Final Project

Overview

- Today: a bunch of final project topics
  - Scripting
  - Level editor
  - Particles
  - Portals
- Last three lectures each cover a major final project topic
  - Networking
  - Advanced rendering
  - Game object systems

Scripting

- What is scripting
  - Using a scripting language
    - High-level & interpreted from source or bytecode
- Scripting Benefits
  - Easy to tweak logic (constants/AI)
  - Faster iteration time (no recompiling)
  - Users can mod original game (community)
  - In-game console
- Scripting Drawbacks
  - Interpreted code is slower
  - Connecting scripts to native objects needs more code

Scripting Languages

- Lua (World of Warcraft, tons of games...)
  - *Really* fast JIT
  - By far the most common choice
- UnrealScript (Unreal Tournament, Gears of War)
  - Integrated into the Unreal Development Kit
- Stackless Python (Eve Online)
  - Many parallel tasks (microthreads)
- Others: Squirrel (Portal 2, L4D), Galaxy (SC2)

Scripting Use Cases

- Scripted callbacks / event handlers
  - Customizable functionality in user-supplied functions
  - Callbacks implemented in native language or script
  - Event handlers respond to game or engine events
- Extending game objects
  - New objects can be written entirely in script
  - May use inheritance or composition
- Script-driven engine systems
  - Reverses relationship between native and scripting language
  - e.g. Panda3D engine written in Python, C++ parts treated like a library

Case Study: Lua

- Compact
  - Small set of libraries
  - Interpreter and standard libraries are under 1mb
- Portable
  - Written entirely in C
  - Runs on desktop devices, mobile devices, embedded microprocessors, etc.
- Easy to embed in C/C++
- Very popular language for games
  - Crysis, Civilization V, World of Warcraft, etc.

Lua: hello.cpp

```
extern "C" {
  #include "lua.h"
  #include "lualib.h"
  #include "lauxlib.h"
}

int main() {
  lua_State *L = lua_open(); // initialize Lua
  luaL_openlibs(L); // set up Lua libraries
  luaL_dofile(L, "hello.lua"); // run "hello.lua"
  lua_close(L); // close Lua
  return 0;
}
```

Lua: hello.lua

```
function hello()
  return "hello there"
end

print(hello())
```

- Recommended Lua tutorials
  - http://luatut.com/
  - http://www.lua.org/pil/ (*Programming in Lua*)

Lua: Engine Integration

- Many existing libraries for more complicated scripting scenarios
- Luabind
  - Template-based C++ library for Lua bindings (like boost::python)
  - Easily import and export functions and classes

```
class Foo {
  void func() {}
};

module(L) [
  class_<Foo>("Foo").def("func", &Foo::func)
];
```

Lua in World of Warcraft

- Used to develop add-ons for UI customization
- Add-ons consist of three parts
  - .toc file: table of contents
  - .xml files: contain UI elements
  - .lua scripts: respond to events
- Handling events
  - Create a frame widget
  - Register to handle certain events
  - Implement OnEvent() handlers for those events

Lua in World of Warcraft

- Simple example
  - Prints "Hello PLAYER_ENTERING_WORLD" to the chat frame when a character zones into a world

```
-- Create the frame to handle the event, register the event
local frame = CreateFrame("FRAME", "AddonFrame")
frame:RegisterEvent("PLAYER_ENTERING_WORLD")

-- Define a custom event handler
local function eventHandler(self, event, ...)
  print("Hello " .. event)
end

-- Set the OnEvent() method to call our event handler
frame:SetScript("OnEvent", eventHandler)
```

Triggers

- Sends a notification when a scenario occurs
  - If player within 5 meters: closeDoor()
  - Generate events handled by an event handler
- May be treated like standard game objects
  - When player collides with trigger: closeDoor()
- Triggers are a perfect match for scripting
  - Events and event handling logic is decoupled
- Implementation strategies
  - Polling
  - Callbacks
  - Signals and slots

Triggers: Signals and Slots

- Implementation of the *observer pattern*
- Trigger generates a *signal*, event handling code in a *slot*
    - Signal and slot need to be connected beforehand
- Implemented in Qt and Boost

Level Editor

- Don't reinvent
  - Rely on existing 3D modeling and rendering packages (3DS, Maya, UnrealEd, Blender)
  - Only provide "glue" to put pieces together
  - Users will import content and use editor to tweak and annotate for game use
- Pick common file formats for exporting
  - May need to use other tools to transform data
  - XML, JSON, OBJ, ...
- Be nice to users
  - Undo/redo is much easier if you design for it first
  - Use a framework designed for UI (e.g. Qt)

Level Formats

- Don't have to use XML!
- Google Protocol Buffers
  - Fast binary format
  - Compile a schema to C++ serialization code
  - Automatically backwards compatible when changed
- JSON
  - Text-based format
  - JsonCpp: A very nice C++ API:

```
Json::Value json;
Json::Reader().parse(file, json);
float x = json["player"]["x"].asFloat();
```

Level Editor: Command Pattern

- Command objects encapsulate modifications to a document (similar to a transaction)
  - Used by Qt (QCommand, QUndoStack)

```
struct InsertCommand : Command {
  Document &doc;
  char c;
  int i;

  void redo() { doc.text.insertAt(i, c); }
  void undo() { doc.text.removeAt(i); }
};
```

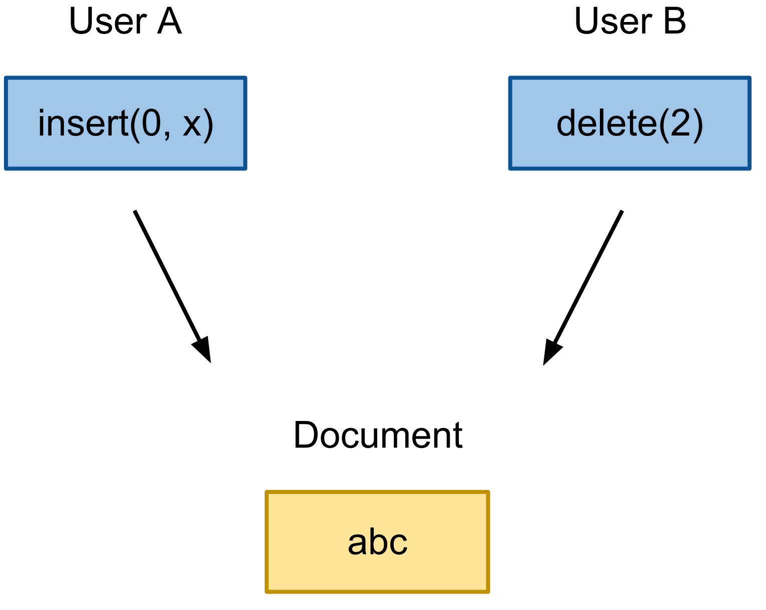
Level Editor: Command Pattern

- Operations on document use commands

```
struct Document {
  stack<Command> stack;
  string text;
  void insert(int i, char c) {
    stack.push(new InsertCommand(i, c, this));
    stack.top().redo();
  }
  void undo() {
    stack.top().undo();
    stack.pop();
  }
};
```

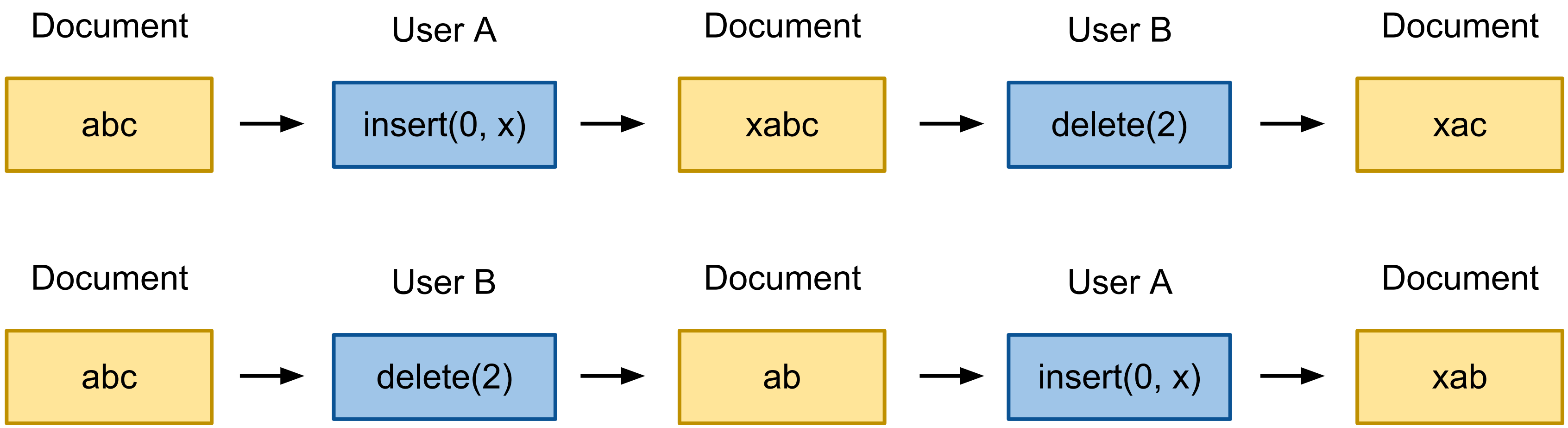
Level Editor: Concurrent Editing

- Consider a document and two simultaneous operations by different users



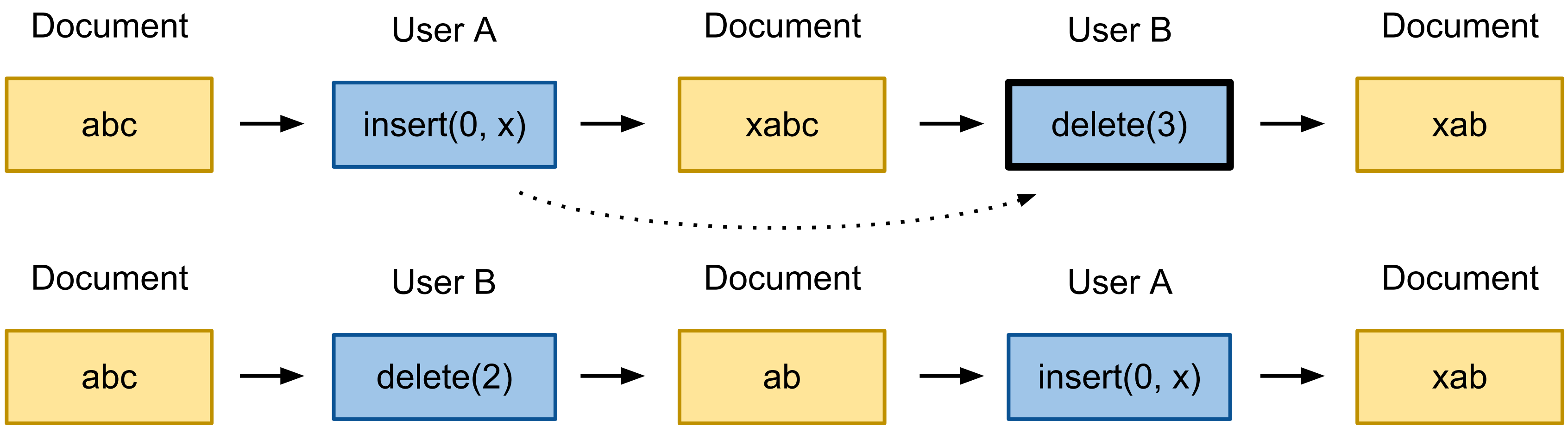
Level Editor: Concurrent Editing

- Order of execution gives different results



Level Editor: Concurrent Editing

- Operational transformation
  - Transform each operation by previous operations to achieve same effect regardless of execution order
  - Works well with command pattern



Particles

- Simulation of volumetric effects
  - Often undergo forces / damping / collisions
  - Parametric equations or force integration
- Each particle has randomized properties
  - Particles achieve a global effect together

Type 1: Billboarded Particles

- Oriented to always face the camera
  - Can read camera parameters directly from OpenGL
  - Good for: smoke, fog, glow, ...

```
float m[16];
glGetFloatv(GL_MODELVIEW_MATRIX, m);
Vector3 dx = Vector3(m[0], m[4], m[8]); // left-right
Vector3 dy = Vector3(m[1], m[5], m[9]); // up-down
Vector3 dz = Vector3(m[2], m[6], m[10]); // front-back
glBegin(GL_QUADS);
glVertex3f(-dx.x, -dx.y, -dx.z);
glVertex3f(-dy.x, -dy.y, -dy.z);
glVertex3f(dx.x, dx.y, dx.z);
glVertex3f(dy.x, dy.y, dy.z);
glEnd();
```

Type 2: Axis-Aligned Particles

- Rotates around an axis to face the camera
  - Use cross product to orient particle
  - Good for: lasers, sparks, trails, ...

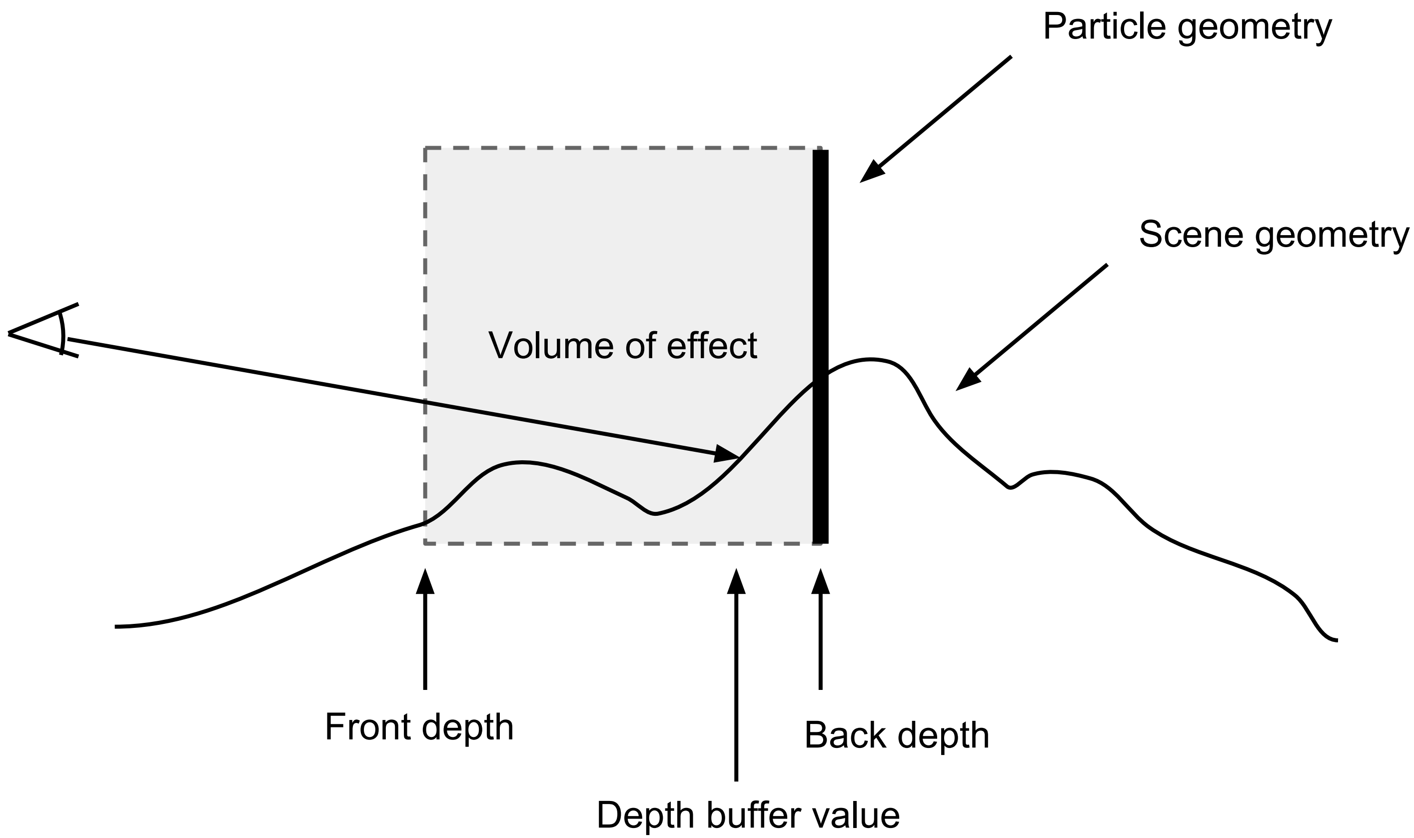
```
Vector3 a, b; // the end points of the axis
Vector3 eye; // the eye of the camera
Vector3 d = (b - a).cross(eye - a).unit();
glBegin(GL_QUADS);
glVertex3fv((a - d).xyz);
glVertex3fv((a + d).xyz);
glVertex3fv((b + d).xyz);
glVertex3fv((b - d).xyz);
glEnd();
```

Type 3: Soft Particles

- Higher quality billboarded particles
  - Removes sharp line where particle intersects geometry
  - Implemented in a shader
  - Reads from the depth buffer (needs separate pass)

Type 3: Soft Particles

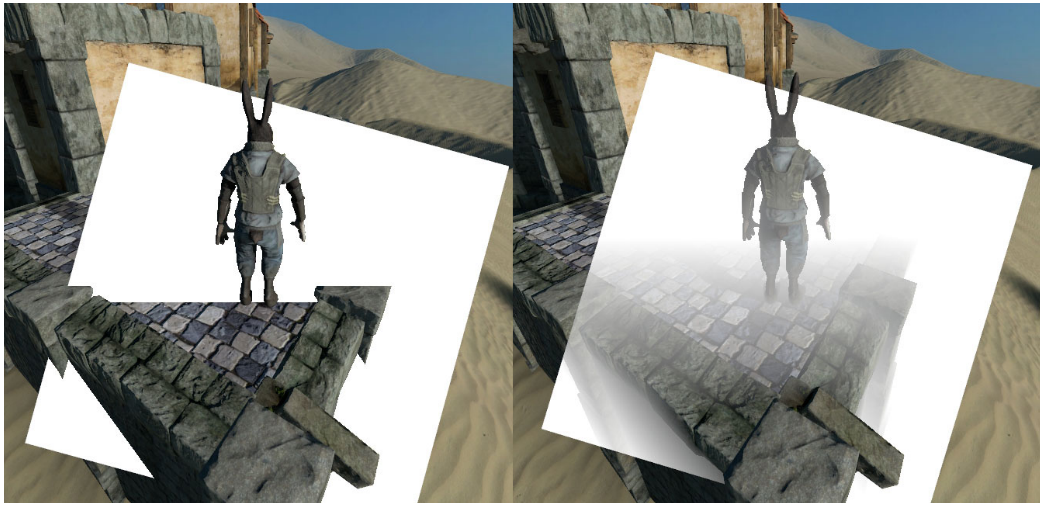
- Linearly interpolate alpha from front to back
  - alpha = (depth - front) / (back - front)



Type 3: Soft Particles

- Linearly interpolate alpha from front to back

```
float depth; // value from depth buffer
float front, back; // front and back particle depths
float alpha = clamp((min(back, depth) - max(0.0, front))
    / (back - front), 0.0, 1.0);
```



Rendering Particles

- Sorting
- Overdraw
  - Number of times a pixel is rendered to
  - Use richer particles (flipbook texture on one particle)
  - Render particles at half screen resolution and upscale on top of screen
    - Need to be careful about depth discontinuities
  - Use opaque particles that modify depth buffer
    - Pixels behind opaque particles will be skipped

Implementing Portals

Implementing Portals

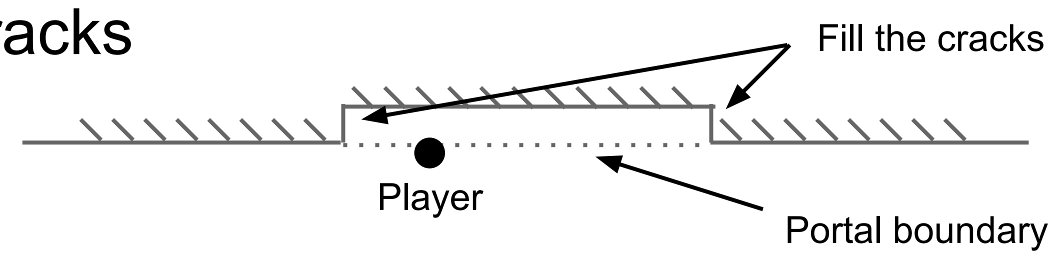
- Need two polygons marked as portals
  - Need matrix transform between them
  - $M_{AB} = M_A^{-1} \cdot M_B$
- Player needs separate coordinate system
  - Player may flip sideways or upside down
  - May want to rotate player upright over time
- When player center crosses portal boundary
  - Transform player center and velocity by M_{AB} or M_{BA}

Implementing Portals

- Problem: collisions
  - Player won't be able to enter portal on wall
  - Player will be touch wall and stop
  - Solution: portal uses "negative" collision volume, allows players to pass through it like a tunnel
- Problem: duplication
  - Player halfway through portal in two places at once
  - Solution: temporarily add player to world twice, once at each location

Rendering Portals

- Need to render room on other side of portal
  - Restrict drawing to inside portal using stencil buffer
  - Clear depth buffer
  - Draw the other side using stencil clipping
  - Recursively draw portals to fixed maximum depth
- Be careful about near clipping plane
  - Player going through portal will intersect portal polygon
  - Solution: back the portal into the wall and fill the cracks



Final Project: This Week

- Figure out detailed engine structure
  - Can share code from cs195u with group members
  - Start with one of your engines or from scratch
- Stub out major engine features
  - Stub out all public functionality
  - Integrate all features into one engine

Final Project: Important Dates

- 4/13: Engine components stubbed out
  - Integrated into one project
- 4/27: Engine features completed
  - Along with initial version of gameplay
  - Playtesting in class
- 5/4: Public playtesting deadline
  - 10+ playtesters per group member
- 5/11: Final game deadline
- TBD: Final showcase

C++ Tip of the Week

- Member pointers
  - Extra kind of pointer that no one knows about!
  - Pointer to non-static member (variable or function)

```
struct Foo {
  int num;
  int func() { return num; }
};

int Foo::*pNum = &Foo::num;
int (Foo::*pFunc)() = &Foo::func;
```

C++ Tip of the Week

- Member pointers
  - Use operators `.*` and `->*` to access the pointed-to member on a specific object

```
Foo foo;
foo.*pNum = 1;
cout << (foo.*pFunc)() << endl;

Foo *pFoo = &foo;
pFoo->*pNum = 2;
cout << (pFoo->*pFunc)() << endl;
```

C++ Tip of the Week

- Pointer to member functions are different
  - Implicit special this parameter (passed in the ecx register in Microsoft's compilers)
  - Won't work when casted to regular function pointer
- Also problems with multiple inheritance
  - Object has different pointer for each base
  - Don't know what `this` is until call time
- Member function pointer needs to store:
  - Address of the function body
  - Offset of the correct base (multiple inheritance)
  - Index of another offset that is stored the vtable (virtual inheritance)

C++ Tip of the Week

- Size of member function pointers
  - Often 2-3x the size of a regular function pointer
  - Casting may change the size of the pointer!

```
struct A {};
struct B : virtual A {};
struct C {};
struct D : A, C {};

cout << sizeof(void (A::*)()) << endl;
cout << sizeof(void (B::*)()) << endl;
cout << sizeof(void (D::*)()) << endl;

// 32-bit Visual C++ 2008:  A=4, B=8, D=12
// 32-bit GCC 4.2.1:        A=8, B=8, D=8
// 32-bit Digital Mars C++: A=4, B=4, D=4
```

Platformer Playtesting!

References

Scripting:

Gregory, Jason. Game Engine Architecture. pp 803-815. AK Peters Ltd, Natick, MA.

http://www.gandogames.com/2010/12/game-event-handling-part-1/

http://www.gandogames.com/2010/12/game-event-handling-%E2%80%93-part-2/

http://www.wowwiki.com/Handling_events

Particles:

http://www.2ld.de/gdc2007/EverythingAboutParticleEffectsSlides.pdf

Portals:

http://en.wikibooks.org/wiki/OpenGL_Programming/Mini-Portal

C++ tip of the week:

http://www.codeproject.com/Articles/7150/Member-Function-Pointers-and-the-Fastest-Possible