Life ‘N Touch

Final Report

Kyle Dobitz and
Jeremy Muehlbauer

Table of Contents

Table of Figures

Table of Equations

Table of Tables

Abstract

The world is brimming with information. In many fields, in order for the data to be useful, the data must be interpreted. The goal of the Life 'N Touch is to provide a platform to eliminate unnecessary user data interpretation by supplying information already interpreted directly to the user. By applying this goal to current technology, the Life 'N Touch provides a heads-up display on which data can be relayed to the user. A specific application demonstrated is a real-time updated street navigation system that determines sections of a route to show based upon the user's current environment and orientation creating a context-aware system. The design is accomplished by integrating a GPS receiver, accelerometer, digital compass, and camera technologies to establish the user's environment. The system consists of a hand held device displaying the environment captured by an attached camera. The captured image of the environment is then drawn upon to show the route from the user's current position to the destination. The user's orientation is then established through accessing the accelerometer and digital compass, enabling the Life 'N Touch to display the portion of the route that the user is currently viewing.

Introduction

As more knowledge becomes available, the amount of information that can be accessed through various means will continue to grow as well. Although a vast wealth of information may be accessible, the information often requires a translation to reality. For instance, although one may be able to find directions from home to the nearest Italian restaurant, the turn-by-turn directions returned by the map database must still be interpreted to locations in reality. The Life ‘N Touch proposes a method to circumvent the unnecessary data interpretation by displaying interpreted data in an intuitive format.

The purpose of the Life ‘N Touch is to provide a platform from which automated data retrieval and interpretation can be performed. The platform can be applied to multiple areas; however, the specific application developed is real-time street navigation. This application is similar to many street navigation tools currently on the market in its ability to provide accurate turn-by-turn navigation; however, the application differs from current trends in the manner in which the turn-by-turn data is conveyed to the user. Instead of creating an internal map structure embedded in a marketed navigation unit, the Life ‘N Touch’s application captures the user’s environment and orientation and overlays pertinent route information on top of the captured environment. Displaying the information in this manner causes the Life ‘N Touch to function similar to heads-up displays (HUD), a transparent display that presents data without obstructing the user's view, used in commercial and military aviation (Figure 1). The application accomplishes this task by receiving data from a variety of interfaced hardware components: a Global Positioning System (GPS) receiver, an accelerometer, a camera, and a digital compass. These components allow accurate route information to be calculated and displayed to the user.

Figure 1 - An example of an HUD used to display the speed of a car to its driver [1]

Design Specifications

The Life ‘N Touch is designed to show proof of concept of a small embedded device with integrated hardware components: an interfaced GPS receiver, digital compass, accelerometer, and camera. The embedded device would have a small form factor, approximately the size of current touch-screen cellular phone technology, facilitating handheld, non-vehicular navigation simulated through the

use of a notebook computer as the embedded microprocessor and display with the interfaced hardware previously mentioned simulating specialized circuits in the embedded device.

The navigation system will operate by first capturing the user's environment and orientation with the camera, accelerometer, and digital compass. A positioning database will then be queried for the current position and destination to be transformed into the user's route. After the data is obtained, it will be translated into an intuitive format (i.e. route line) and overlaid on the user's captured environment with the determined orientation. Other software will function in a similar manner except the information being accessed will be different and, consequently, the data will be translated differently.

Design Overview

The project design must meet the three specifications: capturing the user's environment; retrieving the abstract route data; and integrating that information into an intuitive display. The user's environment is determined by three pieces of information, the orientation, position, and view of the device. Each of these pieces of the user's environment is captured by different hardware devices. Retrieving the abstract route data is achieved by taking the user's current location and desired destination, and retrieving the route data necessary to provide the user to navigate to the destination. This utilizes an internet connection to obtain the necessary data. Once all of this abstract data is collected the device must translate and integrate the information and finally utilize three dimensional libraries to visualize the information on screen in an intuitive format for the user.

Capturing the Environment

The orientation of the environment is the direction the device is pointed as determined by the yaw, or cardinal direction (e.g. North, South) of the device, the pitch or angle of inclination from the

horizon, and the roll or angle of tilt left or right of the device. The cardinal direction of the device is determined utilizing a digital compass, in this case the GPS receiver and digital compass: Wintec WBT-100. The digital compass allows the device to obtain the cardinal direction of the device's orientation by measuring the magnetic field of the earth. The direction is then returned to the program in degrees from north. The pitch and roll of the device is determined from the direction of acceleration due to gravity as measured by the Phidgets 3-axis USB accelerometer (see Appendices A and E). The accelerometer returns the amount of acceleration or force applied by gravity in each of three dimensions (Figure 2). This information is used to develop the angles of roll and pitch as explained in the integration step.

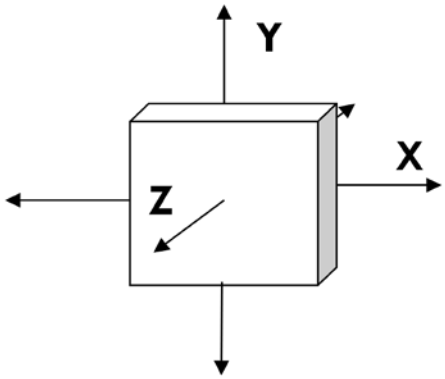

Figure 2 - XYZ dimensions of gravitational acceleration

The position is the geographic location of the device on the earth. This is expressed as angles of latitude, angular displacement from the equator, and longitude, angular displacement from the prime meridian (Figure 3). The user's current location is returned by the Wintec WBT-100 GPS device utilizing a grid of satellites orbiting the earth (see Appendix A).

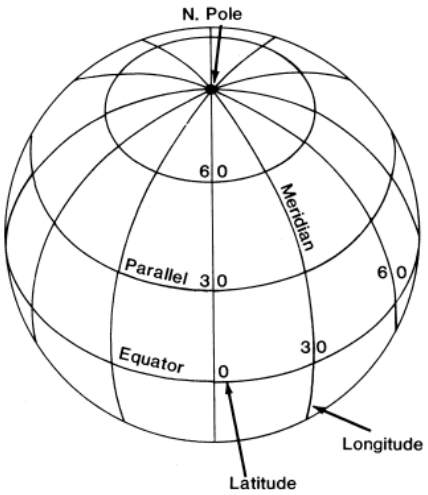

Figure 3 - Latitude and longitude as demonstrated on globe [2]

Finally the view of the device is the visual information from the environment that is currently in front of the device. This information is necessary to give the user a point of reference to make the data displayed more usable. Thus when a data object is drawn on screen the user can compare it to the view in front of him and determine to what in physical space that data object corresponds. Video must be streamed with a high

refresh rate for it to be accurate to the view of the user, and thus the LifeCam VX-6000 web cam (see Appendix F) is utilized in the design to capture the video feed.

Retrieving Route Data

The route data is retrieved in a two step process. First the relevant information necessary for developing a route is collected which includes the user's destination and current location. The second step formulates these pieces of information into a query then queries the route service for a route.

The design utilizes a web interface developed with JavaScript and PHP that asks for the user's destination. When the user submits a new destination (usually in the form of an address), the web interface checks a MySQL database for the user's current position. This position is constantly being updated by the Life 'N Touch every time the GPS device receives a new position. After receiving the user's destination and current location, the device then submits a query to the Google Maps routing service (Figure 4).

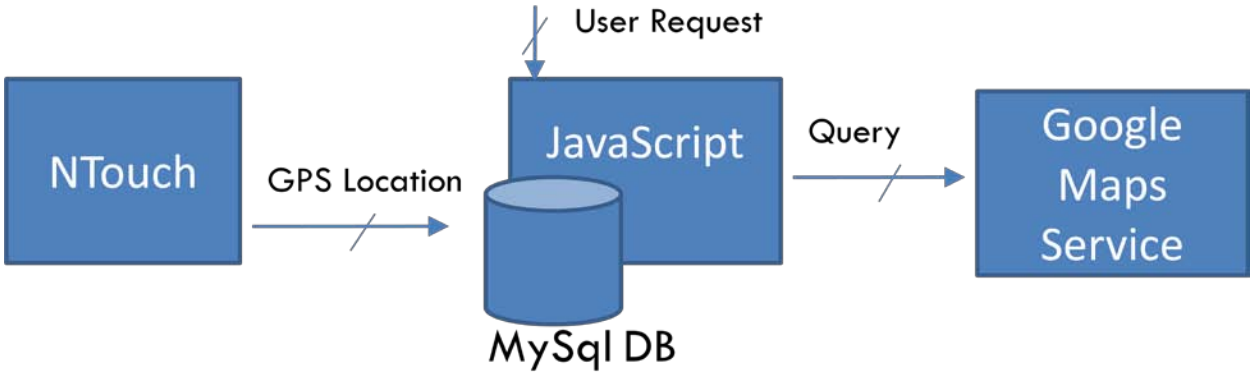

Figure 4 - Route data retrieval utilizing theweb interface

The Google maps routing service then performs two important services to calculate the route. First, it executes geocoding, in which the destination, usually expressed in natural language such as an address, is transformed into longitude and latitude coordinates. Next Google Maps performs routing to

determine a minimal route between the current location and the destination. The route as expressed in a series of latitude and longitude coordinates is then returned to the web interface and subsequently placed into the MySQL database.

Data Integration and Display

The information retrieved from capturing the environment and abstract route data retrieval is then integrated into an intuitive display for the user. The three steps in this integration is the display of the video stream, the calculation and display of the three dimensional route line, and the orientation of that line according to the user's environment.

In order to stream video data and provide a three dimensional route on screen, the Life 'N Touch utilizes the media streaming library DirectShow and the three dimensional graphics library Direct3D. DirectShow allows the Life 'N Touch to interface with the video data streamed from the web cam, which is then converted into a Direct3D texture. This texture is then placed upon a three dimensional polygon (in this case a rectangle) and placed directly in front of the "camera" of the viewing environment at an infinite depth (see Appendix D).

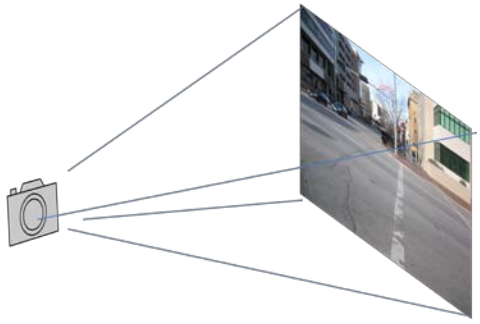

Figure 5 – The captured video stream placed directly in front of virtual "camera"

The route line is displayed as a collection of rectangles that connect the list of route coordinates returned by the route service (Figure 6). Each of these coordinates is returned in latitude and longitude points. These points are converted into distances based upon the radius of the Earth. The distance of a degree in longitude or latitude differs irregularly across the globe because the Earth is neither a perfect sphere nor ellipsoid. This design uses two averages for the

Figure 6 - Route line drawn utilizing rectangles

Earth's radii: the latitudinal radius of 6367.38 km and the longitudinal radius of 6388.84 km. Although these are not highly accurate estimates, any deviations from these values are well within one percent of error, which is acceptable as a one percent difference in distances would not change the display noticeably. Utilizing these radii, the distances from one point on the route to the next is calculated using the following equations derived from the circumference of a circle:

$$Distance_{EW} = \Delta Long R_{Long} \frac{\pi}{180^\circ} \cos(Lat)$$

Equation 1 - East/West distance due to changes in longitude

$$Distance_{NS} = \Delta Lat R_{Lat} \frac{\pi}{180^\circ}$$

Equation 2 - North/South distance due to changes in latitude

The vertices of the rectangles of the line to be displayed are then calculated such that they are two meters wide and connect each point of the route to the next point.

Finally the line is manipulated in relation to the user's environment to display correctly on screen. The route line is first moved approximately two meters down from the graphical environment's "camera" to simulate the height of the device held by a user. The distance of the user from the first point in the line is calculated using Equations 1 and 2 above for transforming latitude and longitude angles into distances. The line is then moved that given distance from the "camera". Finally the angles of orientation are used to spin the line based upon the user's orientation. The accelerometer information captured above is used to calculate the pitch and roll of the device. These are calculated trigonometrically using the ratios of the acceleration due to gravity in each of the three dimensions (Figure 7). Using the orientation angles of yaw, pitch, and roll, the route line is thus oriented

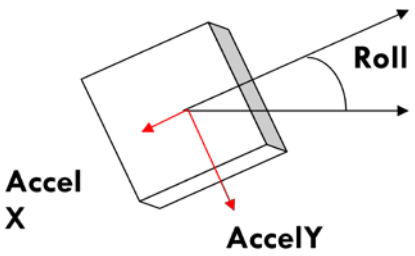

Figure 7 - Roll as determined by acceleration in different dimensions

accordingly or “spun” about the virtual “camera” so the line corresponds to the orientation of the user. Finally all of this is displayed on screen as a video feed with the three dimensional line overlaid on top of it (see Appendix D for more information).

Multithreading

This project utilizes many differing hardware devices as well as an internet connection in retrieving the data necessary. Each of these connections has a limiting speed to which they can return data. The speed of information return also has a higher or lower priority depending on how necessary up to date information is on the device’s utility. Because this device must operate in real time, the access delay of each device represents a bottleneck to performance. The Life ‘N Touch’s design utilizes multithreading to mitigate these limitations. The design utilizes four different threads a route thread, a GPS and compass thread, an accelerometer thread, and a main thread. The table below gives an overview of the four threads and their significant factors (see Appendix C for more information).

Thread	Route	GPS / Compass	Accelerometer	Main (3D Display)
Interface	Network Connection	GPS / Digital compass	Accelerometer	Camera, Graphics card
Protected data	set of route coordinates	current position / Yaw	Pitch and Roll	Utilizes all other protected values
Update rate	30s	20s / 0.5s	<0.1s	0.03s (30 Hz)

Table 1 - Each thread is provided along with the portion of the design the thread is used, the data with which it interacts, and the rate it is updated

The route thread is a thread created to interface with the MySQL database and return the most up to date set of route coordinates. The route is changed every time a user enters a new destination in

the web interface. Because it should take significant time for the user to arrive at a destination or enter a new destination, the device checks for a new route every thirty seconds.

The GPS and compass thread interfaces with the GPS and digital compass hardware component. The GPS and digital compass utilized in this prototype has two different modes of functionality: the GPS mode and the digital compass mode. When in GPS mode the device returns the current location in latitude and longitude, but at a slow rate. When in compass mode the device returns the cardinal direction of the device in a fairly rapid rate (1-2 hertz). Because the project is limited to non-vehicular travel, the current location does not need to be updated especially fast because the individual will not be moving at any considerable speed. The direction the device is pointing in has a direct impact on its performance because the direction will change rapidly as a user turns around or looks around using the device. Thus the design utilizes the GPS mode only every twenty seconds, and stays in compass mode the rest of the time.

The accelerometer thread interfaces with the accelerometer. The orientation of the device is returned by the accelerometer and so the refresh rate must be very high to provide accurate display information. The accelerometer returns data very rapidly (>10 hertz), thus ensuring the pitch and roll values are always up to date.

Finally the main thread initializes all the other threads, and interfaces with all the data returned from the other threads. This thread also stores display information on the graphics card and streams the data returned by the camera. The data shared between this thread and the other threads are protected from data access violations with the utilization of mutexes. Mutexes are global pieces of information that only allow one thread at a time to access a given piece of data. Because of the high performance of the graphics card and video streaming, the main thread gives the overall display a refresh rate at nearly 30 hertz.

Design Alternatives

The current implementation of the Life 'N Touch is not without alternatives that could improve the design; however, in order to produce a workable device under budget and with all of the proposed functionality, some concessions were made. Specifically, these alternatives are: using a digital compass independent of the GPS receiver; using a tablet notebook computer in place of the notebook computer; and using a map database installed on the notebook itself instead of accessing an online database.

The current design for the Life 'N Touch utilizes a combined GPS receiver and digital compass; however, by separating the two devices, the design of the Life 'N Touch could be greatly improved. Currently the GPS receiver/compass must switch modes of operation in order to obtain both the GPS location and the current direction faced. This type of operation causes a delay because the change in operational mode requires a fixed amount of time (i.e. the change is not instantaneous). In a real-time system like the Life 'N Touch, every moment of extraneous delay causes inaccuracies in the display; and during the time to change modes, no data can be acquired from the receiver. In other words, when the receiver is obtaining GPS location, the compass data is not being updated and vice versa. Therefore, this implementation leads to inaccurate data being used in the display.

Figure 8 - By functioning as two components, the GPS receiver causes delays in updating the display and consequently inaccurate displayed data [3]

An alternative to the current design would involve utilizing two pieces of hardware, a GPS receiver and a digital compass, separate from each other. Such an implementation would remove all delays associated with changing modes of operation of the receiver and always provide accurate data for the display. Ultimately this alternative was not used in the design of the Life 'N Touch because all available digital compasses with a USB interface were too expensive and could not be purchased with

the money allotted for the design: the Life ‘N Touch was allotted $500 for components but separate digital compasses cost a minimum of $199, roughly 40% of the budget (see Appendix B).

In addition to using two separate hardware devices for the GPS receiver and the digital compass, one could alternatively execute the software in the Life ‘N Touch on a tablet notebook computer instead of a notebook computer. A tablet notebook would further demonstrate proof of concept of an embedded device not only because of the tablet’s similar form factor but also because of the similar hardware capabilities: because of the size limitations of a tablet, the tablet’s hardware must consequently be smaller and generally less powerful than full size options.

This second alternative was not used in the Life ‘N Touch because the software developed for the street navigation system requires a high performance machine in order to operate smoothly and seamlessly. The software for the street navigation system was initially tested on a tablet; however, the results of the test indicated that the tablet did not have sufficient hardware to operate smoothly. While being executed, applications that use DirectX allocate graphics to memory on a graphics card if available in order to facilitate faster processing: the graphics card can perform the graphics processing while the application continues [4]. In doing so the application is streamlined and gains improved performance speed [4]. The tablet tested did not have an installed graphics card; therefore, the lag of the navigation system was sufficient enough to warrant a migration back to a regular notebook computer. Ultimately the graphics library could be switched for a less hardware demanding library thus allowing the design to work on less powerful devices.

Another alternative to the current implementation of the street navigation system is querying a map database stored locally on the machine running the Life ‘N Touch application as opposed to querying an online database. By using a locally stored database, all dependencies to the internet are severed since the internet is only used as a means to query an online database. The Life ‘N Touch could

then be used anywhere regardless of internet signal or connection allowing a greater freedom to the user.

Although internet dependencies would no longer be a factor, by using a locally stored map database routing and geocoding must be performed by the Life 'N Touch software instead of the online database. In theory, determining the shortest route between a starting point and a destination is merely an extension of Dijkstra's algorithm; however, the nuances of street congestion are not taken into account with Dijkstra's. By allowing the online database, Google Maps, to perform the route calculations for the Life 'N Touch, the resources available to the navigation system are better spent updating the current environment and orientation of the user while the more plentiful resources of the Google Maps web service calculate the shortest route. Unlike routing, geocoding could not be performed by an extension of a known algorithm. Geocoding can be defined as the process of assigning a code (i.e. name, number) to a geographic feature or location [5]. Unless the map database installed on the local machine running the street navigation system had geocoding capabilities, this functionality would be unavailable. The online database, therefore, allows users to input "coffee shop" and receive directions to the nearest coffee shop relative to their location.

Extensions of the Current Design

The Life 'N Touch as a platform can easily be extended to various applications with wide ranges of features. The current street navigation system, for example, can easily be amended to be used as a personal tour guide, directing tourists to famous landmarks and providing additional information and options retrieved from the internet. For instance, if a user is touring Greece near the Parthenon, the Life 'N Touch could provide an internet link to various web sites and services with more in-depth information about the Parthenon and its history. By selecting an alternative database, the current street navigation

system could be used as a trail guide as well, providing a path through various types of terrain as well as locations of landmarks and natural wonders.

By altering the software, the Life ‘N Touch is capable of providing various other services as well. For example, with the current hardware components as well as the time of day and the time of year, the Life ‘N Touch can calculate what stars are visible to the user and outline possible stars of interest or constellations, in effect creating a star registry or star map.

The Life ‘N Touch could also become the ultimate geocaching tool. Geocaching is a recreational activity in which someone "buries" something for others to try to find using a GPS receiver, often thought of as a GPS treasure hunt. “Geocachers” are only provided the latitude and longitude coordinates of the “treasure”. By allowing a geocacher to input his treasure’s location and providing him a route, the Life ‘N Touch would be the ultimate visual guide towards the treasure.

Conclusion

The Life ‘N Touch is an extendable platform that can be used to automatically retrieve data from various sources and interpret it in an intuitive manner that can be displayed to a user. In doing so, the user is not required to interpret the data. By applying this platform to street navigation, users no longer need to understand their current position relative to a destination or where a specific thoroughfare intersects their path. By following the route provided on screen, the user will arrive at the destination without interpreting a street sign, road map, or any data whatsoever. By capturing the environment and orientation of the user, the Life ‘N Touch can pinpoint exactly how much route the user can currently view thereby eliminating displaying unnecessary data that is outside the view of the user.

Appendices

Table of Contents

Table of Figures

Table of Tables

Appendix A - Background Information

Global Positioning System

Originally dubbed the Navigation Satellite Timing and Ranging Global Positioning System (NAVSTAR GPS), the Global Positioning System began development in 1973. By the end of 1994, twenty-four satellites orbited the Earth [6]. Each satellite sends out time data at a given frequency as a microwave signal. Receivers "catch" the propagating satellite signals and convert the difference in signal time from current time into a distance. The basic receiver/satellite interaction is outlined below:

1. A receiver receives a signal from a GPS satellite.
2. It determines the difference between current time and the time submitted by the frequency.
3. It calculates the distance of the satellite from the receiver, knowing that the signal was sent at the speed of light.
4. The receiver receives a signal from an additional two satellites and calculates the distance from them.
5. Knowing its distance from three known locations, the receiver triangulates its position. [6]

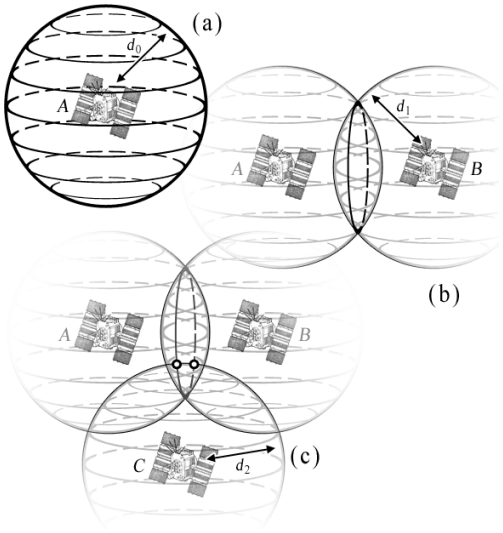

Figure 9 - Determining the position of a GPS receiver by using spheres [6]

The distance computed from satellite A, d_0, by the receiver can be thought of as a sphere with radius d_0. Thus the receiver can be anywhere on the surface of the imagined sphere. By computing a distance from satellite B, d_1, a second sphere can be visualized in which the receiver can be found. Although highly unlikely, if the spheres merely touch the location can already be determined [6]. Regardless, the two spheres must intersect at one point at least to be considered successful calculations. The intersection

creates a circle of locality for the receiver that is always d_0 from the first satellite and d_1 from the second. By utilizing a third sphere from satellite C of radius d_2 that also intersects the previous two spheres, two points of locality can be computed where the third radius intersects the aforementioned circle. One point can be eliminated because its position and velocity will put it outside the realm of possibilities, i.e. above the satellites, giving the receiver its location [6]. An image of this description can be seen in Figure 1.

Accelerometer

An accelerometer is a device which can measure the acceleration on it in one to three dimensions. The most common version of accelerometer is the capacitive accelerometer which measures acceleration by capturing the peak voltage of a flexure capacitor. Flexure capacitors consist of two fixed plates and a shared diaphragm, or seismic mass, between them, in effect creating two capacitors [7]. As the accelerometer accelerates, the diaphragm's position is altered between the fixed plates causing a shift in capacitance because of the change in distance from each plate to the diaphragm [7]. The output peak voltage will consequently vary with input acceleration causing acceleration to be measurable.

Appendix B – Cost Analysis

As stated in design specifications, the implementation of the Life 'N Touch is to be a handheld device that can easily facilitate non-vehicular travel. Therefore, when analyzing the potential development and market cost of the Life 'N Touch, all instruments used in the design must be able to be used in a handheld device.

The parts used in the developmental design are given below:

Component	Price
Processor and Display: Dell laptop	(owned previously) $999.99
GPS Receiver and Digital Compass: Wintec WBT-100	$34.59
Camera: Microsoft LifeCam VX-6000	(borrowed) $127.99
Accelerometer: Phidgets 3-axis USB	$94.99
Total	**$1257.56**

Table 2 - The parts used in the developmental design of the Life 'N Touch and corresponding prices of each component.

When analyzing the prices listed in Table 1, the total price is not the actual priced paid for development. The laptop used as the processor and display was previously purchased outside of the project timeline and the camera was borrowed. Using these factors, the actual developmental price of the Life 'N Touch is $129.58; however, this price does not adequately reflect the unit cost nor market cost of the design.

The following table is a better estimation of the market value of the Life 'N Touch:

Component	Price
Processor and Display: Compaq H3835 iPAQ Pocket PC [8]	$165.00
GPS Receiver: ISM 300 GPS OEM Firmware V324 Module [9]	$29.95
Digital Compass: OS5000-S 3-Axis Digital Compass [10]	$199.00
Camera: Motorola RAZR V3c OEM Camera [11]	$18.99
Accelerometer: ADXL320 Accelerometer Chip [12]	$6.36
Total	**$419.30**

Table 3 – The parts used in an estimated marketed design of the Life 'N Touch and their corresponding prices.

Table 2 better reflects the actual market value of the Life 'N Touch as a platform. Using a Pocket PC as an estimated market value for similar technology (i.e. handheld processing power with memory, a processor, a display, etc.), the remaining components correspond to an actual market value at which they are sold. One will notice that the price of the digital compass outweighs the price of every other component. Digital compass technology uses advanced and complex circuitry and is a small field with little demand for supply. In order for most manufacturers to continue development of said circuits, the price must be set high.

The total price given in Table 2 is purely an estimated value. The components used are real components that could be used to build the Life 'N Touch; however, the prices given are market value for each component and reflect a substantial price increase from the production cost. Assuming the development of each component within one company or one company sub-contracting design to others, the production cost of the Life 'N Touch is substantially less. Assuming a 33% increase in price per

component from production cost to market value, the price of the Life ‘N Touch drops from $419.30 to $314.55. This estimate is only a rough approximation to the true value of the Life ‘N Touch because some functionality in the Pocket PC may not be necessary to the Life ‘N Touch, thereby creating a cheaper processor because of less components.

By comparing the price of the Life ‘N Touch to that of the Apple iPhone ™, one can notice that the prices are comparable: $419.30 for the Life ‘N Touch and $399.99 for the iPhone [13]. This result was desired because it shows that the Life ‘N Touch as a platform is competitive with similar technologies.

Appendix C – Multithreading

Taken from [14].

Multithreading in C++

Multithreading is becoming an increasingly important part of modern programming. One reason for this is that multithreading enables a program to make the best use of available CPU cycles, thus allowing very efficient programs to be written. Another reason is that multithreading is a natural choice for handling event-driven code, which is so common in today's highly distributed, networked, GUI-based environments. Of course, the fact that the most widely used operating system, Windows, supports multithreading is also a factor. Whatever the reasons, the increased use of multithreading is changing the way that programmers think about the fundamental architecture of a program. Although C++ does not contain built-in support for multithreaded programs, it is right at home in this arena.

Because of its growing importance, this chapter explores using C++ to create multithreaded programs. It does so by developing two multithreaded applications. The first is a thread control panel, which you can use to control the execution of threads within a program. This is both an interesting demonstration of multithreading and a practical tool that you can use when developing multithreaded applications. The second example shows how to apply multithreading to a practical example by creating a modified version of the garbage collector from Chapter 2 that runs in a background thread.

This chapter also serves another purpose: it shows how adept C++ is at interfacing directly to the operating system. In some other languages, such as Java, there is a layer of processing between your program and the OS. This layer adds overhead that can be unacceptable for some types of programs, such as those used in a real-time environment. In sharp contrast, C++ has direct access to low-level

functionality provided by the operating system. This is one of the reasons C++ can produce higher performance code.

What Is Multithreading?

Before beginning, it is necessary to define precisely what is meant by the term *multithreading*. Multithreading is a specialized form of multitasking. In general, there are two types of multitasking: process-based and thread-based. A *process* is, in essence, a program that is executing. Thus, *process-based multitasking* is the feature that allows your computer to run two or more programs concurrently. For example, it is process-based multitasking that allows you to run a word processor at the same time you are using a spreadsheet or browsing the Internet. In process-based multitasking, a program is the smallest unit of code that can be dispatched by the scheduler.

A *thread* is a dispatchable unit of executable code. The name comes from the concept of a "thread of execution." In a *thread-based* multitasking environment, all processes have at least one thread, but they can have more. This means that a single program can perform two or more tasks concurrently. For instance, a text editor can be formatting text at the same time that it is printing, as long as these two actions are being performed by two separate threads. The differences between process-based and thread-based multitasking can be summarized like this: Process-based multitasking handles the concurrent execution of programs. Thread-based multitasking deals with the concurrent execution of pieces of the same program.

In the preceding discussions, it is important to clarify that true concurrent execution is possible only in a multiple-CPU system in which each process or thread has unrestricted access to a CPU. For single CPU systems, which constitute the vast majority of systems in use today, only the appearance of simultaneous execution is achieved. In a single CPU system, each process or thread receives a portion of

the CPU's time, with the amount of time determined by several factors, including the priority of the process or thread. Although truly concurrent execution does not exist on most computers, when writing multithreaded programs, you should assume that it does. This is because you can't know the precise order in which separate threads will be executed, or if they will execute in the same sequence twice. Thus, it's best to program as if true concurrent execution is the case.

Multithreading Changes the Architecture of a Program

Multithreading changes the fundamental architecture of a program. Unlike a single-threaded program that executes in a strictly linear fashion, a multithreaded program executes portions of itself concurrently. Thus, all multithreaded programs include an element of parallelism. Consequently, a major issue in multithreaded programs is managing the interaction of the threads.

As explained earlier, all processes have at least one thread of execution, which is called the *main thread*. The main thread is created when your program begins. In a multithreaded program, the main thread creates one or more child threads. Thus, each multithreaded process starts with one thread of execution and then creates one or more additional threads. In a properly designed program, each thread represents a single logical unit of activity.

The principal advantage of multithreading is that it enables you to write very efficient programs because it lets you utilize the idle time that is present in most programs. Most I/O devices, whether they are network ports, disk drives, or the keyboard, are much slower than the CPU. Often, a program will spend a majority of its execution time waiting to send or receive data. With the careful use of multithreading, your program can execute another task during this idle time. For example, while one part of your program is sending a file over the Internet, another part can be reading keyboard input, and still another can be buffering the next block of data to send.

Why Doesn't C++ Contain Built-In Support for Multithreading?

C++ does not contain any built-in support for multithreaded applications. Instead, it relies entirely upon the operating system to provide this feature. Given that both Java and C# provide built-in support for multithreading, it is natural to ask why this isn't also the case for C++. The answers are efficiency, control, and the range of applications to which C++ is applied. Let's examine each.

By not building in support for multithreading, C++ does not attempt to define a "one size fits all" solution. Instead, C++ allows you to directly utilize the multithreading features provided by the operating system. This approach means that your programs can be multithreaded in the most efficient means supported by the execution environment. Because many multitasking environments offer rich support for multithreading, being able to access that support is crucial to the creation of high-performance, multithreaded programs.

Using operating system functions to support multithreading gives you access to the full range of control offered by the execution environment. Consider Windows. It defines a rich set of thread-related functions that enable finely grained control over the creation and management of a thread. For example, Windows has several ways to control access to a shared resource, including semaphores, mutexes, event objects, waitable timers, and critical sections. This level of flexibility cannot be easily designed into a language because the capabilities of operating systems differ. Thus, language-level support for multithreading usually means offering only a "lowest common denominator" of features. With C++, you gain access to all the features that the operating system provides. This is a major advantage when writing high-performance code.

C++ was designed for all types of programming, from embedded systems in which there is no operating system in the execution environment to highly distributed, GUI-based end-user applications

and everything in between. Therefore, C++ cannot place significant constraints on its execution environment. Building in support for multithreading would have inherently limited C++ to only those environments that supported it and thus prevented C++ from being used to create software for nonthreaded environments.

In the final analysis, not building in support of multithreading is a major advantage for C++ because it enables programs to be written in the most efficient way possible for the target execution environment. Remember, C++ is all about power. In the case of multithreading, it is definitely a situation in which “less is more.”

Appendix D – Direct3D

Taken from [15].

World Transform (Direct3D 9)

The discussion of the world transformation introduces basic concepts and provides details on how to set up a world transform.

What Is a World Transform?

A world transform changes coordinates from model space, where vertices are defined relative to a model's local origin, to World Space, where vertices are defined relative to an origin common to all the objects in a scene. In essence, the world transform places a model into the world; hence its name. The following diagram illustrates the relationship between the world coordinate system and a model's local coordinate system.

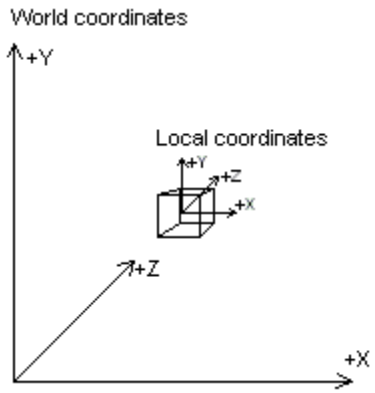

Figure 10 - The difference between a local system of coordinates and a world system of coordinates [15]

The world transform can include any combination of translations, rotations, and scalings.

Setting Up a World Matrix

As with any other transform, create the world transform by concatenating a series of matrices into a single matrix that contains the sum total of their effects. In the simplest case, when a model is at the

world origin and its local coordinate axes are oriented the same as world space, the world matrix is the identity matrix. More commonly, the world matrix is a combination of a translation into world space and possibly one or more rotations to turn the model as needed.

The following example, from a fictitious 3D model class written in C++, uses the helper functions included in the D3DX utility library to create a world matrix that includes three rotations to orient a model and a translation to relocate it relative to its position in world space.

```
/*
 * For the purposes of this example, the following variables
 * are assumed to be valid and initialized.
 *
 * The m_xPos, m_yPos, m_zPos variables contain the model's
 * location in world coordinates.
 *
 * The m_fPitch, m_fYaw, and m_fRoll variables are floats that
 * contain the model's orientation in terms of pitch, yaw, and roll
 * angles, in radians.
 */

void C3DModel::MakeWorldMatrix( D3DXMATRIX* pMatWorld )
{
    D3DXMATRIX MatTemp;  // Temp matrix for rotations.
    D3DXMATRIX MatRot;   // Final rotation matrix, applied to
                         // pMatWorld.

    // Using the left-to-right order of matrix concatenation,
    // apply the translation to the object's world position
    // before applying the rotations.
    D3DXMatrixTranslation(pMatWorld, m_xPos, m_yPos, m_zPos);
    D3DXMatrixIdentity(&MatRot);

    // Now, apply the orientation variables to the world matrix
    if(m_fPitch || m_fYaw || m_fRoll) {
        // Produce and combine the rotation matrices.
        D3DXMatrixRotationX(&MatTemp, m_fPitch);         // Pitch
        D3DXMatrixMultiply(&MatRot, &MatRot, &MatTemp);
        D3DXMatrixRotationY(&MatTemp, m_fYaw);           // Yaw
        D3DXMatrixMultiply(&MatRot, &MatRot, &MatTemp);
        D3DXMatrixRotationZ(&MatTemp, m_fRoll);          // Roll
        D3DXMatrixMultiply(&MatRot, &MatRot, &MatTemp);

        // Apply the rotation matrices to complete the world matrix.
        D3DXMatrixMultiply(pMatWorld, &MatRot, pMatWorld);
    }
}
```

After you prepare the world matrix, call the IDirect3DDevice9::SetTransform method to set it, specifying the D3DTS_WORLD macro for the first parameter.

Note Direct3D uses the world and view matrices that you set to configure several internal data structures. Each time you set a new world or view matrix, the system recalculates the associated internal structures. Setting these matrices frequently-for example, thousands of times per frame-is computationally time-consuming. You can minimize the number of required calculations by concatenating your world and view matrices into a world-view matrix that you set as the world matrix, and then setting the view matrix to the identity. Keep cached copies of individual world and view matrices so that you can modify, concatenate, and reset the world matrix as needed. For clarity, in this documentation Direct3D samples rarely employ this optimization.

Appendix E – ADXL330 Datasheet

See ADXL330_0.pdf.

Appendix F – LifeCam VX-6000 Datasheet

See TDS_LifeCamVX-6000_0606A.pdf.

Works Cited

1. (n.d.). Retrieved April 30, 2008, from http://z.about.com/d/cars/1/0/d/O/sp_06z06_hu3.jpg

2. (n.d.). Retrieved May 3, 2008, from http://www.landtrustgis.org/images/GIS%20Technology/GIS%20data%20basics/5longitudelatitude.png/image

3. (n.d.). Retrieved April 30, 2008, from http://www.shopbot.ca/i-ca/2006/8/76685733_small.jpg

4. Mirza, Y. H., & da Costa, H. (2003, July). *Introducing the New Managed Direct3D Graphics API in the .NET Framework*. Retrieved April 30, 2008, from http://msdn.microsoft.com/en-us/magazine/cc164112.aspx

5. (n.d.). Retrieved April 30, 2008, from http://www.anzlic.org.au/glossary_terms.html

6. Lammerstma, P. F. (2005). *Satellite Navigation.* Utrecht: Utrecht University.

7. PCB Piezotronics Limited.(n.d.). *How Sensors Work*. Retrieved March 25, 2008, from Sensorland: http://www.sensorland.com/HowPage011.htm

8. (n.d.). Retrieved April 28, 2008, from http://www.novatechgadgets.com/ip38hapda.html

9. (n.d.). Retrieved April 28, 2008, from http://www.inventeksys.com/shop/item.aspx?itemid=25

10. (n.d.). Retrieved April 28, 2008, from http://store.oceanserver-store.com/os3axdico2.html

11. (n.d.). Retrieved April 28, 2008, from http://cnn.cn/shop/motorola-razr-camera-p-1048.html

12. (n.d.). Retrieved April 28, 2008, from http://www.sparkfun.com/commerce/product_info.php?products_id=850

13. (n.d.). Retrieved April 28, 2008, from http://gizmodo.com/gadgets/apple/8gb-iphone-price-cut-by-200-4gb-iphone-gone-296705.php

14. (n.d.). Retrieved May 3, 2008, from http://www.devarticles.com/c/a/Cplusplus/Multithreading-in-C/

15. (n.d.). Retreived May 3, 2008, from http://msdn.microsoft.com/en-us/library/bb206365(VS.85).aspx