

**Zachary Davids 100796282**

**Programmer Guide**

**Overall Architecture**

Wolfenstein was developed by adhering as much as possible to the object-orientated philosophy, with game objects being separated entirely from the engine components and any required functionality required by game objects added to those objects respectively. Note that the UML diagram below does not include lighting-related classes as lighting was not used, and all classes below the Level class are interconnected.

**Data Structures and Performance**

Amongst the most extensively used data structures used were C++ vectors, a container with a contiguous storage location whose size can be changed dynamically and function similar to ArrayLists, and to a lesser extent C++ maps, a data structure which stores elements with value mapped to a key which can be used to identify and sort elements. A node object was defined for the XML parser, which contained a string and a position, in order to pass information easily to the Level class. A vertex object was also created to represent a vertex and holds a location as well as a texture coordinate and normal for that specific vertex, this was done to avoid passing multiple vectors to the mesh class. Arrays were used to represent the various state of keyboard and mouse keys.

Thankfully, no performance encounters arose during the development as index-based drawing was implements and any faces or vertices not currently facing the camera were culled. This resulted in a frame rate that remained above 60 FPS, despite having a large number of objects with constantly enabled collision detection properties.

**External Libraries and Artwork**

Many external libraries were used to facilitate the development process:

**The OpenGL Extension Wrangle Library (GLEW):**

“The OpenGL Extension Wrangler Library (GLEW) is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform. OpenGL core and extension functionality is exposed in a single header file.” The GLEW library was used to access openGL functionality.

<http://glew.sourceforge.net/>

**OpenGL Mathematics (GLM):** “OpenGL Mathematics (GLM) is a header only C++ mathematics library for graphics software based on the [OpenGL Shading Language (GLSL)](http://www.opengl.org/documentation/glsl/) specifications.” Using this library significantly shortened development time by allowing me to forgo the definition of vectors, matrices and the operations I needed to perform on them.

<http://glm.g-truc.net/0.9.6/index.html>

**Open Asset Import Library (Assimp):**

Assimp is “a portable Open Source library to import various well-known [3D model formats](http://assimp.sourceforge.net/main_features_formats.html) in a uniform manner. The most recent version also knows how to export 3d files and is therefore suitable as a general-purpose 3D model converter.” Using the Assimp library I enabled by mesh loader to handle a variety of different file extensions used by different modeling programs using the same code.

<http://assimp.sourceforge.net/>

**pugixml:** pugixml is a “light-weight, simple and fast XML parser for C++ with XPath support”. pugixml was used to simplify and expedite the process of loading my levels in using an XML file.

<http://pugixml.org/>

**Simple Directmedia Layer (SDL):** “Simple DirectMedia Layer is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D.” SDL was used to create the window, handle user input, as well as the management of the openGL context.

<https://www.libsdl.org/index.php>

**Simple and Fast Multimedia Library (SFML):**

“SFML provides a simple interface to the various components of your PC, to ease the development of games and multimedia applications. It is composed of five modules: system, window, graphics, audio and network.” SFML was used to handle all of the music and audio related features.

<http://www.sfml-dev.org/>

**Sean T. Barrett (STB) image:** STB is a set of C/C++ libraries that facilitate image loading, audio decoding, image writing to disk, text editing for games, etc. The STB library was used only for loading and decoding the PNG images which constituted the textures and sprites.

<https://github.com/nothings/stb>

**Tools**

Adobe Photoshop CS6, a photo editor suite, was used to create transparency in the sprites and textures.

**Difficulties**

Difficulties arose when attempting to create a consistency amongst creating objects, specifically with their world coordinates. Some objects were drawn at the origin, and others were drawn with an offset. This created confusion when attempting to set-up collision detection as the objects may have visually been in the correct spot, but they were offset in collision detection calculations. Collision detection as a whole was slightly confusing, and using a 3rd party library, such as box2d would have been wiser in hindsight.

**Incomplete**

I felt that the included levels could have had more detail added to them, as well as some of the other 116 tile textures used for level design; this would be done by adding a new property to the XML files indicating a specific texture is used. Collision detection is not up to par, and enemies will see you through some doors, this could be fixed by going back and making sure doors are drawn at the origin. Transparency is also an issue with enemies clipping each other due to draw order, which could be remedied by sorting the enemy vector by position and drawing them according. The HUD has rotation issues, and can be remedied by rotating the transformation matrix along the x-axis.