data Driven Design:
  - Things can get a little hairy



Base pass shader with static lighting: 120 instructions
Base pass shader with only dynamic lighting: 96 instructions
Vertex shader: 33 instructions
Texture samplers: 7/16

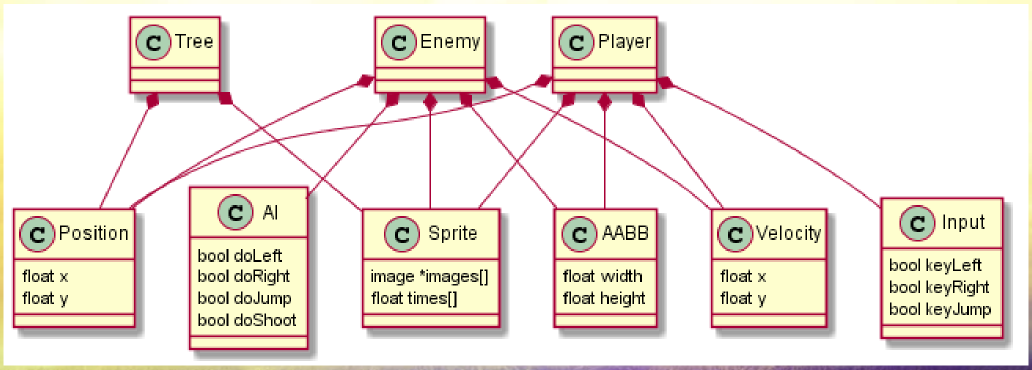
Cool Stuff

- Composition instead of Inheritance



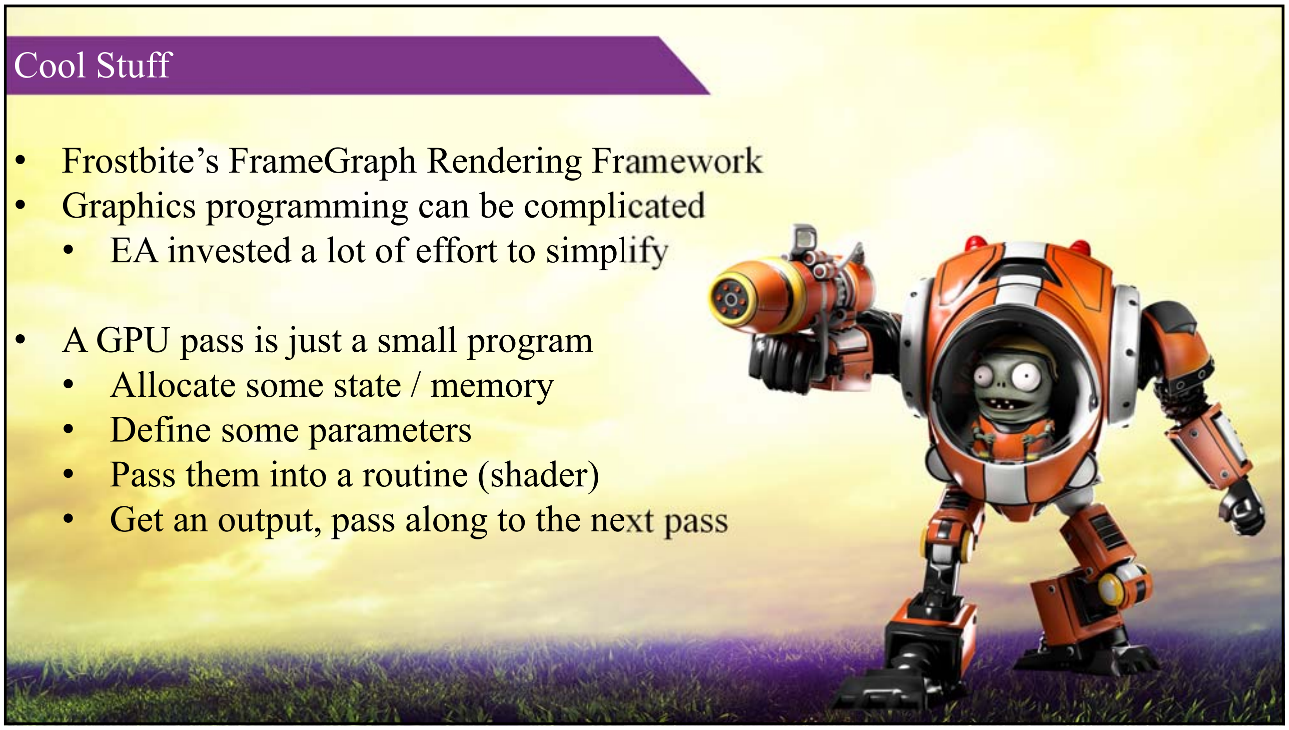
Cool Stuff

- Composition instead of Inheritance



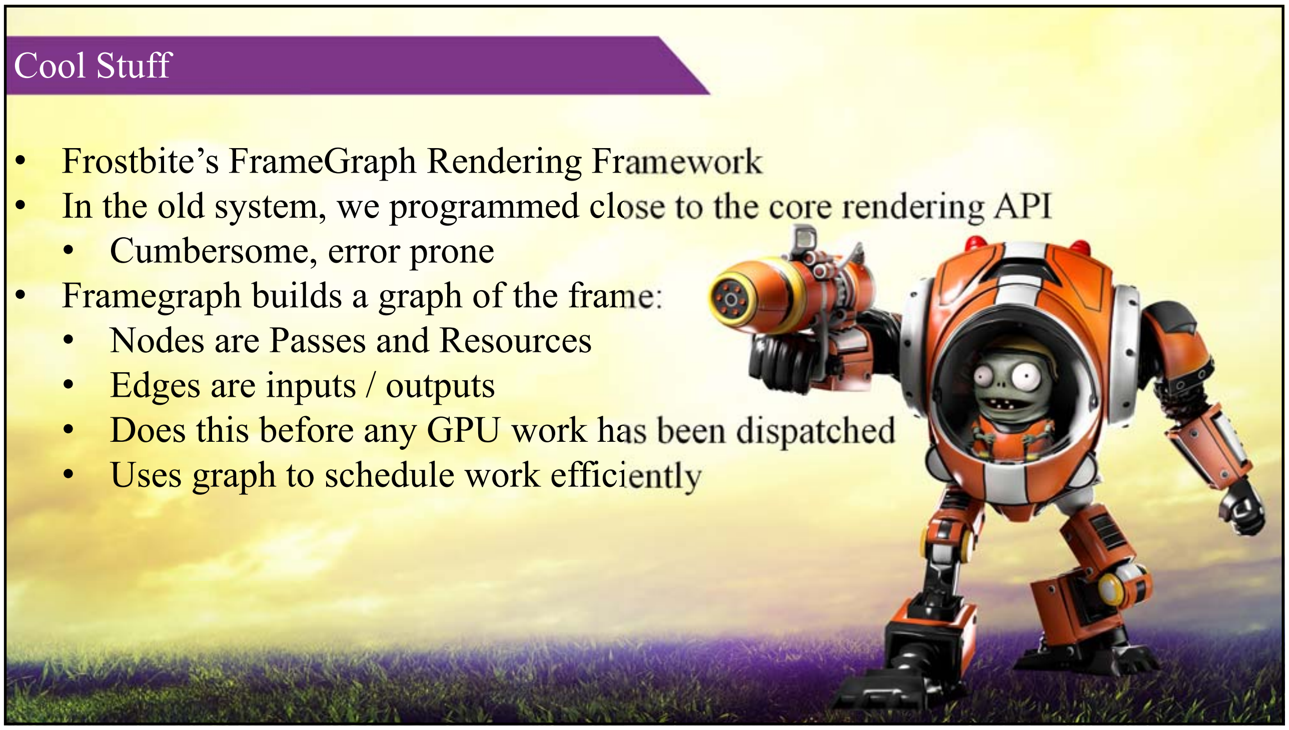
Cool Stuff

- Separation of simulation and presentation
  - Keeps renderers out of game code and vice versa
  - Enables easier parallelization
  - Crucial for things like Replays



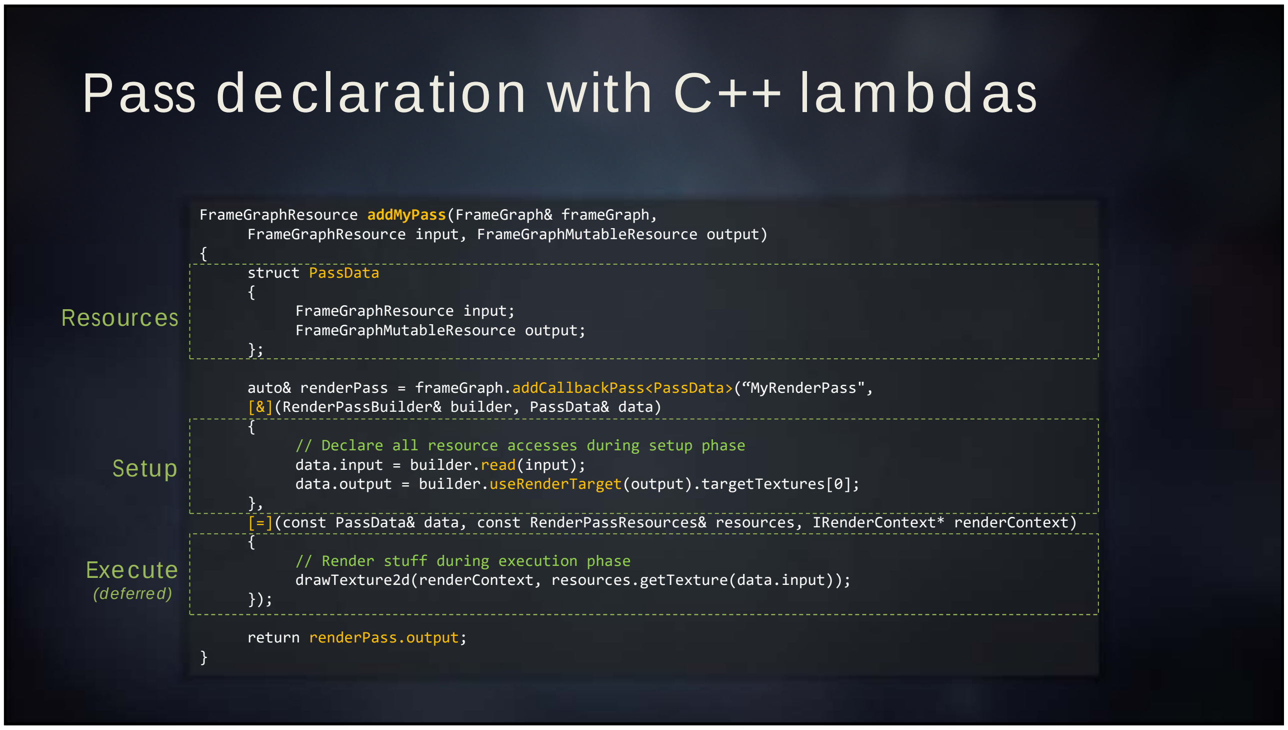
Cool Stuff

- Frostbite’s FrameGraph Rendering Framework
- Graphics programming can be complicated
  - EA invested a lot of effort to simplify
- A GPU pass is just a small program
  - Allocate some state / memory
  - Define some parameters
  - Pass them into a routine (shader)
  - Get an output, pass along to the next pass



Cool Stuff

- Frostbite’s FrameGraph Rendering Framework
- In the old system, we programmed close to the core rendering API
  - Cumbersome, error prone
- Framegraph builds a graph of the frame:
  - Nodes are Passes and Resources
  - Edges are inputs / outputs
  - Does this before any GPU work has been dispatched
  - Uses graph to schedule work efficiently



Pass declaration with C++ lambdas

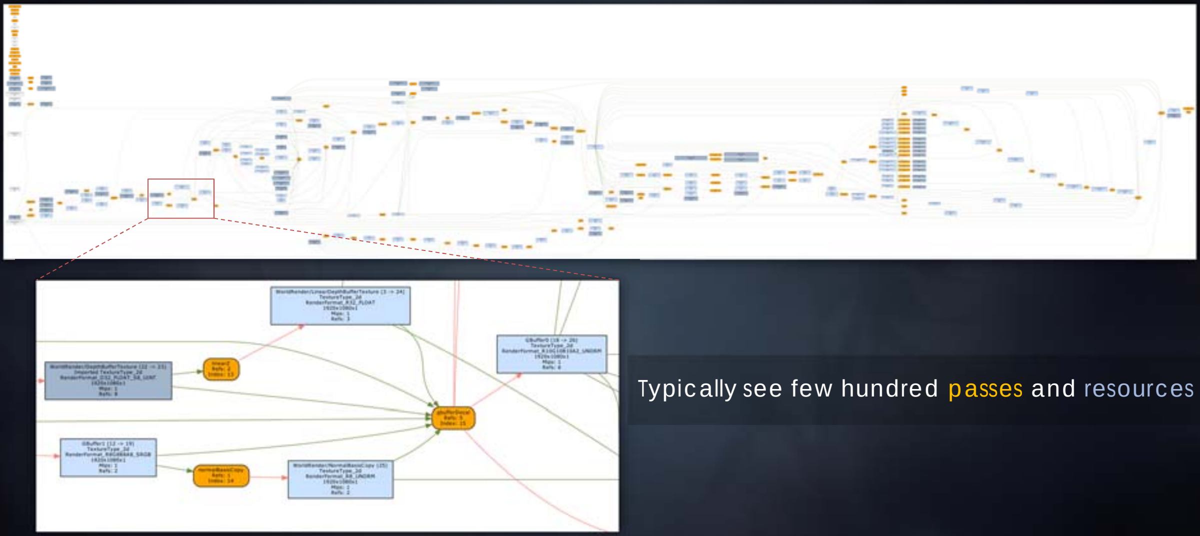
Resources

Setup

Execute
(deferred)

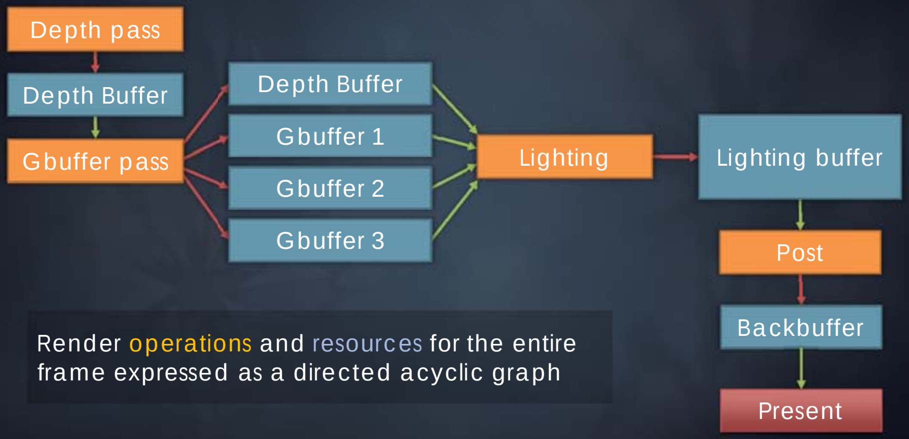
```
FrameGraphResource addMyPass(FrameGraph& frameGraph,
      FrameGraphResource input, FrameGraphMutableResource output)
{
      struct PassData
      {
            FrameGraphResource input;
            FrameGraphMutableResource output;
      };

      auto& renderPass = frameGraph.addCallbackPass<PassData>(“MyRenderPass",
      [&](RenderPassBuilder& builder, PassData& data)
      {
            // Declare all resource accesses during setup phase
            data.input = builder.read(input);
            data.output = builder.useRenderTarget(output).targetTextures[0];
      },
      [=](const PassData& data, const RenderPassResources& resources, IRenderContext* renderContext)
      {
            // Render stuff during execution phase
            drawTexture2d(renderContext, resources.getTexture(data.input));
      });

      return renderPass.output;
}
```

Graph of a Battlefield 4 frame



Typically see few hundred passes and resources

Frame Graph example



Render operations and resources for the entire frame expressed as a directed acyclic graph

Cool Stuff

- Frostbite’s FrameGraph Rendering Framework
- Why bother?
  - Simple API
  - Prune unused passes
  - Transient memory
  - Future – data driven?

Future?

- Cloud
- AI / Machine Learning
- Streaming

Interviews (last slide)

- Not about “getting the right answer”
- We want to see how you:
  - Think / problem-solve
  - Communicate
- Keep your chin up
- Know something you’ve done inside and out
  and talk about it!

Fin

Thanks!
Q&A

cpeters@ea.com

https://www.ea.com/careers

Fin

Campus Recruiter:
Brenna MacLean
brmaclean@ea.com
(mention req # in email)

cpeters@ea.com

https://www.ea.com/careers