**CHAPTER 3**

# THEORETICAL FRAMEWORK

In this project, the project developers will develop the system by following a set of coding principles for a more dynamic and robust code structure. SOLID is a mnemonic acronym introduced by Michael Feathers for the five basic principles for object-oriented programming and design. It will be followed by constructing aesthetics as front-end of the system through several open source platforms and frameworks.

## **Interactive 3D Maps**

Three-dimensional maps use 3D computer graphics to present geographical information, using perspective representations that, to a certain degree, correspond to the real world. The view presented on a 3D map is more natural, intuitive and easier to comprehend than its 2D equivalent. 3D maps may, but do not have to, use real-3D data and volumetric objects. Earth surface representation, that incorporates height information, called 2.5D, complemented with 3D symbols, is decent enough for many applications (Raper, 1989).

3D maps are interactive by definition. Their usability is very restricted without the likelihood of interactive manipulation of the presented view, and unobstructed setting of the wanted perspective. The higher the level of interactivity which is provided, the more useful a 3D map becomes (Góralski, 2009).

However, it is not easy to define what is and what is not a 3D map. As with traditional maps there are diverse types and sorts of geographical presentations that use 3D visualization. A majority of research focuses on the broader subject of application of 3D in geographical visualization, or geovisualization, (MacEachren and Kraak, 2001). Representations used in this discipline range in levels of realism and presented data types. In the geovisualization’s understanding, a 3D map may be a realistic reconstruction of a city, or a planned landscape, as in geospatial virtual environments, (MacEachren et al., 2003).

This system will focus on a 3D campus map of MSU-IIT that present topography of buildings and structures and use a combination of different types of textures and symbols (3D objects and labels, 2D symbols, lines and polygons, text, numbers, points; selectable or not; multimedia and hyperlinks).

## **WebGL**

WebGL (Marrin, 2011) is an extension of HTML5 canvas element, which is now widely used for developing web applications requiring 3D visualization. It is a 3D graphics API, written in low level language and is based on OpenGL ES 2.0. To avoid complex low level programming, several WebGL-based frameworks have been developed, providing ease of development.

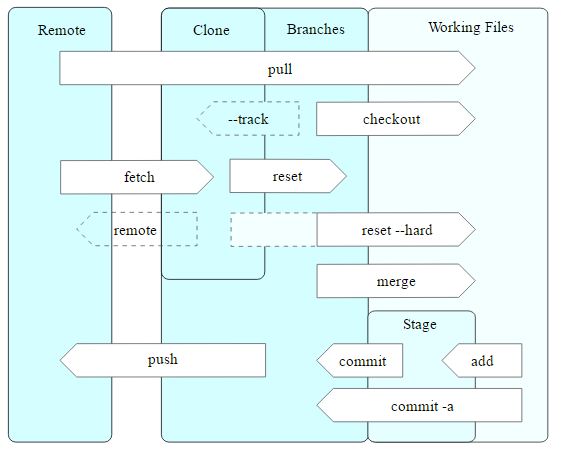
The system greatly uses WebGL in displaying the interactive campus map. It is with this technology that the map can be accessed in any browser anywhere.



**Figure 3.1** Sample WebGL Representation of a Terrain

## **Git Workflow**

By far, the most widely used modern version control system in the world today is Git. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel. A staggering number of software projects rely on Git for version control, including commercial projects as well as open source. Developers who have worked with Git are well represented in the pool of available software development talent and it works well on a wide range of operating systems and IDEs (Integrated Development Environments) (Atlassian, 2005).



**Figure 3.2** Git Operations

## **SOLID Principles**

Since the application requires an extensive but robust number of APIs, extra functionalities might get entangled with the other functions or mismanagement in writing the code would result to an unstable product. With this, the researchers will use the 5 basic principles of object-oriented programming in creating a flexible and ready to use and updatable product.

### **Single Responsibility**

In implementing functions of the system, classes of these functions should be considerably written to one reason to chance. This represents a good way of identifying classes (as to their use) during the design phase of the application.

*Example (might change due to the relevance of code)*

// single responsibility principle - good example

interface IEmail {

public void setSender(String sender);

public void setReceiver(String receiver);

public void setContent(IContent content);

}

interface IContent {

public String getAsString(); // used for serialization

}

class Email implements IEmail {

public void setSender(String sender) {// set sender; }

public void setReceiver(String receiver) {// set receiver; }

public void setContent(IContent content) {// set content; }

}

In the project, single responsibility will be mostly observed in constructing functionalities of the system to minimize confusion but maximizes effectiveness of different tools from different frameworks.

### **Open Close Principle**

Software entities like classes, modules and functions should be *open for extension* but *closed for modifications***.** This creates flexibility on the child classes by introducing abstraction to its parent class methods. So if a functionality of a system can be reused to another, it is applicable through this principle.

### **Liskov's Substitution**

Derived types must be completely substitutable for their base types. This ensures the system that any new derived models or classes to be implemented in the system would not change the behavior or the previous functionalities.

### **Interface Segregation**

The Interface Segregation Principle states that clients should not be forced to implement interfaces they don't use. Instead of one fat interface many small interfaces are preferred based on groups of methods, each one serving one submodule. In the system, the principle will be useful especially when applying tools of different APIs on a single method or functionality of the application.

### **Dependency Inversion**

High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions. With this, the system's modules would be flexible for future modification and/or updates.

## **MVC**

In object-oriented programming development, model-view-controller (MVC) is the name of a methodology or design pattern for successfully and efficiently linking the user interface to underlying data models. The MVC pattern has been prefigured by many developers as a useful pattern for the reuse of object code and a pattern that permits them to significantly lessen the time it takes to develop applications with user interfaces.

The model-view-controller pattern proposes three main components or objects to be used in software development:

* A *Model*, which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface.
* A *View*, which is a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth)
* A *Controller*, which represents the classes connecting the model and the view, and is used to communicate between classes in the model and view.

## **Laravel 5**

Laravel 5, created by Taylor Otwell and intended for the development of web applications following the model–view–controller (MVC) architectural pattern, is a web application framework with expressive, elegant syntax. Laravel attempts to take the pain out of development by easing common tasks used in the majority of web projects, such as authentication, routing, sessions, queueing, and caching. It aims to make the development process a pleasing one for the developer without sacrificing application functionality. It is accessible, yet powerful, providing powerful tools needed for large, robust applications. A superb inversion of control container, expressive migration system, and tightly integrated unit testing support give you the tools you need to build any application with which you are tasked.

## **OpenGL**

Open Graphics Library is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms.

## **OpenStreetMap**

Google Maps is a commercialized data map library whereas its services were exponentially expensive. Meanwhile, OpenStreetMap is an open data project. They release raw data "vector" maps that together with OSM Elements make up the virtual map. With this short discrepancy, OpenStreetMap has the greater potential for larger contribution since data presented are raw and open to all. However, Google Maps has more active services and larger data diversity which means that potentially the developed system architecture has already been established (which also costs more). With this considered, the researchers would use OpenStreetMap as one of the main frameworks to be used in the system.

## **Leaflet**

Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps. It has all the mapping features most developers ever need. It is designed with simplicity, performance and usability in mind. It works efficiently across all major desktop and mobile platforms, can be extended with lots of plugins, has a beautiful, easy to use and well-documented API and a simple, readable source code that is a joy to contribute to.

## **OSM Buildings**

OSM Buildings is a 3D renderer that uses geometry data from OpenStreetMap available under OpenDatabase License. Three.js is also a 3D rendering Library written in Javascript that uses WebGL technology. Although they act on the same purpose, Three.js has more dynamic functionalities than OSM Buildings. However, OSM Buildings works well with OpenStreetMap since every visible element within the OpenStreetMap acts as a rendered 3D and 2D resources.

In the initial development of the system, the researchers started with Three.js with Google Maps as framework. But the extensive library of Three.js and the limited free resources from Google Maps hinders the researchers to control the aesthetic design of the developed system. Thus then the researchers decided to use OSM Building and its related library for the system instead.



**Figure 3.3** OSM Buildings Representation

## **GeoJSON**

GeoJSON is a format for encoding a variety of geographic data structures. A GeoJSON object may represent a geometry, a feature, or a collection of features. This format will be used to most functionalities required by this system since the selected frameworks follows the same format as well as it has human readable syntax.