

University of Nevada  
Reno

# **Terrain Analyzing in a Virtual Environment with Real-time Native Shape Creation**

A thesis submitted in partial fulfillment of the  
requirements for the degree of Master of Science  
with a major in Computer Science.

by

Joseph Richard Jaquish

Dr. Frederick C. Harris, Jr., Thesis advisor

December 2005

**UNIVERSITY  
OF NEVADA  
RENO**

**THE GRADUATE SCHOOL**

We recommend that the thesis  
prepared under our supervision by

**JOSEPH JAQUISH**


entitled


**Terrain Analyzing in a Virtual Environment with  
Real-time Native Shape Creation**

be accepted in partial fulfillment of the  
requirements for the degree of

**MASTER OF SCIENCE**

  
\_\_\_\_\_  
Frederick Harris Jr., Ph. D., Advisor

  
\_\_\_\_\_  
Sergiu Dascalu, Ph. D., Committee Member

  
\_\_\_\_\_  
Scott Bassett, Ph. D., Graduate School Representative

  
\_\_\_\_\_  
Marsha H. Read, Ph. D., Associate Dean, Graduate School

December, 2005

## Abstract

An accurate representation of terrain is crucial for land based operations. Computer technology is helping expert geographers analyze terrain in new and more accurate ways. This thesis presents an application called Taverns which stands for “Terrain Analyzing in a Virtual Environment with Real-time Native Shape Creation”. Taverns will help analysts create an accurate representation of terrain because it is designed to be used in an immersive environment, creating the deepest sense of realism graphics technology can deliver today. Using Taverns, specialists with many years of experience may analyze terrain with a high degree of accuracy based on information that can be gathered through remote sensing techniques. The applications of remote sensing are not limited to Earth, increasing the potential of Taverns to aide specialists in more thoroughly analyzing the surface of other solid surface planets, in particular the surface of Mars.

## Acknowledgements

The efforts depicted are sponsored by the Department of the Army, Army Research Office; the content of the information does not necessarily reflect the position or the policy of the federal government, and no official endorsement should be inferred. The Desert Terrain Project (ARO# DAAD19-03-1-0159) and CAVE Project (ARO# N61339-04-C-0072) have funded the work presented.

The path to this point in my life is not of my strength alone. If not for the love and support of wonderful people in my life, I would not have such privileges bestowed upon me. I am privileged with the committee that has accepted my request to support me and guide me through my graduate experience. To Dr. Scott Bassett and Dr. Sergiu Dascalu, thank you for your seemingly endless compassion and support. Few students are so lucky to experience your influence. To Dr. Frederick Harris Jr., my graduate advisor, I express a thank you from my heart deeper than any I have expressed in my life. Thank you. I pray that the consequences of your wisdom and grace are evident to all whom I meet. Be sure that you are all role models I shall not forget. I am privileged to work with the peers placed at my side. To Michael Penick and Michael Dye, understand that your qualities are not hidden from those whom you meet. I shall carry the character and personality I see in you both with me for a very long time. To Jeffery Stuart, know that I speak for all when I say your skills lie in the shadows of your character. I pray that our friendship shall remain. You may all be sure that I will boast of such admirable influences on my life. I am privileged to be loved by my remarkable family. You have carried me in weakness and been my light darkness. I pray you forgive me of my falls, for in your arms I am merely a fledgling new to flight. Be sure that your love has made me who I am today.

# Contents

<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>List of Figures</b>	<b>v</b>
<b>List of Tables</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 GIS and Virtual Reality</b>	<b>4</b>
2.1 Geographic Information Systems . . . . .	4
2.2 Virtual Reality . . . . .	13
2.3 The Desert Terrain Project . . . . .	23
2.4 Need for 3D . . . . .	23
<b>3 Requirements Specification and Modeling</b>	<b>26</b>
3.1 Functional Requirements . . . . .	27
3.2 Non-Functional Requirements . . . . .	27
3.3 Primary Use Cases . . . . .	28
3.4 Requirements Traceability Matrix . . . . .	28
3.5 UML Scenarios . . . . .	30
<b>4 Implementation Design</b>	<b>33</b>
4.1 Architecture and Algorithms . . . . .	34
4.2 User-Interface Prototype . . . . .	49
<b>5 Conclusions and Future Work</b>	<b>60</b>
5.1 Conclusions . . . . .	60
5.2 Future Work . . . . .	61
<b>Bibliography</b>	<b>64</b>

# List of Figures

2.1	Example data layers for a particular location [22]. . . . .	7
2.2	Relationship of the <code>Shx</code> , <code>Shp</code> and <code>Dbf</code> files. . . . .	11
2.3	Process of drawing a shape (A), erasing the hole (B) and filling it with another shape (C). . . . .	12
2.4	Example of potentially restricted air space over a key facility [18]. . .	13
2.5	Head mounted VR display [36]. . . . .	17
2.6	Projection based VR display [36]. . . . .	17
2.7	Virtual objects move relative to the user utilizing a head tracking system [36]. . . . .	22
2.8	Lava flows on the surface of Mars [30]. . . . .	24
2.9	Rotated view of the lava flows on Mars [30]. . . . .	24
2.10	Classic alluvial fan [41]. . . . .	25
3.1	Taverns primary use case diagram. . . . .	29
4.1	Taverns class structure. . . . .	34
4.2	Loading configuration data. . . . .	37
4.3	Creating an image texture from a shapefile. . . . .	39
4.4	Algorithm for creating a texture of a shapefile. . . . .	40
4.5	Example point locations around a shape. . . . .	41
4.6	Potential “hole” shapes inside a “main” shape. . . . .	41
4.7	The same shape drawn “clockwise” and “counterclockwise”. . . . .	43
4.8	Find two points closest from shape <i>A</i> and shape <i>B</i> . . . . .	44
4.9	A shape with a hole and the two closest points $pA$ and $pB$ . . . . .	45
4.10	Shape <i>A</i> after hole <i>B</i> has been “fixed”. . . . .	45
4.11	Determining attribute data types. . . . .	48
4.12	The Sight, general design and in action. . . . .	48
4.13	Plotting points undo and redo example. . . . .	51
4.14	Common desktop editing commands Taverns supports. . . . .	52
4.15	Common 3D navigation commands Taverns supports. . . . .	53
4.16	Taverns specific editing commands. . . . .	53
4.17	Sight motion keys. . . . .	54

4.18	Example attribute list with Taverns in VR. . . . .	56
4.19	Simple shape on a typical terrain texture. . . . .	57
4.20	Simple shape on a data layer texture. . . . .	58
4.21	Simple shape on a data layer in wire frame view. . . . .	59

# List of Tables

3.1	Functional requirements (Section 3.1) mapped to use cases (Section 3.3).	30
3.2	Primary scenario, create a new shapefile. . . . .	31
3.3	Secondary scenario, the user creates no shapes. . . . .	31
3.4	Secondary scenario, not all shapes have assigned attributes. . . . .	32



# Chapter 1

## Introduction

Training through experience is a valuable asset to military forces. Throughout most of history, the most effective training scenarios for a soldier were actual scenarios themselves. As one can guess, actual scenarios for military forces are quite often costly experiences of war and death. We are fortunate to live in a day and age where research and development promises to provide militaries more experience while avoiding the horrors of war. The advantages that technology provides with regards to training personnel have proved incredibly successful and continue to grow in potential. For this reason, significant research and development continues creating more and more realistic training environments for military personnel.

One simply cannot exaggerate the necessity of military training, it is utterly essential [38, 42]. Generally, the more realistic training is to the soldier, the better prepared that soldier will be when an actual scenario arises [24, 26, 34]. Creating a realistic training environment is an intensive process involving significant planning, coordination, time and money. Resources such as equipment, fuel, vehicle parts, and maintenance time are used extensively making it difficult to provide consistent, thorough training for everyone.

The field of computer graphics and visualization has become crucial for the creation of realistic training environments. With modern technology, the military can train its personnel in computer simulated situations, thus alleviating much of the need for travel and equipment transportation. Also, personnel can work their way through a simulation several times, honing their skills while using minimal resources.

Visualization techniques help to establish realism, but quite often, real-world geographic data is also necessary to achieve a fully immersive and realistic simulation. The geographic data represents a true-to-life landscape in the simulated environment. The data can be visualized via computer graphics and virtual reality techniques. The end goal is to give personnel the feeling they are at the real-world location, even though they are actually in a simulated environment.

In order to obtain accurate information about a location, specialists in geographic and geological disciplines must first visit and thoroughly analyze the location. This is viewed as a necessary expenditure of time and resources. However, situations occur where the resources and access necessary to collect ample data is simply not available. This lack of data can endanger missions, especially when specific training, such as helicopter flight, requires accurate knowledge and representation of an area.

Remote sensing methods allow geographic experts to gather data without ever visiting the location, and later analyze this new data off-site [4, 10]. One such piece of data is a Digital Elevation Model (DEM) [5]. The name has pragmatic significance in that it consists of spatially explicit digital values of height data. Other important data are Geographic Information Systems (GIS) vector layers [27]. GIS vector layers are maps (i.e. represent a location) that store two-dimensional shapes (polygons) with attributes. Attributes can be assigned from a variety of sources including satellite imagery, site visits, current databases, and so on. DEMs are also referred to as GIS raster layers, but from this point forward, any mention of a GIS layer refers to a GIS vector layer.

There are current software packages available that provide geographic experts the tools to analyze data layers. ESRI<sup>TM</sup> [16] currently provides a wide array of GIS software packages such as ArcGIS<sup>TM</sup> [14] and ArcScene<sup>TM</sup> [15]. As useful and accurate as these packages have proven to be, they unfortunately do not allow the expert to analyze the terrain while viewing it in 3D. This thesis introduces a revolutionary application called Taverns which stands for “Terrain Analyzing in a Virtual Environment with Real-time Native Shape Creation”. Taverns is especially designed

to allow the analysis of data layers in a three dimensional view on both the standard desktop monitor and in a virtual environment. The virtual environment will provide the expert user another level of realism unattainable on desktop monitors.

Taverns is actually the first large step in a project called Desert Terrain [21, 22]. One of the goals of this project is to model dust behavior of desert regions as accurately as possible and use these predicted behaviors for training simulations. As mentioned above, better realism in training environments produces more prepared personnel for the real-world scenario. In order for the specialists to accurately model dust behavior, they must have an accurate representation of the terrain. Experts of geographic disciplines will use Taverns to create this accurate representation.

The remainder of this thesis is structured as follows: Chapter 2 presents background information on GIS methods and applications, computer graphics, and immersive virtual environments. Chapter 3 outlines application requirements and modeling specifications for Taverns. We present the implementation design of Taverns in Chapter 4. Finally, we finish with conclusions and the strong potential for future work in Chapter 5.

# Chapter 2

## GIS and Virtual Reality

This thesis presents a project that combines a number of disciplines ranging from geographic information systems to visual psychology. This chapter presents these disciplines in detail enough to understand their role for this project. Section 2.1 discusses geographic information systems which are the driving force behind this project. Section 2.2 discusses a popular field of computer graphics known as virtual reality and the methods for creating realistic environments through illusion. Section 2.3 introduces the overarching project called Desert Terrain of which this project is a large portion. Finally, Section 2.4 presents several arguments for why a three dimensional view is crucial for accurate terrain analysis.

### 2.1 Geographic Information Systems

#### What is GIS

GIS is a very broad term that stands for Geographic Informations Systems [27]. To word it differently, it is a collection of systems for geographic information. These systems revolve around the creation and management of spatial data. It is an extensive field that allows users to integrate, store, edit, analyze and display geographic data. The practical applications of GIS are immense ranging from scientific investigations to military training.

ESRI<sup>TM</sup> is a predominant provider of GIS software [16]. One significant software package ESRI supplies is ArcGIS<sup>TM</sup> [14]. ArcGIS is incredibly functional allowing the

user to load a vast array of data formats, combining information, converting information, etc. One may think of ArcGIS as a very powerful editor for spatial information. ArcScene™ [15] enables geographers to view GIS data in three dimensions. The user has quite a bit of control over how the data is viewed, but it lacks the support for drawing “Shapes” on the data displayed in 3D.

Shapes, as will be detailed in further chapters, are used to represent features of spatial and geographic data and are widely used in terrain analysis. Currently, software packages only allow the creation of shapes while viewing the data in two dimensions, that is to say, no elevation data is rendered while the user is creating shapes. For example, if a geographer wanted to outline a mountain range with a shape and label it the “Rocky Mountains,” the user would be forced to estimate the location of the mountains because the slopes of the mountains would not be rendered. Essentially, the geographer would be estimating the location of the mountain range with a “birds-eye” aerial photo. Though expert geographers have a large amount of resources available to make close estimations of tasks such as the example, inaccuracies will develop. What is needed is an application that will read current GIS data formats, render the data in a three dimensional view, allow the creation of shapes on the 3D data, and finally save the newly created shapes to GIS data formats.

## **Remote Sensing**

A tremendous amount of data is available for any one location. Geographers gather vast amounts of data ranging from elevation data, moisture content, vegetation density, soil density, even magnetic field information. Often this data is extracted from an expert “on-site,” that is to say, the expert must travel to the location to start extracting these types of data features. The process of gathering this data without actually visiting the location is referred to as Remote Sensing [4, 10].

New remote sensing techniques are always being researched in order to determine what other types of data can be gathered with imaging equipment on board aircraft or satellites. When it is found that remote sensing can gather a new type of data, it is

very important to ensure that the gathered data is accurate. This most often requires an expert to visit the location which is quite necessary in the interest of proving the new remote sensing technique is accurate.

Many remote sensing techniques have proven to be fast and reliable which is strong support for continued research. Another strong argument is the fact that not all areas of the world are available for a site visit. Military applications often require information about terrain, yet political and other complications will restrict experts from visiting the site. This is when remotely gathered data becomes invaluable.

Once a variety of data is gathered with remote sensing techniques, geographers may organize this data in a more “research friendly” environment and use it to extract features that would normally require a site visit. The preferred method for viewing several types of data pertaining to one location is to create layers from the data. Current GIS software allows the user to customize how the layers are presented in order to more easily determine differences and similarities between the layers, which is vital for extracting further features. Figure 2.1 shows an example of several data layers that represent different types of data pertaining to the same location [21, 22].

### **Digital Elevation Model**

Elevation data is one of the many data layers that may be remotely gathered from a location. A Digital Elevation Model (DEM) is a table of height values that represents the contour of a particular location [5]. DEMs come in numerous formats but they must all contain at least the following data:

1. A starting reference location
2. Number of height values per row
3. Separation distance between each row value
4. Number of height values per column
5. Separation distance between each column value
6. The height data itself

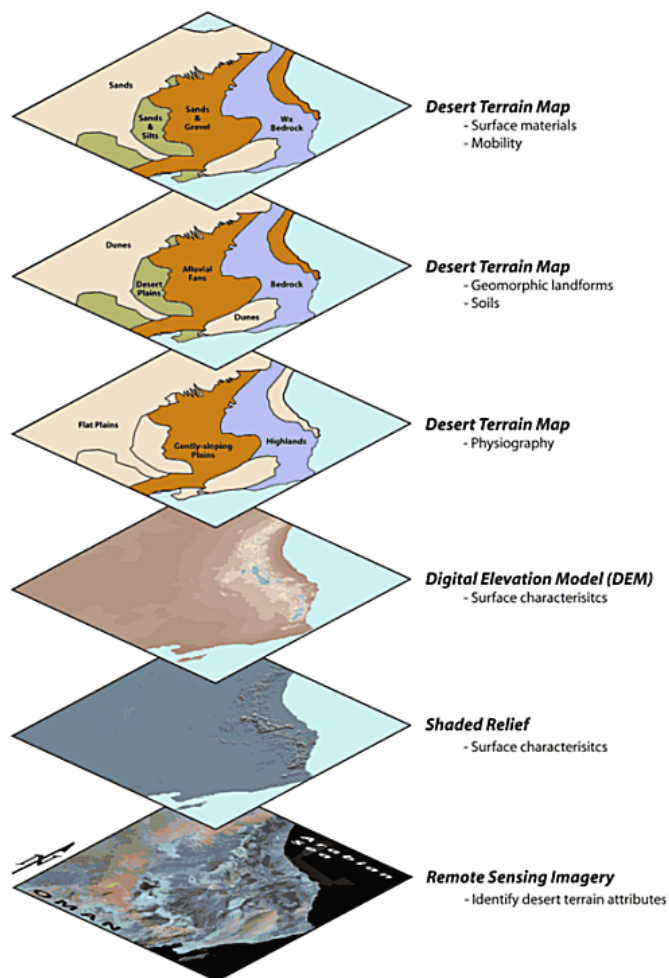


Figure 2.1: Example data layers for a particular location [22].

Items 3 and 5 are often consolidated into one value. If not, the the values are normally the same because they represent the “resolution” of the DEM. For example, if the separation value is 10 meters, then the DEM is referred to as a 10 meter resolution DEM. Each individual DEM file format will specify the numerical representation (eg: feet, meters, etc.).

The resolution of the DEM is definitely an important piece of information, but it does not help us understand how detailed the DEM is without also knowing the size of the terrain the DEM represents. A 10 meter resolution DEM is very high detail for an area the size of California for example, but fairly low resolution for a small county.

Other metadata may be included with individual DEM file formats, however the data listed above is a minimum. With this data, an application can render and manage the terrain quite easily. We will briefly discuss the popular BIL format for representing DEM data because future chapters will reference this format. The BIL interpretation of DEM data is actually a collection of three files, but are collectively required for the BIL format [39].

The data file itself is a simple binary raster file that stores the height values in row major form. The metadata for the DEM is contained in a Header file and a World file, described below.

The header file is the first of two files that contains metadata for the data file. It is an ASCII text file with values following keywords. The majority of the necessary metadata for the DEM is stored in this file. The pieces of information that will be referenced in following chapters are listed below:

1. BYTEORDER: Byte order of the height values
2. NROWS: Number of rows in the data table
3. NCOLS: Number of columns in the data table
4. NBITS: Number of bits for each height value

The World file contains the rest of the necessary metadata to accurately render and manage the terrain. There are no keywords in this file, the reader must store the values appropriately. The pertinent values that will be referenced in future chapters are listed below:

1. Row resolution value
2. Column resolution value
3. X location of the upper-left corner (in georeference coordinates)
4. Y location of the upper-left corner (in georeference coordinates)

With the data contained in a DEM, the contour of the terrain can be rendered to scale portraying a realistic image of the terrain. Though simply visualizing the terrain is useful, more can be calculated about the terrain with the georeference coordinates that specify exactly where in the world the terrain is located.



## The Shapefile

The “shapefile” is a very common and widely used data layer format. As may can guess, a “shapefile” stores “shapes” and it is with these shapes that geographers represent spatial and geographic data [13]. There are three primitive shape types: Points, Lines and Polygons. Points can be used to, for example, record the locations of objects such as wells or buildings. Lines may be used to represent roads or airways. Polygons can be used to cover an entire area. Multiple polygons can cover multiple areas, perhaps marking state boundaries and thus a shapefile consisting of polygons that represent state boundaries would represent an entire country.

Shapes themselves are powerful visualization tools, but alone are not very useful for computation purposes. Geographers assign “attributes” to shapes in order to specify particular properties between, for example, one state and another state. A geographer may outline a state with one polygon shape and assign it an attribute of 10,000 which represents the state’s population. Another shape can be created and assigned an attribute of 100,000 which also represents population. With the polygon representing state area and one attribute per polygon representing state population, functions can be implemented to determine the relationship of state size to population, for example. Multiple attributes can be assigned to any one shape which increases the potential to determine similarities and differences between shapes.

Points and Lines are very useful shape types, but this thesis does not discuss any functionality involving these types. From this point, any reference to a “shape” will refer to the Polygon type shape. A Polygon shape is merely a collection of points that start and end with the same point. When an application renders a Polygon, line segments are drawn between the points and the interior of the Polygon is shaded to show that it is not a line loop, but a filled Polygon.

We can now discuss the Shapefile itself. As stated earlier, shapefiles store shapes and the attributes associated with them. But like the BIL representation of a DEM, one shapefile is a collection of three files: the file that contains the shapes (**Shp** file), an index file into the **Shp** file (**Shx** file), and the file containing attributes for each

shape (**Dbf** file).

Index files are indicated by the suffix “shx”. This binary file is used to access the shapes in the **Shp** file, therefore the number of shapes in the **Shx** file must match the number in the **Shp** file. The **Shx** file makes it very easy to obtain data for individual shapes, or make small changes without recreating the entire shapefile. Figure 2.2 shows how the **Shx** file points to each shape in the **Shp** file.

The file containing attributes for each shape is indicated by the suffix “dbf” which follows the dBase IV file format [2]. This is also a binary file and because one shape may be assigned multiple attributes, this file stores the attributes in a table. The values in every row in the **Dbf** file are assigned to the corresponding shape in the **Shp** file. That is to say, row  $n$  in the **Dbf** file corresponds to shape  $n$  in the **Shp** file (Figure 2.2).

Attributes are stored as text strings regardless of what they are meant to represent. The **Dbf** file therefore stores a flag in the header for each column to specify how the values in each column should be translated. For example, if a column of values is meant to represent dust potential of the soil in a shape, that value could quite possibly be a floating point value. The **Dbf** file would store each digit as a character and once each character is read in, the entire string must be converted to a floating point number. The dBase IV format supports a variety of types ranging from “Dates” to “Memos,” but the rest of this thesis will only refer to Integers, Floats and Alpha-Numeric Strings.

The **Shp** file contains the actual spatial data, the polygons themselves. As stated earlier, Polygon shape types are a set of points. When the points are connected in order, a concave or convex polygon is created. The format specification states that polygons with self-intersecting lines are not supported [13]. This is not entirely accurate because it is current GIS software that may work improperly with any polygons that contain self-intersecting lines. The **Shp** file itself just stores raw point data.

A simple **Shp** file of polygons will contain header information about the shapes followed by the data pertaining to each shape. **Shp** files are a bit more complicated

because they support “holes” within the shapes. A hole within a shape is an area that does not share the same attributes as the surrounding shape. Holes within shapes are filled in with other shapes stored in the same **Shp** file. Figure 2.2 shows that within the **Shp** file, there are two shapes, 1 and 2. **Shp** 1 has a hole and **Shp** 2 is specified as the shape to fill this hole.

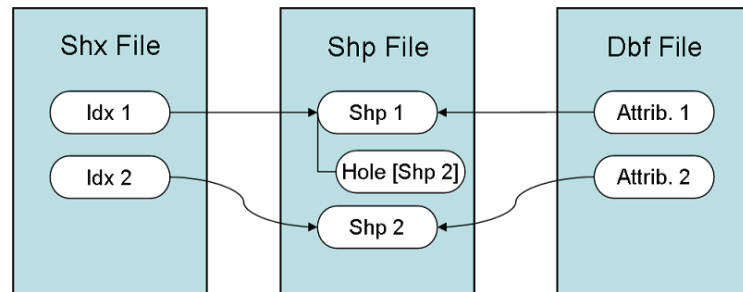


Figure 2.2: Relationship of the Shx, Shp and Dbf files.

It is very important to represent and manage holes correctly in order to obtain accurate computations that will be performed with the shapefile. If a shape stored in the **Shp** file has one or more holes, then the points for each hole are ultimately duplicated in the **Shp** file. Referring back to Figure 2.2, the data in the **Shp** file would appear as follows:

1. Header information
2. Points for Shp 1
  - Points for Shp 2
3. Points for Shp 2

For each shape, the points of the holes are necessary in order to determine the area of the shape that is *not* part of the current shape. For example, in order to render a shape, the portions that are specified as holes must be left open because they are not part of the shape (future chapters examine the function of “fixing” holes to render them properly). Therefore, if a shape is drawn with holes empty, and the shapes that fill the holes are specified later in the **Shp** file, then ultimately holes will be correctly filled in with the correct shape. Figure 2.3 helps illustrate these steps.

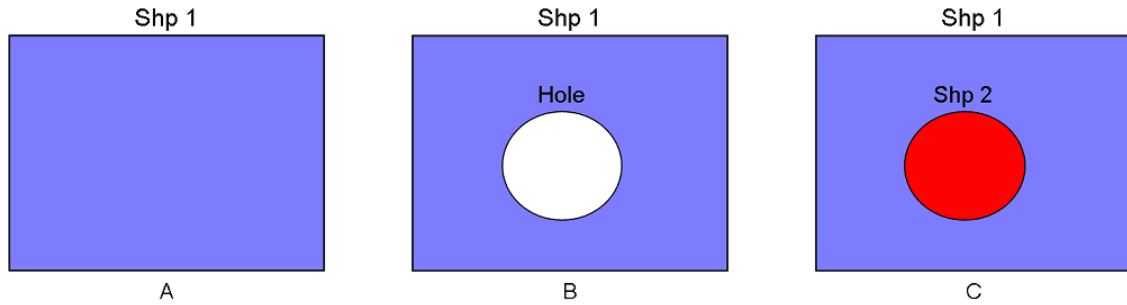


Figure 2.3: Process of drawing a shape (A), erasing the hole (B) and filling it with another shape (C).

Figure 2.3 is a trivial example of a shape containing a hole, however the reader must not underestimate the usefulness of creating shapes that represent useful information. Shapefiles and attributes themselves are vague, it is the context in which they are being used that gives meaning to the shapes and attributes. If the shapes from Figure 2.3 are put in the context of restricted and unrestricted airspace, new meaning begins to appear. Figure 2.4 uses Shp 1 to represent unrestricted airspace and Shp 2 to represent restricted airspace over a key facility (such as the US Pentagon in Arlington, VA. Photo courtesy of Google Earth™ [18]).

The Shp, Shx and Dbf files presented here form the foundation for one individual data layer. A collection of data layers forms the basis of a solid geographic information system, which revolves around the creation and management of spatial data. The practical applications of well managed, accurate spatial data are immense and continue to grow.

The demand for accurate geographic data has spawned the idea of terrain analysis in a virtual environment. This will give expert geographers another level of realism while extracting features from the terrain. We can now discuss the system that will create this level of reality purely through illusion.

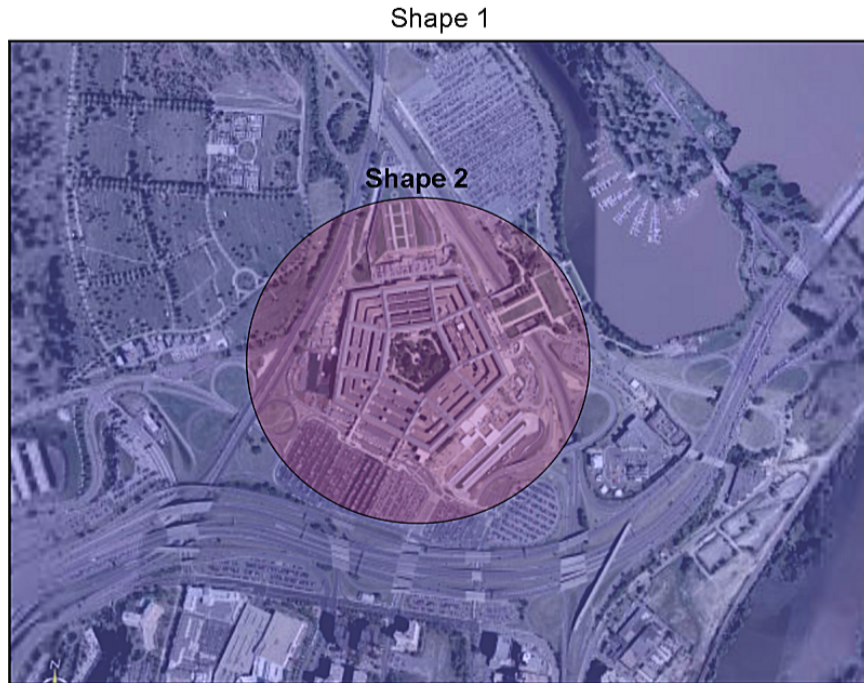


Figure 2.4: Example of potentially restricted air space over a key facility [18].

## 2.2 Virtual Reality

Virtual Reality is a very specialized field of computer graphics. The methods and applications of virtual reality are tremendous [36], thus this section will discuss areas of virtual reality that are immediately related to this project.

The term “Virtual Reality” itself is very broad, referring to the idea of representing something as realistically as possible with computer technology. Playing 3D games on a standard desktop monitor is a type of virtual reality, and indeed with improving physics models, rendering methods and more powerful computer hardware, a desktop monitor will portray very realistic scenes. However, the term “Virtual Reality” is generally used to refer more specifically to an “Immersive Environment”. We will use the term “Immersive Environment” to help portray the idea that the user is not merely seeing a realistic scene, but is immersed into the representation of the environment.

The feeling of being immersed into the environment is achieved by an entire

system that is specifically designed to appeal to the user's senses. These environments today strive primarily to appeal to the visual (sight), aural (hearing) and haptic (touch) senses [3, 9]. Appealing to our aural senses is fairly straight forward. Today's surround sound technology is very realistic with a high quality system, however sound can take away from the immersive feel if used improperly from the context of the application. Appealing to our haptic senses is much more complicated. Immersive environments can use props to create an entire stage the user may interact with. The user may also wear equipment designed to restrict and allow muscle movement. However, this project does not focus on appealing to haptic senses so little discussion will follow related to this topic (the reader may learn a good deal more from [3], [9], and [36]).

The goal of the project presented in this thesis is to appeal to the user's visual senses, particularly those that perceive depth. An excellent design for creating depth illusions has been developed over the years which creates a firm foundation for all immersive environments. But before we can discuss how the system is designed to trigger our depth perception, we must have an understanding of how humans perceive depth.

## **Depth Cues**

Many people will attest that we see in 3D. This is not entirely accurate because the mechanics of our eyes gather light on the retina, a two dimensional surface [29]. The process of creating a three dimensional scene takes place after our eyes gather light information. It would be more accurate to say that we "see" in two dimensions and "perceive" in three dimensions. Precisely how our eyes gather light is not in the scope of this project, but how we interpret two dimensional data from the retina into a three dimensional scene is directly related to this project because immersive environments are designed to appeal to the user's depth perception. Once we know how this works, we can understand how to appeal to it (covered in the next section).

Items that we use to gather depth information from an image are known as Depth

Cues. Depth cues help us answer the question, “How do I know object  $A$  is further away than object  $B$ ?” There is no single answer, but a combination of answers that vary depending on what the person is viewing. We will discuss three key categories of depth cues: motion, monoscopic, and stereoscopic depth cues.

Depth cues we receive through relative object motion are also referred to as “motion parallax”. The idea is very simple: as we move through an environment, the movement of objects is a function of proximity. The best example is while traveling in a vehicle, close objects move past the vehicle quickly and objects further away appear to have almost no motion at all. If the image the viewer is looking at changes in an organized manner, motion depth cues will be present. However when viewing a photograph, motion cues are not present, viewers must use other cues to perceive depth in the image.

Monoscopic depth cues can be described in two ways. They may first be described as the only depth cues present when we look at our environment with one eye, that is, if the viewer is stationary. Even with one eye, motion depth cues are present if the viewer or environment gradually changes position. To be a bit more general in the description of monoscopic depth cues, we can say that they are the only cues present when viewing an image presented on a flat surface. Flat surfaces include printed photographs, computer monitors, even the retinas in our eyes. Monoscopic depth cues are always present, whether working in a real-world environment or working with a desktop monitor, whether looking with one eye or two eyes. The following list describes several types of monoscopic depth cues with a brief description. This list is only meant to provide examples of monoscopic depth cues, more information may be found in [32]:

1. Occlusion: When one object is covering another object, we say it is “occluding” that object. This tells us that the occluded object is farther away.
2. Familiar Size: When an object is familiar to us, we compare the perceived size to the expected size. We can therefore determine that the smaller an object appears, the farther away it is.
3. Texture Gradient: We see more detailed textures of objects closer to us. As an object’s texture decreases in detail, we presume it is farther away. Atmospheric

conditions such as fog or haze will increase the rate at which texture detail diminishes, thus creating the illusion that objects are actually farther away than they really are.

4. Linear Perspective: The ever popular examples of looking down a road or train tracks accurately describes this type of depth cue. Two parallel lines will appear to converge in the distance, which helps us assess the distance of objects positioned near the parallel lines as well.
5. Shadows: Shadows are an extension of Occlusion however there are many factors that account for the creation of shadows. If these factors are unknown to the viewer, illusions will occur (we will see an example of a powerful illusion due to shadows in following sections).

We can now describe stereoscopic depth cues, perhaps more popularly referred to as “binocular vision”. Stereoscopic depth cues require the viewer to look at an actual three dimensional environment, are those that require two eyes. These cues rely on our ability to calculate the differences and similarities between the two images captured by our eyes. Our eyes capture two different images because they are offset from one another which is called “binocular disparity”. Stereoscopic depth cues are very powerful and we learn to rely on them for many of our daily activities. However these cues are only reliable within a certain distance, roughly six meters. Outside of this range, stereoscopic cues are ineffective because the binocular disparity is so minute, the differences between the two images is insignificant. We must then rely on other depth cues, such as motion parallax and monoscopic cues, to determine distance.

With this brief understanding of how we perceive depth in our environment, we can now explain exactly how to design an environment, a Virtual Environment, meant to appeal to our depth perception.

## **Environment Design**

The term “Virtual Environment” is used to refer to the entire system of hardware and software that is orchestrated together to create an immersive environment. An environment may be a head-mounted display as shown in Figure 2.5, or an elaborate projection based display as shown in in Figure 2.6. The project presented in this thesis



is designed for use in a projection based system to allow the user to feel physically immersed in the environment.



Figure 2.5: Head mounted VR display [36].

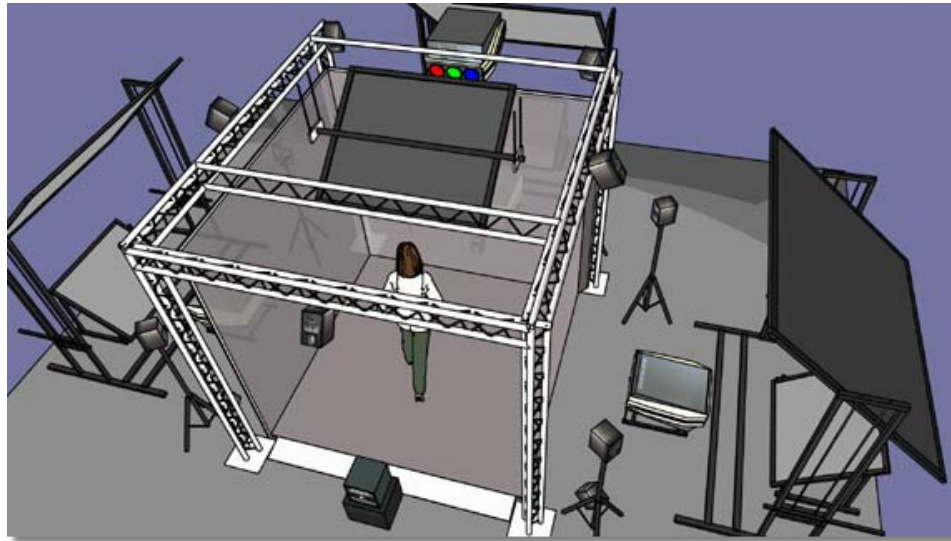


Figure 2.6: Projection based VR display [36].

Projection based display systems are large, stationary environments and as can be determined from Figure 2.6, require a carefully built system. The user in Figure 2.6 is surrounded by four projection screens (left, right, front and bottom), thus this type

of environment is referred to as a four-wall environment. The minimum requirements to create a single-wall, projection based virtual environment are listed below:

1. One rear projection screen
2. Projector(s):
  - (a) One projector for active stereo
  - (b) Two projectors for passive stereo, with light filters
3. Glasses:
  - (a) Shuttering glasses for active stereo
  - (b) Filtering glasses for passive stereo

Optional items shown in Figure 2.6 that are not required to visually immerse the user into the environment, but are fairly standard with projection-based displays are listed below:

1. Mylar (or front faced glass) mirrors
2. Surround sound speaker system

As mentioned earlier, immersion with sound technology is not in the scope of this project. Sometimes the mylar mirrors are required because of space limitations. The larger the projection screens, the more distance is required between the screens and the projectors, and the mirrors help to reduce this distance.

The user will only see the display on the projection screens. Having large display screens fills the user's field of view and allows the user to move about the environment freely. With multi-wall environments, the ability to join screens together makes the user feel even more immersed into the environment because the user may now freely look to the left, right, even top and behind in six-wall environments, to see what is displayed in that relative direction. With fewer projection screens, the user is forced to control the camera to change the looking direction, which takes away from the immersive experience.

We can now determine exactly how the virtual environment appeals to our depth perception. It is a combination of the projectors and the appropriate glasses for the

projectors. Remember in the previous section regarding depth cues that we can already gather depth information on a standard desktop monitor with motion (that is, when the 3D application moves the virtual camera) and monoscopic cues. The projectors and user glasses are used solely to trigger our stereoscopic depth cues. The projectors create what is simply referred to as a “stereo” projection, and the glasses utilize the stereo projection to trigger our depth cues.

The goal is to simulate the data our eyes would gather in a real-world environment. As mentioned in the previous section, the offset of our eyes causes two different images to be sent through the image processing pathways of our brain. We then calculate differences and similarities between these two images to interpret depth information. The virtual environment forces our eyes to see two different images of the same computer generated scene. There are two methods used for sending these images to the appropriate left or right eye which are known as “passive stereo” and “active stereo”.

Passive stereo requires two projectors to display the scene. One projector is responsible for displaying information for the left eye, and the other projector displays information for the right eye. Since both of these images show directly over each other on the projection screens, filters must be used to distinguish what image should be sent to the left and right eyes. The glasses the user wears will filter the light so only light meant for the left eye is received by the left eye, and likewise for the right eye. But to determine exactly what light is meant for the left and right eyes, it must be filtered as it shines from the appropriate projector. Passive stereo projections are generally imperfect because filtering the light precisely is incredibly difficult. Passive projections are excellent for general demos and testing purposes.

Active stereo projections help diminish the imperfections present in passive projections. Active projections use only one projector, so no filtering is required. However these projectors are quite expensive because they must refresh at twice the rate of normal projectors. Now, only one projector is responsible for displaying images for the left and right eyes. The glasses are known as “shuttering” glasses because they

essentially “open” one eye to allow light to pass through while “closing” the other eye. The glasses are synchronized with the projector so light intended for the left eye passes through an “open” left eye slot in the glasses, and not through the right eye slot. When the projector refreshes, the new image is intended for the right eye, thus the glasses will “shutter,” now opening the right eye slot and closing the left.

Both passive and active stereo projections ultimately create an illusion which triggers our stereoscopic depth cues. Users now feel as though the scene truly has depth because it is no longer perceived as a flat surface. A well designed stereo view is the first step to immersing a user into a virtual environment. How the system receives input from the user and how the user interacts with the system are two significant topics required to create a deeper sense of immersion. These topics are covered briefly, staying within the scope of this project.

## **Input and Immersion**

Input refers to any means of sending information to the computer system managing the immersive environment. Input must be sent to the computer in order for the environment to respond appropriately to the user’s actions.

Common input devices such as joysticks and joypads are often present in immersive environments. These are a form of physical control devices. They serve the purpose of being intuitive devices that take little time to learn. The result of input from these devices is evident based on the actions of the objects being displayed. For example, if a joystick were responsible for navigation controls, then leaning one of the control sticks left may turn the camera to the left, or move the camera left. Either way, the feedback is instantaneous which is crucial for fluid interaction [37].

These devices also serve the purpose of performing tasks that we are otherwise incapable of accomplishing. For example, the user may be immersed in a real-world representation of a large landscape and wish to move quickly from one side of the large landscape to the other. One may think that a simple joystick is the perfect device to navigate over large areas, however devices such as these do not create the

sensation of being immersed in the environment like other methods do.

These environments have created a new realm of interaction. Because desktop interaction methods and devices are so common, it is difficult for one to think of what it means to interact immersively in virtual environments. This introduces the idea of “User Monitoring” in order to more fully immerse the user into the virtual environment. The idea of monitoring the actions of the user gives light to what it means to be immersed into the environment. Sherman and Craig [36] give several definitions of immersion:

**immersion** sensation of being in an environment; can be a purely mental state or can be accomplished through physical means: physical immersion is a defining characteristic of virtual reality; mental immersion is probably the goal of most media creators.

**mental immersion** state of being deeply engaged; suspension of disbelief; involvement.

**physical immersion** bodily entering into a medium; synthetic stimulus of the body’s senses via the use of technology; this does not imply all senses or that the entire body is immersed/engulfed.

These definitions help us understand that we often mentally immerse ourselves into what we enjoy such as literature, music and cinema. We are always physically immersed in the world in which we live, but the essence of a virtual environment is to immerse a user into a world that is not accessible, or perhaps not even real.

Monitoring the actions of the user allows us to create environments that will respond to those actions. Many devices and methods are available to track the user, but the one method we will introduce is “head tracking”. Remember that the goal of this project is to appeal to the user’s visual senses, and a virtual environment with a well designed stereoscopic view will appeal to the three types of depth cues mentioned previously.

Head tracking creates a new way for the user to receive motion depth cues. If

the user were to use physical control devices, then the world may be altered with this device and thus the user will effectively receive motion depth cues. However, head tracking not only relieves the need for controls to view the world, but creates a deeper sense of immersion for the user. While the user moves freely around the virtual environment, the system will respond by appropriately adjusting the viewpoint to match that of the user. Figure 2.7 shows four views of a user within a virtual environment utilizing a head tracking system. The different views show that the rendered objects “move” relative to the user as one would expect if placed in a similar real-world situation.

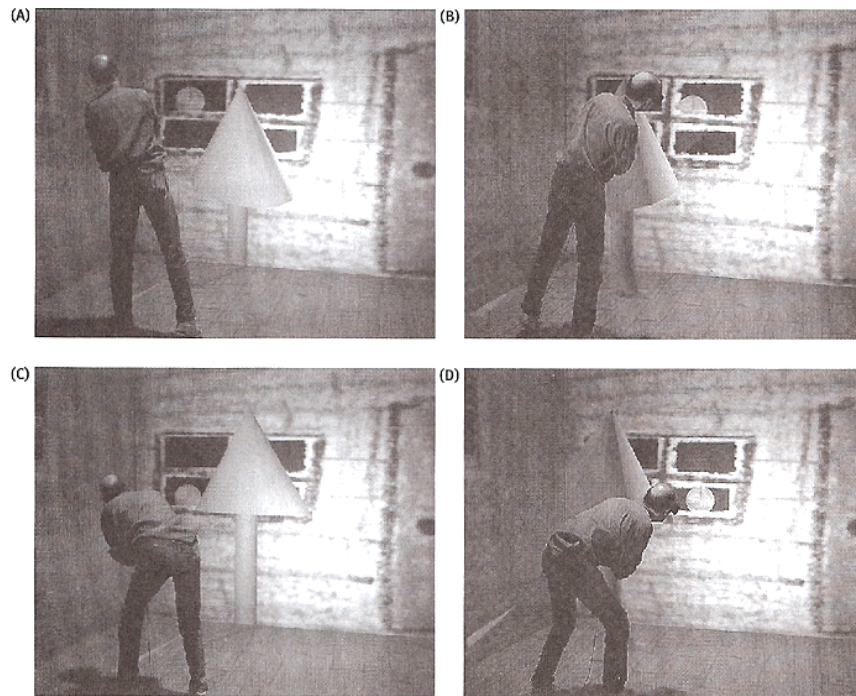


Figure 2.7: Virtual objects move relative to the user utilizing a head tracking system [36].

With proper head tracking, the user is completely unaware that input is being supplied to the system, and the foundation for a physically immersive environment is now in place.

## 2.3 The Desert Terrain Project

The Desert Research Institute in Reno, Nevada has undertaken the task of modeling and forecasting the conditions of desert landscapes [20, 21, 22]. There are two key components of this project:

1. Forecast desert regions
2. Predict the effects of the region on equipment and personnel

The project intends to forecast and model desert regions as accurately as possible using data gathered via remote sensing techniques. As mentioned previously, current GIS software does not allow analysts to create representations of terrain in three dimensional view which will cause the accuracy of the analysis to suffer.

This thesis introduces an application designed to give expert geographers the ability to analyze terrain in a three dimensional view. However, since accuracy is crucial for this project, the application will also allow the user to analyze terrain in a virtual environment.

## 2.4 Need for 3D

### Depth Illusions

We are subject to a significant illusion when viewing aerial photographs. As mentioned briefly in the section regarding depth cues, shadows are a powerful method for determining distance in an image. However, we make assumptions when we use shadows to determine depth, assumptions such as the number of light sources, and the location of the light sources. If this information is not presented to the viewer, then it is entirely possible that the viewer will make incorrect presumptions of height and depth areas in an image.

Figure 2.8 is a landscape of the surface of Mars showing several lava flows on the terrain. Many other planetary photos such as these may be viewed at the online NASA Photo Journal [30]. Having read that the elevated areas are in fact lava flows, your assumptions of light number and source are confirmed.



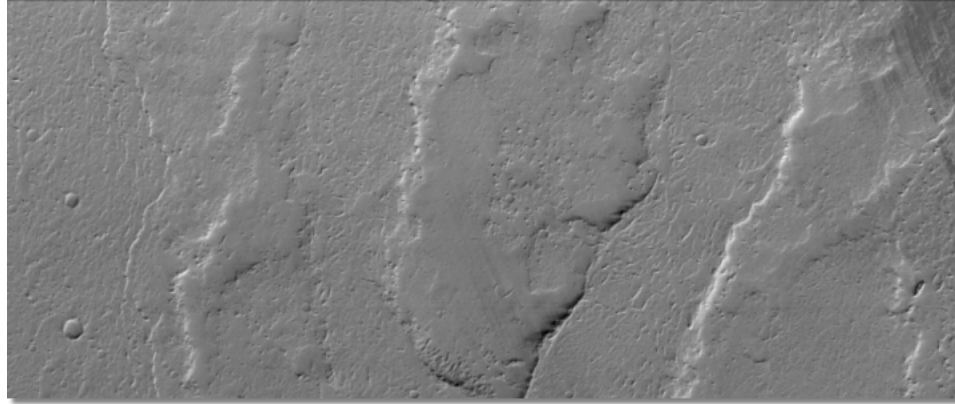


Figure 2.8: Lava flows on the surface of Mars [30].

Now we present the illusion. The assumptions of light number and source have already been confirmed and thus will remain in place when a similar image is viewed. Figure 2.9 portrays the very same image from Figure 2.8, however it has been rotated 180 degrees. If this classic illusion has taken effect, the reader should now see depressed areas where there were elevated areas in the previous figure [32]. Though expert geographers are generally very familiar with the terrain they are analyzing, they too may suffer from this illusion. This is strong support for performing analysis while viewing the terrain in a three dimensional view.

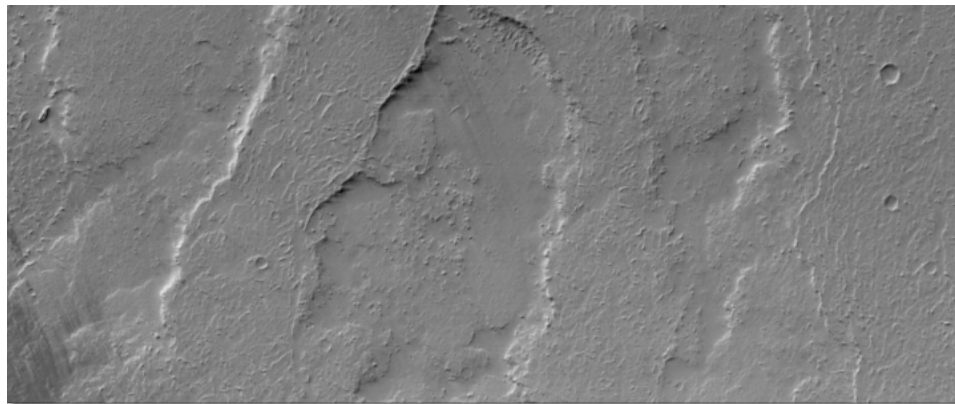


Figure 2.9: Rotated view of the lava flows on Mars [30].



## Desert Terrain

The Desert Terrain project is particularly interested in a terrain feature called an Alluvial Fan. Figure 2.10 portrays a classic example of an alluvial fan [28, 41]. Alluvial fans are created as water flow through a canyon releases sediment at the opening of the canyon. Several aspects of the project require close examination of these fans which can be very difficult to locate on terrain rendered in 2D. A 3D rendition of the terrain greatly increases the geographer's ability to locate areas where alluvial fans would typically be located, and help the geographer more accurately distinguish the fan itself.



Figure 2.10: Classic alluvial fan [41].

## Chapter 3

# Requirements Specification and Modeling

As we have seen from Chapter 2, in order for the Desert Terrain project to be a success, expert geographers need to be able to analyze terrain as accurately as possible. The most accurate method for analyzing terrain is by going to the location and starting the process of extracting features from there. Going to the site location is simply not an option at times, so geographers must extract features from data gathered via remote sensing techniques.

To give expert geographers the ability to extract features from a given piece of terrain as accurately as possible, without sending the expert to the location, this thesis proposes an application named Taverns. The name is an acronym that stands for “Terrain Analyzing in a Virtual Environment with Real-time Native Shape Creation”. The reader should now have a fairly clear understanding of the title and the purpose of the application with the background presented in Chapter 2.

This chapter will formally introduce Taverns by specifying the requirements, specifications and modeling characteristics for the application. Because Taverns is a prototype still in the beginning stages of development, we follow common software engineering practices while developing the application [1, 40]. The following sections present core elements that create a solid foundation for Taverns that shall remain in place during the development process.

### 3.1 Functional Requirements

The motivation for Taverns comes from the specialists at the Desert Research Institute as a large step in the Desert Terrain project [20, 21, 22]. It is therefore important for the developers to meet the needs of these specialists. Certain requirements have been created for Taverns which ultimately lead to the ability to analyze terrain rendered in 3D. Below are listed the minimum requirements for the application that make this final goal possible:

1. Taverns shall load and display GIS data layers
2. Taverns shall display the GIS data layers in 3D
3. Taverns shall allow the user to free fly around the 3D world
4. Taverns shall allow the user to toggle between data layers being displayed
5. Taverns shall have a wire frame view to more easily see the contour of the terrain
6. Taverns shall allow the user to draw shapes on the 3D terrain
7. Taverns shall provide undo and redo capability when creating shapes
8. Taverns shall allow the user to edit attributes about the created shapes
9. Taverns shall save the created shapes in the GIS standard shapefile format
10. Taverns shall save the created shapes in its own format
11. Taverns shall load both its own format and shapefile format to continue working
12. Taverns shall function in a desktop environment and virtual environment

### 3.2 Non-Functional Requirements

The developers have created other requirements for the application that fit more appropriately in the category of non-functional requirements. Requirements such as these specify how the application will be implemented and place constraints on the system. Much research in the field of human-computer interaction focuses on the seamless usability of an application and as such, these non-functional requirements should be nearly invisible to the user [37].

1. Taverns shall be written in C++

2. Taverns shall render objects with OpenGL
3. Taverns shall be cross platform compatible
4. Taverns shall provide pertinent feedback in the HUD (Heads Up Display)
5. Taverns shall be as automated as possible
6. Taverns shall allow the user to create shapes in any order
7. Taverns shall be highly configurable
8. Taverns shall receive input from a variety of different devices
9. Taverns shall format the shape and attribute information before it is saved

### **3.3 Primary Use Cases**

Use cases are a very powerful method for portraying essential items the application is responsible for implementing and supporting. Well planned use cases are also the basis for determining what classes and modules will be required during the implementation process. The diagram shown in Figure 3.1 portrays the intended users of the application, and where the output of the application shall be directed. Both of these pieces of information are important to establish for the development process. If intended users, use cases, and output direction have not been clearly established early in the development process, inconsistencies and flaws will develop resulting in an application that may be unusable, or produces flawed output. Taverns is a piece of a much larger project, therefore the effectiveness of this application will significantly influence the success of the entire project.

### **3.4 Requirements Traceability Matrix**

With established functional requirements and established use cases, we create a requirements traceability matrix to associate the requirements and use cases together. It is very important to map these together because the users will expect functionality from the application, but in the way of use cases. Similarly, the developers will provide use cases for the users, but by means of the functional requirements. Table 3.1

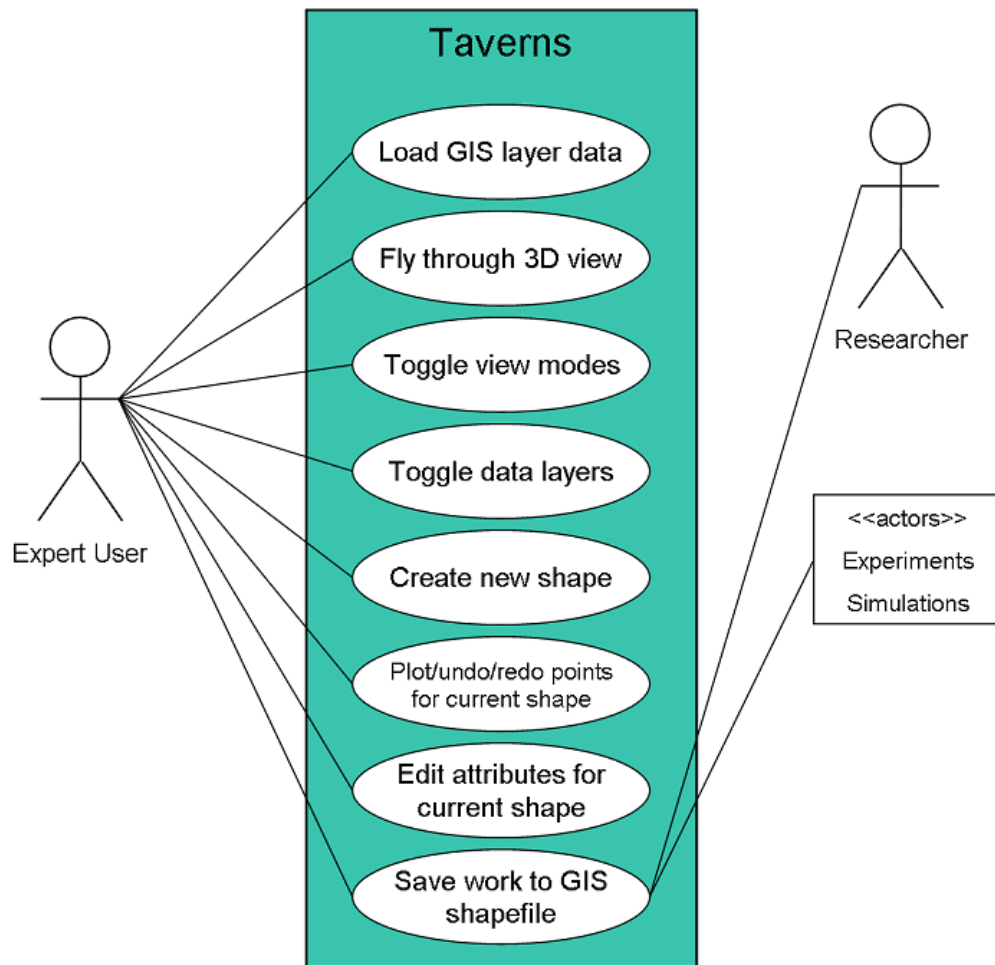


Figure 3.1: Taverns primary use case diagram.

maps the use cases that the user will expect from the program to the functional requirements the program must fulfill.

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8
FR1	X							
FR2		X						
FR3		X						
FR4				X				
FR5			X					
FR6					X	X		
FR7						X		
FR8							X	
FR9								X
FR10								X
FR11								X
FR12	X							

Table 3.1: Functional requirements (Section 3.1) mapped to use cases (Section 3.3).

### 3.5 UML Scenarios

We can combine the requirements and use cases set forth in previous sections to create potential scenarios that the user will encounter while using the application. Scenarios help the developers determine how the user and the application will interact with each other through the duration of the program. Possible errors and improvements may then be determined before the development process has progressed too far, making changes to past work incredibly tedious.

The scenarios presented in this section are in common UML form to portray the typical usage of Taverns. The full scenario of loading data, using the data to create a shapefile, and using the created shapefile for further studies is presented in Table 3.2. The two most plausible secondary scenarios, which are possible deviations from the primary scenario, are described in Tables 3.3 and 3.4.

<b>Scenario: Create a new shapefile.</b>
<b>Actors:</b> Expert User
<b>Requirements:</b> <ol style="list-style-type: none"> <li>1. At least one GIS data layer loaded successfully.</li> <li>2. At least one shape has been created to save.</li> <li>3. Paths to save to specified in the configuration file.</li> </ol>
<b>Primary scenario:</b> <ol style="list-style-type: none"> <li>1. The user will specify the data layers to load in the configuration file.</li> <li>2. The user will specify the files to save to in the configuration file.</li> <li>3. The system will read the configuration file and check for errors.</li> <li>4. The system will load the data layers and store them in memory in the order specified in the configuration file.</li> <li>5. The system will display the first data layer in 3D.</li> <li>6. The system will wait for user input.</li> <li>7. The system will respond to user input according to the use cases as described in Section 3.3.</li> </ol>
<b>Secondary scenario:</b> <ol style="list-style-type: none"> <li>1. The user creates no shapes.</li> <li>2. The user creates some shapes, but not all have been assigned attributes.</li> </ol>
<b>Postconditions:</b> <ol style="list-style-type: none"> <li>1. The created shapefile is ready for use in further studies.</li> </ol>

Table 3.2: Primary scenario, create a new shapefile.

<b>Scenario: The user creates no shapes.</b>
<b>Actors:</b> Expert User
<b>Requirements:</b> <ol style="list-style-type: none"> <li>1. At least one GIS layer loaded successfully.</li> </ol>
<b>Secondary scenario:</b> <ol style="list-style-type: none"> <li>1. The user will press the key command to save work.</li> <li>2. The system will detect if any shapes have been created.</li> <li>3. The system will find that none have been created.</li> <li>4. The save operation will terminate.</li> </ol>
<b>Postconditions:</b> <ol style="list-style-type: none"> <li>1. No new files will be created.</li> </ol>

Table 3.3: Secondary scenario, the user creates no shapes.

<b>Scenario: The user creates some shapes, but not all have been assigned attributes.</b>
<b>Actors:</b> Expert User
<b>Requirements:</b> 1. At least one GIS layer loaded successfully.
<b>Secondary scenario:</b> 1. The user will press the key command to save work. 2. The system will detect if any shapes have been created. 3. The system will find that some shapes have been created 4. The system will find that the number of shapes and number of attributes do not match. 5. The system will assign empty string attributes to the shapes without any assigned attributes. 6. The save operation will continue as normal.
<b>Postconditions:</b> 1. The user resumes work uninterrupted.

Table 3.4: Secondary scenario, not all shapes have assigned attributes.



# Chapter 4

## Implementation Design

With the requirements, specifications and modeling characteristics in place, we can discuss the methodology for fulfilling the functional requirements presented in Section 3.1, thus providing the users with the use cases presented in Section 3.3.

It is important to remember that Taverns must be as portable as possible. For this reason, it will make use of as few toolkits and packages as possible. Taverns could make use of packages such as Demeter (terrain manager) [12], or Open SceneGraph (scene graph) [6] however these toolkits, as toolkits are designed to do, alleviate responsibility from the programmer and take control of tedious tasks such as camera management, lighting calculations, and model loading. As advantageous as these toolkits are, they typically involve different setup methods per operating system, that is, if they are available for multiple operating systems. They also may take control of some aspects of the application that the programmer must have control of to make the application usable in a virtual environment. Open SceneGraph is an example of a package that may take control of the camera. This makes it very difficult to use the application in a virtual environment. Fortunately, there are also toolkits available specifically for displaying an application in a virtual environment such as FreeVR [35] which will be discussed in the next section.

Section 4.1 first presents the class structure of Taverns, along with explanations of pertinent algorithms in the appropriate class. Section 4.2 examines our efforts to balance the functionality and intuitiveness of Taverns on both desktop and virtual environments.

## 4.1 Architecture and Algorithms

Taverns is a wonderfully object oriented application. Figure 4.1 portrays the high-level structure of Taverns with a description of each module following the figure. Taverns is still a prototype, so it does not implement any advanced optimization algorithms. However there are several computational geometry algorithms that will be described in the section that implements the algorithm [11, 31].

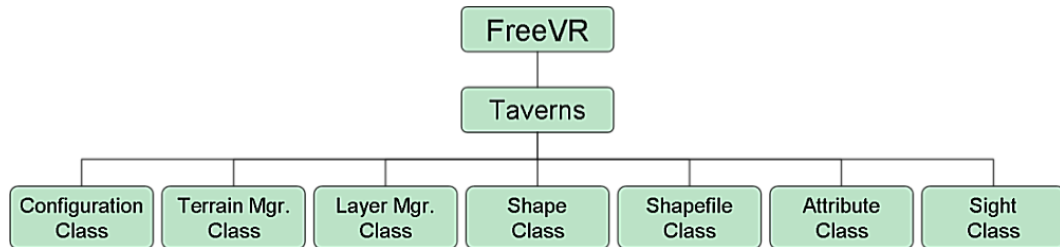


Figure 4.1: Taverns class structure.

### FreeVR and Taverns

FreeVR is a software toolkit designed for managing Taverns while running in a virtual environment [35]. Within the scope of this project, FreeVR provides functions for two important aspects of virtual environments as mentioned in Chapter 2: user input and camera management.

Remember from Section 2.2 that input may be from physical control devices and from tracking devices. FreeVR accepts input from both devices but not all input is sent to the same location. The input from control devices such as joysticks and joypads is sent immediately to Taverns. Once Taverns receives the input from FreeVR, it must respond appropriately. Taverns manages the sub-classes which collectively creates the functional application.

In terms of tracking devices, this project will only focus on head tracking for the time being. FreeVR will receive input from head tracking devices and use this to adjust the camera to match the perspective of the viewer.

Section 2.2 also describes the projectors used in virtual environments and how they must display images for both our left and right eyes. It is FreeVR that coordinates the images sent to each projector. Though Taverns is responsible for rendering the virtual world, FreeVR is responsible for placing the camera in the virtual world to not only match the perspective of the viewer, but alternate positions which simulates the views for our left and right eyes.

FreeVR itself has an extensive configuration file. It allows us to configure two key components, the first of which is the number of walls in the virtual environment the application will run in. Secondly, we must specify whether the environment is a passive stereo or active stereo projection system. FreeVR will manage the camera appropriately for either case. Figure 2.7 demonstrates how the user can move in the virtual environment and FreeVR will update the virtual camera position accordingly (that is, if a head tracking system is being utilized).

With the use of FreeVR, the implementation of the following sections is unaffected. The configuration file allows us to specify a large amount of parameters specific to individual virtual environments, and FreeVR will work according to these parameters.

### **Configuration Class**

Taverns must be versatile. For any given sample of terrain that is to be analyzed, the amount of current data about the terrain varies. The expert user may wish to use a large amount of current data to extract more features. Or, the user may wish to simply start with the terrain itself and extract features. The most practical way to obtain this versatility is with a configuration file. Automating, validating and error checking the data specified in the configuration file is all done through this class. Taverns requires certain pieces of information before it will allow the user to begin work. Figure 4.2 displays the flow of loading the configuration data and sending that data to the appropriate class.

First and foremost, the user must provide elevation data. One source DEM file

(as described in Section 2.1) must be specified. The DEM data alone is not very useful without a data layer to texture over the rendered terrain mesh, therefore the user must also specify an image file to be textured over the terrain (most often an aerial photo but the user may specify any image to be textured). The texture specified with the elevation data is the first data layer the user will see when the display loads. Finally, once the data is validated, it is sent to the terrain manager class.

Next the user may specify data layers that will be used to extract features from the terrain. Taverns does not require any data layers because the amount of data already gathered regarding a sample of terrain varies. Quite often the only data geographers have is elevation data and an aerial photo and the process of extracting features begins there. If a layer is specified, the user must specify a shapefile, an attribute file (Section 2.1), and an image texture of the shapefile. The user may specify any number of layers to load and Taverns will allow the user to toggle between them while creating shapes. Once validated, the layers are sent to the layer manager class.

In order to load and save work done, the user must specify the paths to the appropriate shapefiles to load from and save to. These are data layers just like the ones mentioned above and the configuration class will read, validate, and send them to the layer manager appropriately. When the configuration class sends these layers, it must specify to the layer manager that they are layers to load from and save to and it is the layer manager's responsibility to manage them appropriately.

Lastly, the user must specify a small amount of information regarding the computer environment Taverns will run on. The user must first specify if the application will run on a desktop environment, or in a virtual environment. If it is to run on a desktop environment, the user must also specify whether to run the application in full screen mode or not. This section is highly expandable due to the potential for other items to be added later, for example different input devices.

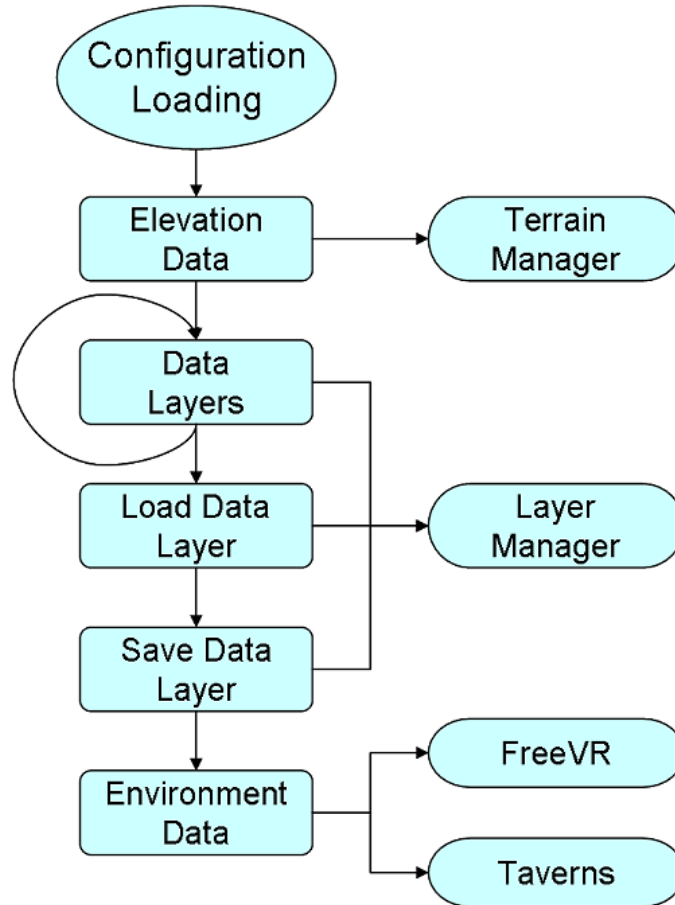


Figure 4.2: Loading configuration data.

### Terrain Manager Class

Accurately and efficiently managing the terrain is crucial for many reasons. For one, the user wants the application to run smoothly. Rendering the terrain inefficiently will cause a significant drop in the refresh rate. Only simple rendering algorithms are currently implemented for the prototype application, however this class will be responsible for the future optimizations described in Chapter 5.

More important than rendering the terrain efficiently is maintaining data accuracy. A solid foundation for data accuracy must be established. If the terrain is rendered improperly, it is quite possible that inconsistencies will develop. If flaws evolve between the output of Taverns and current GIS software, the entire point of

the application is defeated.

The most reliable way to maintain consistency is to render the terrain at the very same location specified in the header of the DEM file. As described in Section 2.1, the header of a DEM file specifies the upper-left location of the terrain in georeferenced coordinates, and all of the necessary information to render the terrain to scale. If we render the terrain to scale starting from the upper-left location, then all of the shapes the user creates will be placed at the very location the user expects them to be and no calculations are required to move them to the appropriate georeferenced location. If future developers alter the terrain in any way, perhaps scale the terrain or render it in a different location, then the shapes must be scaled and/or moved appropriately so as to appear over the same location when loaded in current GIS software.

### **Layer Manager Class**

As mentioned in Chapter 2, remote sensing techniques gather a wide variety of data and the preferred method for viewing this data is in layers. The layer manager is responsible for organizing these textures into those that the user will:

1. View while creating shapes
2. Load shapes from to edit
3. Save edited shapes to

If the paths to save work to exist, they will be overwritten. If the paths to the load work from exist, the work is loaded, otherwise an empty work set is created. The same files to load from and save to may be specified supposing the user wants to continue working on a set of shapes. Taverns does not keep handles to the files open, all of the data is stored in memory and only written to file when the user chooses to save the work.

Other than elevation data, shapefiles are the only layer format Taverns currently supports. In order to view this data on the rendered terrain, an image of the shapefile is created and that image is textured on the terrain. The user may specify an image

to use as the texture for the data layer which will save time in the loading process. If no texture is specified, the layer manager will create one.

Because the header information of a shapefile contains the bounding box for the shapes, the texture making algorithm that the layer manager currently has implemented begins at the top-left part of the bounding box and scans left to right, top to bottom. A pixel is written to the image file according to the shape each pixel is contained within. But each shape has one “key” attribute that determines which shapes are similar and which are different. So before the scanner can begin, each unique “key” field of the attribute table is assigned a random color (checking first that the color is not already chosen) so all shapes that share the same “key” value are colored the same. Figure 4.3 helps visualize the algorithm described in Figure 4.4.

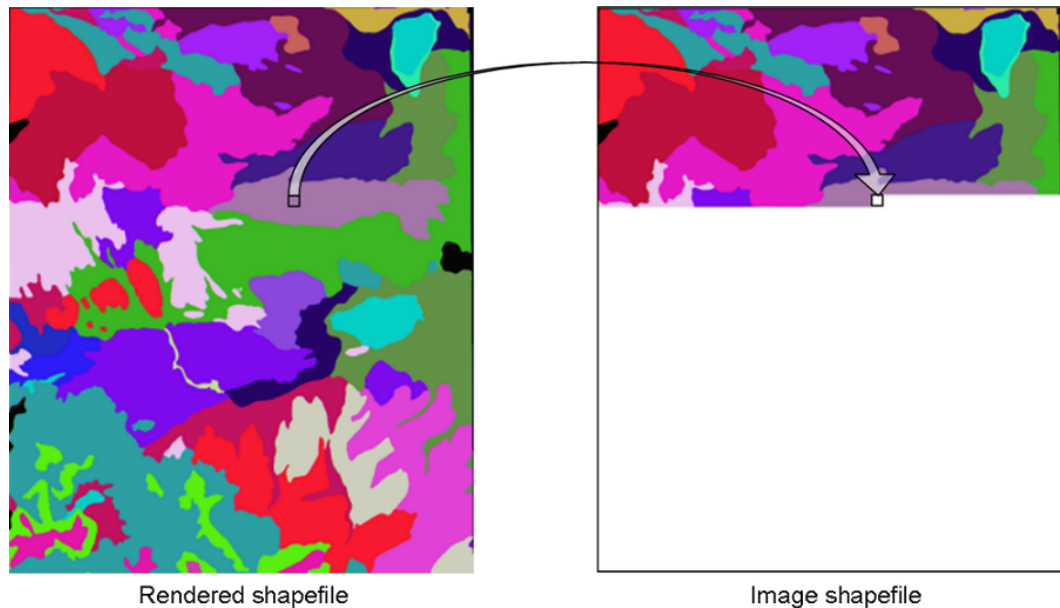


Figure 4.3: Creating an image texture from a shapefile.

From Y Max.  $\rightarrow$  Y Min.  
 From X Min.  $\rightarrow$  X Max.  
 $s$  = The shape point (x, y) is in  
 $k$  = The key field value for shape  $s$   
 $c$  = The color for key  $k$   
 Write color  $c$  to the image

Figure 4.4: Algorithm for creating a texture of a shapefile.

## Shape Class

Shapes themselves are relatively simple, but there are many computational geometry functions that apply directly to shapes which warrants a subsystem directly for them.

The most often used algorithm Taverns implements is one that tests if a point is within a shape or not. It is a simple algorithm, but very crucial to the entire function of the application. In order to test if a point lies within a shape, we create an imaginary vertical line from the point and ending past the highest point of the shape. We then count the number of times this newly created line segment intersects the border of the shape. This algorithm therefore relies utterly on the ability to detect if two line segments intersect. Such an equation is presented in the popular “Introduction to Algorithms” textbook [8]. If the number is even, then the point lies outside the shape. If the number is odd, then the point lies inside the shape. Figure 4.5 shows examples for several point locations with circles drawn around the intersection points. One may quickly notice that because this algorithm works with concave shapes, it will work equally well with convex shapes.

Following this algorithm is one that determines if a shape lies within another shape. This is a simple extension of the one above. The algorithm must be given two shapes, a “main” shape and a “hole” shape. This will return true if the hole shape lies within the main shape, or false otherwise. A shape lies within another shape if all of its points are inside the main shape. As soon as one point is found that is not inside the main shape, the algorithm will return false. Figure 4.6 shows three test



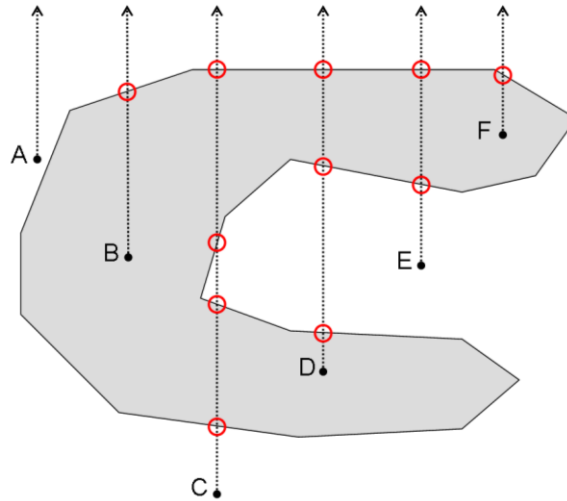


Figure 4.5: Example point locations around a shape.

cases of a “main” shape and potential “hole” shapes. The algorithm will return true for hole A because all of its points are inside the main shape. False will be returned for shapes B and C because not all of their points lie inside the main shape.

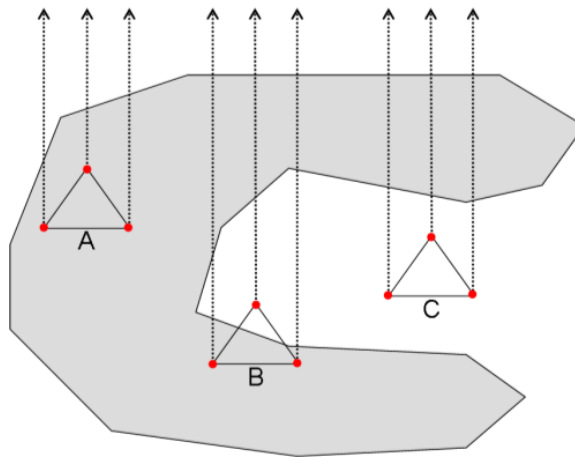


Figure 4.6: Potential “hole” shapes inside a “main” shape.

As described in Section 2.1, shapes and holes are stored as a collection of points within shapefiles. What the section did not mention is that the points stored in the shapefile must be ordered a particular way. We will refer to the order of the points as “clockwise” or “counterclockwise” however these terms are not entirely accurate, we

use them for brevity purposes. For each shape in the shapefile, the points must be ordered in a clockwise fashion. The shapefile format gives a definition for a clockwise shape:

The neighborhood to the right of an observer walking along the ring in vertex order is the neighborhood inside the polygon [13].

To put it another way, if we create line segments between all of the points, starting from the first point through the last point, the area to the right of each created line segment is the interior of the shape. This method helps ensure distinct designation between shapes and holes within the shapes. The points of holes must be ordered in counterclockwise fashion. The specification document need not give a definition for counterclockwise order because it is quite the opposite of the clockwise definition.

With the points of shapes specified in a clockwise and the points of holes specified in a counterclockwise fashion, it is easy to distinguish shapes from their corresponding holes. The next step is to determine whether a set of points is specified in clockwise or counterclockwise order.

Our Shape class determines whether a shape is drawn “clockwise” or “counterclockwise”. See Figure 4.7 for an example of the same shape drawn clockwise and counterclockwise. We can determine whether a shape is clockwise or counterclockwise by first finding the point with the maximum Y value. If there are multiple points level with each other, we must find the point with the lowest X value. We end up with the top-most left-most point of the shape. It is from this point that we begin to test whether the shape is clockwise or counterclockwise.

We must now find the “next” point and the “previous” point and this is relative to the order of the points. So in the left shape in Figure 4.7, the top-most left-most point is point 4, the next point is point 5 and the previous point is point 3. Finally, a simple dot product will tell us if angle  $B$  turns to the right or to the left. If the angle turns right, the shape is clockwise. For the right shape in Figure 4.7, the top-most left-most point is point 9, thus the next point must be point 1 and the previous point is point 8. Angle  $B$  turns to the left which makes the shape a counterclockwise shape.

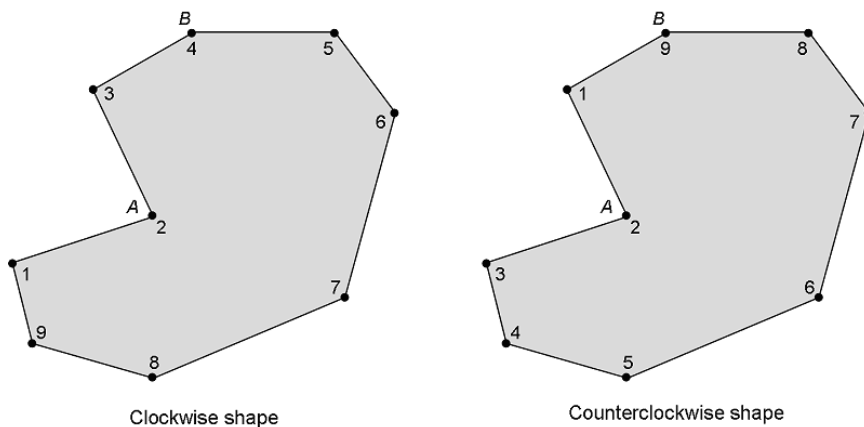


Figure 4.7: The same shape drawn “clockwise” and “counterclockwise”.

In both shapes in Figure 4.7, angle  $A$  is drawn to show that this method works for concave shapes. For convex shapes, the algorithm need only pick the first three points and determine what direction the angle between them turns. Angle  $A$  shows this will not work for concave shapes. The algorithm’s run time is increased for concave shapes because it must find the target top–most left–most point. Fortunately, the necessity for determining whether a shape is clockwise or counterclockwise need only be performed before the user saves the created shapes.

When Taverns loads a shapefile, it must “fix” the holes of shapes. As mentioned earlier, determining if a point lies within a shape is a common function within Taverns, therefore holes must be “fixed” for this function to work properly. To fix a hole, we must add the points of the hole to the shape it resides within. A series of steps is performed to accomplish this. We must first define the main shape as shape  $A$  and the hole shape as shape  $B$ .

Now we can define two points,  $pA$  and  $pB$ . These two points are the two with the minimum distance between them of all the points of shape  $A$  and shape  $B$ . To put it another way,  $pA$  is a point from shape  $A$  and  $pB$  is a point from shape  $B$ , and they are the two closest together. We must find the two closest points so the shape combination step does not overlap any edges of the shapes. Figure 4.8 describes this process. Once  $pA$  and  $pB$  have been identified, we can continue to step two which is

```

D = 1000000
From 0 → points in shape A
  From 0 → points in shape B
    f = distance(current A point, current B point)
    if D > f
      pA = current A point
      pB = current B point
      D = f

```

Figure 4.8: Find two points closest from shape  $A$  and shape  $B$

combining the two shapes between  $pA$  and  $pB$ .

Combining the two shapes  $A$  and hole  $B$  together requires several steps. Refer to Figure 4.9 to help understand these steps. We must create a new array of points to represent the new shape, which will be the main shape  $A$  combined with the hole shape  $B$ . The order in which the points are added to the new points array is the most important part. While referring to Figure 4.9, the starting point of shape  $A$  is  $1A$  and the end point of shape  $A$  is  $4A$ . The same is true for shape  $B$ . The points must be inserted in this order:

1.  $\text{Start}(A) \rightarrow pA$
2.  $pB \rightarrow \text{Start}(B)$
3.  $\text{End}(B) \rightarrow pB$
4.  $pA \rightarrow \text{End}(A)$

The resulting point order from the example is as follows:

$1A \ 2A \ 2B \ 1B \ 4B \ 3B \ 2B \ 2A \ 3A \ 4A$

The steps 1–4 added the following points to the array:

1.  $1A \rightarrow 2A$
2.  $2B \rightarrow 1B$
3.  $4B \rightarrow 2B$
4.  $2A \rightarrow 4A$

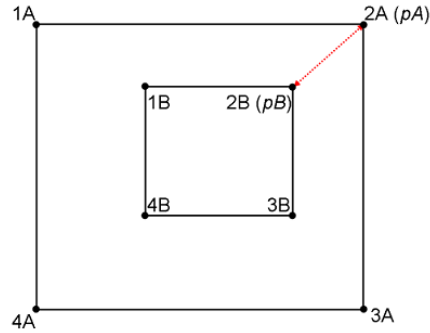


Figure 4.9: A shape with a hole and the two closest points  $pA$  and  $pB$ .

Once this operation is complete, the hole is now merely a concave portion of the shape. Figure 4.10 portrays what the shape will look like when rendered and an exaggerated version to help visualize how the resulting shape is merely a concave shape. The function that determines if a point lies within the shape can now work properly.

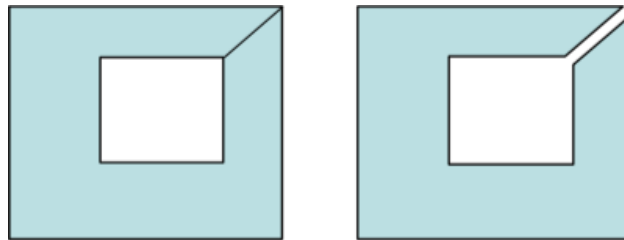


Figure 4.10: Shape  $A$  after hole  $B$  has been “fixed”.

### Shapefile Class

The shapefile itself is a complicated matter due to the numerous ways shapes can be created. This class provides functions for reading, writing, displaying, formatting, etc. Also important is the fact that the user can quite possibly create the shapes in a manner that is inconsistent with the format of the shapefile. This class must fix these inconsistencies to adhere to the shapefile format.

As described in Section 2.1, the shapes contained in a shapefile are formatted according to the format specification [13]. Taverns should not require the user to draw

the shapes according to this format. Even though the format is not complicated, the user need not learn the format in order to draw the shapes.

This Shapefile class is primarily responsible for managing a set of Shape classes. The Shape class, described in the previous section, is responsible for the algorithms performed on the shapes. This class does provide two vital functions: reading and writing shapefiles.

What makes the reading and writing of shapefiles fairly complicated is the fact that a shapefile may not necessarily contain the polygon type “shape” to which this paper has been referring. The format supports a variety of object such as points, lines, poly-lines, etc. Currently, Taverns only supports shapefiles that contain shapes of type “polygon”.

With our Shape class implemented, reading a shapefile is fairly simple. The reader need only read one shape at a time and store it in an instance of our Shape class which will perform the appropriate functions such as fixing holes.

Writing shapefiles is a bit more tedious. Because the user may place the points for each shape in any order, this class must first format all of the user’s work before it can be saved. A copy of the created shapes is made so the user may resume work without noticing all of the formatting that takes place during the save operation. The following steps must be performed before the shapes can be saved:

1. Make every shape clockwise
2. For every shape, find all of the shapes inside of it. Shapes inside of other shapes must be marked as holes.
3. For every hole shape:
  - (a) Make a copy of the hole shape
  - (b) Reverse the hole copy so it is now counterclockwise
  - (c) Associate this shape as a hole of the shape it is inside of

Now that these steps have been performed, all of the shapes are formatted according to the specification. Lastly, the class will calculate the overall bounding box of all the shapes. The data is now ready to be saved to the shape file. But as described in Section 2.1, an index file into the shapefile must also be created. This is

not complicated and this Shapefile class will create an index file for the newly created shapefile.

### **Attributes Class**

Attributes, though directly related to shapes, must be managed separately. Like the Shapefile class, the Attribute class provides the important functions of reading, writing and formatting.

As mentioned in Section 2.1, the attribute files store the data as mere text strings. It is the header information that specifies what type of data each text string represents. Taverns currently does not perform any calculations that need to determine whether a value is an integer type, floating type or string type. This makes loading and displaying the data simple. But if calculations are to be added to Taverns during later development, this Attribute class still stores all of the necessary information needed in order to convert the text data to the appropriate data type.

While working within Taverns, the user assigns attributes to the created shapes. The user has the ability to create alpha-numeric attributes allowing the creation of text strings without restriction. This means when the attributes are saved, the header data that determines what data type the attributes represent must be determined.

Attribute files support a wide variety of data types such as “Memos” and “Dates”. However, Taverns only looks for three standard data types: integers, floats and strings. Because the attributes are stored in a table (one row in the table corresponds to one shape, and shapes may have more than one attribute), the algorithm will scan down each column and determine what all of the values in the column should be. If all of the text strings can be converted to integers, then a flag is set in the header information for the row that it contains integer values. If one text string is found that cannot be converted to an integer but can be converted to a floating point number, then the flag is set that the column contains floats. If just one of the values is an alpha-numeric string, then the flag is set that the column contains strings. Figure 4.11 shows the process of this algorithm.

```

T = Integer
From 0 → Number of Columns
  From 0 → Number of Rows
    If T is of type Integer
      If value (row, col) cannot be converted to an integer
        T = Float
      Continue to next row
    If T is of type Float
      If value (row, col) cannot be converted to a float
        T = String
      Stop
Column data type = T

```

Figure 4.11: Determining attribute data types.

### Sight Class

The “Sight” is not a complicated tool, but necessary for Taverns to be usable in a virtual environment. A standard mouse cursor common to most desktop computers is typically not available in a virtual environment. The mouse cursor is designed for 2D displays, quite contrary to the essence of a virtual environment. The Sight will always follow the contour of the terrain which will make it very easy for the user to see where shapes are created. Figure 4.12 shows a diagram of the Sight and an image of how it will follow the contour of the terrain. The center point is what the user will use to determine where points are placed while creating shapes. The surrounding points help display the contour of the terrain around the location of the center point.

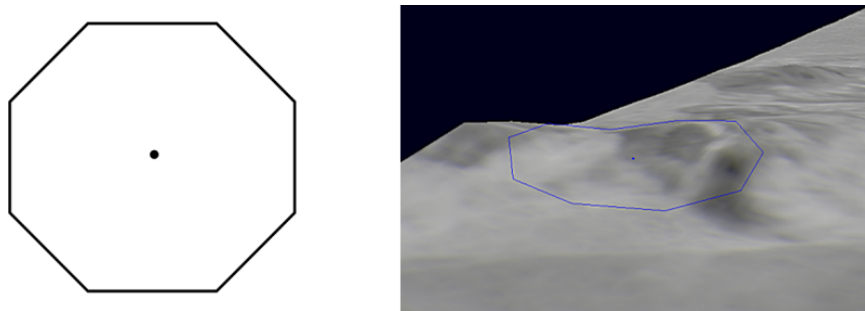


Figure 4.12: The Sight, general design and in action.



## 4.2 User–Interface Prototype

Designing effective interfaces for a user is no small task. This section will describe the attempts to make Taverns an expert user’s application while still holding to common human–computer interaction practices [37].

We must look at Taverns as an editor: a shape editor. Like all editors, specific tools are implemented to suit the purposes of that editor. The following two sections describe new tools the user must utilize while working with Taverns within a virtual environment, and editing shapes in general.

Taverns must be usable on both a standard desktop computer and in a virtual environment. It is believed that the virtual environment will provide a bit more realism than a desktop environment, thus the expert user will be able to more accurately analyze the terrain. However, virtual environments tend to be very cumbersome and expensive, so the option for running the application on a desktop computer is very important. Future sections describe the similarities and differences between interaction with Taverns on a standard desktop computer and in a virtual environment.

### **The Sight**

Section 4.1 has introduced the Sight and its implementation. The user will not see how the Sight is implemented, but how it will behave. The Sight is necessary because a mouse cursor common to desktop systems is not present in virtual environments. And because Taverns is to be as portable as possible, using the cursor with one system and the Sight with another creates inconsistencies that should be avoided. Therefore, Taverns will hide the mouse cursor when running on desktop systems and the user will create shapes with the Sight on both desktop systems and in the virtual environment. The Sight is rendered to always follow the contour of the terrain so the user can quickly and easily see where shapes are created.

The user can control the Sight in two ways. Taverns provides two ways because the terrain rendered is often very large, however the shapes need to be created

with fine detail. Implementation of both fast and slow Sight motion through only one method is rather tedious, but more importantly, it is difficult for the user to understand exactly how the Sight will behave.

Whether running Taverns on a desktop or virtual environment, two methods will be provided to move the Sight both slowly (incrementally) and quickly (accelerating). If the user chooses to accelerate the Sight, it will begin slowly at first and the speed will increase while the user holds down the controls. This is to move the Sight over the large terrain quickly. When the user releases the controls, the Sight will decelerate but at a higher rate than it accelerates. It is very important that the user does not feel the Sight is out of control. The second method will move the Sight in small increments which will allow shapes to be created with a high degree of accuracy.

Obviously the Sight shall move “forward”, “backward”, “left” and “right” but these directions **must** be relative to the direction the camera is looking in order for Taverns to be as intuitive as possible. One immensely popular computer game did not take this into account when implementing one of their aerial weapons meant for guiding munitions to the ground (Unreal Tournament 2004<sup>TM</sup> [17]). This has proved horribly confusing and difficult to manage.

## Editing Shapes

As mentioned, Taverns currently only supports shapes of type “polygon”. A polygon is merely a set of points rendered in a simple connect-the-dot fashion. But as good editors should, Taverns contains more metadata about the shapes the user is creating.

Taverns supports undo and redo operations for each shape the user creates. While the user plots points for the shapes, a plotted point may be un-done or re-done. Undo and redo operations are always important in editors and are no less important with shapes.

Each shape has its own undo and redo history by merely storing a flag that represents the “last” point of the shape. For example, if the user plots twenty points for a shape and presses the undo command five times, the flag to the “last” point

will be decremented five times and only fifteen points will be rendered to the screen. The redo history is now implicitly stored with this flag, which can be incremented with every redo operation up to the actual last point of the shape. Figure 4.13 goes through an example of plotting several points, using the undo function to place one of the points at a different location, then using the redo function to continue working where the user last plotted a point. The arrow above the points P0-P9 is the flag pointing to the “last” point.

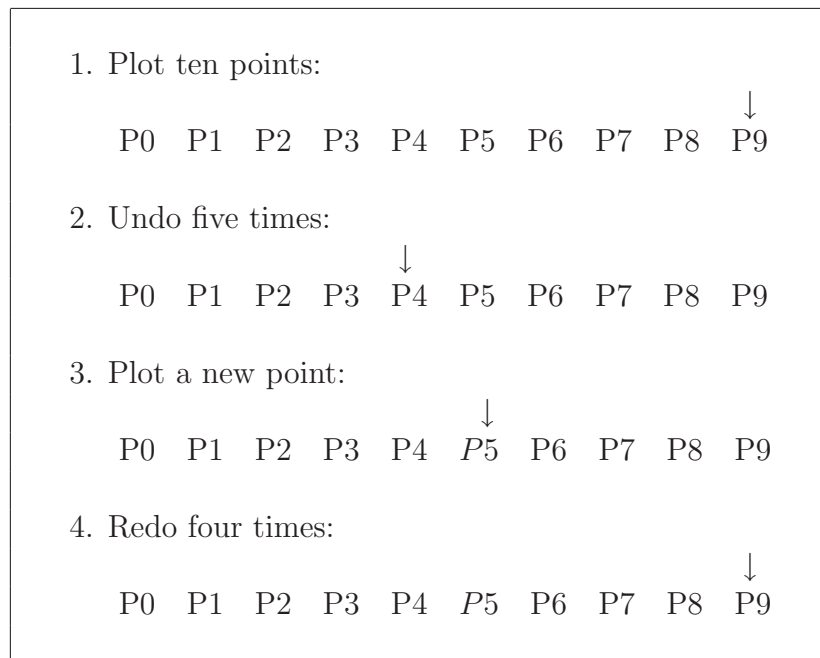


Figure 4.13: Plotting points undo and redo example.

After these steps, point P5 is replaced with the new point the user plotted. With these undo and redo functions, the user can manipulate shapes easily and quickly. Also, when the user chooses to save the shapes, the points after the “last” flag are not saved to the shapefile. Remember from Section 4.1 that a copy of the shapes is made and those shapes are truncated and formatted so the user may resume work with the same undo/redo history for each shape.

## Desktop

A standard keyboard is all Taverns requires to run on a desktop system. But since a mouse is so common with desktop systems, some of the keyboard commands may also be sent through the mouse. Taverns is a professional application specifically designed for expert users. This does not necessarily mean the keyboard commands are numerous and difficult to learn, many professional and industry caliber applications support keyboard commands. Advanced text editors are a prime example of powerful applications meant for more professional expert users. VIM<sup>TM</sup> [7] for example is designed to be purely keyboard based. UltraEdit<sup>TM</sup> [19] is a wonderfully powerful text editor that may be managed purely from the keyboard. However as mentioned, because a mouse is so common, it will respond to input from the mouse as well.

Taverns is very much like a text editor, in fact when the user chooses to edit an attribute, it will respond like a text editor by capturing key strokes and adding characters to the attribute string appropriately. Taverns has kept the popular key commands for standard operations. Common editor commands in both VIM and UltraEdit that Taverns supports are listed below:

Ctrl+S	Save
Ctrl+Z or 'U'	Undo
Ctrl+Y or 'R'	Redo

Figure 4.14: Common desktop editing commands Taverns supports.

Game engines have common commands for navigating in a 3D world, and because Taverns displays the data in 3D, the user must also have intuitive controls for navigation. Common gaming navigation commands that Taverns supports are listed in Figure 4.15:

In order to “look” or “turn” the camera up, down, left or right, the user may press the up, down, left or right arrows respectively. However, since it is far more common to look with the mouse in today’s PC games, Taverns supports mouse look

'W'	Fly Forward
'S'	Fly Backward
'A'	Fly Left
'D'	Fly Right
'C'	Move Down
Space	Move Up

Figure 4.15: Common 3D navigation commands Taverns supports.

as well.

Taverns, as a shape editor, must allow the user to create and manipulate shapes. Therefore, commands are required that are not standard with other applications. This does not mean they need be complicated, just new with Taverns. The commands with a brief explanation of their function are listed in Figure 4.16.

'F'	Locate the Sight by positioning the camera over it
'P'	Plot a point for the current shape
Tab	Outline the current data layer shape and show its attributes
Enter	Select one of the shapes the user has created to edit
Page Up	View the next data layer in the list of loaded layers
Page Down	View the previous data layer in the list of loaded layers
F1	Toggle the help menu on/off
F2	Toggle textured wire frame on/off
F3	Toggle wire frame overlay on/off
F4	Toggle follow terrain mode on/off
F5	Create a new shape and set focus to it
F6	Wait for input to select a shape for editing
F7	Wait for input to select one of the attribute columns
F8	Add a new column to the attribute table
F9	Edit the header of the current attribute column
F10	Edit the entry of the current attribute column
F11	Create debugging report for developers
F12	Take screenshot

Figure 4.16: Taverns specific editing commands.

Fast motion of the Sight is intended to be performed with the number pad, quite often on the right hand side of a standard keyboard. However there are variations of

keyboards and in particular, laptop keyboards are often designed without a traditional number pad to save space. Figure 4.17 explains the characters that will move the Sight.

'8'	Accelerate forward
'5'	Accelerate backward
'4'	Accelerate left
'6'	Accelerate right

Figure 4.17: Sight motion keys.

As mentioned previously, the user may use the mouse to move the Sight in small increments to more precisely plot points for shapes. To a small degree, the user may move the mouse quickly and the Sight will move a proportionally further amount, but it is intended to be used slowly for higher accuracy when creating the shapes.

### **Virtual Environment**

All of the implemented functions that make Taverns an interactive application on a desktop system must also apply in a virtual environment. This makes the application portable and equally functional on both systems. However, Taverns cannot be interacted with in the same way on both desktop systems and in a virtual environment because keyboards and mice, which are so common for desktop computers, are not standard input devices for virtual environments. With this in mind, all of the functions described in the previous section have been implemented to respond to multiple devices which makes Taverns usable in a virtual environment. Most input devices such as joysticks or joypads have a relatively small number of buttons, especially compared to all of the possibilities keyboards and mice supply. This may result in a steeper learning curve, but the accuracy gained while analyzing the terrain in the virtual environment is a worthy trade off.

The biggest difference between the desktop Taverns and the virtual environment Taverns is how the user will assign attributes to the shapes created. On the desktop

version it is quite simple, Taverns can easily detect when the user has entered a typing mode and Taverns will respond to keyboard input very much like a text editor. This allows the user to easily enter arbitrary attributes. While running in a virtual environment, Taverns cannot accept keyboard input thus attributes must be created a different way. A common option is a virtual keyboard. The design of a keyboard with all of the keys outlined are drawn on the screen and the user can move a cursor restricted to the virtual keyboard and enter text. This is normally cumbersome and another option more fine tuned to Taverns itself is proposed here.

The geographers who will use Taverns generally have some background information about the terrain before they begin creating shapes. They also know the kind of features they will be looking for in the terrain. Based on these assumptions, Taverns requires the user to create a list of the attributes that may be assigned to the shapes the user creates. Taverns will read the file of possible attributes and display them in a menu when the user chooses to assign an attribute to a shape. The list of possible attributes will be stored in memory so the user may alter the file and reload it during runtime. Figure 4.18 shows an edited view of how the menu will appear in the virtual environment.

The developers will continue to assess this method of assigning attributes and strive for an optimal combination of fluidity and functionality as long as Taverns is in development.

## Screen Shots

The following are screen shots of the latest version of Taverns on a desktop system. Some of the text may not display clearly so the reader may view the images at the Taverns home page [23].

Figure 4.19 displays a simple shape and the points that compose the shape. It is created on a detailed terrain landscape. Status text is shown on the top-left hand side of the screen. Attributes for the shape are centered at the top of the screen. Information about the current data layer is shown at the bottom of the screen.

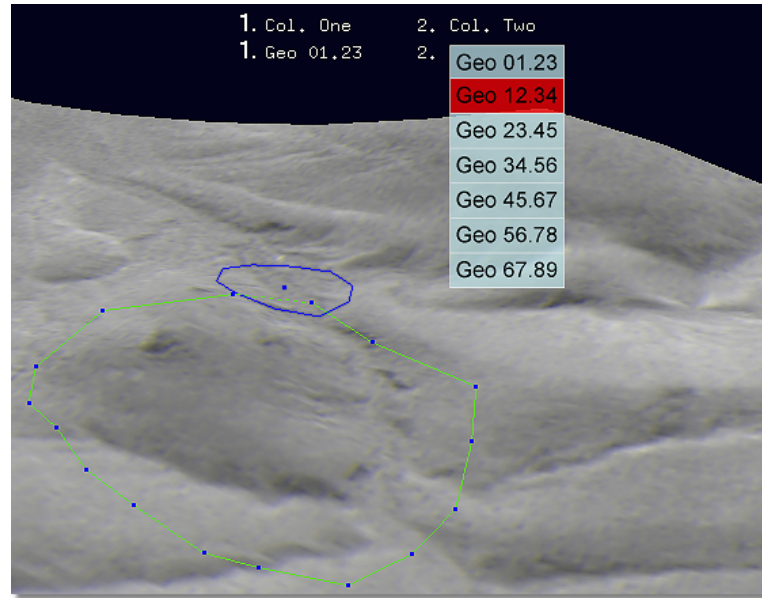


Figure 4.18: Example attribute list with Taverns in VR.

Figure 4.20 displays the same shape as the previous figure, but now a shapefile layer is textured on the terrain. The shapefile texture is averaged with the terrain texture from the previous figure so the user can more easily see features of the terrain as well as the data the shapefile layer represents. From the shapefile layer itself, the user can choose to see the attributes of the shape the Sight is located inside of. This allows the expert user to view any number of data layers already gathered about the terrain and use this information to extract more features.

Figure 4.21 is the same scene as Figure 4.20 but the terrain is drawn in wire frame. The wire frame view allows the expert user to more accurately determine where the slopes in the terrain are.



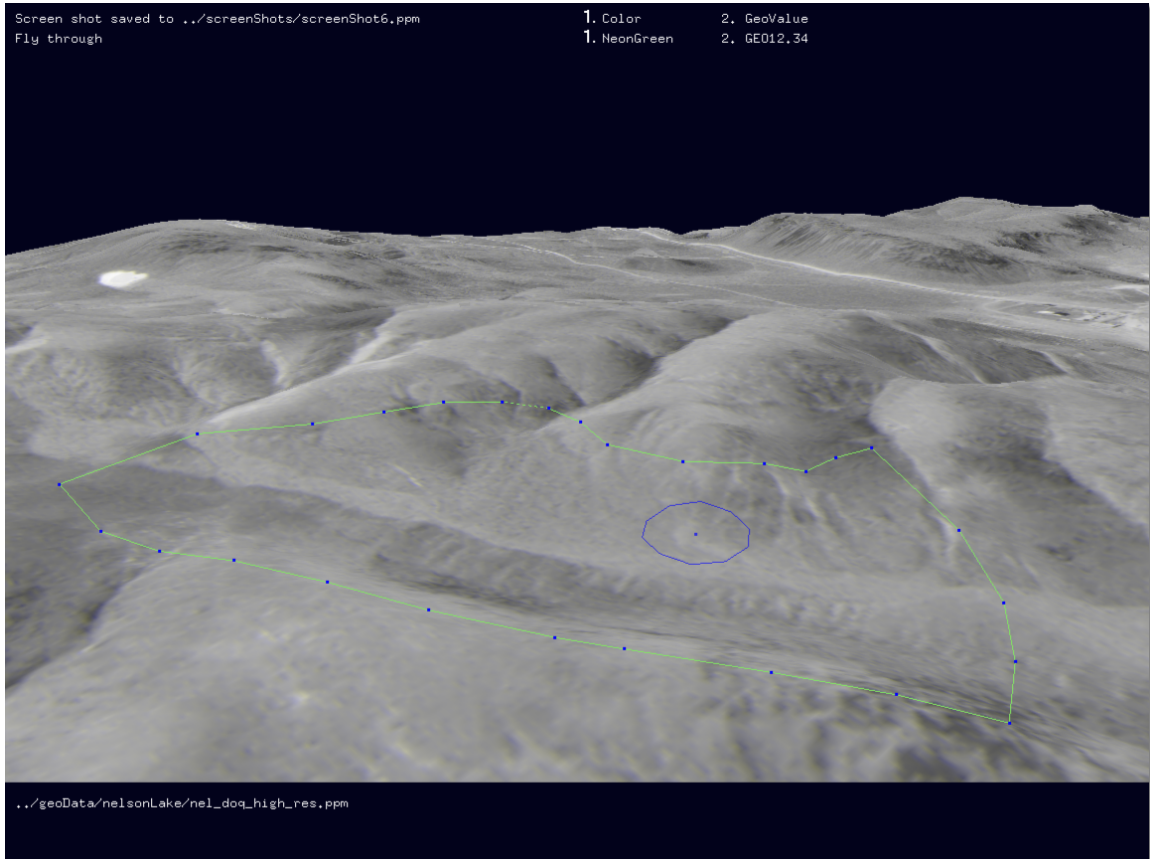


Figure 4.19: Simple shape on a typical terrain texture.

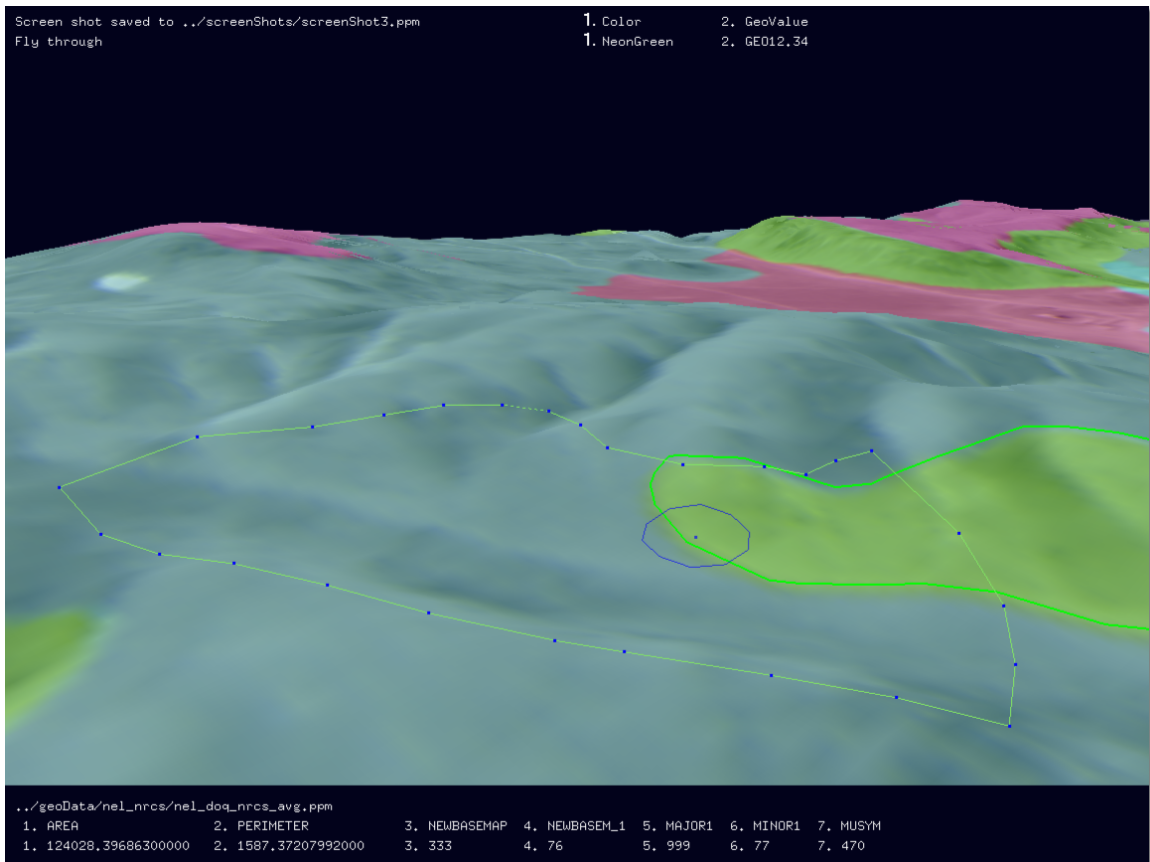


Figure 4.20: Simple shape on a data layer texture.

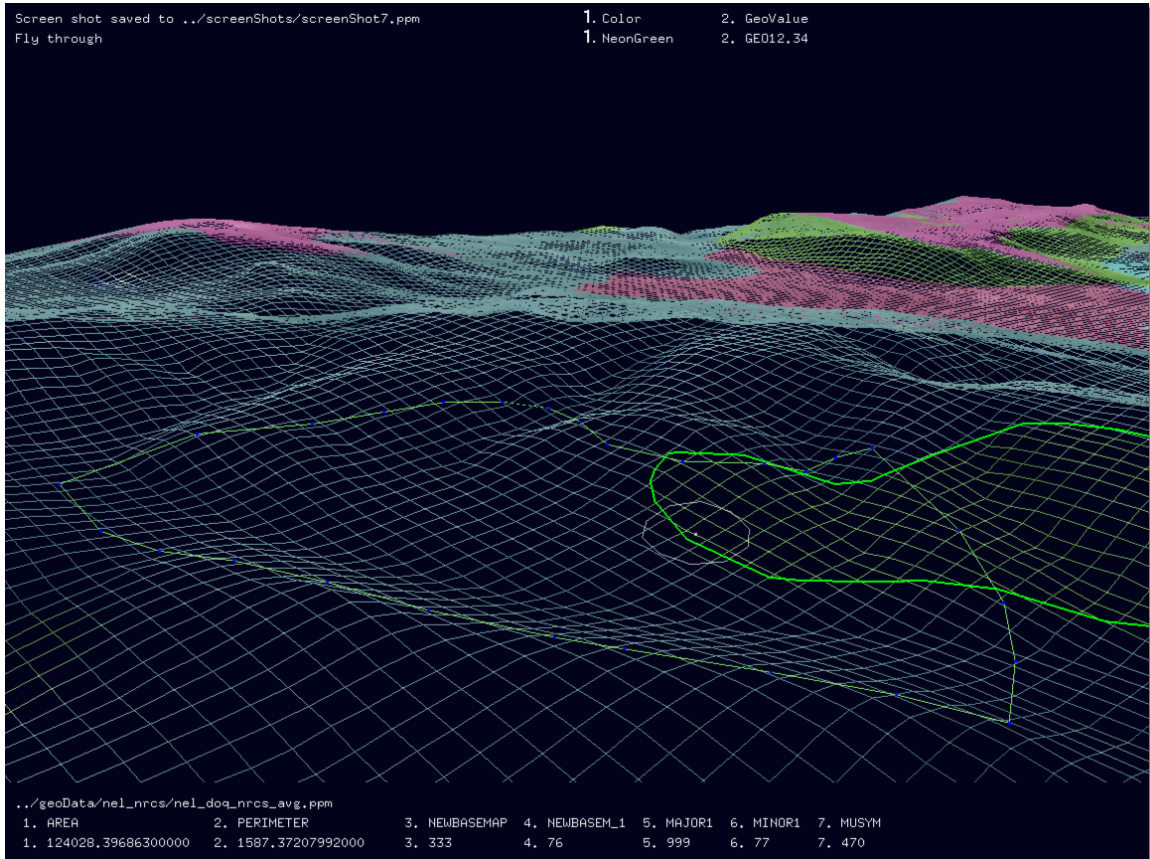


Figure 4.21: Simple shape on a data layer in wire frame view.

# Chapter 5

## Conclusions and Future Work

### 5.1 Conclusions

Much collaboration between the developers and intended users has occurred with Taverns on a desktop system. As quoted by one user, “Taverns rocks the house”. This shows strong support for the future of Taverns. However, little user testing has occurred with Taverns in a virtual environment. As mentioned in Chapter 2, Taverns is a stepping stone for the larger Desert Terrain project, thus much collaboration with the intended users must continue in order to ensure Taverns is a usable, reliable and accurate application to ultimately become an asset in the Desert Terrain project.

The core elements of Taverns as described in Chapter 3 are now in place. The author concludes that Taverns has successfully accomplished the intended functions as listed in Section 3.1. The intended use cases as listed in Section 3.3 are operational and may be performed in a manner similar to the functionality current editors provide users. The developers have attempted to hold to strict human–computer interaction techniques in order to ensure Taverns will conform to current software packages it is meant to supplement. The implementation is stable and ready for the future work items listed in the next section. With this solid foundation, Taverns has proven to be a worthy supplement for current geographic information systems.

## 5.2 Future Work

The future developers of this application are highly encouraged to review and consider the advancement of Taverns based on the recommendations in the following sections. Taverns has already proven valuable, implemented as a prototype application. With the addition of the following features, it will soon become a standard.

### Optimizations

Most of the functions implemented in this project may be optimized to a large extent. The future readers are highly encouraged to reference rendering efficiency methods by means of mesh simplification [25] and triangle decimation [33]. Rendering the terrain more efficiently will allow for more fluid interaction as well as allow much larger data sets to be viewed at one time.

The computational geometry functions described in Chapter 4 are vital for functionality. Optimizing these functions will greatly increase the speed at which most of the implemented algorithms are performed, for example, our texture making algorithm. The algorithm for creating a texture from a shapefile has significant room for optimization, but other methods still rely on the ability to determine whether a point lies in a polygon or not. The simple function of determining whether or not a point lies in a shape should be the first of all the current algorithms to be optimized.

### Parallelization

Not only can many of the current algorithms be optimized, there is strong potential for parallelizing them as well. For example, when the configuration class completes, loading the data layers may be done in parallel which would greatly increase loading time. Many of the functions that are performed on a shape may be parallelized as well, for example finding all of the hole shapes for every shape. Also, virtual environments are normally run by a multi-processor computer system. Utilizing this hardware would be extremely advantageous.

## **Robustness**

Shapefiles are renowned for being “notoriously temperamental”. Creating generic reading and writing functions for them is not easy due to the numerous possibilities for error within the files. Also, because so many current GIS software packages work with shapefiles, we must attempt to account for the intricacies of the output from each application. The best way to find inconsistencies between our shapefile functions and the large databases of GIS data is to simply test data layer after layer in the application.

Similarly, future developers must continually check the output of Taverns and the usability of created shapefiles in current GIS software. One may argue it is much more important to create robust output because the purpose of Taverns is to produce shapefiles that will be used in further experiments and simulations.

## **Interface Development**

As mentioned in the previous section, a fair amount of user testing has occurred with Taverns on a desktop system. Determining the usability of Taverns in a virtual environment is still underway. If Taverns is to be used as a fluid, intuitive editor in a virtual environment, future developers must continually get feedback from the intended users to ensure Taverns will be an asset and not a hindrance.

Recent brainstorming has given rise to the idea of implementing speech recognition algorithms as a way of assigning attributes within the virtual environment. As mentioned in Chapter 2, attributes assigned to shapes do not follow any particular format. The expert geographer assigns attributes depending on the context of the research study. The menu of possible attributes proposed in Section 4.2 is a possible method for assigning attributes, however well implemented speech recognition algorithms would allow a multitude of attributes to be assigned, and perhaps more importantly, would further immerse the user into the virtual environment. Should the speech recognition prove reliable, the developers may extend its capability to support a wide array of verbal commands related to terrain analysis.

Regarding the user's ability to view different data layers, the user may only scroll through the list of loaded layers. In order to view an aerial photo and a shapefile texture at once, our Layer Manager class simply averages the two images together. This gives the user little control over how the data is viewed. Future developers are encouraged to explore multi-texturing methods and increase the user's control regarding what data layers will be viewed and how they will be viewed.

### **Further Brainstorming**

We have seen from Chapter 2 that one key part of the Desert Terrain project is locating alluvial fans. These fans have common characteristics and future developers may consider the possibility of implementing an algorithm to locate and create shapes around potential alluvial fans.

Another option related to automating certain tasks would be to implement an algorithm that will create an output shapefile based on the similarities and/or differences of several input shapefiles. Expert geographers will use Taverns to perform precisely this task themselves and years of experience goes into accomplishing this task. However if certain parameters are passed to the algorithm that will determine relationships between shapefiles and create output shapefiles based on these relationships, it may prove to be an asset.

A significant amount of data about Mars has been gathered and plans are currently underway to begin a more thorough analysis of Mars with Taverns. Rendering the surface to scale in a virtual environment will immerse the user into the an incredibly realistic rendition of Mars, so realistic in fact that the next level would be to visit the planet itself.

Related to immersing the user in the virtual environment, the idea was mentioned of actually being able to feel the granularity of the soil at different locations in the terrain. Little research has been performed regarding methods and technologies that could make this a possibility, but the idea furthers support for virtual environments and ultimately immersing users ever more into the realm of virtual reality.

# Bibliography

- [1] J. Arlow and I. Neustadt. *UML and the Unified Process: Practical Object-Oriented Analysis and Design*. Addison Wesley, 1st edition, 2002.
- [2] Erik Bachmann. Xbase File Format Description. <http://www.clicketyclick.dk/databases/xbase/format/>, Accessed October 18. 2005.
- [3] F. P. Brooks. Grasping Reality Through Illusion—Interactive Graphics Serving Science. In *CHI '88: Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 1–11. ACM Press, 1988.
- [4] James B. Campbell. *Introduction to Remote Sensing*. The Guilford Press, 3rd edition, 2002.
- [5] Geo Community. USGS SDTS format Digital Elevation Model data (DEM). <http://data.geocomm.com/dem/>, Accessed June 13. 2005.
- [6] OSG Community. Open SceneGraph. <http://www.openscenegraph.org/>, Accessed November 7. 2005.
- [7] Vim Community. Vim. <http://vim.org/>, Accessed October 20. 2005.
- [8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. McGraw-Hill, 2nd edition, 2001.
- [9] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart. The CAVE: Audio Visual Experience Automatic Virtual Environment. *Commun. ACM*, 35(6):64–72, 1992.
- [10] Paul J. Curran. Remote Sensing of Foliar Chemistry. volume 30 of *Remote Sensing of the Environment*, pages 271–278, December 1989.
- [11] M. de Berg, M. van Kreveld, M. Overmars, and O. Shewartzkopf. *Computational Geometry - Algorithms and Applications*. Springer, 2nd edition, 1998.
- [12] Terrain Engine. Demeter Terrain Engine. <http://www.terrainengine.com/>, Accessed November 3. 2005.
- [13] ESRI. ESRI Shapefile Technical Description. <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>, Accessed March 17. 2005.
- [14] ESRI. ArcGIS. <http://www.esri.com/software/arcgis/>, Accessed November 3. 2005.



- [15] ESRI. ArcScene. [www.esri.com/news/arcuser/0103/files/3display.pdf](http://www.esri.com/news/arcuser/0103/files/3display.pdf), Accessed November 3. 2005.
- [16] ESRI. ESRI GIS and Mapping Software. <http://www.esri.com/>, Accessed November 3. 2005.
- [17] Epic Games Inc. Unreal Tournament 2004. <http://unrealtournament.com/>, Accessed October 20. 2005.
- [18] Google Inc. Google Earth. <http://earth.google.com/>, Accessed November 14. 2005.
- [19] IDM Computer Solutions Inc. UltraEdit. <http://idmcomp.com/>, Accessed October 20. 2005.
- [20] Desert Research Institute. Desert Research Institute. <http://www.dri.edu/>, Accessed November 8. 2005.
- [21] Desert Research Institute. Identifying Terrain Similarities Between the World's Deserts and the US Army's Desert/Hot Weather Test Site, Yuma Proving Ground (YPG), Southwestern Arizona. [http://www.dees.dri.edu/Projects/emcd\\_CAG.htm](http://www.dees.dri.edu/Projects/emcd_CAG.htm), Accessed November 8. 2005.
- [22] Desert Research Institute. Integrated Forecasting of Desert Terrain Conditions for Military Operations. <http://www.dri.edu/Home/Features/text/DesertTerrain.htm>, Accessed November 8. 2005.
- [23] Joseph Jaquish. Taverns. <http://www.mrjoehimself.com/school/thesisWork/web/>, Accessed November 6. 2005.
- [24] Randolph M. Jones, John E. Laird, Paul E. Nielsen, Karen J. Coulter, Patrick G. Kenny, and Frank V. Koss. Automated Intelligent Pilots for Combat Flight Simulation. *AI Magazine*, 20(1):27–41, 1999.
- [25] L. Kobbelt, S. Campagna, and H.P. Seidel. A General Framework for Mesh Decimation. Graphics Interface Conference, 1998.
- [26] Ming C. Leu, Mochael G. Hilgers, Sanjeev Agarwal, Richard H. Hall, Terry Lambert, Robert Albright, and Kyle Nebel. Training in Virtual Environments for First Responders. ASEE Midwest Section Meeting, 2003.
- [27] Michael R. Meuser. What is GIS and How Does it Work? <http://www.gis.com/whatisgis/whatisgis.html>, Accessed June 13. 2005.
- [28] Andy Mitchell. *The ESRI Guide to GIS Analysis*. Environmental Systems Research Institute Inc., Redlands, California, 1st edition, 1999.
- [29] Keith L. Moore and Arthur F. Dalley. *Clinically Oriented Anatomy*. Lippincott Williams & Wilkins, 5th edition, 2006.
- [30] NASA. Planetary Photo Journal. <http://photojournal.jpl.nasa.gov/catalog/PIA03021>, Accessed November 22. 2005.

- [31] Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, 2nd edition, 1998.
- [32] J. O. Robinson. *The Psychology of Visual Illusion*. Dover Publications Inc., 1998.
- [33] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of Triangle Meshes. ACM SIGGRAPH, pages 65–70, July 1992.
- [34] Donna Miles/American Forces Press Service. Former POW learns value of military training. *TRADOC News Service*, October 2003.
- [35] Bill Sherman. FreeVR. <http://www.freevr.org/>, Accessed November 5. 2005.
- [36] William R. Sherman and Alan B. Craig. *Understanding Virtual Reality—Interface, Application and Design*. Morgan Kaufmann Publishers, San Francisco California, 1st edition, 2003.
- [37] Ben Shneiderman and Catherine Plaisant. *Designing the User Interface—Strategies for Effective Human-Computer Interaction*. Addison Wesley, 4th edition, 2005.
- [38] R.D. Smith. Essential Techinques for Military Modeling and Simulation. volume 1 of *Simulation Conference Proceedings*, pages 805–812, 1998.
- [39] Softwright. TopoScript .BIL File Format. [http://www.softwright.com/faq/support/toposcript\\_bil\\_file\\_format.html](http://www.softwright.com/faq/support/toposcript_bil_file_format.html), Accessed April 2. 2005.
- [40] Ian Sommerville. *Software Engineering*. Addison Wesley, 7th edition, 2004.
- [41] USGS. Death Valley National Park Alluvial Fans. <http://wrgis.wr.usgs.gov/docs/parks/deva/rfan.html>, Accessed November 22. 2005.
- [42] Gwendolyn H. Walton, Robert M. Patton, and Douglas J. Parsons. Usage Testing of Military Simulation Systems. Winter Simulation Conference, pages 771–779, 2001.