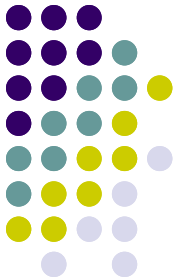
Original Pitch



Game Performance Optimization

Ed Keenan

November 3, 2009

In the Beginning…

There were Monkeys

Soon they began to learn

Use tools

Eventually Typewriters

Since they can type….

They could program

Code monkey was born

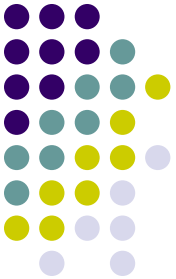
It’s not bad…

Code Monkeys get money

Imagine Ninjas

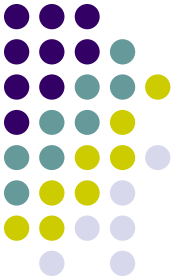
- A warrior specially trained in a variety of unorthodox arts of war.

Roles of Ninjas



- ***Sabotage***
  - Deliberate act of weakening the enemy
- ***Espionage***
  - Obtain information that is secret
- ***Scouting***
  - Exploring to gain information
- ***Assassination***
  - Targeted Killing
- ***Guerilla Warfare***
  - Uses ambush and mobility in attacking vulnerable targets

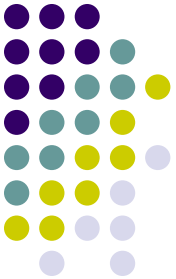
The Enemy



- ***Resource & Performance*** grabbing issues in ***Game Development***
- Using the Ninja duties
  - Gather recon on performance issues ***Scouting***
  - Disrupt old behavior through ***Sabotage***
  - Gather information through testing ***Espionage***
  - Eliminate performance spikes ***Assassination***
  - Refactor and attack weak points ***Guerilla Warfare***

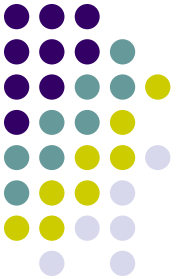
Code Ninja is born

The Class



- ***GAM 391/491***
  - ***Game Performance Optimization***
  - Cross-reference for graduate students
- Issue
  - Game performance and optimization is one of the ***MOST*** important issues that modern game console developers face today
- Goal
  - Introduce these problems, using real-world Game examples to understand and improve these issues.

Game Optimization



- Topics using actual System Game Code
  - Extended matrix instruction set
  - Dynamic memory usages
  - Increasing run-time systems to very large scale
  - C++ language enhancements and extensions
  - Streaming & File I/O
  - Profiling and metrics
- Large final project:
  - Refactor existing Particle System to improve performance and minimize resource usage

Become a Code Ninja

GAM 391/491 Topics
Game Performance Optimization

Game performance and optimization are one of the ***MOST*** important issues that modern game console developers face

- Topics using actual System Game Code
  - Extended matrix instruction set
  - Dynamic memory usages
  - Increasing run-time systems to very large scale
  - C++ language enhancements and extensions
  - Streaming & File I/O
  - Profiling and metrics
- Large final project:
  - Refactor existing Particle System to improve performance and minimize resource usage

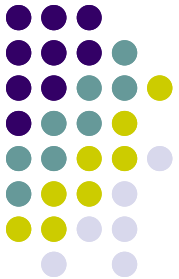
Become a Code Ninja

GAM 391/491 Topics
Game Performance Optimization

Prereqs: C++, Data Structures
Linear Algebra, Graphics, Hardware Knowledge is a plus but not req'd

Ed Keenan ekeenan2@cdm.depaul.edu
Former Executive Technology Director of Midway Games

Class Overview



Optimized C++

CSC 361 / CSC 461

Ed Keenan

9 January 2019

13.0.6.2.10 Mayan Long Count

Overview

- Contacts
- Philosophy
- Syllabus
- Details
- Software Development
- First Assignments

Contact

- Email
  - Use Piazza – private post
    - Last Resort
      - ekeenan2@cdm.depaul.edu
      - Do not forget the **2** in my name
  - 830 CDM
    - Gated community (offices behind locked doors)
    - Need to call for access (phone next to reception table)
  - Phone
    - 312-362-6747

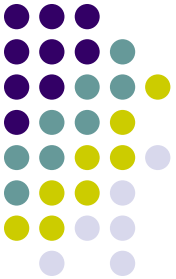
Background info on the Class?

- Why did you take this class?
- Is anyone scared or worried about this class?
- Any questions for me?
  - Bring it on.

I think…

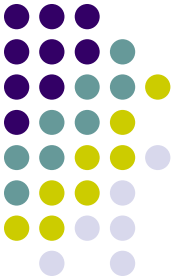
- Optimized software development is: specialized ***APPLIED*** software engineering
  - You can learn more if you
    - practice and experiment often
    - work individually
    - learn new material
    - iterate on previous work
    - compete with peers
    - take a class from Keenan

Interactive Class



- Participation is a Big part of Class
  - Expect to be involved
  - Not a passive Class
- It's a Programming Class
  - We **program**
  - We **PROGRAM**
  - We ***PROGRAM***

Interactive Class

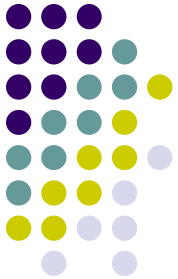


- We will have:
  - Code reviews
    - Oh yes…
    - ***Yum Yum***
  - Debate
  - Demonstrations
  - Discussions
  - Challenges
- We are all learning
  - It's OK to make mistakes.
  - The only mistake is ***NOT*** trying
- Remember:
  - I am a ***professional***

Prereqs

- Data Structures in Java or C++
  - CSC 301, CSC 383, CSC 393, CSC 403
- Computer Systems I
  - CSC 373, CSC 406

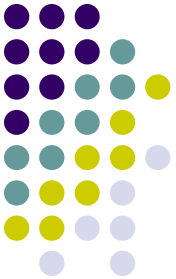
Syllabus



→ Syllabus

Behold its glory!

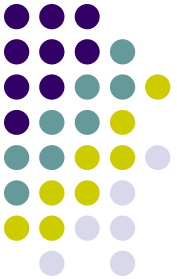
Philosophy



- Learning new material can be hard.
- People learn at different rates.
  - Your background or experience may be different.
- You may need to review
  - C++,
  - Math,
  - OS system libraries.

- THAT's OK.

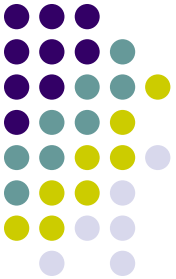
Philosophy cont.



- It’s the level of knowledge that you leave this class with, not the bump in the roads in the process.
  - What does that mean to you?

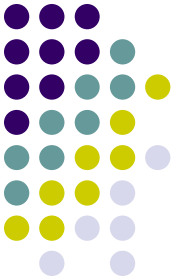
Extras:

Since I am a nice guy…



- Anyone expecting to interview for a coding job in the next 1-30 years?
  - I've interviewed over hundreds of people
  - Reviewed thousands resumes
  - I can help
- Books and material
  - I can recommend material to help you improve in software development
    - Architecture, design, language, project management and more

Season Programmers

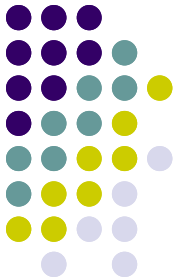


- Movie
  - Cadillac DTS

Thank You!

Did I mention it's a programming class?

Learn C++



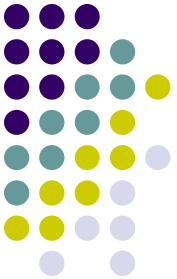
Optimized C++

Ed Keenan

Goals

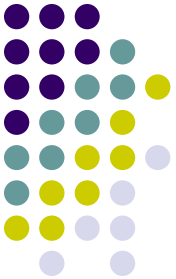
- Quick and Dirty C++ Bootcamp
  - Everyone knows Java
  - Everyone should know a little C (pointers)
- Let's get going fast

Quick and Dirty



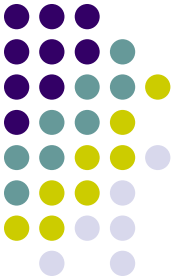
- Highlighting issues in C++
  - Look up in Reference Book
  - Start a separate thread for each item

Headers - Prototypes



- C++ separates prototypes from the body of the code.
  - Headers are processed in the preprocessor phase.
    - Effectively including every #include into the *.cpp file.
    - So it becomes a very large file that is processed
- Every function needs its prototype defined in the header.
  - Allows the body to be defined in a separate location
  - Separation of duties
- Final executable code does not carry any symbol information, it's more or less pure machine code.
  - Need a way to describe the interface of a piece of code, that is separate from the code itself.
  - Description is in the header file.

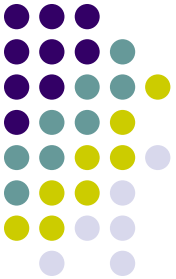
Headers - Guards



- Since headers can be included into any file
  - Headers can also be nested
    - Headers including other headers
  - Same prototype can be included multiple times
  - Header guards prevent this from happening
- Do not rely on pragmas
  - Use:

```
#ifndef  HEADER_NAME_H
#define HEADER_NAME_H
      // header goes here
#endif
```

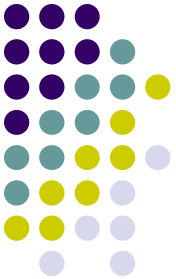
Scope of classes



- Access specifier keywords
  - public
  - protected
  - private
- Control the access of the methods and data
  - Defined in the header / prototype definition
- Can be used on individual methods or as a list

```
private void foo();
public:
        float getX();
        float getY();
```

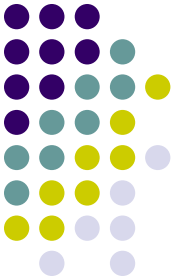
Access of inheritance



```
class A
{
public:
    int x;
protected:
    int y;
private:
    int z;
};
```

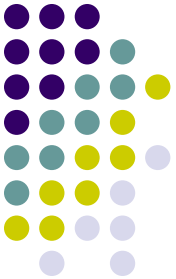
```
class B : public A
{
    // x is public
    // y is protected
    // z is not accessible from B
};

class C : protected A
{
    // x is protected
    // y is protected
    // z is not accessible from C
};

class D : private A    // 'private' is default for classes
{
    // x is private
    // y is private
    // z is not accessible from D
};
```

Big four



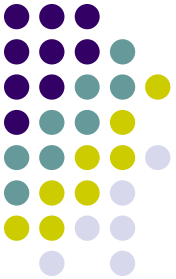
- Four functions are default created by the compiler
  - Default Constructor
    - ***Dog();***
  - Copy Constructor
    - ***Dog(const Dog &);***
  - Assignment Operator
    - ***Dog & operator = (const Dog &);***
  - Destructor
    - ***~Dog()***
- There is actually 6 with C++11 – more later in quarter

Big four – DEFINE them



- Do not EVER use the default implementations implicitly (Pick ONE)
  - Define them yourself
  - Specify that you want to use them
- Copy Constructor – example
  - Dog(const Dog &) = default;
  - Dog(const Dog &) = delete;
  - Dog(const Dog &) { // your implementation }

Heap vs Stack



- You can instantiate a class on the Stack or on the Heap
  - Determines who owns the memory
  - Different responsibility for the programmer

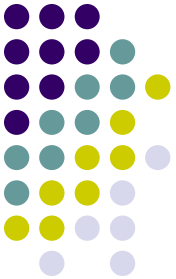
- Stack

```
void foo()
{
   Dog fido;
   fido.setX(5);
}
```

- Heap

```
void foo()
{
   Dog *pDog = new Dog();
   pDog->setX(5);
}
```

Value, Reference, Pointer



- Prototype:

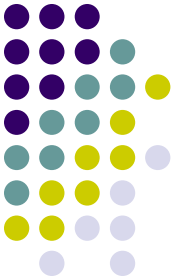
```
void foo( Dog dog )
```

- Calling:

```
Dog fido;
foo(fido);
```

- Discussion:
  - If Dog is 1000 Bytes, then 1000 Bytes are copied to foo function.
  - If fido is modified in foo, no effect in the calling function

Value, Reference, Pointer



- Prototype:

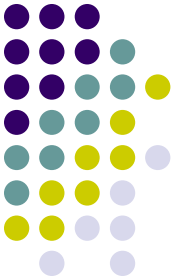
```
void foo( Dog *dog )
```

- Calling:

```
Dog fido;
foo(&fido);
```

- Discussion:
  - If Dog is 1000 Bytes, only the pointer is copied 4 bytes to the foo function.
  - If fido is modified in foo, it changes the value in the calling function

Value, Reference, Pointer



- Prototype:

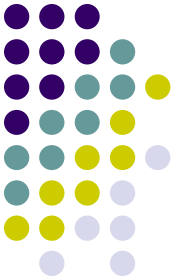
```
void foo( Dog &dog )
```

- Calling:

```
Dog fido;
foo(fido);
```

- Discussion:
  - If Dog is 1000 Bytes, only the reference(pointer) is copied 4 bytes to the foo function.
  - If fido is modified in foo, it changes the value in the calling function

References are Pointers



- Pointer are references, References are pointers
  - That's the truth (same in Java and C#)
- References are:

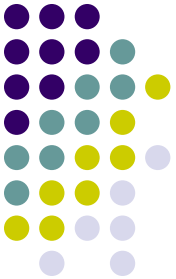
```
Dog & R
```

  is the same as

```
Dog * const P
```

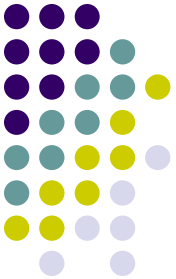
- Difference
  - Difference is that R is guaranteed to be pointing to a Dog object, where pointer P may not be pointing to a Dog.
  - With pointers you can change P, but references are constant pointers that prevent the ability to change the address.
  - Syntax sugar on accessing.
    - references uses:
      - (dot) instead of `->` (arrow)

Printf



- Learn and embrace printf()
  - Its faster and easier
  - *Do not* use ***cout()***
- Even Java now has a printf()
- Formatting contest
  - Demo

Miscellaneous



- Const
- Defines
- Preprocessor
- No protection Arrays
- Memory Leak
- 3 types of inheritance, not only public (java/c#)
- Virtual, abstract, override, final
- No interfaces
- Null is not null, its 0

Thank You!

- Easy?