CS425 GAME PROGRAMMING

Lecture 02 – Game Engine and Event-based Systems

Today

- Today's Agenda
  - **C++ Review**
    - **(**Read GEA 3.1: C++ Review and Best Practices)
  - What is a Game Engine?
  - SDL2

- Reminder
  - PA00 is out and due next week

File Types: .cpp and .h

- *Source* files
  - .c, .cxx, .cpp extensions
  - Contain the bulk of your program's source code

- *Header* files
  - Special kind of source file (.h, hxx, hpp)
  - Used to share information, such as type declarations and function prototypes, between other source files
  - Not seen by the compiler.
    - A preprocessor replaces each *#include* statement with the contents of the corresponding header before sending the code to the compiler
    - Add #pragma once to ensure that each class/function is only defined once
    - For a templated class, both definition and implementations are in .h file

File Types: .obj, .lib, .dll

- *Object* files (.obj or .o) are created as a result of the compiler and contain the machine code
  - Relocatable: the memory addresses at which the code resides have not yet been determined
  - Unlinked: any external references to functions and global data that are defined outside of the source files have not yet been resolved
- *Libraries* are collections of object files
  - Easier to re-use and move around
- Object files and libraries are linked into an executable by the linker
  - Results in fully resolved machine code that can be loaded and run by the operating system.
  - Calculates the final relative addresses of all of the machine code
  - Ensures that all external references to functions and global data made by each object file is properly resolved

File Types: .obj, .lib, .dll

- *Static library* (.lib or .a): linked and included to your code a compile time
- *Dynamic link library* (.dll or .so): a special kind of library that acts like a hybrid between a regular static library and an executable
  - Like a library, it contains functions that can be called by any number of different executables
  - Like an executable, it can be loaded by the operating system independently. It contains some start-up and shut-down code
  - Any references to external functions or data are resolved at run-time
  - Can be updated without changing the executables that use them

- Static or Dynamic libraries must be used with .h files

Key C++ Concepts

- New and Delete
- Virtual functions
  - Virtual destructor?
- Abstract class and abstract functions
- Operator overloading ([ ], <<, <, >, …)
- Functions with default values
- Inline functions
- Templates
- IO (#include <iostream>, <fstream>, <sstream>, …)
  - std::cout, std::cin, std::endl, std::flush, …
- The Standard Template Library (STL)
  - vector<>, list<>, string, queue<>, heap, …
  - hash<>, map<>, set<>, unordered_maps<>, unordered_set<>, …
  - Loop through the elements with iterator or auto

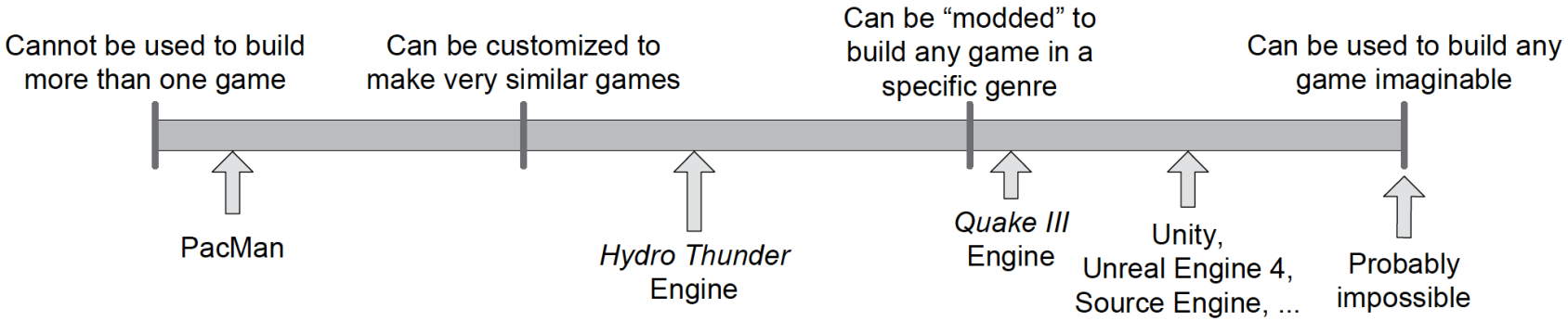
Today

- Today's Agenda
  - C++ Review
  - **What is a Game Engine?**
  - SDL2

What is a Game Engine?

- Software that is extensible and can be used as the foundation for many different games without major modification

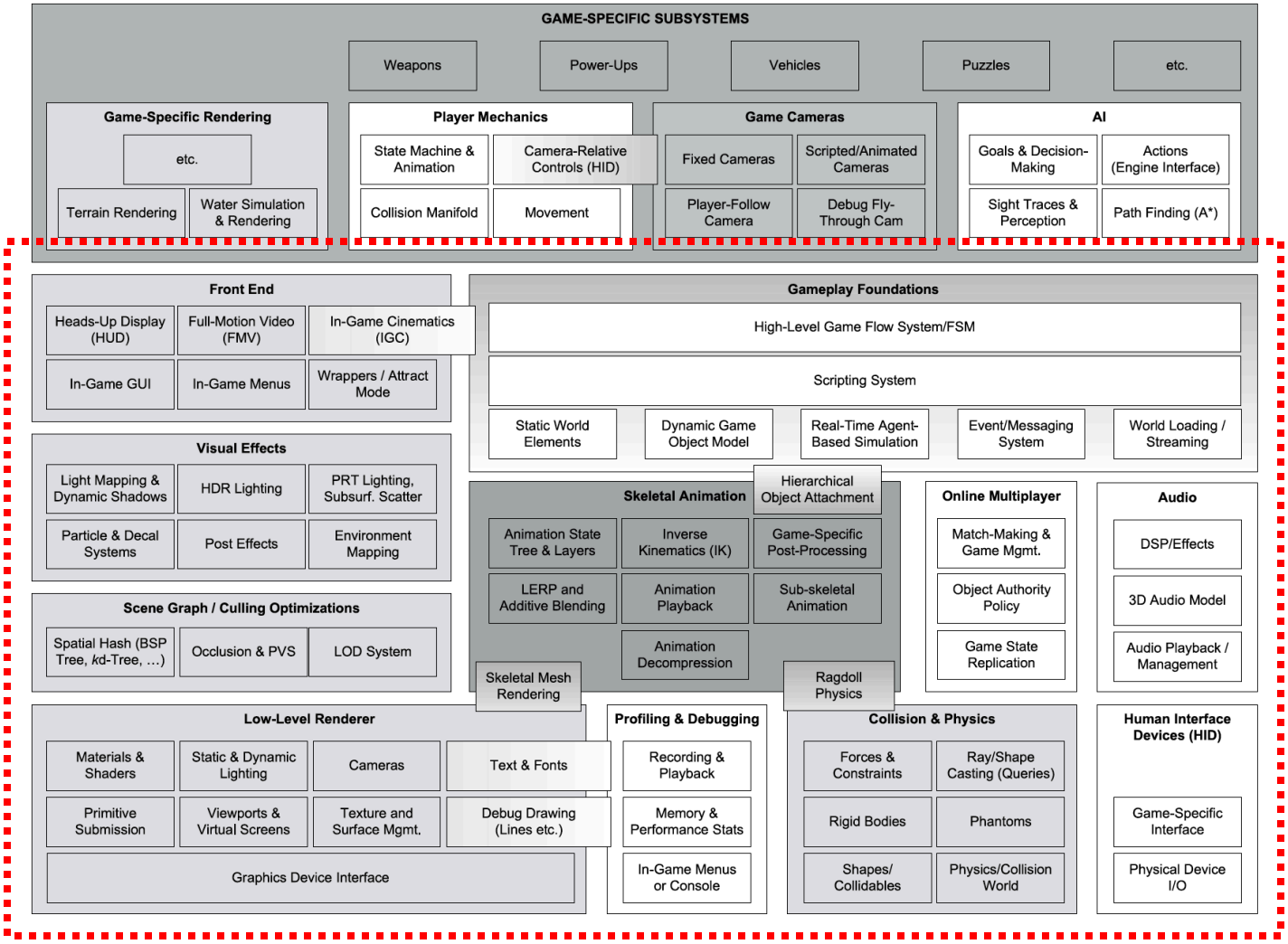
Game Engine Reusability



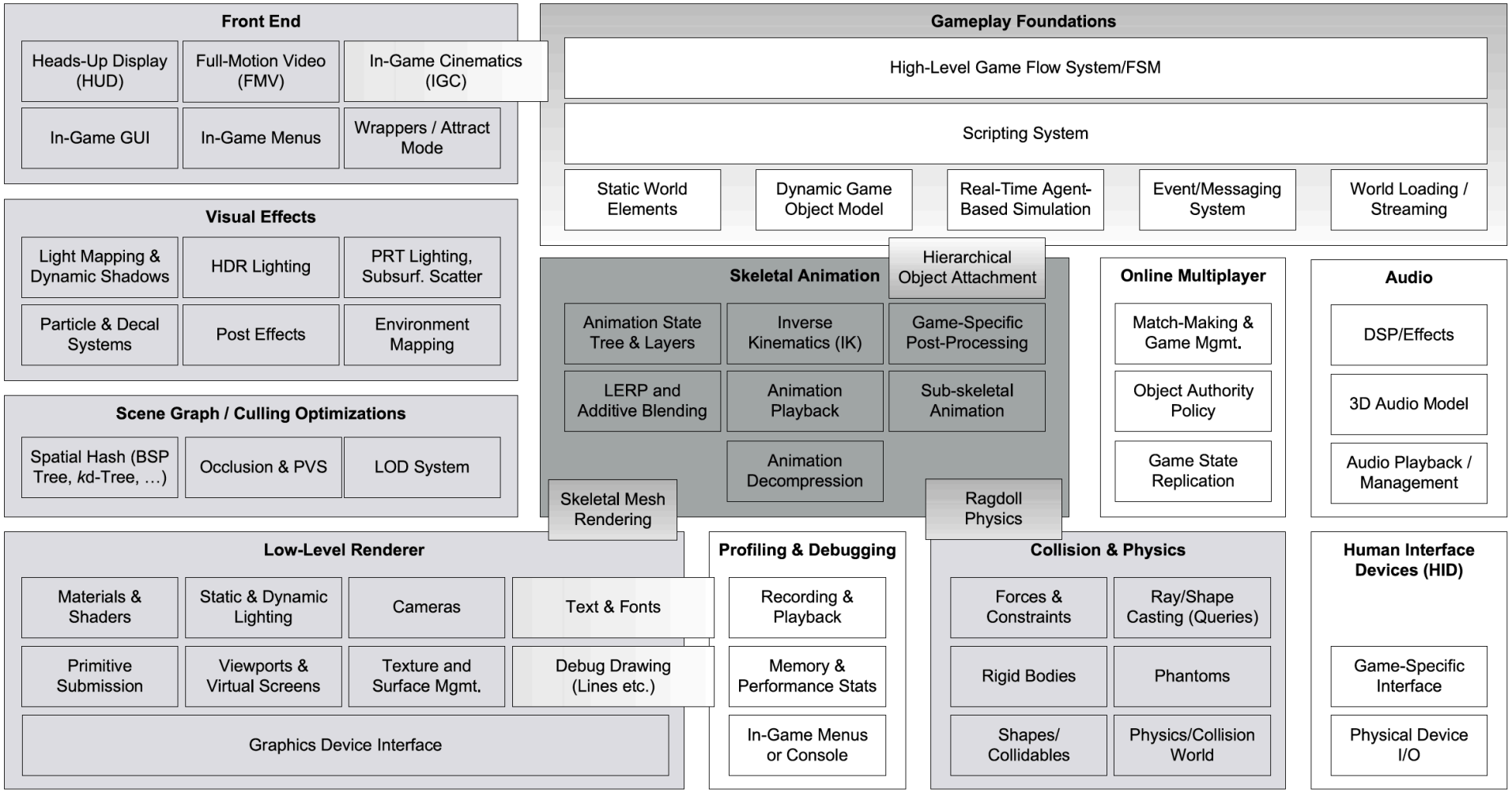
Game Engines

- The Quake Family of Engines
  - https://github.com/id-Software/Quake-2
  - highly recommend downloading these engines and analyzing the source code in Visual Studio
- Unreal Engine
- The Half-Life Source Engine
- DICE's Frostbite
- Rockstar Advanced Game Engine (RAGE)
- CRYENGINE
- PhyreEngine
- XNA Game Studio
- Unity
- …

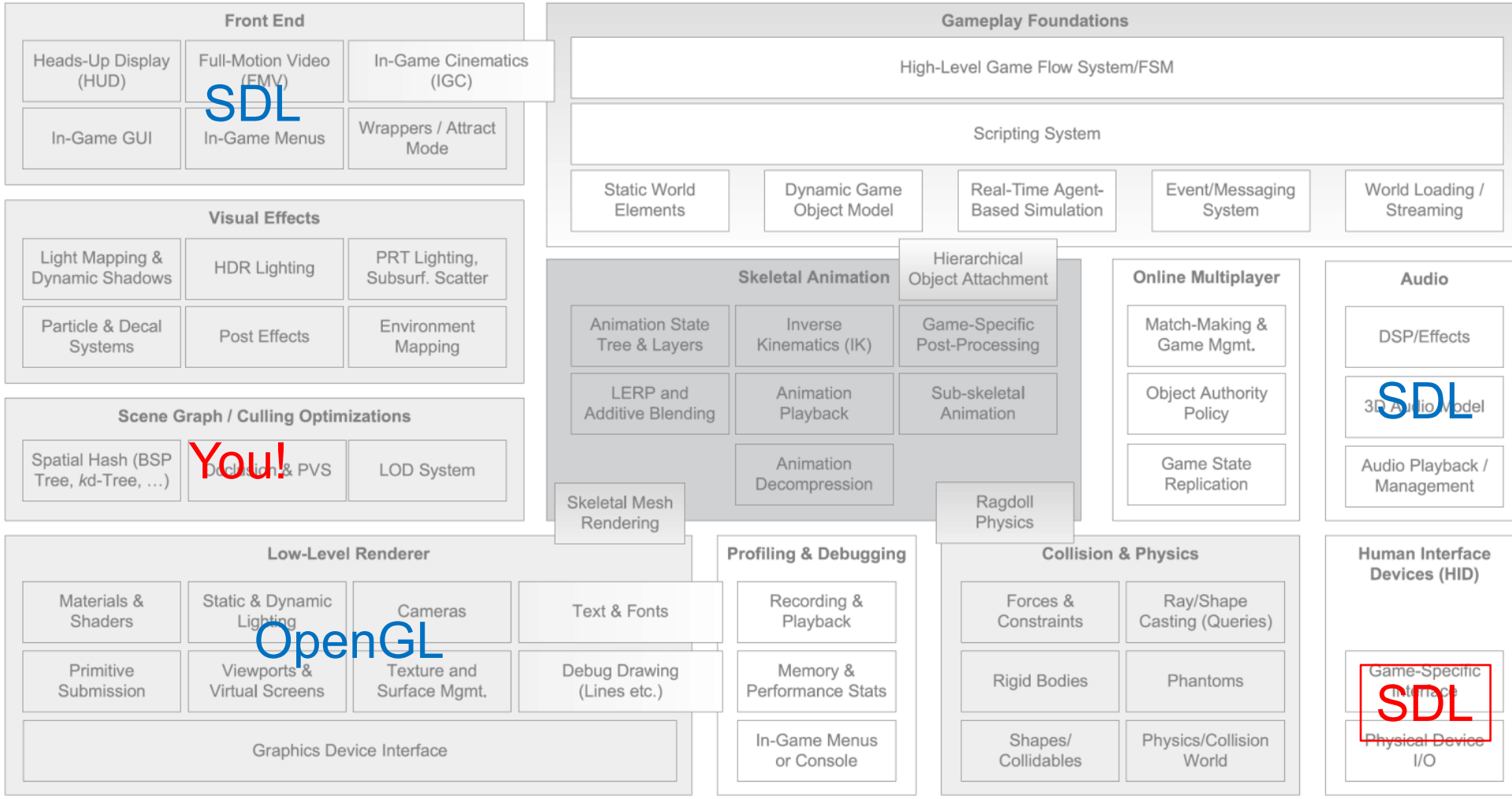
Runtime Game Engine Architecture



Runtime Game Engine Architecture



Runtime Game Engine Architecture



Subsystems in Game Engine

```
RenderManager          gRenderManager;
PhysicsManager         gPhysicsManager;
AnimationManager       gAnimationManager;
TextureManager         gTextureManager;
VideoManager           gVideoManager;
MemoryManager          gMemoryManager;
FileSystemManager      gFileSystemManager;
```

Common Design Patterns of Subsystems

```
class RenderManager
{
public:
    RenderManager()
    {
        // start up the manager...
    }

    ~RenderManager()
    {
        // shut down the manager...
    }

    // ...
};

// singleton instance
static RenderManager gRenderManager;
```

Common Design Patterns of Subsystems

```
class RenderManager
{
public:
    // Get the one and only instance.
    static RenderManager& get()
    {
        // This function-static will be constructed on the
        // first call to this function.
        static RenderManager sSingleton;
        return sSingleton;
    }

    RenderManager()
    {
        // Start up other managers we depend on, by
        // calling their get() functions first...
        VideoManager::get();
        TextureManager::get();

        // Now start up the render manager.
        // ...
    }

    ~RenderManager()
    {
        // Shut down the manager.
        // ...
    }
};
```

Common Design Patterns of Subsystems

```
class RenderManager
{
public:
    RenderManager()
    {
        // do nothing
    }

    ~RenderManager()
    {
        // do nothing
    }

    void startUp()
    {
        // start up the manager...
    }

    void shutDown()
    {
        // shut down the manager...
    }

    // ...
};
```

Brute-force manual start-up and shut-down method

```
int main(int argc, const char* argv)
{
    // Start up engine systems in the correct order.
    gMemoryManager.startUp();
    gFileSystemManager.startUp();
    gVideoManager.startUp();
    gTextureManager.startUp();
    gRenderManager.startUp();
    gAnimationManager.startUp();
    gPhysicsManager.startUp();
    // ...

    // Run the game.
    gSimulationManager.run();

    // Shut everything down, in reverse order.
    // ...
    gPhysicsManager.shutDown();
    gAnimationManager.shutDown();
    gRenderManager.shutDown();
    gFileSystemManager.shutDown();
    gMemoryManager.shutDown();

    return 0;
}
```

Example: OGRE

- OgreRoot.h

```
class _OgreExport Root : public Singleton<Root>
{
    // <some code omitted...>

    // Singletons
    LogManager* mLogManager;
    ControllerManager* mControllerManager;
    SceneManagerEnumerator* mSceneManagerEnum;
    SceneManager* mCurrentSceneManager;
    DynLibManager* mDynLibManager;
    ArchiveManager* mArchiveManager;
    MaterialManager* mMaterialManager;
    MeshManager* mMeshManager;
    ParticleSystemManager* mParticleManager;
    SkeletonManager* mSkeletonManager;
    OverlayElementFactory* mPanelFactory;
    OverlayElementFactory* mBorderPanelFactory;
    OverlayElementFactory* mTextAreaFactory;
    OverlayManager* mOverlayManager;
    FontManager* mFontManager;
    ArchiveFactory *mZipArchiveFactory;
    ArchiveFactory *mFileSystemArchiveFactory;
    ResourceGroupManager* mResourceGroupManager;
    ResourceBackgroundQueue* mResourceBackgroundQueue;
    ShadowTextureManager* mShadowTextureManager;

    // etc.
};
```

Example: OGRE

- OgreRoot.cpp

```
Root::Root(const String& pluginFileName,
           const String& configFileName,
           const String& logFileName) :
    mLogManager(0),
    mCurrentFrame(0),
    mFrameSmoothingTime(0.0f),
    mNextMovableObjectTypeFlag(1),
    mIsInitialised(false)
{
    // superclass will do singleton checking
    String msg;

    // Init
    mActiveRenderer = 0;

    // create log manager and default log file if there
    // is no log manager yet
    if(LogManager::getSingletonPtr() == 0)
    {
        mLogManager = new LogManager();
        mLogManager->createLog(logFileName, true, true);
    }

    // dynamic library manager
    mDynLibManager = new DynLibManager();
    mArchiveManager = new ArchiveManager();

    // ResourceGroupManager
    mResourceGroupManager = new ResourceGroupManager();

    // ResourceBackgroundQueue
    mResourceBackgroundQueue
        = new ResourceBackgroundQueue();

    // and so on...
}
```

Naughty Dog's Uncharted and The Last of Us

- uses a similar explicit technique for starting up its subsystems

```
Err BigInit()
{
    init_exception_handler();

    U8* pPhysicsHeap = new(kAllocGlobal, kAlign16)
        U8[ALLOCATION_GLOBAL_PHYS_HEAP];
    PhysicsAllocatorInit(pPhysicsHeap,
        ALLOCATION_GLOBAL_PHYS_HEAP);

    g_textDb.Init();
    g_textSubDb.Init();
    g_spuMgr.Init();

    g_drawScript.InitPlatform();
```

Then a wide range of operating system services, third-party libraries and so on ….

```
    memset(&g_discInfo, 0, sizeof(BootDiscInfo));
    int err1 = GetBootDiscInfo(&g_discInfo);
    Msg("GetBootDiscInfo() : 0x%x\n", err1);
    if(err1 == BOOTDISCINFO_RET_OK)
    {
        printf("titleId       : [%s]\n",
            g_discInfo.titleId);

        printf("parentalLevel : [%d]\n",
            g_discInfo.parentalLevel);
    }

    g_fileSystem.Init(g_gameInfo.m_onDisc);

    g_languageMgr.Init();
    if (g_shouldQuit) return Err::kOK;

    // and so on...
```

Today

- Today's Agenda
  - C++ Review
  - What is a Game Engine?
  - **SDL2**

SDL2

- It has in fact multiple libraries, make sure to use the necessary library
  - SDL2
    - Core SDL2 functions
  - SDL2_image
    - For loading png and other image formats
  - SDL2_ttf
    - For loading true type font
  - SDL2_mixer
    - For sound and music
  - SDL2_network
    - For basic network communication
  - …

SDL2

- Basic Structure of SDL2 code

```
#include <SDL.h>
int main(int argc, char *argv[])
{
        SDL_Init( … );
        SDL_CreateWindow( … );

        //Load game resources

        //the Game Loop
        while( true )
        {
          SDL_Event e;
          while( SDL_PollEvent( &e ) != 0 )
          {
                  //handle event e
          }
        }
        SDL_DestroyWindow( … );
}
```

SDL2 events

- SDL_Event
  - SDL2 defines many events for IO, GUI and timer
  - Most important data: event.type
- SDL_PollEvent(SDL_Event * event)
- SDL_PushEvent(SDL_Event * event)
  - You can also define your own event

```
SDL_Event user_event;
user_event.type = SDL_USEREVENT;
user_event.user.code = 2;
user_event.user.data1 = NULL;
user_event.user.data2 = NULL;
SDL_PushEvent(&user_event);
```

SDL2 Drawing

```
#include <SDL.h>
int main(int argc, char *argv[])
{
        SDL_Init( … );
        SDL_CreateWindow( … );
        SDL_Renderer * gRenderer = SDL_CreateRenderer( … )
        //Load game resources
        //the Game Loop
        while( true )
        {
          SDL_Event e;
          while( SDL_PollEvent( &e ) != 0 )
          {
                  //handle event e
          }
          SDL_SetRenderDrawColor( gRenderer, …);
          SDL_RenderClear( gRenderer );
          //Render current frame
          SDL_RenderPresent( gRenderer );
        }
        SDL_DestroyWindow( … );
}
```

That is All

- Today we have:
  - C++ Review
    - Read GEA Chapter 3.1 (C++ Review and Best Practices)
  - What is a Game Engine?
    - Read GEA Chapter 1.3 (What is a Game Engine?) and Chapter 6.1 (Subsystem Start-Up and Shut-Down)
  - SDL2

- Reminder
  - PA00 is out and due next week (9/4)