**ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ**

**ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ**

# Σύστημα Πλοήγησης βασισμένο στο Google Earth

Γιάννης Π. Παπαδάκης

Εξεταστική Επιτροπή:

Αντώνης Δεληγιαννάκης Καθηγητής (επιβλέπων)

Μίνως Γαροφαλάκης, Καθηγητής

Κατερίνα Μανιά, Καθηγήτρια

**Χανιά 2014**

# Περίληψη

Σε αυτή την εργασία αναπτύχθηκε ένα application βασισμένο πάνω σε υπηρεσίες της Google Maps με σκοπό την τρισδιάστατη πλοήγηση χρησιμοποιώντας ειδικότερα γεωγραφικά δεδομένα. Στα πλαίσια της εφαρμογής υλοποιήθηκαν τα εξής ζητούμενα: Πρώτον η δημιουργία ενός περιβάλλοντος όπου ο χρήστης μπορεί να αλληλεπιδρά με το σύστημα πλοήγησης, θέτοντας τα επιθυμητά στοιχεία αναχώρησης και προορισμού και η εφαρμογή να επιστέφει τη διαδρομή χρησιμοποιώντας το 3D Google Earth plugin. Επίσης με την επεξεργασία των Spasial data μέσω της υπηρεσίας Google Maps, ο χρήστης έχει την δυνατότητα να συλλέγει περεταίρω πληροφορίες για το ταξίδι του όπως το όνομα των οδών και των δρόμων, τα σημεία από τα οποία περνάει, στοιχεία του δρόμου μονής διπλής κατεύθυνσης, αυτοκινητόδρομος κ.τ.λ.

Το δεύτερο κομμάτι που υλοποιήθηκε είναι η περιστροφική κάμερα κατά την διάρκεια πλοήγησης. Καθώς το Google Earth API μας παρέχει την δυνατότητα ενσωματωμένης κάμερας, εντούτοις δεν είναι εφικτή η χρήση και η περιστροφή της κατά την διάρκεια του ταξιδιού παρά μόνο όταν αυτό είναι σταματημένο. Με βάση αυτό το πρόβλημα υλοποιήθηκε κάμερα η οποία επιτρέπει στο χρήστη να την περιστέφει στο χώρο ανεξάρτητα αν το navigation system βρίσκεται σε κίνηση ή όχι παρέχοντας όλες τις επιθυμητές πληροφορίες.

Τρίτο και τελευταίο κομμάτι ανάπτυξης είναι η ενσωμάτωση της υπηρεσίας Street View στο σύστημα πλοήγησης. Το Street View Services παρέχει την δυνατότητα φωτογραφικής απεικόνισης περιοχών μη πραγματικού χρόνου από το επίπεδο του εδάφους. Η ενσωμάτωση αυτή, παράλληλα με το navigation system καθιστά τη πλοήγηση για το χρήστη μία πλήρης και λεπτομερής εφαρμογή παρέχοντας όλα τα απαραίτητα στοιχεία.

Τέλος η καταλληλότερη γλώσσα προγραμματισμού για την ανάπτυξη της εφαρμογής επιλέχθηκε η client-side language JavaScript, η οποία χρησιμοποιείται ευρέος για την κατασκευή και υλοποίηση διαφόρων εφαρμογών στο Web.

.

# Πρόλογος

Για την διεκπεραίωση της παρούσας διπλωματικής θα ήθελα να ευχαριστήσω αρχικά το σύνολο των ανθρώπων που με έφεραν σε επαφή τόσο με το θεωρητικό όσο και με το πρακτικό κομμάτι της εργασίας, καθώς και για τη γενικότερη προσφορά και στήριξη από μέρους τους.

Ιδιαίτερα σημαντική θεωρώ την συμβολή του αδερφού μου Παπαδάκη Γιώργο οποίος είναι Ηλεκτρονικός Μηχανικός και Μηχανικός Η/Υ και μεταπτυχιακός στο Ε.Μ.Π και ασχολείται πάνω σε Γεωγραφικά Συστήματα Πληροφοριών, ο οποίος αποτέλεσε έναυσμα για να ασχοληθώ με το συγκεκριμένο θέμα, όπως και σημείο αναφοράς στη διαδικασία υλοποίησής της.

Ευχαριστώ την κοινότητα του google maps web services για τις προτάσεις και την βοήθειά τους στην ανάπτυξη της 3D Google Earth navigation system εφαρμογής.

Ευχαριστώ τους καθηγητές κ. Μ. Γαροφαλάκη και κα. Κ. Μανιά του Πολυτεχνείου Κρήτης για τη συμμετοχή τους στην επιτροπή αξιολόγησης και την επιστημονική αξιολόγηση της εργασίας μου.

Τέλος, ευχαριστώ θερμά τον επιβλέποντα καθηγητή κ. Αντώνη Δεληγιαννάκη για την ανάθεση της διπλωματικής καθώς και για την καθοδήγησή του κατά τη διάρκεια της υλοποίησης της εφαρμογής αλλά και της σχεδιαστικής επεξεργασίας της εργασίας.

**1 Εισαγωγή**

1.1 Στόχος της Εργασίας

1.2 Περιγραφή της Εργασίας

**2 Γενική Περιγραφή**

2.1 Εισαγωγή

2.2 Χωρικά Δεδομένα

2.3RESTful Πρωτόκολλο Επικοινωνίας

2.4 Mashup Web Εφαρμογή

2.5 Συστήματα Πλοήγησης

2.6 Γλώσσα Δυναμικού Περιεχομένου JavaScript

**3. Google Maps API Web Services**

3.1 Εισαγωγή

3.2 Google Maps API Web Services

3.3 Google Directions API

3.3.1 Directions Αιτήματα

3.3.2 Παράμετροι Αιτημάτων

3.3.3 Χρήση Σημείων Διαδροομής

3.3.4 Σύστημα Μέτρησης

3.3.5 Κωδικός Περιοχής

3.3.6 Directions Απαντήσεις

3.3.7 Directions Στοιχεία Απαντήσεων

4**. Street View Services**

4.1 Εισαγωγή

4.2 StreetView Panorama

4.3 Street View Τοποθεσία και Προσανατολισμός(POV)

4.4 Street View Controls

4.5 Άμεση Πρόσβαση σε Street View Δεδομένα

4.5.1 Street View Service Αιτήματα

4.5.2 StreetView Service Απαντήσεις

**5. Google Earth API**

5.1 Εισαγωγή

5.2 Google Earth Plugin

5.2.1 Χρήση Google Earth API

5.3 Placemarks(Σημεία)

5.3.1 Βασικό Placemark

5.4 Κάμερα Controls

5.4.1 Εισαγωγή

5.4.2 Τρέχον σημείο

5.4.3 Οριζόντια Κίνηση Κάμερας(Panning)

5.4.4 Οριζόντια Μετατώπιση Δίπλα στο Τρέχον Σημείο

5.5.5 Κλίση Κάμερας

5.5.6 Zoom Κάμερας

5.5 Events

5.5.1 Εισαγωγή

5.5.2 Event Listeners Συναρτήσεις

**6. 3D Google Earth Σύστημα Πλοήγησης**

6.1 Εισαγωγή

6.2 3D Google Earth Σύστημα Πλοήγησης Manual

6.2.1 Σχειαστική Διεπαφή

6.2.2 Υπηρεσίες Web

6.2.3 Controls

6.3 Πλοήγηση

6.4 Λειτουργηκότητα

6.4.1Φόρτωση Επιθυμητών Directions

6.4.2 Σχεδιασμός Διαδρομής

6.4.3 Ανάκτηση Διαδρομής

6.4.4 Υπολογισμός Απόστασης Σημείων

6.4.5 Εστίαση Κάμερας

6.4.6 Δημιουργία Λίστας Στοιχείων Διαδρομής

6.4.7 Controls

6.4.8 Street View

6.4.9 Υλοποίηση Simulator

6.4.10 Κίνηση Κάμερας

**7. Συμπεράσματα Προτάσεις**

**Βιβλιογραφία**

**ΠΑΡΑΡΤΗΜΑ**

## ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

**Σχήμα 2.1:**Συλλογή, επεξεργασία και χρήση δεδομένων GIS

**Σχήμα 2.2**: SOA και RESTful πρωτόκολλα επικοινωνίας

**Σχήμα 2.3:** Mashup αρχιτεκτονική εφαρμογής

**Σχήμα 2.4:** Navigation System Google Earth

**Σχήμα 4.1:** Διασταύρωση στο Manchester της Αγγλίας δείχνωντας από 9 διαφωρετικές γωνίες

**Σχήμα 6.1:** Σχεδιασμός του kmlString και χρωματισμός

**Σχήμα 6.2:** Έλεγχος steps με δείκτη

**Σχήμα 6.3:** Βέλτιστη απόσταση μεταξύ δύο σημείων

**Σχήμα 6.4:** Επίκεντρη γωνία φ με ακτίνα R και τόξο S

**Σχήμα 6.5:** Εσωτερικό γινόμενο δύο διανυσμάτων σε x,y άξονα.

**Σχήμα 6.6:** Γεωγραφικό Σύστημα Συντεταγμένων

**Σχήμα 6.7:** Τριγωνομετρική απεικόνιση σημείου στο χώρο

**Σχήμα 6.8:** Προσανατολισμός κάμερας

**Σχήμα 6.9:** Κίνηση των frames ανά 5sec

**Σχήμα 6.10:** Κίνηση των frames ανά 15sec

**Σχήμα 6.11 :** Υπογισμός σημείου με βάση το segmentTime

## ACTIVITY DIAGRAM

**Activity diagram 6.1:** Επιλογή Control buttons

**Activity diagram 6.2:** Επιλογή ταχύτητας

**Activity diagram 6.3** Επιλογή Street View

## ΠΙΝΑΚΑΣ ΣΥΝΤΟΜΟΓΡΑΦΙΩΝ(αλφαβητικά)

| | |
|---|---|
| **API** | Application Programming Interface |
| **ccTLD** | country code top-level domain |
| **DEM** | Digital Elevation Model |
| **DOM** | Document Object Model |
| **GIS** | Geographic Information System |
| **GPS** | Global Positioning System |
| **HTTP** | Hypertext Transfer Protocol |
| **HTML** | HyperText Markup Language |
| **JSON** | JavaScript Object Notation |
| **REST** | Representational State Transfer |
| **SOAP** | Simple Object Access Protocol |
| **URL** | Uniform Resourse Locator |
| **UI** | User Interface |
| **WebGl** | Web Graphics Library |
| **WSDL** | Web Service Definition Languages |
| **XML** | eXtensible Markup Language |

# Κεφάλαιο 1

# Εισαγωγή

## 1.1 Στόχος της Εργασίας

Στόχος της εργασίας είναι η κατασκευή ενός Web 3D navigation με την χρήση του Google Earth API, καθώς και υπηρεσίες όπως την Street View Services και της Google Maps API Web Services. Ένα application το οποίο δεν είναι standalone δηλαδή μία ανεξάρτητη εφαρμογή, αλλά αποτελεί ένα Mashup Web application, βασιζόμενη σε Restful Web Services αρχιτεκτονική, χρησιμοποιώντας ειδικότερα Spatial data Services.
Το ζητούμενο της διπλωματικής είναι η δημιουργία ενώς συστήματος πλοήγησης στο οποίο ο χρήστης μέσω ενός ενοποιημένου περιβάλλοντος, το οποίο είναι απλό και εύχρηστο στην λειτουργία του, υποβάλει ερωτήσεις στις υπηρεσίες και αντίστοιχα δέχεται τα αποτελέσματα της επιθυμητής αναζήτησης μέσω απαντήσεων. Η δυνατότητα που παρέχει το σύστημα πλοήγησης είναι να δίνει στον χρήστη διάφορα στοιχεία του δρόμου και πληροφορίες σχετικά με το περιβάλλον του ταξιδιού. Αυτό επιτεύχθηκε μέσω της συλλογής και επεξεργασίας των χωρικών δεδομένων που προσφέρουν οι υπηρεσίες της Google. Επίσης αυτό το οποίο ζητήθηκε και υλοποιήθηκε στην εφαρμογή αυτή είναι η δημιουργία κάμερας, η οποία έχει την δυνατότητα να κινείται στο χώρο είτε λειτουργώντας περιστροφικά είτε αυξομειώντας την υψομετρική διαφορά ταυτόχρονα με την διαδικασία πλοήγησης. Τέλος το τρίτο κομμάτι ανάπτυξης της εφαρμογής αποτελεί την ενσωμάτωση της υπηρεσίας της Street View στο σύτημα πλοήγησης, η οποία παρέχει φωτογραφικές λήψεις υψηλής ευκρίνειας και επιτρέπει στους χρήστες να εξερευνήσουν και να περιηγηθούν σε μια περιοχή, με πανοραμικές φωτογραφίες που έχουν τραβηχτεί από το επίπεδο του εδάφους. Με αυτό τον τρόπον υλοποήθηκε ένα πλήρες 3D Google Earth σύστημα πλοήγησης το οποίο παρέχει όλα τα επιθυμητά στοιχεία που μπορεί ο οποιοσδήποτε χρήστης να χρειαστεί κατά την διάρκεια ενός ταξιδιού.
Η υλοποίηση της εφαρμογής βασίστηκε σε γλώσσα παραγωγής δυναμικού περιεχομένου JavaScript, η οποία είναι από τις πιο δημοφιλείς client-side γλώσσες προγραμματισμού ηλεκτρονικών υπολογιστών στον Παγκόσμιο Ιστό και ανάπτυξή της πραγματοποιήθηκε σε πριβάλλον Mozilla Firefox.

## 1.2 Περιγραφή της Εργασίας

Στο επόμενο κεφάλαιο (Κεφ.2) η εργασία αυτή θα ασχοληθεί αρχικά με το τι εννούμε όταν χαρακτηρίζουμε μία εφαρμογή Mashup, τι είναι Restful Web interface design, spatial data-services, navigation system καθώς και μία ανάλυση σε γλώσσες παραγωγής δυναμικού περιεχομένου όπως JavaScript. Στο $3^{ο}$ ,$4^{ο}$ και $5^{ο}$ κεφάλαιο θα αναλυθούν τα API's και τα Services της Google τα οποία χρησιμοποιήθηκαν για να υλοποιηθεί η εφαρμογή. Στο $6^{ο}$ Κεφάλαιο περιγράφεται αναλυτικά το σύστημα πλοήγησης μέσω του Navigation Manual καθώς και η λεπτομερής επεξήγηση της εφαρμογής. Τέλος το $7^{ο}$ Κεφάλαιο εξάγει τα συμπεράσματα της εργασίας αλλά και προτάσεις για περεταίρω υλοποιήση.

# Κεφάλαιο 2

# Γενική περιγραφή

## 2.1 Εισαγωγή

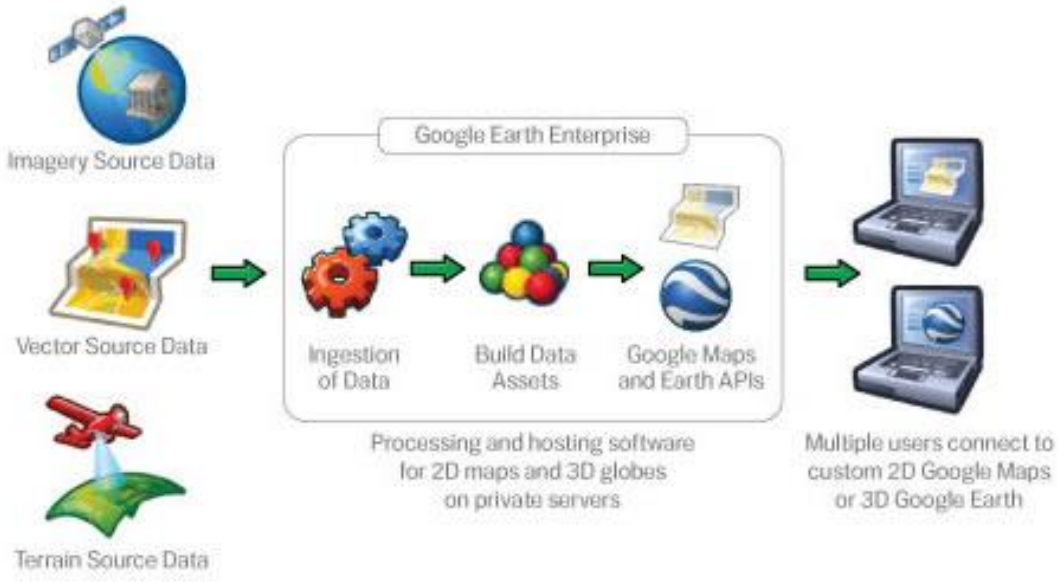
Το navigation system που υλοποιήθηκε είναι ένα application το οποίο συλλέγει δεδομένα από διάφορες υπηρεσίες του διαδυκτίου καθιστώντας την μια Mashup Web application, βασιζόμενη σε Restful Web services αρχιτεκτονική χρησιμοποιώντας ειδικότερα Spatial data. Τα δεδομένα αυτά συνδιάζονται πάνω σε μία ενιαία πλατμφόρμα χρησιμοποιώντας γλώσσα παραγωγής δυναμικού περιεχομένου την JavaScript δημιουργώντας το κατάλληλο interface για τον χρήστη .

## 2.2 Χωρικά Δεδομένα

Spatial data ή αλλιώς geospatial data είναι τα δεδομένα ή οι πληροφορίες που προσδιορίζουν τα σημεία γεωγραφικών στοιχείων πάνω στην Γη. Τα στοιχεία αυτά μπορεί να είναι φυσικά ή τεχνητά, ωκεανοί, βουνά κ.τ.λ. Τα spatial data συνήθως αποθηκεύονται τοπολογικά ή με γεωγραφικές συντεταγμένες και μπορούν να χαρτογραφηθούν και αναλύονται μέσω GIS. γνωστό ευρέως και ως Σύστημα Γεωγραφικών Πληροφοριών. Το GIS είναι ένα σύστημα διαχείρισης χωρικών δεδομένων και συσχετισμένων ιδιοτήτων. Στην πιο αυστηρή μορφή του είναι ένα ψηφιακό σύστημα, ικανό να ενσωματώσει, αποθηκεύσει, προσαρμόσει, αναλύσει και παρουσιάσει γεωγραφικά συσχετισμένες πληροφορίες. Η λειτουργία του GIS στηρίζεται σε μία database η οποία, αποτελείται από μια σείρα πληροφοριακών δεδομένων, τα οποία αφορούν την ίδια την γεωγραφική περιοχή. Η βάση αυτή μπορεί να χρησιμοποιηθεί από διαφορούς χρήστες για την κάλυψη διάφορων αναγκών όπως να αποτυπώνουν μέσω Η/Υ στο διαδίκτυο μία περίληψη του πραγματικού κόσμου, να δημιουργούν διαδραστικά ερωτήσεις χωρικού ή περιγραφικού χαρακτήρα, να αναλύουν τα χωρικά δεδομένα, να τα προσαρμόζουν και να τα αποδώσουν σε αναλογικά μέσα όπως εκτυπώσεις χαρτών και διαγραμμάτων ή σε ψηφιακά μέσα όπως αρχεία χωρικών δεδομένων και διαδραστικούς χάρτες στο Διαδίκτυο.

Τα τελευταία χρόνια, χάρη στην εξέλιξη της τεχνολογίας, έχουμε την κυκλοφορία ψηφιακών χαρτών με δυνατότητες εντοπισμού της γεωγραφικής θέσης (στίγματος) από δορυφόρους. Η λειτουργία των χαρτών αυτών βασίζεται στο GPS. Όταν παρέχονται επιπλέον δυνατότητες υπολογισμού μιας διαδρομής, τότε ονομάζονται και ηλεκτρονικοί πλοηγοί. Στο διάγραμμα που ακολουθεί επεξηγεί τις μεθόδους και τα εργαλεία που χρησιμοποιούνται για να επέλθει το τελικό αποτέλεσμα, που δεν είναι άλλο από την παραγωγή ενός χάρτη με την παροχή της απαραίτητης πληροφορίας. Το αποτέλεσμα της όλης διαδικασίας είναι ουσιαστικά η διαδρομή που ακολουθούν τα δεδομένα όταν γίνεται η κλήση στον server από τον χρήστη, για να αποτελέσουν τα βασικά χωρικά στοιχεία (μετά από επεξεργασία από τα κατάλληλα εργαλεία) της χαρτογραφικής εφαρμογής που αποτελεί και το τελικό ζητούμενο. Με αυτό τον τρόπο

λοιπόν, χρησιμοποιώντας GIS στοιχεία και spatial services όπως Google Maps, Directions και Geocoding υλοποιήθηκε ένα mashup application που παρέχει μία RESTful διεπαφή.

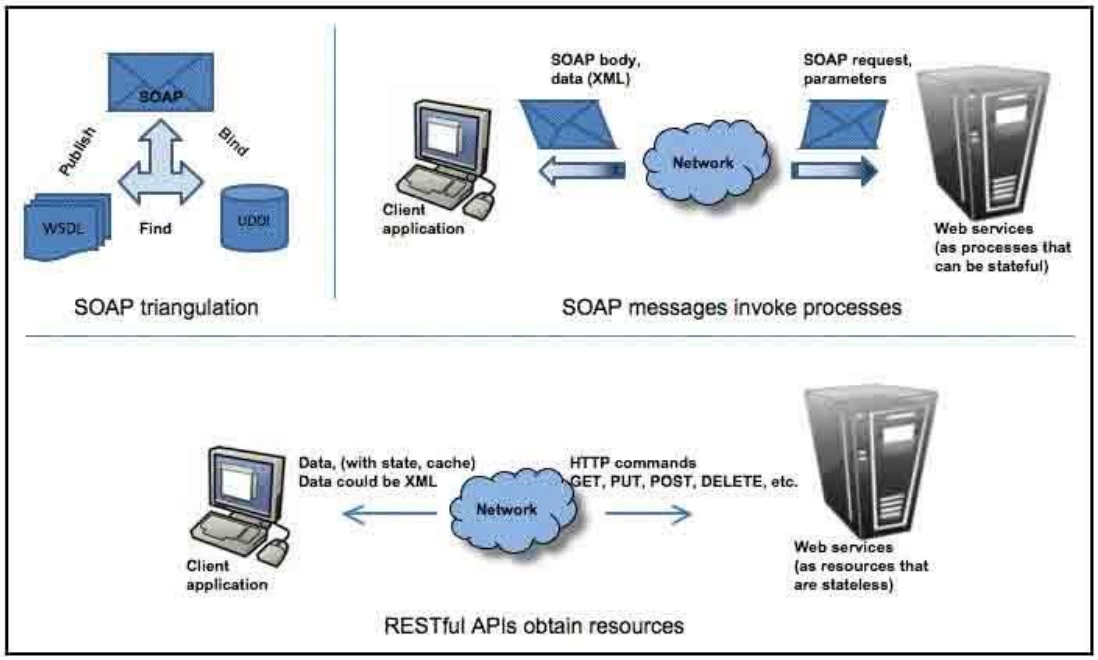



Σχήμα 2.1: Συλλογή, επεξεργασία και χρήση δεδομένων GIS

## 2.3 RESTful Πρωτόκολλο Επικοινωνίας

Το πρωτόκολλο επικοινωνίας REST είναι ένα σύνολο αρχιτεκτονικών κανόνων με τους οποίους μπορούμε να σχεδιάσουμε υπηρεσίες Web. Οι κανόνες αυτοί εστιάζουν στις υπηρεσίες του συστήματος που θεωρούνται πόροι συμπεριλαμβανομένου του τρόπου με τον οποίο οι πόροι αυτοί μετρέπονται σε διευθύνσεις και μεταφέρονται μέσω του HTTP πρωτοκόλλου. Τα τελευταία χρόνια το REST έχει αναδειχθεί ως κυριίαρχο Web design service μοντέλο εκτοπίζοντας ως επι το πλείστον το SOAP και WSDL-based interface design, λόγω ότι είναι πολύ απλό, αξιόπιστο και ευέλικτο στην χρήση του. Πιο συγκεκριμένα το REST service είναι lightweight με την έννοια ότι παρέχει μόνο το απετούμενο output σε μορφή json/xml στο οποίο οι παράμετροι καθορίζονται στο UriTemplate σε σχέση με το SOAP το οποίο είναι heavyweight δηλαδή παρέχει πρόσθετες πληροφορίες οι οποίες μπορούν να το επιβαρύνουν καθιστώντας το πιο αργό. Επίσης τα REST αιτήματα μπορούν να δημιουργηθούν γρήγορα χωρίς την επιπρόσθετη ενθυλάκωση που απαιτείται από το SOAP και να αποθηκευτούν εύκολα από την υπάρχουσα υποδομή μειώνοντας το φορτίο των servers και το κόστος εύρους ζώνης.
Η REST-style αρχιτεκτονική υποστηρίζει client-server μοντέλο . Οι clients στέλνουν HTTP αιτήματα στους servers, οι οποίοι παρέχουν πόρους όπως αρχεία HTML, επεξεργάζονται τα αιτήματα και στέλνουν πίσω τις κατάλληλες απαντήσεις σε μορφή json ή xml. Με αυτό τον τρόπο οι χρήστες μπορούν είτε να εισάγουν αιτήματα σαν ένα απλό URL και να πάρουν τις αντίστοιχες απαντήσεις, είτε να καλούν τις λειτουργίες της επιθυμητής υπηρεσίας προγραμματιστικά χρησιμοποιώντας την κατάλληλη γλώσσα και να επεξεργάζονται το απαντητικό xml ή json έγγραφο πάλι

προγραμματιστικά. Πάνω σε αυτό το Web μοντέλο βασίστηκε η υλοποίηση της εφαρμογής, στην οποία ο χρήστης στέλνει αιτήματα σε ένα Web server μέσω της εφαρμογής που υλοποιήθηκε και ο server στέλνει πίσω τις αντίστοιχες απαντήσεις εφόσον πρώτα έχει επεξεργαστεί τα δεδομένα. Η εφαρμογή είναι μία mushup web application η οποία συνδιάζει διαφορετικά δεδομένα και υλοποιήσεις από διαφορετικές πηγές.




Σχήμα 2.2: SOA και RESTful πρωτόκολλα επικοινωνίας

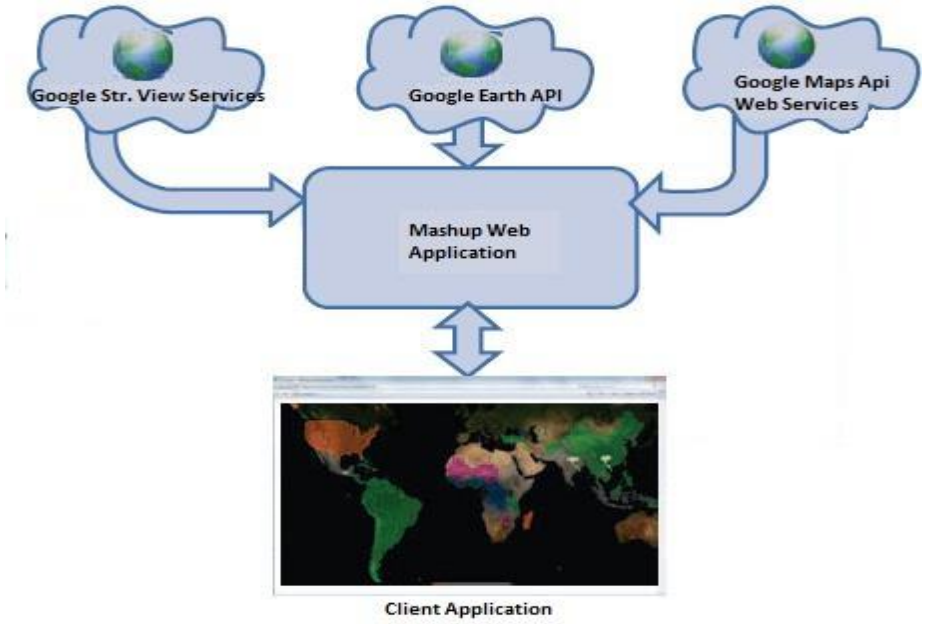
## 2.4 Mashup web Εφαρμογή

Όπως αναφέραμε στην προηγούμενη ενότητα η εφαρμογή που αναπτύχθηκε είναι μία mushup web application, μια διαδικτυακή εφαρμογή που συνδυάζει δεδομένα από περισσότερες από μία πηγές σε ένα ενιαίο εργαλείο με σκοπό την δημιουργία ενός κοινού interface. Μπορούμε να αναφέρουμε ότι τα Mashups κατηγοριοποιούνται σε τρεις γενικές μορφές τα consumer mashups, τα data mashups, και τα business mashups. Ο πιο γνωστός τύπος είναι τα consumer mashups, που εξηγείται καλύτερα από τις πολλές εφαρμογές των Google Maps και πιο συγκεκριμένα τα Mapping mashups, τα οποία χρησιμοποιούν μια υπηρεσία παροχής χαρτών (π.χ. Google, Yahoo) συνδυάζοντας δεδομένα με σημεία πάνω στο χάρτη. Με αυτόν τον συνδιαστικό τρόπο παροχής υπηρεσιών πετυχαίνουμε τα εξής:

- σύνθεση διαφορετικών υπηρεσιών raw data
- απλοποίηση υπηρεσιών API
- φιλικό περιβάλλον στον απλό χρήστη

Όσον αφορά την συνθέση υπηρεσιών στην συγκεκρικένη εφαρμογή που υλοποιήθηκε, χρησιμοποιήθηκαν εκτενέστερα οι υπηρεσίες που παρέχει η Google.

Πιο συγκεκριμένα η εφαρμογή αναπτύχθηκε πάνω σε Google Maps API Web Services, Google Earth API καθώς και Google Street View Services. Αντλώντας μόνο τα βασικά στοιχεία των παραπάνω API's-services για την υλοποίηση της εφαρμογής, επιτεύχθηκε ένα σύνθετο αλλά απλό για τον χρήστη navigation system που επιτρέπει να αλληλεπιδρά με τις υπηρεσίες, εισάγοντας απλά τα επιθυμητά δεδομένα και λαμβάνοντας τα αντίστοιχα αποτελέσματα, χωρίς να χρειάζεται ο χρήστης να έχει κάποια προγραμματιστική ή άλλη εξιδικευμένη γνώση πάνω σε αυτές τις υπηρεσίες. Στο σχήμα που ακολουθεί παρατηρούμε τη μορφή αρχιτεκτονικής mushup που χρησιμοποιήσαμε για την υλοποιήση της εφαρμογής




Σχήμα 2.3: Mashup αρχιτεκτονική της εφαρμογής

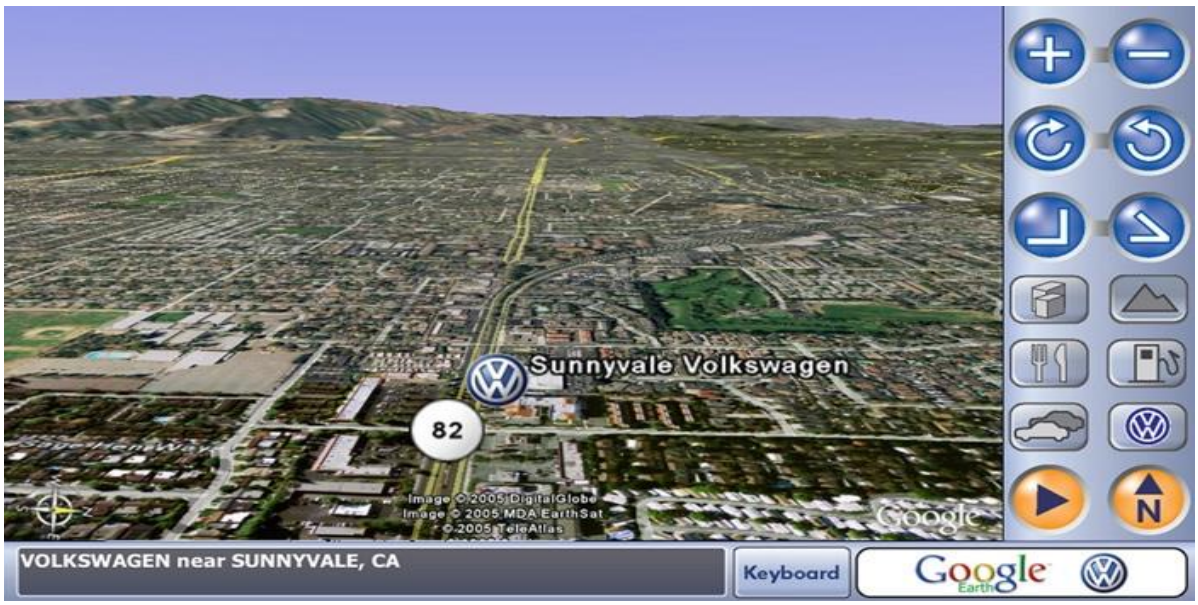
## 2.5 Συστήματα Πλοήγησης

Ένα σύστημα πλοήγησης προσδιορίζει τη θέση του χρήστη στον πλανήτη. Υπάρχουν διάφορα συστήματα πλοήγησης που χρησιμοποιούνται ανάλογα με τις εκάστοτε ανάγκες στα οποία βρίσκονται αποθηκευμένοι ψηφιακοί χάρτες, ο υπολογισμός και η ενημέρωση των οποίων γίνεται με τη βοήθεια δορυφόρων. Ένα σύστημα πλοήγησης βοηθά τους οδηγούς αυτοκινήτων, μοτοσικλετιστές, ποδηλάτες και περιηγητές να φθάσουν στον προορισμό τους. Είναι ένα τεχνικό σύστημα που μπορεί να παρέχει γεωγραφικά στοιχεία υπό τη μορφή οδικών, εναέριων, θαλασσίων ή τοπολογικών χαρτών.

Υπάρχουν διάφορα είδη συστημάτων πλοήγησης:

- ενσωματωμένα στο αυτοκίνητο με χάρτες διαφόρων χωρών
- PND με χάρτες για μία ή περισσότερες χώρες
- συσκευές μοτοσικλέτας
- συσκευές για το ποδήλατο ή την πεζοπορία
- συσκευές για συνδυασμένη χρήση
- συσκευές για υποβρύχια χρήση
- λογισμικό για φορητούς υπολογιστές και κινητά

Τα συστήματα πλοήγησης χρησιμοποιούν χάρτες που έχουν χαρτογραφηθεί ειδικά για το σκοπό αυτό. Οι χρήστες συχνά χρησιμοποιούν τα λεγόμενα «waypoints». Ένα σημείο αναφοράς είναι ένα σημείο στη Γη που έχει αποθηκευθεί στο χάρτη με τις συντεταγμένες του. Οι χρήστες μπορούν να το ορίσουν μόνοι τους ή να κατεβάσουν σημεία αναφοράς άλλων από το διαδίκτυο. Όταν το σύστημα πλοήγησης λάβει εντολή εύρεσης της διαδρομής, προσδιορίζει με τη βοήθεια δορυφόρων την εκάστοτε θέση και στη συνέχεια υπολογίζει τα σημεία που συνδέονται μεταξύ τους. Αντίστοιχα εμφανίζονται η διαδρομή και η θέση στον ψηφιακό χάρτη. Η εφαρμογή που υλοποιήθηκε, βασίστηκε πάνω σε αυτή την λογική των συστηματων πλοήγησης χρησιμοποιώντας το 3d viewer και τα directions API της Google . Το 3d viewer της Google παρέχει την δυνατότητα να απεικονήσει τρισδιάστατα γραφικά όπως κτήρια, βουνά, γέφυρες, δρόμους κ.τ.λ χάρη στο DEM της Google Earth το οποίο είναι ένα ψηφιακό μοντέλο 3διάστατης απεικόνισης της επιφάνειας και των κτηρίων ενός εδάφους. Τα directions API απεναντίας είναι εργαλεία τα οποία υπολογίζουν αποστάστεις μεταξύ σημείων στέλοντας αιτήματα προς τις υπηρεσίες της Google Maps. Η εφαρμογή αυτή ενοποιήσε τις διαφορετικές αυτές υπηρεσίες δημιουργώντας ένα ενιαίο περιβάλλον 3d χάρτη πλοήγησης για τον χρήστη που επιθυμεί να πλοηγηθεί.




Σχήμα 2.4: Navigation System Google Earth

## 2.6 Γλώσσα Δυναμικού Περιεχομένου JavaScript

Η εφαρμογή που υλοποιήθηκε, βασίστηκε εξ' ολοκλήρου σε γλώσσα προγραμματισμού JavaScript, η οποία είναι πολύ δημοφιλής στους web developers καθώς είναι απλή στην σύνταξη της και υποστηρίζεται απ' όλους τους δημοφιλείς browsers. Η JavaScript είναι μια αντικειμενοστραφής scripting γλώσσα (object-oriented scripting language) που χρησιμοποιείται για δώσει πρόσβαση σε αντικείμενα, μεταξύ του πελάτη(client) και λοιπές εφαρμογές. Χρησιμοποιείται κυρίως με την μορφή client-side JavaScript, δηλαδή στην μεριά του πελάτη, και εφαρμόζεται ως μια ολοκληρωμένη συνιστώσα του web browser, επιτρέποντας την ανάπτυξη ενισχυμένων διεπαφών και δυναμικών ιστοσελίδων. Η JavaScript είναι μια διάλεκτος του ECMAScript και χαρακτηρίζεται ως μια δυναμική, weakly typed, prototype-based γλώσσα με πρώτης κατηγορίας λειτουργίες (first-class function).

Η κύρια χρήση της JavaScript είναι η συγγραφή κώδικα και λειτουργιών που ενσωματώνονται ή περιλαμβάνονται από έγγραφα HTML και αλληλεπιδρούν με το DOM ενός εγγράφου. Με την βοήθεια της JavaScript μπορούμε να πετύχουμε:

- Πολυμερή έγγραφα με πλαίσια
- Επαναφόρτωση μέρους του παραθύρου
- Δημιουργία έγγραφων με αλληλεπίδραση
- Περισσότερος έλεγχος στην αλληλεπίδραση με το χρήστη
- Έγγραφα με μνήμη
- Ζωντανά έγγραφα
- Μηνύματα που ολισθαίνουν
- Ρολόγια
- Χρονικός μηχανισμός αντίστροφης μέτρησης
- Έγγραφα με αυτόματη ενημέρωση κ.α.

Τέλος επειδή ο κώδικας JavaScript μπορεί να εκτελεστεί τοπικά σε έναν browser και όχι σε έναν απομακρυσμένο διακομιστή, ανταποκρίνεται στις ενέργειες των χρηστών με μεγάλη ταχύτητα, κάνοντας την JavaScript πιο ευέλικτη και responsive.

# Κεφάλαιο 3

## Google Maps API Web Services

## 3.1 Εισαγωγή

Το Google Maps API είναι μια διεπαφή προγραμματισμού εφαρμογών της Google το οποίο δίνει τη δυνατότητα ενσωμάτωσης ενός δυναμικού χάρτη σε ιστοσελίδες με τη χρήση Javascript. Το API παρέχει έναν αριθμό εργαλείων για το χειρισμό χαρτών και την προσθήκη περιεχομένου σε αυτούς μέσω διαφόρων υπηρεσιών, όπως και στην επίσημη ιστοσελίδα του Google Maps (http://maps.google.com/), επιτρέποντας έτσι τη δημιουργία εύρωστων εφαρμογών που εκμεταλλεύονται γεωγραφικές πληροφορίες.
Βασικό στοιχείο οποιασδήποτε Google Maps API εφαρμογής είναι ο ίδιος ο χάρτης, ο οποίος φορτώνεται και αρχικοποιείται με συγκεκριμένες διαστάσεις (ύψος και πλάτος), κέντρο (γεωγραφικό πλάτος, γεωγραφικό μήκος), επίπεδο εστίασης και τύπο (κανονικό, δορυφορικό, υβριδικό). Στη συνέχεια είναι δυνατή η αλληλεπίδραση με το χάρτη μέσω διαφόρων μηχανισμών όπως είναι τα events, τα controls, τα overlays αλλά και συναρτήσεις επεξεργασίας των χαρακτηριστικών του.
Στην ενότητα αυτή θα αναλυθεί το Google Directions API, μία από τις web υπηρεσίες που βασίστηκε η διπλωματική.

## 3.2 Google Maps API Web Services

Το Google Maps API παρέχει υπηρεσίες web ως διεπαφή για την υποβολή αιτημάτων Maps API δεδομένων από εξωτερικές υπηρεσίες και τη χρήση τους σε εφαρμογές για χάρτες. Οι υπηρεσίες web χρησιμοποιούν HTTP αιτήματα σε συγκεκριμένες διευθύνσεις URL, θέτοντας παραμέτρους URL στις υπηρεσίες. Σε γενικές γραμμές, οι υπηρεσίες αυτές επιστρέφουν δεδομένα από το HTTP είτε ως JSON ή XML για parsing ή/και processing.

Ένα τυπικό Web Service αίτημα δίνεται γενικά από την παρακάτω φόρμα:

`http://maps.googleapis.com/maps/api/service/output?parameters`

όπου service δηλώνει το συγκεκριμένο service(π.χ directions) που απαιτήθηκε και το output δηλώνει το format(json ή xml) . Μία από τις web services που βασίστηκε η διπλωματική μας είναι το Google Directions Api.

## 3.3 Google Directions API Services

Το Google Directions API είναι μία υπηρεσία η οποία υπολογίζει διαδρομές(directions) μεταξύ τοποθεσιών (locations) χρησιμοποιώντας HTTP αιτήματα. Οι διαδρομές μπορεί να επιλεχθούν με ποικίλους τρόπους μεταφοράς όπως οδήγησης, πεζοπορίας, ποδηλασίας, Μέσα Μαζικής Μεταφοράς κ.τ.λ. Τα directions μπορεί να καθορίζουν αφετηρίες, προορισμούς, σημεία, είτε ως text strings(π.χ "Chicago, IL" ή "Darwin, NT, Australia") είτε ως γεωγραφικές συντεταγμένες. Το

Directions API μπορεί να επιστρέψει πολλαπλά directions χρησιμοποιώντας σειρά από σημεία(waypoints). Γενικά το service αυτό έχει σχεδιαστεί για να υπολογίζει static addresses δηλαδή διευθύνσεις που είδη είναι γνωστοποιημένες το Google Maps application και όχι σχεδιασμένο για να απαντάει στον χρήστη σε real time.

### 3.3.1 Directions Αιτήματα

Ένα Directions Api αίτημα συντάσεται με την εξής μορφή:

`http://maps.googleapis.com/maps/api/directions/output?parameters`

όπου το output μπορεί να είναι μία από τις παρακάτω μορφές:

- json
- xml

### 3.3.2 Παράμετροι Αιτημάτων

Ορισμένοι παράμετροι που χρησιμοποιούνται στο URL είναι απαιτούμενοι, ενώ άλλοι επιλέγονται προεραιτικά. Οι απαιτούμενες παράμετροι που χρησιμοποιούνται με τις αντίστοιχες τιμές τους είναι οι εξής:

#### Απαιτούμενες Παράμετροι

- origin — Διεύθυνση (address) ή γεωγραφικές συντεταγμένες (textual latitude/longitude), τιμές που χρησιμοποιούνται για τον υπολογιασμό των διαδρομών. Αν βάλουμε σαν παράμετρο μια διεύθυνση σαν string, το Direction service θα αποκωδικοποιήσει το string και θα το μετατρέψει σε γεωγραφικό πλάτος/μήκος συντεταγμένων με σκοπό να υπολογίσει τις διαδρομές. Αν περάσουμε συντεταγμένες οι τιμές θα παραμείνουν ως έχουν.

- destination — Διεύθυνση (address) ή γεωγραφικές συντεταγμένες (textual latitude/longitude), τιμές που χρησιμοποιούνται για τον υπολογιασμό των διαδρομών. Αν βάλουμε σαν παράμετρο μια διεύθυνση σαν string, το Direction service θα αποκωδικοποιήσει το string και θα το μετατρέψει σε γεωγραφικό πλάτος/μήκος συντεταγμένων με σκοπό να υπολογίσει τις διαδρομές. Αν περάσουμε συντεταγμένες οι τιμές θα παραμείνουν ως έχουν.

- sensor — Υποδεικνύει εάν ή όχι τα directions αιτήματα προέρχονται από συσκευή με location sensor. Η τιμή αυτή θα πρέπει να είναι true ή false.

## Προαιρετικές Παράμετροι

- mode — Καθορίζει το travel mode (walking, transit, bicycling driving(default value).

- waypoints — Καθορίζει ένα πίνακα από σημεία. Τα waypoints αλλάζουν την διαδρομή μέσα από συγκεκριμένα locations τα οποία είτε είναι συντεταγμένες είτε διευθύνσεις οι οποίες θα αποκωδικοποιηθούν. Τα waypoints υποστηρίζουν μόνο διαδρομές μέσω οδήγησης, πεζοπορίας και ποδηλασίας.

- alternatives — Αν είναι επιλογή true, τότε το Direction servise καθορίζει ενναλακτικές διαδρομές.

- avoid — Υποδεικνύει ποιές διαδρομές με συγκεκριμένα χαρακτηριστικά θα πρέπει να αποφεύγονται όπως π.χ

    - tolls -Υποδεικνύει ότι η υπολογισμένη διαδρομή θα πρέπει να αποφεύγει υπερηψωμένους δρόμους και γέφυρες.

    - highways-Υποδεικνύει ότι οι υπολογισμένη διαδρομή θα πρέπει να αποφεύγει τους αυτοκινητόδρομους.

Έχουμε την δυνατότητα να επιλέξουμε μία διαδρομή, η οποία παίρνει και τις δυο τιμές παράλληλα για παράδειγμα avoid=tolls|highways.

- language — Η γλώσσα στην οποία θα επιστρέψει το αποτέλεσμα.

- unit — Καθορίζει το unit system κατά την εμφάνιση των αποτελεσμάτων.

- region — Καθορίζει τον Κωδικό Περιοχής

- departure time — Υποδεικνύει τον επιθυμητό χρόνο αναχώρησης

- arrival time — Υποδεικνύει τον επιθυμητό χρόνο άφιξης

Παρακάτω επισυνάπτονται μερικά Directions αιτήματα:

1. Το παρακάτω αίτημα(request) επιστρέφει driving directions θέτοντας σαν αφετηρία το Toronto, Ontario και προορισμό το Montreal , Quebec.

    http://maps.googleapis.com/maps/api/directions/json?origin=Toronto&destination=Montreal&sensor=false

2. Αλλάζοντας τις παραμέτρους mode και avoid μπορεί να τροποποιηθεί το αρχικό αίτημα από driving(default value) και να επιστέψει directions π.χ για bicycling τα οποία directions θα αποφευγουν τους αυτοκινητόδρομους. Το παρακάτω αίτημα υποδεικνύει την υλοποίηση αυτή:

   http://maps.googleapis.com/maps/api/directions/json?origin=Toronto&destination=Montreal&sensor=false&avoid=highways&mode=bicycling

3. Το παρακάτω παράδειγμα ψάχνει για διαδρομές M.M.M θέτοντας σαν αφετηρία το Brooklyn, New York και προορισμό το Queens, N.Y. Όταν πρόκειται για M.M.M αίτημα πρέπει να καθορίσουμε και τις ποραμέτρους departure_time ή/και arrival_time. Ακολουθεί το παράδειγμα:

   http://maps.googleapis.com/maps/api/directions/json?origin=Brooklyn&destination=Queens&sensor=false&departure_time=1343605500&mode=transit

### 3.3.3 Χρήση Σημείων Διαδρομής

Στον υπολογισμό μίας διαδρομής μέσω από το Directions API μπορούμε να χρησιμοποιήσουμε waypoints για driving, walking, bicycling διαδρομές αλλά όχι για transit. Τα σημεία διαδρομής μας επιτρέπουν να καθορίσουμε διαδρομές μέσω επιπλέων περιοχών από τα οποία και θα περάσουμε κατά την πλοήγησή μας. Τα waypoints αποτελούνται από μία ή και περισσότερες διευθύνσεις ή περιοχές διαχωρισμένες μέσω του (|) χαρακτήρα. Για παράδειγμα το ακόλουθο URL αρχικοποιεί ένα Directions αίτημα για μία διαδρομή μεταξύ Boston, MA και Concord, MA μέσω Charlestown και Lexington, με αυτή την σειρά.

http://maps.googleapis.com/maps/api/directions/json?origin=Boston,MA&destination=Concord,MA&waypoints=Charlestown,MA|Lexington,MA&sensor=false

Το Directions API υπολογίζει by default τα waypoints με την σείρα τα οποία δίνονται. Προαιτετικά μπορούμε να θέσουμε optimize: true σαν πρώτο στοιχείο μέσα στη waypoints παράμετρο οπού επιτρέπει το Directions να βελτιστοποιήσει την διαδρομή αναδιατάσσοντας τα waypoints με πιο έξυπνο τρόπο.

### 3.3.4 Σύστημα Μονάδας Μέτρησης

Τα αποτελέσματα του Directions περιέχουν text μέσα στα distance πεδία για να υποδεικνύουν την απόσταση ενός συγκεκριμένου step μιας διαδρομής. Από προεπιλογή το text αυτό χρησιμοποιεί το unit system της εκάστοτε περιοχής ή της

χώρας. Τα σημεία αυτά εκφράζονται μέσω συντεταγμένων παρά μέσω διευθύνσεων σύμφωνα πάντα με το σύστημα μέτρησης.

Για παράδειγμα, μια διαδρομή από το "Chicago, IL" σε "Τορόντο, Οντάριο" θα εμφανίσει τα αποτελέσματα σε μίλια, ενώ το αντίστροφο δρομολόγιο θα εμφανίσει τα αποτελέσματα σε χιλιόμετρα. Μπορούμε να ορίσουμε εμείς το unit system θέτοντας units παραμέτρους, βάζοντας μία από τις ακόλουθες τιμές

- metric — προσδιορίζει το σύστημα μονάδς μέτρησης. Η τιμή που επιστρέφεται για τις αποστάσεις είναι σε χιλιομέτρα και σε μέτρα.
- imperial — προσδιορίζει το Αγγλικό σύστημα μονάδας μέτρησης. Η τιμή που επιστρέφεται για τις αποστάσεις είναι σε μίλια και σε πόδια

### 3.3.5 Κωδικός Περιοχής

Μπορούμε να θέσουμε το Direction service με αυτό τον τρόπο, που θα μας επιστέφει μία συγκεκριμένη περιοχή χρησιμοποιώντα παραμέτρους κωδικών περιοχής. Οι παραμετροι αυτοι πέρουν ccTLD στοιχεία. Οι περισσότεροι ccTLD κώδικες είναι ISO 3166-1 κώδικες με καποίες εξαιρέσεις. Γεια παράδειγμα ένα αίτημα διαδρομής από το ‘’Toledo’’ στην ‘’Madrid’’ επιστέφει το εξής:

http://maps.googleapis.com/maps/api/directions/json?origin=Toledo&destination=Madrid&region=es&sensor=false

```
{
  "status": "OK",
  "routes": [ {
    "summary": "AP-41",
    "legs": [ {
        ...
    } ],
    "copyrights": "Map data ©2010 Europa Technologies,
Tele Atlas",
    "warnings": [ ],
    "waypoint_order": [ ]
  } ]
}
```

### 3.3.6 Directions Απαντήσεις

Τα Directions επιστρέφουν απαντήσεις με την μορφή json το οποίο υποδηλώνει το output μέσα στο URL path.

## JSON Output

Ένα παράδειγμα από ένα HTTP αίτημα το οποίο υπολογίζει την διδαρομή από το Chicago, IL στο Los Angeles, CA μέσω δύο waypoints Joplin, MO και Oklahoma City, OK:

http://maps.googleapis.com/maps/api/directions/json?origin=Chicago,IL&destination=Los+Angeles,CA&waypoints=Joplin,MO|Oklahoma+City,OK&sensor=false

Το αποτέλεσμα του JSON δίνεται παρακάτω:


```
{
  "status": "OK",
  "routes": [ {
    "summary": "I-40 W",
    "legs": [ {
      "steps": [ {
        "travel_mode": "DRIVING",
        "start_location": {
          "lat": 41.8507300,
          "lng": -87.6512600
        },
        "end_location": {
          "lat": 41.8525800,
          "lng": -87.6514100
        },
        "polyline": {
          "points": "a~l~Fjk~uOwHJy@P"
        },
        "duration": {
          "value": 19,
          "text": "1 min"
        },
        "html_instructions": "Head \u003cb\u003enorth\u003c/b\u003e on
\u003cb\u003eS Morgan St\u003c/b\u003e toward \u003cb\u003eW Cermak
Rd\u003c/b\u003e",
        "distance": {
          "value": 207,
          "text": "0.1 mi"
        }
      },
      ...
      ... additional steps of this leg
    ...
    ... additional legs of this route
      "duration": {
        "value": 74384,
        "text": "20 hours 40 mins"
      },
      "distance": {
        "value": 2137146,
        "text": "1,328 mi"
      },
```

```
      "start_location": {
        "lat": 35.4675602,
        "lng": -97.5164276
      },
      "end_location": {
        "lat": 34.0522342,
        "lng": -118.2436849
      },
      "start_address": "Oklahoma City, OK, USA",
      "end_address": "Los Angeles, CA, USA"
    } ],
"copyrights": "Map data ©2010 Google, Sanborn",
    "overview_polyline": {
      "points":

"a~l~Fjk~uOnzh@vlbBtc~@tsE`vnApw{A`dw@~w\\|tNtqf@l{Yd_Fblh@rxo@b}@xxS
fytAblk@xxaBeJxlcBb~t@zbh@jc|Bx}C`rv@rw|@rlhA~dVzeo@vrSnc}Axf]fjz@xfF
bw~@dz{A~d{A|zOxbrBbdUvpo@`cFp~xBc`Hk@nurDznmFfwMbwz@bbl@lq~@loPpxq@b
w_@v|{CbtY~jGqeMb{iF|n\\~mbDzeVh_Wr|Efc\\x`Ij{kE}mAb~uF{cNd}xBjp]fulB
iwJpgg@|kHntyArpb@bijCk_Kv~eGyqTj_|@`uV`k|DcsNdwxAott@r}q@_gc@nu`CnvH
x`k@dse@j|p@zpiAp|gEicy@`omFvaErfo@igQxnlApqGze~AsyRzrjAb__@ftyB}pIlo
_BflmA~yQftNboWzoAlzp@mz`@|}_@fda@jakEitAn{fB_a]lexClshBtmqAdmY_hLxiZ
d~XtaBndgC"
    },
    "warnings": [ ],
    "waypoint_order": [ 0, 1 ],
    "bounds": {
      "southwest": {
        "lat": 34.0523600,
        "lng": -118.2435600
      },
      "northeast": {
        "lat": 41.8781100,
        "lng": -87.6297900
      }
    }
  } ]
}
```


### 3.3.7 Directions Στοιχεία Απαντήσεων

Οι Directions απαντήσεις περιέχουν δύο βασικά στοιχεία:

- status — περιέχει metadata δεδομένα από το αίτημα.
- routes — περιέχει πίνακα από routes από το σημείο έναρξης στο σημείο προορισμού

## Status Codes:

Το status πεδίο στις Directions απαντήσεις περιέχει τη κατάσταση της απάντησης καθώς επίσης και debugging πληροφορίες για θα βοηθήσουν να εντοπίσουμε γιατί απέτυχε το Direction service. Το status πεδίο μπορεί να περιέχει τις εξής τιμές:

- OK — υποδεικνύει ότι η απάντηση έχει έγκυρο αποτέλεσμα.
- NOT_FOUND — υποδεικνύει τουλάχιστον ένα από τα σημεία που αναφέρονται στα αιτήματα αφετηρίας, προορισμού ή σημεία δεν μπορεί να κωδικοποιηθεί γεωγραφικά.
- ZERO_RESULTS — υποδεικνύει την μη εύρεση διαδρομής μεταξύ αφετηρίας και προορισμού.
- MAX_WAYPOINTS_EXCEEDED — υποδεικνύει ότι πάρα πολλά σημεία δόθηκαν στο αίτημα. Ο μέγιστος επιτρεπόμενος αριθμός σημείων είναι 8 μαζί με το αφερτηρίας και προορισμού ( το Google Maps API for Business μπορεί να έχει μέχρι και 23 σημεία).
- INVALID_REQUEST — υποδεικνύει άκυρο αίτημα εξαιτίας μίας άκυρης παραμέτρου ή τιμή παραμέτρου.
- OVER_QUERY_LIMIT — υποδεικνύει ότι το service έχει λάβει πολλά αιτήματα από την εφαρμογή μας εντός της επιτρεπόμενης χρονικής περιόδου.
- REQUEST_DENIED — υποδεικνύει την απόρριψη του Direction service από την εφαρμογή μας.
- UNKNOWN_ERROR — υποδεικνύει ότι ένα αίτημα διαδρομής δε μπορεί να υλοποιηθεί εξαιτίας σφάλματος του διακομιστή(server error).

## Routes:

Όταν το Directions API επιστρέφει αποτελέσματα τα τοποθετεί μέσα σε ένα json routes πίνακα. Ακόμα και αν το service δεν επιστρέψει αποτελέσματα δηλαδή δεν υπάρχει σημείο αναχώρησης ή/και προορισμού, το Directions API θα επιστρέψει ένα αδείο πίνακα routes. Κάθε στοιχείο περιέχει ένα μοναδικό αποτέλεσμα από το καθορισμένο σημείο αφετηρίας και προορισμού. Η κάθε διαδρομή μπορεί να αποτελείται από ένα ή και περισσότερα legs ανάλογα με το κατά πόσον τα waypoints προσδιορίζονται. Επίσης το route περιέχει πνευματικά δικαιώματα και προειδοποιήσεις που πρέπει να εμφανίζονται στον χρήστη σαν πληροφορίες δρόμου.
Κάθε πεδίο του route μπορεί να περιέχει τα εξής πεδία:

- summary — περιέχει ένα μικρο text περιγραφής της διαδρομής(route).
- legs[] — περιέχει έναν πίνακα, ο οποίος περιέχει με την σειρά του πληροφορίες για ένα leg του δρομολογίου, μεταξύ δύο τοποθεσιών της συγκεκριμένης διαδρομής. Ξεχωριστό leg δημιουργείται για κάθε σημείο

(waypoints) ή για κάθε προορισμό. Μία διαδρομή χωρίς σημεία θα περιέχει έναν ακριβώς πίνακα leg. Κάθε leg αποτελείται από μία σειρά από steps.

- waypoint_order — περιέχει έναν πίνακα που δείχνει την σειρά των σημείων της διαδρομής. Τα σημεία αυτά μπορούν να αναδιαταχθούν εάν η παράμετρος optimize: στο αίτημα έχει επιλεχθεί true(optimize: true).

- overview_polyline — περιέχει ένα object που κατέχει έναν πίνακα από κωδικοποιημένα σημεία που αντιπροσωπεύουν κατά προσέγγιση(εξομάλυνση) την διαδρομή από τα αποτελέσματα των directions.

- bounds — περιέχει την οριοθέτηση της δαιδρομής.

- copyrights — περιέχει τα copyright της διαδρομής.

- warnings[] — περιέχει έναν πίνακα από προειδοποιήσεις που πρέπει να εμφανίζονται κατά την διάρκεια της διαδρομής

## Legs:

Κάθε στοιχείο μέσα στο legs πίνακα καθορίζει ένα μοναδικό leg του ταξιδιου, από την αφητηρία του στον προορισμό του. Διαδρομές που δεν αποτελούνται από σημεία, ορίζονται από ένα και μόνο leg, ενώ για διαδρομές που ορίζονται από ένα ή και περισσότερα σημεία, περιέχουν ένα ή και περισσότερα legs.

Κάθε πίνακας legs μπορεί να έχει τα εξής στοιχεία:

- steps[] — περιέχει έναν πίνακα από steps που δίνουν πληροφορίες σχετικά για κάθε ξεχωριστό steps του κάθε leg της διαδρομής.
- distance — υποδεικνύει την συνολική απόσταση που καλύπτεται μέσα σε κάθε leg και σαν πεδίο παίρνει τα εξής στοιχεία:
    - value — υποδεικνύει την απόσταση σε μέτρα.
    - Text — περιέχει μία αναπαράσταση της απόστασης εκφρασμένο σε μονάδα μέτρησης. Το πεδίο αυτό μπορεί να μην υφίσταται εάν δεν υπαρχει η αποσταση αυτή.
- duration — υποδεικνύει το συνολικό χρόνο για κάθε leg και σαν πεδίο παίρνει τα εξής στοιχεία:
    - value — υποδεικνύει το χρόνο σε sec.
    - text — περιέχει μία αναπαράσταση της διάρκειας. Το πεδίο αυτό μπορεί να μην υφίσταται εάν δεν υπαρχει η αποσταση αυτή.

- duration_in_traffic— υποδεικνύει την συνολική διάρκεια του εκάστοτε leg λαμβάνοντας υπόψην τις συνθήκες κυκλοφορίας. Η παράμετρος duration_in_traffic θα επιστέψει τιμή αν και μόνο αν είναι αληθείς τα εξης:

    1. Το directions αίτημα συμπεριλαμβάνει departure_time παράμετρο.
    2. Το αίτημα συμπεριλαμβάνει Maps για Business client και signature παραμέτρους.
    3. Η κατάσταση της κυκολφορίας συνπεριλαμβάνεται στο επιλεγμένο δρομολόγιο.
    4. Η παράμετρος duration_in_traffic συμπεριλαμβάνει τα εξής πεδία:
        - value — υποδεικνύει το χρόνο απόστασης σε sec.
        - Text — περιέχει σε human-readable αναπαράσταση της διάρκειας.

- arrival_time_— υποδεικνύει το χρόνο άφιξης για το leg για transit directions. Το αποτέλεσμα επιστρέφει σαν Time object με τις εξής ιδιότητες:
    - value — ο χρονος ορίζεται σαν JavaScript Date object.
    - text — ο χρονος ορίζεται ως string.
    - time_zone — περιέχει το time zone του κάθε σταθμού.
- departure time — περιλαμβάνει το χρόνο αναχώρησης για το leg που ορίζεται ως Time object και είναι μόνο για transit directions.
- start_location — περιέχει τις γεωγραφικού πλάτους/μήκους συντεταγμένες του σημείου εκκίνησης του leg.
- end_location — περιέχει τις γεωγραφικού πλάτους/μήκους συντεταγμένες του σημείου άφιξης του leg.
- start_adress — περιέχει human-readable διεύθυνση (συνήθως διεύθυνση οδού) αντανακλώντας το start_location.
- end_adress — περιέχει human-readable διεύθυνση (συνήθως διεύθυνση οδού) αντανακλώντας το end_location.

## Steps

Κάθε στοιχείο που εσωκλείεται στον πίνακα steps προσδιορίζει ένα μοναδικό step της πλοήγησης και αποτελεί την μικρότερη μονάδα του direction's route. Το step περιγράφει μία συγκεκριμένη, απλή εντολή όπως π.χ ‘‘Turn left at W. 4th St’’. Το step δεν περιγράφει μόνο την εντολή, αλλά περιέχει και πληροφορίες σχετικά με την διάρκεια της απόστασης συσχετίζοντας το ένα step με το ακόλουθο. Π.χ ένα step υποδηλώνεται ως ‘‘Merge onto I-80 West’’ και μπορεί να περιέχει μία διάρκεια ‘‘37 miles’’ and ‘‘40 minutes’’, υποδεικνύοντας ότι το επόμενο step απέχει 37 μίλια και 40 λεπτά από το επόμενο step.

Κάθε step που εσωκλείεται μέσα σε ένα steps πεδίο-α μπορεί να περιέχει τα εξής στοιχεία:

- html_instructions — περιέχει οδηγίες format για αυτό το step, σε μορφή HTML text string.
- distance — περιέχει την απόσταση από ένα step στο επόμενο. Το πεδίο αυτό μπορεί να μην είναι αναγνωρίσημο αν η διαδρομή δεν υπάρχει.
- duration — υποδεικνύει τον κατάλληλο χρόνο που χρειάζεται το step να εκτελεστεί μέχρι το επόμενο step. Το πεδίο αυτό μπορεί να μην είναι αναγνωρίσημο αν η διαδρομή δεν υπάρχει.
- start_location — υποδεικνύει το σημείο εκκίνησης του step σε μορφή συντεταγμένων latitude και longitude.
- end_location — υποδεικνύει το σημείο άφιξης του step σε μορφή συντεταγμένων latitude και longitude
- Polyline — έναν πίνακα από κωδικοποιημένα σημεία που περιγράφουν αναλυτικά τα ενδιάμεσα σημεία για κάθε step

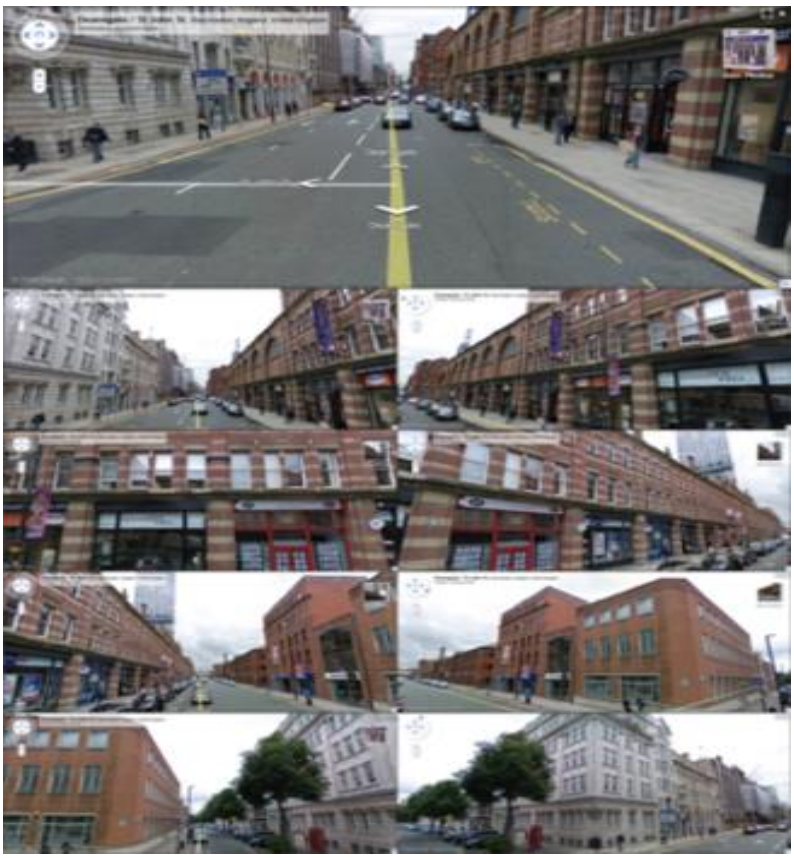
# Κεφάλαιο 4

## Street View Services

## 4.1 Εισαγωγή

### Google Street View Service

Το Google Street View είναι ένα service που χρησιμοποιείται σε συνδιασμό με το Google Maps και Google Earth και παρέχει panoramic view 360$^{o}$ μοιρών από καθορισμένους δρόμους καλύπτοντας παράλληλα όλες τις γύρω περιοχές τους. Η Street View κάλυψη είναι η ίδια με εκείνη για την εφαρμογή Google Maps application (http://maps.google.com/). Ένα δείγμα Street View δίνεται στο παρακάτω σχήμα:



Σχήμα 4.1: Διασταύρωση στο Manchester της Αγγλίας
δείχνωντας από 9 διαφωρετικές γωνίες

Το Google Maps API Javascript παρέχει μια υπηρεσία του Street View για την απόκτηση και το χειρισμό των εικόνων που χρησιμοποιούνται στο Google Maps Street View. Σε αντίθεση με το API V2, η υπηρεσία του Street View στο Maps API

Javascript V3 υποστηρίζεται εγγενώς στο πρόγραμμα περιήγησης. Στην ενότητα αυτή θα αναλυθεί το το Google Street View Services της Google στο οποίο βασίστηκε η ανάπτυξη και η υλοποίηση της εφαρμογής.

## 4.2 Street View Panoramas

Οι εικόνες του Street View υποστηρίζονται μέσω της χρήσης StreetViewPanorama object, το οποίο παρέχει ένα interface στον χρήστη. Κάθε χάρτης περιέχει μία προεπιλεγμένη Street View panorama, η οποία μπορεί να ανακτηθεί με την κλήση getStreetView() map's method. Όταν προσθέτονται τα Street View Control στο χάρτη θέτοντας την επιλογή streetViewControl σε true, αυτόματα συνδέουμε το control στην αυτή την προεπιλεγμένη Street View panorama. Μπορούμε επίσης να δημιουργήσουμε το δικό μας StreetViewPanorama object χρησιμοποιώντας το object αυτό στο χάρτη μας αντί του προεπιλεγμένου, θέτοντας τη streetView map's παράμετο κατάλληλα στο object κονστράκτορα.

## 4.3 Street View Τοποθεσία και Προσανατολισμός(POV)

Ο StreetViewPanorama κονστράκτορας, μας επιτρέπει να θέσουμε το Street View location και το Point of View(POV) χρησιμοποιώντας StreetViewOptions παράμετρους. Μπορούμε να καλέσουμε τις μεθόδους setPosition() και setPov() πάνω στο object μετά το construction για να αλλάξουμε το location και το POV. Το Street View location καθορίζει την τοποθεσία της κάμερας για την εικόνα αλλά όχι και τον προσανατολισμό της. Για αυτόν το σκοπό το StreetViewPov object καθορίζει τρεις παραμέτρους:

- heading (default 0) καθορίζει την γωνία περιστροφής γύρω από το σημείο τοποθέτησης της κάμερας σε μοίρες σε σχέση με την Βορά. Headings μετρώνται δεξιόστροφα, με την φορά του ρολογιού.
- pitch (default 0) καθορίζει την γωνία απόκλησης, από την αρχική της προεπιλεγμένη θέση που είναι συνήθως flat horizontal. Pitch γωνίες μετρώνται σε θετικές τιμές προς τα πάνω ($+90^o$ μοίρες πάνω και ορθογώνια σε σχέση με το default pitch ) και σε σε αρνητικές τιμές προς τα κάτω ($-90^o$ μοίρες κάτω και ορθογώνια σε σχέση με το default pitch ).
- zoom (default 1) καθορίζει το zoom level. Περισσότερα Street View locations υποστηρίζουν zoom level από 0 ως 3 επίπεδο zoom.

## 4.4 Street View Controls

Όταν εμφανίζεται το StreetViewPanorama, εμφανίζονται παράλληλα μία σειρά από controls τα οποία είναι προεπιλεγμένα στο panorama. Τα controls αυτά μπορούν να ενεργοποιηθούν ή και να απενεργοποιηθούν θέτοντας τα κατάλληλα πεδία τους μέσα στο Street View's StreetViewPanoramaOptions true ή false.

- panControl - παρέχει τον τρόπο περιστροφής για το panorama. Αυτό το control εμφανίζεται ως προεπιλογή πυξίδας pancontrol και μπορεί να αλλαχθεί η θέση του παρέχοντας τα κατάλληλα PanControlOptions μέσα στα panControlOptions πεδία.
- zoomControl - παρέχει τρόπο να κάνει zoom στην εικόνα. Αυτό το control εμφανίζεται ως προεπιλογή κάτω από το pancontrol και μπορεί να αλλαχθεί η εμφάνισή του παρέχοντας τα κατάλληλα ZoomControlOptions μέσα στα zoomControlOptions πεδία.
- adressControl - παρέχει ένα textual overlay που υποδεικνύει την διεύθυνση της αντίστοιχης περιοχής και μπορεί να αλλαχθεί η εμφάνισή του παρέχοντας τα κατάλληλα StreetViewAddressControlOptions μέσα στα addressControlOptions πεδία.
- linksControl – παρέχει βέλη πορείας στην εικόνα για πλοήγηση στις panorama εικόνες.

## 4.5 Άμεση Πρόσβαση σε Street View Δεδομένα

Το Street View Service παρέχει την δυνατότητα να καθορίζουμε την διαθεσιμότητα της Street View data ή να μας επιστρέφει πληροφορίες πάνω σε συγκεκριμένα panoramas, χωρίς την απαιτείται άμεσος χειρισμός ενώς χάρτη. Αυτό μπορεί να υλοποιηθεί με την χρήση StreetViewService object, που παρέχει ένα interface με τα δεδεμένα που είναι αποθυκευμένα στην υπηρεσία StreetView της Google.

### 4.5.1 Street View Service Αιτήματα

Η πρόσβαση στο Street View service είναι ασύχρονη, δεδομένου ότι το Google Maps API χρειάζεται να καλέσει έναν εξωτερικό server. Γι αυτό τον λόγο θα πρέπει να χρησιμοποιήσουμε μία callback συνάρτηση για να εκτελέση του αιτήματος. Η callback συνάρτηση αυτή επεξεργάζεται το αποτέλεσμα
Πρέπει να αρχικοποιηθούν δύο τύποι αιτημάτων στο StreetViewService:

- getPanoramaById() - επιστρέφει panorama data κάνοντας reference ID που προσδιορίζει μοναδικά το panorama.
- gePanoramaByLocation() – αναζήτηση για panorama data για μία συγκεκριμένη περιοχή δίνοντας LatLng και ακτίνα σε μέτρα. Εάν η ακτίνα είναι 50 μέτρα ή λιγότερο, η επιστροφή panorama θα είναι το πλησιέστερο panorama με την δεδομένη θέση.

### 4.5.2 Street View Service Απαντήσεις

Τόσο η getPanoramaById() και η getPanoramaByLocation() προσδιορίζουν μία callback συνάρτηση, η οποία θα εκτελεστεί κατά την ανάκτηση του αποτελέσματος από το Street View Service. Αυτή η callback συνάρτηση επιστρέφει ένα σύνολο από panorama data σε ένα StreetViewPanormaData object και ένα StreetViewStatus code δηλώνοντας το status του αιτήματος με αυτή την σειρά
Ένα StreetViewPanormaData object περιέχει meta-data πληροφορίες για το Street View panorama ακολουθώντας την εξής φόρμα:

```
{
 "location": {
    "latLng": LatLng,
    "description": string,
    "pano": string
  },
  "copyright": string,
  "links": [{
      "heading": number,
      "description": string,
      "pano": string,
      "roadColor": string,
      "roadOpacity": number
    }],
  "tiles": {
    "worldSize": Size,
    "tileSize": Size,
    "centerHeading": number
  }
}
```

Αυτά τα δεδομένα δεν είναι από μόνα τους ένα StreetViewPanorama object. Για την δημιουργία ενός Street View object για αυτά τα δεδομένα, πρέπει να δημιουργηθεί ένα StreetViewPanorama και να καλέσουμε setPano() κάνωντας passing το ID στο location.pano πεδίο.
Το status code μπορεί να επιστέψει τις ακόλοθες τιμές:

- OK – αναδυκνύει ότι το service ένα κατάλληλο panorama.
- ZERO_RESULTS – αναδυκνύει ότι το service δεν μπορεί να βρει ένα κατάλληλο panorama που να συμφωνεί με τα κριτήρια.
- UNKNOWN_ERROR – αναδυκνύει ότι το Street View αίτημα δε μπορεί να συνεχίσει την διαδικασία για αγνωστους λόγους

# Κεφάλαιο 5

# Google Earth API

## 5.1Εισαγωγή

### Google Earth

Το Google Earth είναι μια εικονική υδρόγειος, η οποία παλαιότερα ήταν γνωστή ως Keyhole. Αναπτύχτηκε αρχικά από την Keyhole, Inc., η οποία στη συνέχεια αγοράστηκε από την Google το 2004. Το Keyhole μετονομάστηκε σε Google Earth το 2005 και είναι πλέον διαθέσιμο για χρήση σε προσωπικούς υπολογιστές. Το Google Earth τοποθετεί δορυφορικές εικόνες, αεροφωτογραφίες και πληροφορίες GIS σε ένα μοντέλο 3D της Γης. Το Google Earth είναι μια εφαρμογή τρισδιάστατων γραφικών η οποία επιτρέπει την προβολή αεροφωτογραφιών και δορυφορικών εικόνων της Γης από ψηλά με μεγάλη λεπτομέρεια. Η Google πήρε τα χαρακτηριστικά του Google Maps και τα ένωσε με τις δυνατότητες του Keyhole. Έτσι, το Google Earth επιτρέπει όχι μόνο την προβολή των εικόνων αλλά και πολλών στρωμάτων δεδομένων που παρέχονται από την Google και την διαδικτυακή κοινότητα. Τα δεδομένα αυτά περιλαμβάνουν αντικείμενα όπως πάρκα, ποταμούς, σύνορα κρατών, τοποθεσίες εθνικών μνημείων και πολλές χιλιάδες άλλα σημεία. Το Google Earth επιτρέπει στους χρήστες να ψάξουν για διευθύνσεις, να εισάγουν συντεταγμένες, ή απλά να χρησιμοποιήσουν το ποντίκι για να μεταβούν σε μια τοποθεσία. Πολλές μεγάλες πόλεις παρέχονται με ιδιαίτερα υψηλή ανάλυση ώστε να μπορεί κανείς να διακρίνει κτήρια, σπίτια ή και αυτοκίνητα.
Στην ενότητα αυτή θα αναλυθεί το το Google Earth API στο οποίο βασίστηκε η ανάπτυξη και η υλοποίηση της εφαρμογής:

## 5.2 Google Earth Plugin

Το Google Earth Plugin υποστηρίζεται από τις εξής πλατφόρμες:

- **Microsoft Windows (XP, and Vista)**
  - Google Chrome 5.0+
  - Internet Explorer 7.0+
  - Firefox 3.0+
  - Flock 1.0+
- **Apple Mac OS X 10.5 and higher (Intel)**
  - Google Chrome 5.0+
  - Safari 3.1+
  - Firefox 3.0+

### 5.2.1 Χρήση Google Earth API

Για να κάνουμε load το Google Earth Plugin στην web σελίδα πρέπει να κάνουνε τα εξής βήματα:

1. Φόρτωση Google Earth API.
2. Δημιουργία DIV element για το plugin.
3. Υλοποίηση συναρτήσεων για αρχικοποιήση του plugin.
4. Κλήση της συνάρτησης όταν η σελίδα έχει φορτωθεί.

Τα παραπάνω βήματα θα αναλυθούν παρακάτω:

#### 1. Φόρτωση Google Earth API.

Βάζουμε το ακόλουθο tag στο <head> της HTML σελίδας :

```
<script type="text/javascript"src="https://www.google.com/jsapi"></script>
```

Το tag script δείχνει σε ένα JavaScript αρχείο με μία μέθοδο την google.load που χρησιμοποιείται για να φορτώσει επιμέρους Google APIs.

Μέσα σε ένα νέο <script> tag, καλούμε:

```
google.load("earth","1");
```

Με αυτό τον τρόπο κάνουμε load το earth module στο google.earth namespace, υποδευκνύοντας την έκδοση 1. Για να καθορίσουμε την τελευταίο version του API γράφουμε "1.x".

#### 2. Δημιουργία DIV element για το plugin

Το Google Earth Plugin γίνεται load μέσα ένα DIV element με μοναδικό id. Με αυτό τον τρόπο τοποθετούμε το DIV στο <body> της σελίδας:

```
<div id="map3d"style="height:400px;width:600px;"></div>
```

Στο παραπάνω παράδειγμα το DIV με id= map3d είναι το target element για το plugin.

## 3. Υλοποίηση συναρτήσεων για αρχικοποιήση του plugin

Σε αυτό το στάδιο θα πρέπει να υλοποιηθούν συναρτήσεις με την εξής σειρά:

- Προσπάθεια για δημιουργία ενός instance από το plugin
- Μία callback συνάρτηση η οποία θα καλείται όταν το plugin instance έχει δημιουργηθει με επιτυχία
- Μία callback συνάρτηση η οποία θα καλείται όταν το plugin instance δε μπορει να δημιουργηθεί.

Για το 1$^{ο}$ μπούλετ η συνάρτηση υλοποιείται ως εξής:

```
function init() {
  google.earth.createInstance('map3d', initCB,
failureCB);
}
```

Η google.earth.createInstance περιέχει τρεις τιμές:

1. Το DIV element.
2. Την callback συνάρτηση initCB η οποία επιστρέφει το success.
3. Την callback συνάρτηση failureCB η οποία επιστρέφει το failure.

Οι callback συνάρτηση η οποία εκτελεί το success περιέχει τον κώδικα που απαιτείται για το set up, τα objects και views που θα εμφανιστούν όταν το plugin instance θα φορτωθεί στον browser. Η function αυτή θα πρέπει να περιέχει και την GEWindow.setVisibility συνάρτηση, θέτοντας το window visibility με true έτσι ώστε το plugin ναι είναι ορατό μέσα στο DIV του.

Για το 2$^{ο}$ μπούλετ η συνάρτηση υλοποιείται ως εξής:

```
function initCB(instance) {
   ge = instance;
   e.getWindow().setVisibility(true);
}
```

Οι callback συνάρτηση η οποία εκτελεί το failure περιέχει κώδικα ο οποίος έχει να κάνει με την μη ικανότητα δημιουργίας instance του plugin. Ο error κώδικας διέρχεται στην callback function και μπορεί να επαναλυφθεί στην σελίδα ή να εμφανιστεί σαν alert.

Για το 3$^{ο}$ μπούλετ η συνάρτηση υλοποιείται ως εξής:

```
function failureCB(errorCode) {
}
```

## 4. Κλήση της συνάρτησης όταν η σελίδα έχει φορτωθεί

Το google namespace περιέχει την setOnLoadCallback() function η οποία καλεί την init() συνάρτηση όταν η HTML σελίδα και τα απαιτούμενα APIs έχουν γίνει load. Χρησιμοποιώντας αυτήν την function διασφαλίζεται ότι το plugin δεν θα γίνει load εφόσον οι DOM σελίδες δεν ολοκληρωθούν πρώτα. Η υλοποίησή της γίνεται ως εξής google.setOnLoadCallback(init);

### Παράδειγμα κώδικα

Το ολοκληρωμένο παράδειγμα για loading το Google Earth Plugin στην web σελίδα είναι το εξής:

```
<html>

      <head>
        <title>Sample</title>
        <script type="text/javascript" src="https://www.google.com/jsapi">
      </script>
        <script type="text/javascript">
          var ge;
          google.load("earth", "1");

          function init() {
            google.earth.createInstance('map3d', initCB , failureCB);
          }

          function initCB(instance) {
            ge = instance;
            ge.getWindow().setVisibility(true);
          }

          function failureCB(errorCode) {
          }

          google.setOnLoadCallback(init);
        </script>

      </head>
      <body>
        <div id="map3d" style="height: 400px; width: 600px;"></div>
      </body>
</html>
```

## 5.3 Placemarks(Σημεία)

Ένα placemark δείχνει ένα σημείο στην επιφάνεια της σφαίρας. Ένα basic;ο placemark συμπεριλαμβάνει το στάνταρ placemark εικονίδιο και μία γεωγραφική τοποθεσία. Ένα placemark μπορεί να συμπεριλαμβάνει:

- Μία περιγραφή
- Ένα ειδικό εικονίδιο

### 5.3.1 Βασικό Placemark

Το βασικό placemark συμπαιριλαμβάνει μόνο πληροφορία για μία γεωγραφική περιγραφή δε χρειάζεται όνομα ούτε περιγραφή. Ο παρακάτω κώδικας τοποθετεί το βασικό placemark εικονίδιο(πινέζα) στην επιφάνεια της Γης στις καθορισμένες συντεταγμένες

```
// Create the placemark.
var placemark = ge.createPlacemark('');

// Set the placemark's location.
var point = ge.createPoint('');
point.setLatitude(12.345);
point.setLongitude(54.321);
placemark.setGeometry(point);

// Add the placemark to Earth.
ge.getFeatures().appendChild(placemark);
```

## 5.4 Κάμερα Controls

### 5.4.1 Εισαγωγή

Το 'view' στο Google Earth είναι η εικόνα που βλέπουμε στο plugin παράθυρο. Το 'camera' control που παρέχει το plugin είναι η θέση του θεατή στο χώρο. Υπάρχουν δύο τρόποι για να ορίσουμε το view:

- **Camera:** Με τη χρήση του camera view, το σημείο που έχει οριστεί καθορίζει τη θέση του viewer στο χώρο. Σετάροντας το γεωγραφικό πλάτος και το μήκος μετακινούμε τη κάμερα στο συγκεκριμένο σημείο. Το viewer κινείται σε x,y,z άξονες μέχρι να ανακτηθεί το επιθυμητό view.
- **LookAt:** Με τη χρήση του LookAt, το σημείο που έχει οριστεί είναι το σημείο που παρατηρείται.

### 5.4.2 Τρέχον View

Μπορεί να γίνει η χρήση της copyAslookAt() function που επιστρέφει το γεωγραφικό πλάτος και μήκος του σημείου που κοιτάει η κάμερα, το υψόμετρο στο οποίο η κάμερα είναι, την κλίση της, και το προσανατολισμό της πυξίδας σε μοίρες. Η τιμή που επιστέφει το lookAt:

```
var lookAt = ge.getView().copyAsLookAt(ge.ALTITUDE_RELATIVE_TO_GROUND)
```

### 5.4.3 Οριζόντια Κίνηση Κάμερας(Panning)

Μπορούμε να μεταφερθούμε από το τρέχων σημείο σε ένα άλλο σημείο διατηρώντας την κλίση, το εύρος, τον προσανατολισμό και την υψομετρική τιμή

```
// Get the current view.
var lookAt = ge.getView().copyAsLookAt(ge.ALTITUDE_RELATIVE_TO_GROUND);

// Set new latitude and longitude values.
lookAt.setLatitude(36.584207);
lookAt.setLongitude(-121.754322);

// Update the view in Google Earth.
ge.getView().setAbstractView(lookAt);
```

Εναλλακτικά μπορούμε να ορίσουμε ένα lookAt ή camera object και να το σετάρουμε

```
// Create a new LookAt.
var lookAt = ge.createLookAt('');

// Set the position values.
lookAt.setLatitude(36.584207);
lookAt.setLongitude(-121.754322);
lookAt.setRange(5000.0); //default is 0.0

// Update the view in Google Earth.
ge.getView().setAbstractView(lookAt)
```

### 5.4.4 Οριζόντια Μετατώπιση Δίπλα στο Τρέχον Σημείο

Το view μπορεί να κινηθεί σε σημείο δίπλα στο τρέχον.Το παρακάτω παράδειγμα μετακινεί τη κάμερα κατά $25^{o}$

```
// Get the current view.
var lookAt = ge.getView().copyAsLookAt(ge.ALTITUDE_RELATIVE_TO_GROUND);

// Add 25 degrees to the current latitude and longitude values.
lookAt.setLatitude(lookAt.getLatitude() + 25.0);
lookAt.setLongitude(lookAt.getLongitude() + 25.0);

// Update the view in Google Earth.
ge.getView().setAbstractView(lookAt);
```

### 5.5.5 Κλίση Κάμερας

Το lookAt παίρνει τιμή μεταξύ $0^{o}$ κ' $90^{o}$ με τις $0^{o}$ να ορίζουν την κάθετη όψη και τις $90^{o}$ την οριζόντια.

```
// Get the current view.
var lookAt = ge.getView().copyAsLookAt(ge.ALTITUDE_RELATIVE_TO_GROUND);

// Add 15 degrees to the current tilt.
lookAt.setTilt(lookAt.getTilt() + 15.0);

// Update the view in Google Earth.
ge.getView().setAbstractView(lookAt);
```

### 5.5.6 Zoom Κάμερας

Το zoom in-out ρυθμίζεται από το εύρος(range) attribute για το lookAt

```
// Get the current view.
var lookAt = ge.getView().copyAsLookAt(ge.ALTITUDE_RELATIVE_TO_GROUND);

// Zoom out to twice the current range.
lookAt.setRange(lookAt.getRange() * 2.0);

// Update the view in Google Earth.
ge.getView().setAbstractView(lookAt);
```

# Κεφάλαιο 6

## 3D GoogleEarth Σύστημα Πλοήγησης

## 6.1 Εισαγωγή

Στο προηγούμενο κεφάλαιο περιγράφτηκαν τα API's και τα Services της Google πανω στα οποία βασίστηκε η ανάπτυξη της εφαρμογής. Στο κεφάλαιο αυτό θα περιγραφεί αναλυτικά η υλοποίηση του navigation system. Πιο συγκεκριμένα θα αναλυθούν οι αλγόριθμοι πάνω στους οποίους βασίστηκε η υλοποίηση του application και θα φτιαχτεί ένα εγχειρίδιο πλοήγησης το οποίο θα περιέχει πληροφορίες και συνοπτικές οδηγίες στον χρήστη για την εφαρμογή αυτή.

## 6.2 3D GoogleEarth Σύστημα Πλοήγησης Manual

Το 3D GoolgeEarth navigation system είναι μία turn-by-turn εφαρμογή για χρήση Υ/Η μέσω web. Μέσω του συστήματος πλοήγησης, ο χρήστης έχει την δυνατότητα να υποβάλει τα επιθυμητά αιτήματα αναχώρησης και προορισμού, και να παίρνει τις ανάλογες επιθυμητές οδηγίες της επιλεγμένης διαδρομής. Επίσης μέσω της χρήσης κάποιων συγκεκριμένων control buttons, η εφαρμογή παρέχει στον χρήστη δυνατότητες όπως τον υπολογισμό της ταχύτητας της πλοήγησης, το σταμάτημα ή και την επανάληψη της διαδρομής, καθώς και την περιστροφή της κάμερας και την αυξομείωση της υψομετρικής διαφοράς κατά την διάρκεια του επιλεγμένου ταξιδιού. Επίσης το application παρέχει την υπηρεσία StreetView σε τρεις διαφορετικές επιλογές. Η εφαρμογή αναπτύχθηκε με σκοπό την εύκολη και άμεση αναζήτηση και είναι εύχρηστη και κατανοητή στον χρήστη.

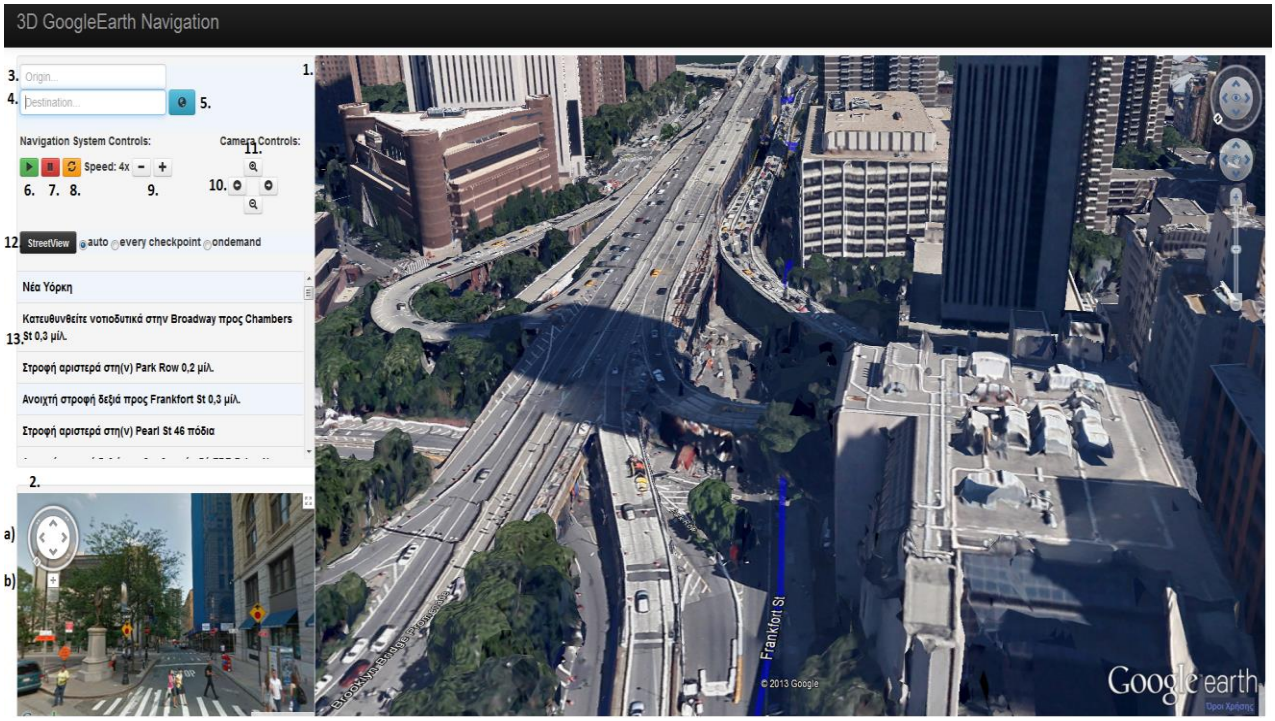
### Χαρακτηριστικά πλοήγησης

Το 3D navigation system συγκεντρώνει τα παρακάτω χαρακτηριστικά:

- Εισαγωγή δεδομένα αναχώρησης και προορισμού.
- Εμφάνιση της συνολικής διαδρομής.
- Το όνομα των οδών και των δρόμων.
- Τα σημεία από τα οποία περνάει(checkpoints)
- Στοιχεία του δρόμου(μονής –διπλής κατεύθυνσης, αυτοκινητόδρομος κ.τ.λ).
- Την μερική χρονική διάρκεια από το ένα checkpoint στο άλλο, αλλά και το συνολικό χρόνο που χρειάστηκε για να φτάσει στον προορισμό του.
- Χωρικά δεδομένα(πόλεις, κτήρια, πάρκα, βουνά, θάλασσες κ.τ.λ).

## 6.2.1 Σχεδιαστική Διεπαφή

Η Σχεδιαστική διεπαφή που υλοποιήθηκε για την εφαρμογή είναι το παρακάτω:




## 6.2.2 Υπηρεσίες Web

Παρακάτω παρατείθονται οι Google υπηρεσίες

1. **Υπηρεσία Google Earth**:

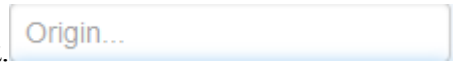
2. **Υπηρεσία StreetView**:
   Δυνατότητα πλοήγησης μέσω φωτογραφικών εικόνων.

   a) **StreetView compass** – Πυξίδα περιστροφής της κάμερας κατά $360^{o}$ .

   b) **StreetView zoom controls –** Δυνατότητα zoom in/out της φωτογραφίας.

### 6.2.3 Controls

3. **Origin** text:

   Εισάγωγη ως δεδομένα ως αφετηρία. 


4. **Destination** text:

   Εισάγωγη ως δεδομένα ως άφιξη. 


5. **Find route** button:

   Αποστολή αιτήματος στις Google υπηρεσίες. 

6. **Start** button:

   Εκίνηση της πλοήγησης. 

7. **Pause** button:

   Παύση της πλοήγησης. 

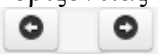
8. **Reset** button:

   Επανεκίνηση της διαδρομής. 

9. **Speed+-** button:

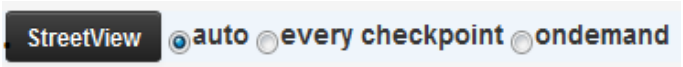
   Αυξομείωση της ταχύτητας της πλοήγησης. 

10. **Look left-right** button:

    Δυνατότητα οριζόντιας περιστροφής 360$^{o}$ της κάμερας κατά το μήκος της διαδρομής. 

11. **Zoom in-out** buttons: Αυξομείωση της υψομετρικής διαφοράς. 

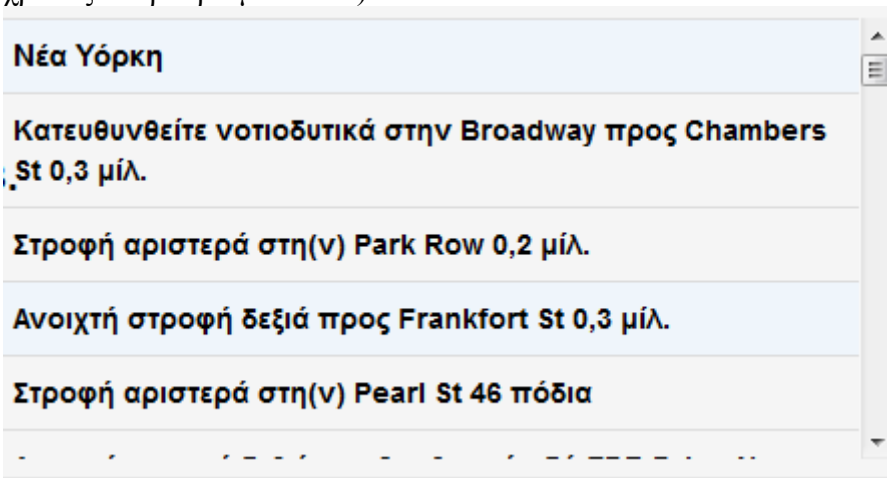
12. **StreetView** button:




- **auto** radio button – αυτόματη επιλογή εμφάνισης και αλλαγής της φωτοφραφίας StreetView υπηρεσίας.
- **every checkpoint** radio button – επιλογή εμφάνισης και αυτόματης αλλαγής σε κάθε checkpoint της φωτογραφίας της StreetView υπηρεσίας.
- **ondemand** radio button – επιλογή εμφάνισης StreetView υπηρεσίας και αλλαγής της φωτογραφίας κατ' επιλογή μέσω του StreetView button.

13. **Route details**:

Εμφάνιση χαρακτηριστικών της διαδρομής (διεύθυνση δρόμων, στοιχεία πορείας, χρόνος προορισμού κ.τ.λ).



## 6.3 Πλοήγηση

Ο χρήστης αρχικά εισάγει ως δεδομένα την πόλη που επιθυμεί να ξεκινήσει(και την πιθανή οδό) αλλά και το επιθυμητό σημείο προορισμού στα πεδία «origin» και «destination». Στην συνέχεια πατώντας το κουμπί «Find Route», εμφανίζεται η διαδρομή και τα checkpoints από τα οποία περνάει. Τα checkpoints είναι οι πινέζες (placemarks) πάνω στη 3D σφαίρα και περιέχουν στοιχεία της διαδρομής (διεύθυνση, στοιχεία πορείας, χρόνος προορισμού κ.τ.λ). Πατώντας το «Start» button ο χρήστης μπορεί να πλοηγηθεί τρισδιάστατα στη Google Γη ενώ χρησιμοποιώντας παράλληλα τα «zoum-in», «zoum-out» camera controls έχει την δυνατότητα να αλλάζει την υψομετρική διαφορά και να δει από ψηλά την πορεία του παρέχοντας μία δορυφορική, πιο συνολική εικόνα της διαδρομής. Αντίστοιχα με τα «look left», «look right» camera controls, έχει την δυνατότητα ο πλοηγός να κοιτάει εκατέρωθεν της διαδρομής και να λαμβάνει δεδομένα πέραν του δρόμου όπως κτήρια, πάρκα θάλασσες βουνά κ.τ.λ. Επίσης η εφαρμογή παρέχει στον χρήστη την ικανότητα αυξομείωσης της ταχύτητας με τη χρήση του button «Speed» επιταχύνοντας ή επιβραδύνοντας με αυτό τον τρόπο την διαδρομή. Μία άλλη δυνατότητα που παρέχει η εφαρμογή είναι η ικανότητα του χρήστης να μεταφέρεται από το ένα σημείο στο άλλο. Πατώντας απλά ένα οποιδήποτε σημείο «checkpoint», η εφαρμογή μετατοπίζει τη θέση από ένα σημείο στο επόμενο επιθυμητό κατευθείαν χωρίς να χρειαστεί να πλοηγηθεί από την αφετηρία και να φτάσει σε αυτό και στην συνέχεια πατώντας το «Start» να αρχίσει το ταξίδι του από το σημείο αυτό.
Με την ενεργοποιήση της υπηρεσίας Google Street View μέσω του «StreetView» button, ο χρήστης έχει τρεις επιλογές εμφάνισης της εικόνας και πλοήγησης ανάλογα με το πώς αυτός επιθυμεί. Μέσω των controls που παρέχει το Street View μπορεί να επιδράσει στην εικόνα είτε κάνοντας «zoum in-out» είτε κάνοντας περιήγηση στο χώρο με τη χρήση της πυξίδας. Επίσης μπορεί να πλοηγηθεί μέσω αυτής πατώντας απλά πάνω στο δρόμο. Σε κάθε περίπτωση όταν σταματάει το navigation σταματάει και το Street View στο σημείο εκείνο που έχουμε κάνει παύση μέσω του button «Pause» εάν και εφόσον υπάρχει φωτογραφική λήψη της περιοχής εκείνης. Αν ο χρήστης πλοηγηθεί μόνο μέσω του Street View ενώ το navigation system είναι σε παύση, όταν ενεργοποιηθεί ξανά το navigation, τότε το Street View επανέρχεται και πάλι στο σημείο εκείνο από το οποίο έχει ξεκινήσει και το σύστημα πλοήγησης. Με αυτό τον τρόπο ο χρήστης δε μπερδεύεται κατά την πλοήγηση και η εφαρμογή δουλεύει αξιόπιστα.

## 6.4 Λειτουργικότητα

### 6.4.1 Φόρτωση των Επιθυμητών Directions

Όπως αναφέρθηκε στην προηγούμενη ενότητα, ο χρήστης εισάγει τα δεδομένα προορισμού στα πεδία «origin» και «destination», στη συνέχεια πατάει το κουμπί «Find Route», και εμφανίζεται η επιθυμητή διαδρομή στο Google Earth μαζί με τα χαρακτηριστικά του ταξιδιού. Ουσιαστικά το κουμπί καλεί την συνάρτηση goDirections(), η οποία εκτελείται πίσω από το UI. Η συγκεκριμένη συνάρτηση καλεί

το Direction services της Google Maps και στέλνει ένα αίτημα(request) με τη μορφή που είδαμε στην ενότητα 3.3.1. Το Google Directions API services στέλνει πίσω απάντηση(response) με την μορφή json. Το αρχείο αυτό περιέχει τα δεδομένα τα οποία χρησιμοποιήθηκαν για την κατασκευή της εφαρμογής. Πιο συγκεκριμένα και όπως αναλύθηκε στην ενότητα 3.3.6 το json αρχείο επιστρέφει ένα πίνακα routes[] με ένα μόνο legs[] πίνακα που περιέχει τα δεδομένα σημεία αναχώρησης και προορισμού. Ο πίνακας legs[] με την σειρά του περιέχει πίνακες από step[] όπου κάθε step περιέχει συγκεκριμένα χαρακτηριστικά του ταξιδιού(τα οποία και εμφανίζουμε στην αριστερή στήλη της διαδρομής) καθώς και μια σειρά από κωδικοποιημένα σημεία που συνθέτουν το γεωγραφικό polyline της διαδρομής. Εφόσον έχει ληφθεί η απάντηση παίρνουμε τα steps και το polyline για επεξεργασία.

```
function goDirections(){
..
..
directions = new google.maps.Directions(ge, null);
..
// getting steps and polyline for process
directions.load('from:'+$('#from').val() +'to '+$('#to').val(),
   {getSteps: true, getPolyline: true});

}
```

Το επόμενο βήμα είναι να φτιαχτεί η διαδρομή και να σχεδιαστεί στη σφαίρα έτσι ώστε να είναι ορατή στο χρήστη. Για το λόγο αυτό χρησιμοποιήθηκαν τα events της Google Maps JavaScript API. Τα events της Google Maps δημιουργούν γεγονότα μόλις ενεργοποιηθούν και αυτό γίνεται μέσω event listeners. Οι συναρτήσεις αυτές παίρνουν σαν ορίσματα τρεις παραμέτρους. Το object το οποίο θα ακούσει, το event και μία callBack(cb) function η οποία εκτελείται μόλις το event ενεργοποιηθεί. Πιο συγκεκριμένα το object εδώ είναι το directions το event είναι το load και η cb function είναι η directionsLoaded η οποία και θα αναλυθεί παρακάτω.

```
google.maps.Event.addListener(directions, 'load', directionsLoaded);
```

### 6.4.2 Σχεδιασμός της Διαδρομής

Η directionsLoaded() είναι μία callBack συνάρτηση, οι οποία ζωγραφίζει την διαδρομή στο Google Earth και τοποθετεί τα σημεία (placemarks) στον πίνακα. Αρχικά πραγματοποιεί Geocoding μετατρέποντας τις φυσικές διευθύνσεις αφετηριάς και προορισμού σε συντεταγμένες. Κατόπιν με βάση αυτές τις συντεταγμένες φτιάχνει το αρχικό και το τελευταίο σημείο. Στη συνέχεια και αφού έχουμε ανακτήσει και τις συντεταγμένες του κάθε step[i] (θα αναλυθεί παρακάτω πως), δημιουργεί και τα ενδιάμεσα σημεία με τις αντίστοιχες περιγραφές (descriptions). Με βάση τις παραπάνω συντεταγμένες δημιουργεί το polyline "lineStringKml" το οποίο είναι η γραμμή πλοήγησης στο Google Earth. Αυτή έπειτα σχεδιάζεται στη σφαίρα (γη) με τα κατάλληλα χαρακτηριστικά χρώμα, πάχος κ.τ.λ. Ταυτόχρονα ο χρήστης πλοηγείται

στο αρχικό σημείο αναχώρησης. Η συνάρτηση που υλοποιεί την πλοήγηση είναι η flyToLatLng().

```
// Creating the lineString route on the ground
var lineStringKml = '<LineString><coordinates>\n';
for (var i = 0; i < path.length; i++)
    lineStringKml +=
        path[i].loc.lng().toString() + ',' +
        path[i].loc.lat().toString() +
        ',10\n';
lineStringKml += '</coordinates></LineString>';

//parseKml requires a KML string, and returns a KmlFeature object.
var routeLineString = ge.parseKml(lineStringKml);
var linePlacemark = gex.dom.addPlacemark({
      lineString: {
 // Draw me
        path: routeLineString,
        altitudeMode: ge.ALTITUDE_CLAMP_TO_GROUND,
        tessellate: true
      },
      style: {
        line: { color: '88ff0000', opacity: 0.5, width: 10 }
      }
    });
//fly to 1rst step
flyToLatLng(steps[0].loc);
```

Η συνάρτηση flyToLatLng() παίρνει το αρχικό step[i]. Κάνωντας χρήση των συναρτήσεων της Google Earth API για Camera Control και πιο συγκεκριμένα τη συνάρτηση ge.createLookAt() φτίαχνουμε το camera Object, το οποίο θα πάρει τις συνταγμένες του αρχικού step[i] και δίνοντας στη συνέχεια τα κατάλληλα χαρακτηριστικά κάμερας (ύψος, γωνία κ.τ.λ ) θα το προβάλει στο Google Earth.
Ακολουθεί παράδειγμα πως σχεδιάζουμε και ζωγραφίζουμε το lineString με βάση τις εκάστοτες συντεταγμένες.

**Draw a line <LineString>**

```
Color the line<Style>, <LineStyle>
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
<Document>
<Placemark>
 <LineString>
  <coordinates>
   135.2, 35.4, 0.
   135.4, 35.6, 0.
   135.2, 35.6, 0.
   …
   …
   …
  </coordinates>
 </LineString>
 <Style>
  <LineStyle>
   <color> #88ff0000 </color>
  </LineStyle>
 </Style>
</Placemark>
</Document>
</kml>
```

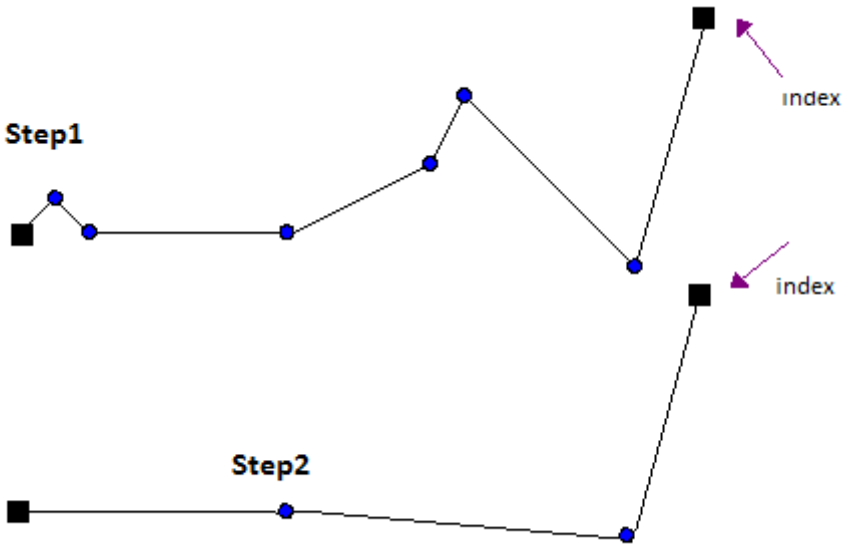

Σχήμα 6.1: Σχεδιασμός του kmlString και χρωματισμός

Όπως αναφέρθηκε στην προηγούμενη παράγραφο, προκειμένου να τοποθετηθούν τα placemark και να δημιουργηθεί το polyline “lineStringKml”, θα πρέπει να είναι γνωστές οι συντεταγμένες του κάθε step[i] αλλα και όλα τα ενδιάμεσα σημεία. Οι συναρτήση buildPathStepArrays() καλείται μέσα στην συνάρτηση της directionsLoaded() και υλοποιεί τον παραπάνω σκοπό.

```
function directionsLoaded() {
…
code
…
…
// build the path and steps from the google.maps.Directions route
buildPathStepArrays();
…
…
}
```

### 6.4.3 Ανάκτηση Διαδρομής

Για να ανακτήσουμε την επιθυμητή διαδροομή πρέπει να επεξεργαστούμε την απάντηση που μας επιστρέφει η Google Maps Directions API Servise σε μορφή json. Το json αυτό αρχείο, όπως έχει αναφερθεί και πιο πάνω, περιέχει πληροφορίες της επιλεγμένης διαδρομής όπως το σημείο εκκίνησης και το σημείο άφιξης με τα χαρακτηριστικά του κάθε σημείου, το κάθε step[i] με τα δικά τους στοιχεία δρόμου και το polyline που είναι κωδικοποιημένα σημεία τα οποία συνθέτουν την διαδρομή. Εφόσον δεν έχουμε επιλέξει ενδιάμεσα σημεία το αποτέλεσμα είναι να παίρνουμε έναν και μόνο πίνακα route[](βλ.Κεφ3.3.7). Η buildPathStepArrays που υλοποιήθηκε ανακτά όλα τα παραπάνω δεδομένα δημιουργώντας δύο πίνακες, τον πίνακα steps[] και τον πίνακα path[] και στην συνέχεια κάνει get το route, τις κορυφές του polyline και τον αριθμό των step[i]. Ύστερα περνάει το κάθε step[i] μέσα στο πίνακα steps[] αλλά και και ολόκληρο το path μέσα στον πίνακα path[]. Ο τρόπος αυτός πραγματοποιείται μέσω ενός δείκτη που εξετάζει αν η κορυφή του κάθε path[i] είναι και η τελευταία της διαδρομής ή όχι.




Σχήμα 6.2: Έλεγχος steps με δείκτη

Ο πίνακας steps[] περιέχει το κάθε step[i] μαζί με τα χαρακτηριστικά του, όπως το location που προσδιορίζει τις συντεταγμένες του, την περιγραφή του και την απόσταση.

```
//bulding steps[i]
steps.push({
      loc: step.getLatLng(),
      desc: step.getDescriptionHtml(),
      distanceHtml: step.getDistance().html,
      pathIndex: path.length
    });
```

Στην συνέχεια εκτελείται έλεγχος για όλα τα ενδιάμεσα σημεία του polyline του κάθε step[i] αντίστοιχα και υπολογίζει την απόσταση αλλά και τον χρόνο μεταξύ δύο διαδοχικών σημείων της διαδρομής και φτιάχνει τον αντίστοιχο πίνακα path:

```
// bulding entire path
path.push({
        loc: loc,
        step: i,
        distance: distance,
        duration: step.getDuration().seconds *distance/stepDistance
      });
```

Ο υπολογισμός της διάρκειας(duration) μεταξύ του path[i] και path[i+1] υπολογίζεται ως εξής. Θα πρέπει να πάρουμε τη συνολική διάρκεια του εκάστοτε step[i] και να την πολλαπλασιάσουμε με τον ρυθμό μετατόπισης από το ένα σημείο στο άλλο.

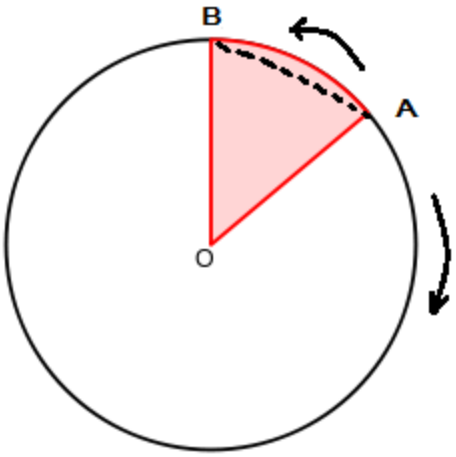
Ο παραπάνω τύπος προκύπτει από την απλή μέθοδο των τριών. Για Παράδειγμα αν για να πάμε 10m χρειαζόμαστε 60sec για κάθε 1m χρειαζόμαστε $x = 60\text{sec} \cdot 1m / 10m \Rightarrow x = 0{,}6\text{sec}$

## 6.4.4 Υπολογισμός Απόστασης Σημείων

```
//calculate distance between to points
hdistance(loc, polyline.getVertex(j + 1));
```

Η hdistance υπολογίζει την απόσταση μεταξύ δύο σημείων. Αρχικά παίρνει σαν παραμέτρους τις τιμές loc, polyline.getVertex(j + 1) , όπου είναι τα αντίστοιχα δύο σημεία του polyline, σε αυτό που βρισκόμαστε και σε αυτό που θέλουμε να πάμε. Το κάθε σημείο περιγράφεται από την γεωγραφική του θέση με βάση το γεωγραφικό μήκος (lon), το γεωγραφικό πλάτος του (lat) και το υψόμετρό του (alt).

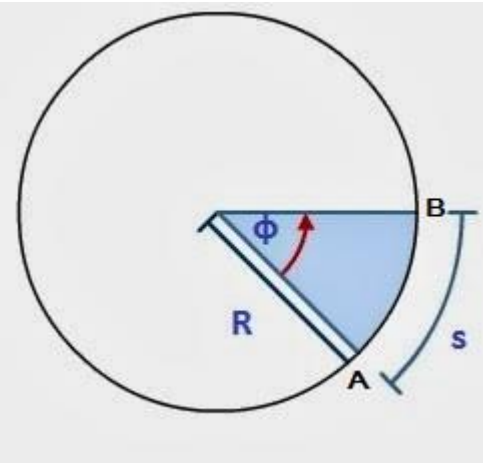
Για να πάμε από το ένα σημείο Α στο σημείο Β ο καλύτερος τρόπος είναι η ευθεία. Στη σφαίρα όμως δεν υπάρχουν ευθείες και επειδή δύο σημεία χωρίζουν την σφαίρα σε δύο τόξα, το τόξο με το μικρότερο μήκος αποτελεί και την βέλτιστη απόσταση.




Σχήμα 6.3: Βέλτιστη απόσταση μεταξύ δύο σημείων

Αν πάρουμε σα παράθεση ότι η Γη είναι ένας τέλειος κύκλος τότε με βάση τον ορισμό γωνίας ενός κύκλου, ορίζουμε γωνία φ το πηλίκο του μήκους S του τόξου AB που περιέχεται μεταξύ των ευθειών προς την ακτίνα R του κύκλου. (βλ.Σχήμα 6.4)

Δηλαδή $\phi = S / R$




Σχήμα 6.4: Επίκεντρη γωνία φ με ακτίνα R και τόξο S

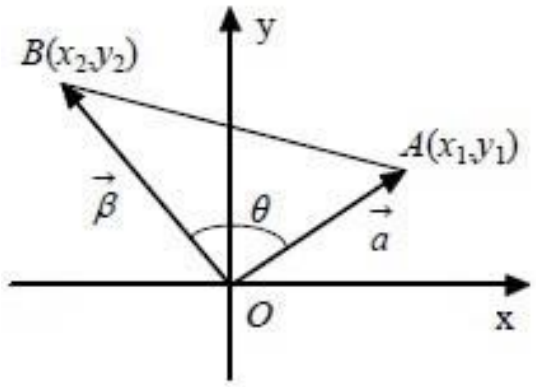
Άρα $S_{AB} = R \cdot \phi$ ή αλλιώς $d = r \cdot \Delta_\sigma$ όπου $\Delta_\sigma$ η επίκεντρη γωνία ενός κύκλου και d το μήκος του τόξου των δύο σημείων, δηλαδή η απόσταση από το σημείο path[i] στο path[i+1]. Η ακτίνα της Γης είναι γνωστή, το ζητούμενο του προβλήματος ήταν να υπολογιστεί η γωνία $\Delta_\sigma$ που σχηματίζουν τα δύο σημεία . Η 3D vector algebra formula είναι η εξίσωση εκείνη η οποία θα δώσει την τιμή της γωνίας και περιγράφεται από τον μαθηματικό τύπο

$$\Delta_\sigma = \arccos(n_1 \bullet n_2)$$

όπου $n_1 \bullet n_2$ δηλώνει το εσωτερικό γινόμενο δύο διανυσμάτων(dot), και arccos συνάρτηση του τόξου συνημιτόνου. Το εσωτερικό γινόμενο δύο διανυσμάτων n διαστάσεων **a** = [$a_1$, $a_2$, ..., $a_n$] and **b** = [$b_1$, $b_2$, ..., $b_n$] ορίζεται ως:

$$\mathrm{a} \cdot \mathrm{b} = \sum_{i=1}^{n} a_i \cdot b_i = a_1 b_1 + a_2 \cdot b_2 + \ldots + a_n \cdot b_n$$

Από τους παραπάνω τύπους προκύπτει ότι το εσωτερικό γινόμενων των διανυσμάτων είναι ίσο με το άθροισμα των γινομένων των ομώνυμων συντεταγμένων τους και το αποτέλεσμα είναι ένας πραγματικός αριθμός.




$$\vec{\alpha} \cdot \vec{\beta} = x_1 x_2 + y_1 y_2$$

Σχήμα 6.5:Εσωτερικό γινόμενο δύο διανυσμάτων σε x,y άξονα.

Για να μετατραπούν οι γεωγραφικές συντεταγμένες των δύο σημείων σε σημεία του χώρου έτσι ώστε να υπολογστεί το εσωτερικό γινόμενο αρχικά τις μετατρέπουμε σε σφαιρικές συντεταγμένες, κάνοντας την χρήση των παρακάτω τύπων.

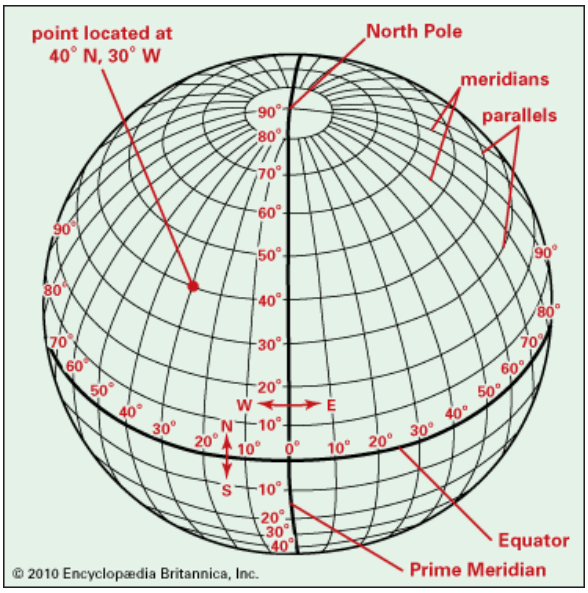
$$\theta^o = latitude \cdot \pi / 180$$

$$\phi^o = longitude \cdot \pi / 180$$

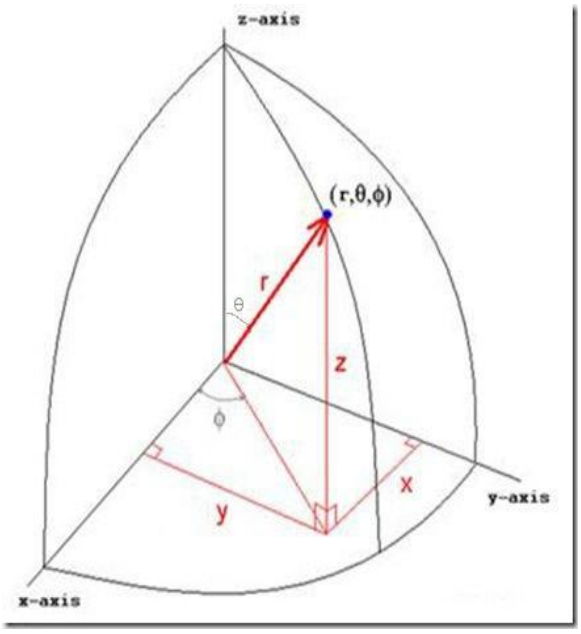
Με αυτή την μέθοδο έχουμε τα σημεία (r,θ,φ) σ'ένα σύστημα σφαιρικών συντεγμένων. Για να τα εκφράσουμε σε σημεία του χώρου δηλαδή της μορφής (x,y,z) κάνουμε χρήση των γεωμετρικών και τριγωνομετρικών τύπων όπως ο νόμος του ημιτόνου, συνημιτόνου και της εφαπτομένης όπως προκύπτουν από ένα ορθογώνιο τρίγωνο(βλ. Σχήμα 6.7).

$x = r\sin\theta\cos\varphi$
$y = r\sin\theta\sin\varphi$
$z = r\cos\theta$

Όπου γωνία φ είναι το longitude, το θ είναι το latitude και το r είναι η ακτίνα της γης r=6378100m.



Σχήμα 6.6: Σφαιρικό Σύστημα Συντεταγμένων



Σχήμα 6.7: Τριγωνομετρική μετατροπή σημείου στο χώρο

Με βάση λοιπόν τις (x,y,z) συντεταγμένες του path[i] και path[i+1], υπολογίσουμε το εσωτερικό γινόμενο δύο διανυσμάτων. Έφοσον γνωρίζουμε το εσωτερικό γινόμενο έπειτα υπολογίσουμε τη γωνία των διανυσμάτων και τέλος την απόσταση των δύο σημείων δηλαδή:

```
var angle = Math.acos(dot);
distance = Earth_radius * angle;
```

### 6.4.5 Εστίαση Κάμερας

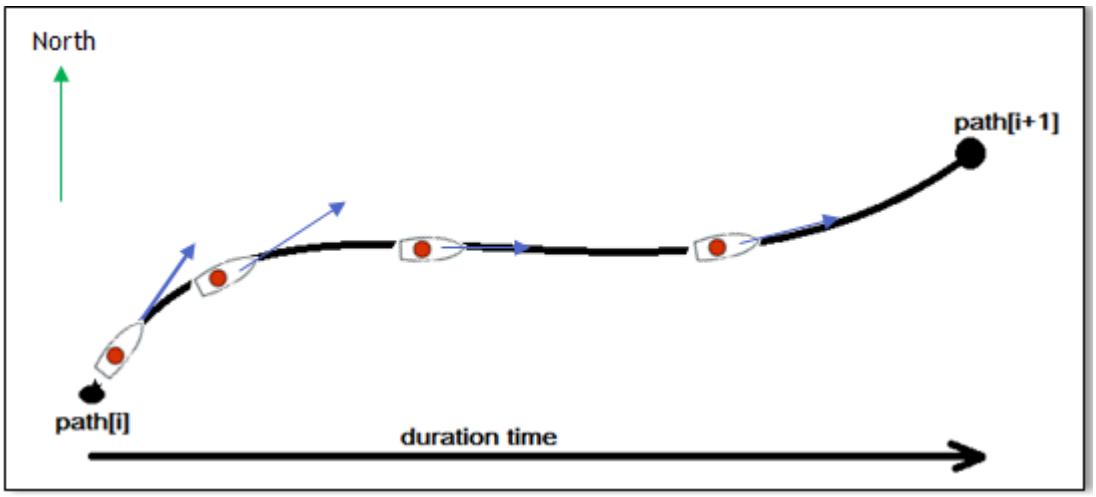
Ο χρήστης όπως έχει αναφερθεί έχει την δυνατότητα να πάει από ένα σημείο στο άλλο απλά κάνοντας κλικ το αντίστοιχο step στην αριστερή λίστα με τα χαρακτηριστικά της διαδρομής. Για να παρέχει η εφαρμογή την συγκεκριμένη δυνατότητα θα πρέπει να πάρουμε αρχικά το επιθυμητό step[i]. Η συνάρτηση flytoStep() παίρνει σαν παράμετρο το step[i] αυτό. Το επόμενο βήμα που πρέπει να υλοποιηθεί είναι να δώσουμε τις κατάλληλες τιμές στις παραμέτρους της κάμερας προκειμένου να δούμε το επιθυμητό step[i] στην οθόνη. Κάνοτας και δω χρήση των συναρτήσεων της Google Earh API για το control της κάμερας η συνάρτηση δημιουργεί το lookAt object και το σετάρει με γεωγραφικά χαρακτηριστικά του step[i], που είναι το γεωγραφικό πλάτος , μήκος και ύψος καθώς και τον τρόπο με τον οποίο η κάμερα θα κοιτάξει το σημείο αυτό. Εκτός από την γωνία που κοιτάμε το σημείο και το εύρος της κάμερας, πρέπει να καθοριστεί ο προσανατολισμός ή αλλιώς να υπολογίσουμε το heading.

Το πρόβλημα που επιλύθηκε εδώ ήταν η μεταβολή της εστίασης της κάμερας κατά την κίνησή της. Για παράδειγμα, έστω ότι θέλουμε να πάμε από το σημείο Α στο σημείο Β. Εφόσον η Γη είναι σφαιρική, ο πιο αποτελεσματικός τρόπος για να ταξιδέψουμε είναι υπό μορφή τόξου. Ως εκ τούτου, εάν καθορίσουμε μία τοξοειδής διαδρομή από το σημείο Α στο σημειό Β, το τρέχων heading θα ποικίλει καθώς διασχίζουμε το τόξο. Το τελικό heading θα διαφέρει από το αρχικό αλλάζοντας συνεχόμενα μοίρες ανάλογα την απόσταση και το υψόμετρο. Π.χ αν πηγαίναμε από την Βαγδάτη η οποία είναι 35°N,45°E στην Οσάκα(πόλη της Ιαπωνίας) που βρήσκεται 35°N,135°E θα ξεκινούσαμε με heading $60^o$ και θα καταλήγαμε σε heading $120^o$. Η μόνη λύση στο πρόβλημα αυτό είναι να ρυθμιστεί η κάμερα με τέτοιο τρόπο έτσι ώστε το heanding να διορθώνει το προσανατολισμό του εστιάζοντας πάντα στη διαδρομή
Πάνω σε αυτή τη λογική υλοποιήθηκε η συνάρτηση getHeading(), η οποία κάνει χρήση της haversing Navigation formula, μία μαθηματική εξίσωση η οποία υλοποιεί την παραπάνω μέθοδο διορθώντας το heading.

$$tc1 = \mathrm{mod}(a\tan 2(\sin(lon2 - lon1)\cdot\cos(lat2),$$
$$\cos(lat1)\cdot\sin(lat2) - \sin(lat1)\cdot\cos(lat2)\cdot\cos(lon2 - lon1)))$$

όπου:

lon1 =γεωγραφικό ύψος του σημείου Α
lat1 = γεωγραφικό μήκος του σημείου Α
lon2 = γεωγραφικό ύψος του σημείου Β
lat2 = γεωγραφικό μήκος του σημείου Β
tc1 = κατεύθυνση του σημείου Β από το σημείο Α



Σχ.6.8 Προσανατολισμός της κάμερας

### 6.4.6 Δημιουργία Λίστας Στοιχείων Πλοήγησης

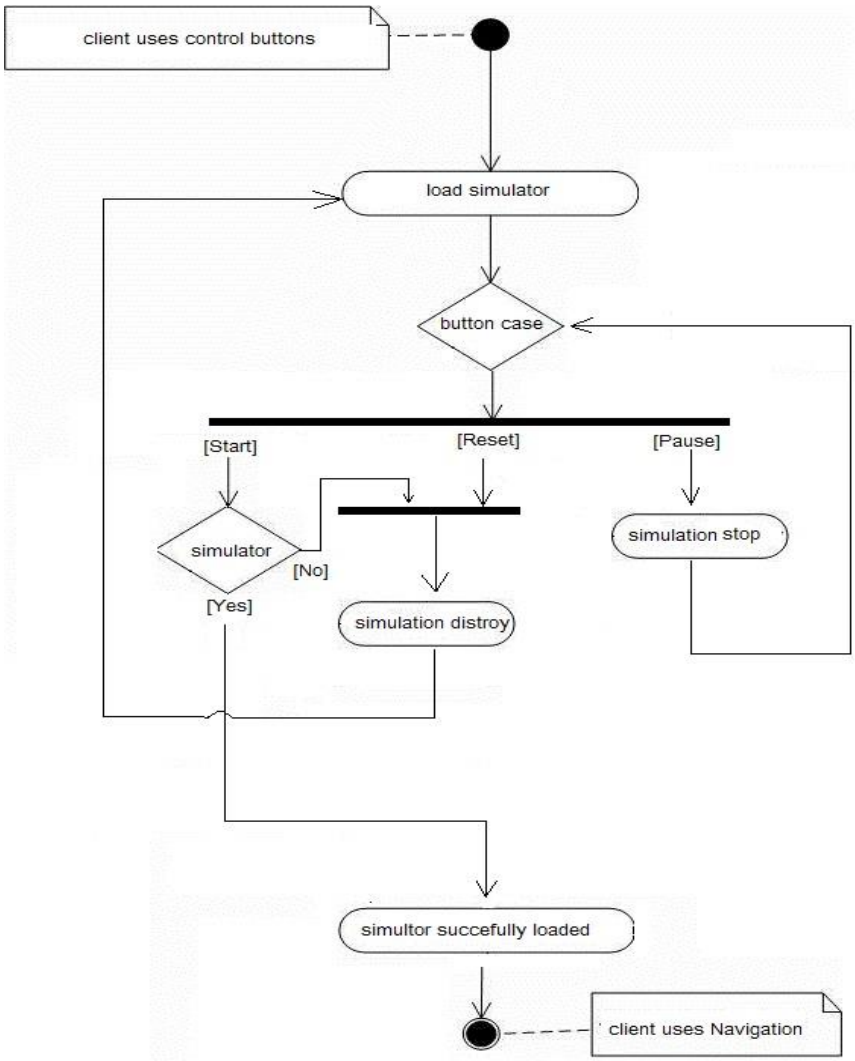
Η buildDirectionList() δημιουργεί τα HTML elements για την αριστερή λίστα του συστήματος πλοήγησης. Αρχικά έχει δημιουργηθεί ένα table στην HTML το οποίο είναι άδειο. Μόλις φορτωθεί το route η συνάρτηση αυτή παίρνει το route,τα steps και κάνει Geocoding την διεύθυνση της αναχώρησης καθώς και την διεύθυνση προορισμού. Στην συνέχεια γεμίζει το table με γραμμές, για κάθε step(αρχικό, τελικό και τα ενδιάμεσα), τα οποία παρέχουν τις πληροφορίες της πλοήγησης.

```
//bulding navigation direction list
$('#route-details').append('<tr
id="start"><td>'+start.address+'</td></tr>');

$('#route-details').append('<tr id="item'+i+'"
class="itemlist"><td>'+step.desc+' ' +step.distanceHtml+'</td></tr>');


$('#route-details').append('<tr
id="end"><td>'+end.address+'</td></tr>');
```

### 6.4.7 Controls

Στην συνέχεια και εφόσον έχει φορτωθεί ο χάρτης και τα αντίστοιχα directions, φορτώνεται και ο simulator και ο χρήστης μπορεί να πλοηγηθεί πατώντας το κουμπί «Start», να πατήσει «Pause» ή να κάνει «Reset» το ταξίδι του. Η controlSimulation()

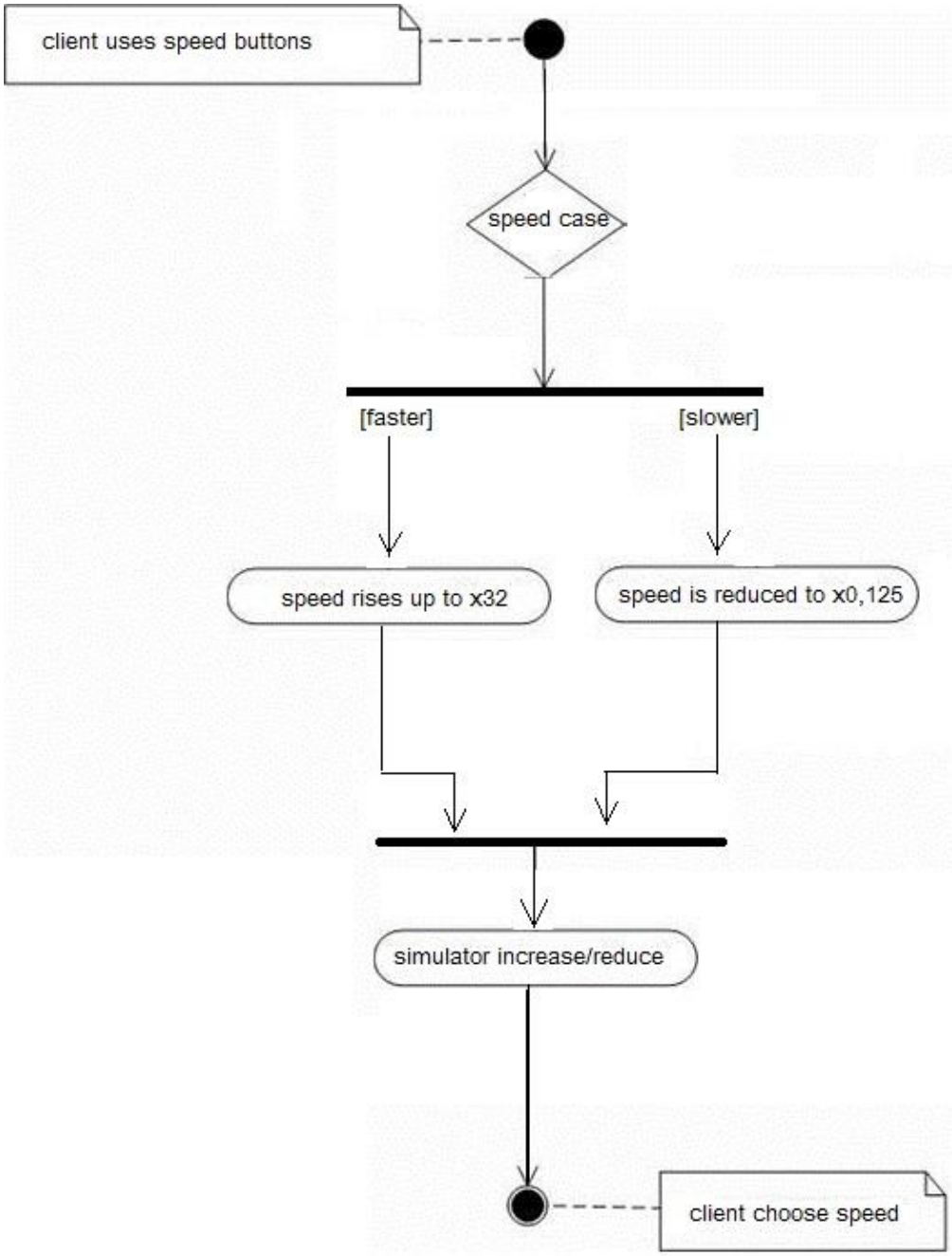
που υλοποιεί τα παραπάνω χρησιμοποιεί την switch case command ανάλογα με την επιλογή. Έτσι έχουμε:




Activity diagram 6.1 Επιλογή Control button

Επίσης μπορεί να επιταχύνει ή να επιβραδύνει την πλοήγηση και να ενεργοποιήσει την υπηρεσία StreetView .

Η επιτάχυνση εκτελείται επίσης με cases, case slower κ΄ faster. Τα όρια της ταχύτητας τα οποία κινείται ο simulator είναι 0,125<speed<32.



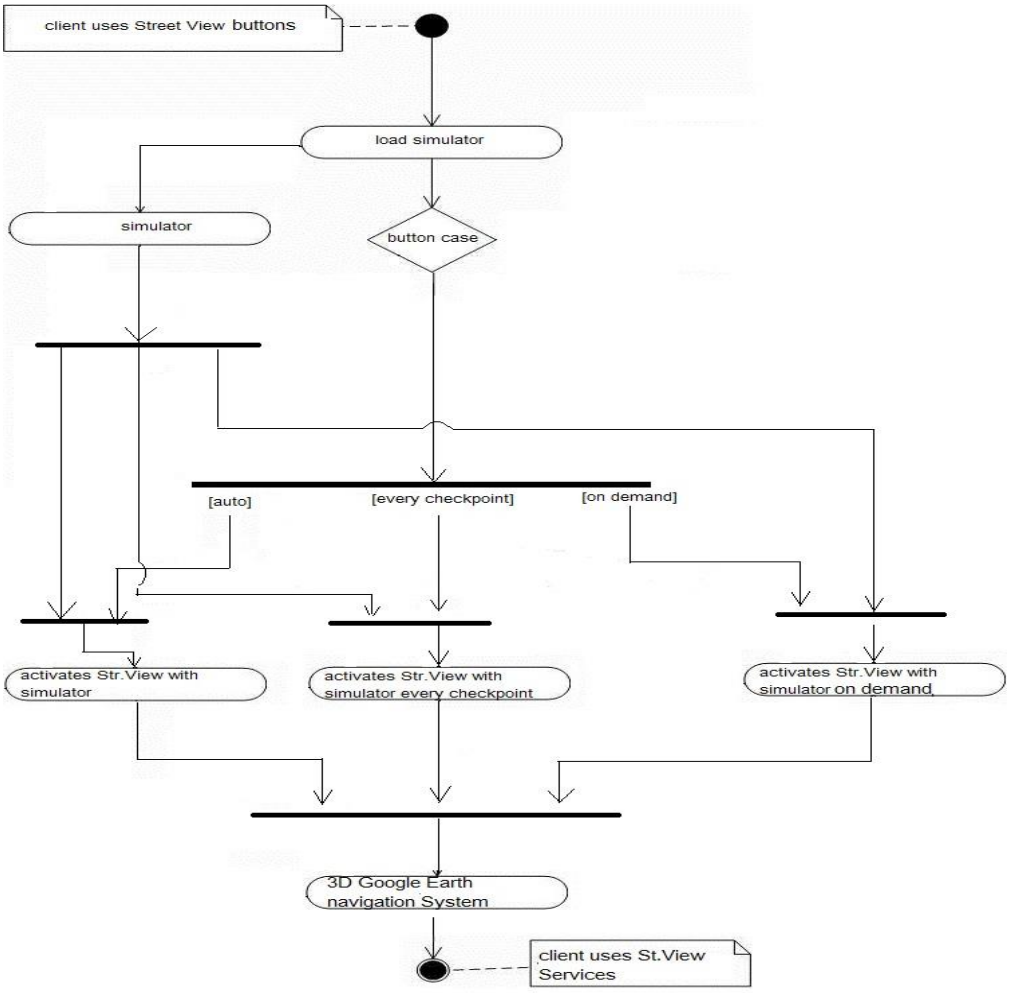

Activity diagram 6.2 Επιλογή ταχύτητας

### 6.4.8 StreetView

Αντίστοιχα η StreetView υπηρεσία εκτελείται από την συνάρτηση showPano(). Και στις τρεις επιλογές που έχει ο χρήστης για να καλέσει το service αυτό('auto', 'every checkpoint', 'ondemand'), η παραπάνω συνάρτηση δημιουργεί ένα StrviewPanorama Object το οποίο παίρνει σα παραμέτρους το location του σημείου που είναι ο

simulator καθώς και το Heading και τα φορτώνει στην οθόνη σα φωτογραφία. Στην περίπτωση που η Street View υπηρεσία είναι στην επιλογή 'auto' θα πρέπει να βρεθεί ο κατάλληλος ρυθμός rendering των φωτογραφιών, έτσι ώστε να μην αργούν αλλά ουτέ και να εναλλάσονται πολύ γρήγορα με αποτέλεσμα να μην συμβαδίζει η φωτογραφία με τον simulator. Συνεπώς ο χρόνος ανανέωσης της φωτογραφίας της υπηρεσίας StreetView θα πρέπει να είναι εκρφαρασμένος συναρτήση του χρόνου κίνησης του simulator. Ύστερα από χρονικούς πειραματισμούς, η ontick() συνάρτηση που υλοποιήθηκε ορίζει ότι η ανανέωση των φωτογραφιών StreetView θα πρέπει να γίνονται αφού έχουν περάσει 6sec κίνησης του simulator. Με βάση αυτή τη χρονική περίοδο, πετυχαίνεται η σωστή και συμβατή χρήση της Street View υπηρεσίας και του simulator.

Η Επιλογές που παρέχει το UI στον χρήστη μέσω του StreetView είναι η εξής:




Activity diagram 6.3 Επιλογή Street View

## 6.4.9 Υλοποποίηση Simulator

Όπως αναφέρθηκε και πιο πάνω η συνάρτηση controlSimulation() καλεί τον simulator ο οποίος ξεκινάει την πλοήγηση στο χάρτη. Ο simulator εκτελείται μέσω ενός google.earth.addEventListener ο οποίος παίρνει τρεις παραμέτρους. Το google

earth plugin object, το event στο οποίο ακούει και την callback σύναρτηση που καλεί όταν ολοκληρωνεται το event. Πιο συγκεκριμένα:

```
//Google Earth Listener function
window.google.earth.addEventListener(this.ge,'frameend',this.tickListener);
```

Το frameend event ενεργοποιείται όταν το googleEarth έχει τελειώσει το rendering της εικόνας πλοήγησης. Το event αυτό θα καλεστεί πολλές φορές διαδοχικά, καθώς η εικόνα πλοήγησης θα αλλάζει(re-rendering), δημιουργώντας σταδιακές αλλαγές στο παράθυρο πλοήγησης (frames) για ομαλή (smooth) κίνηση στο χώρο.

Η cb συνάρτηση tickListener() καλεί την συνάρτηση tick() :

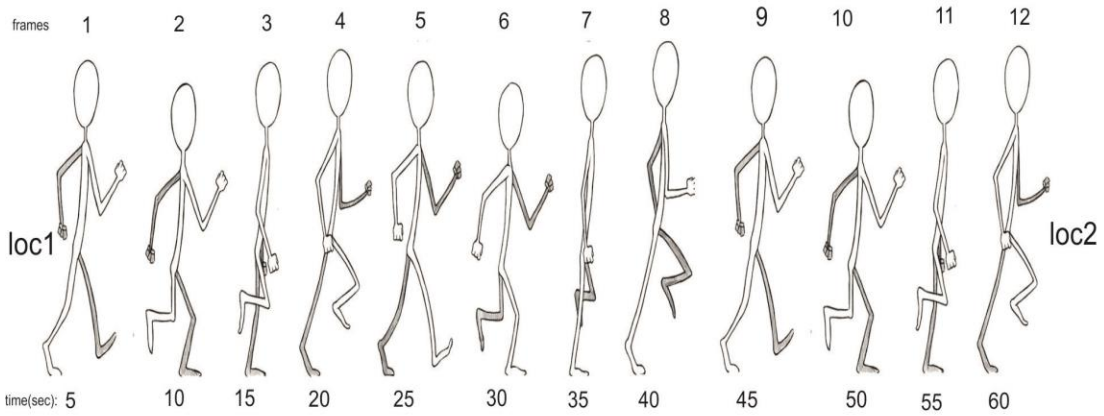
```
//tickListener callback function
tickListener = function() {
                        if (me.doTick)
                          me.tick();
                      };
```

Η συνάρτηση tick() ουσιαστικά δίνει τον παλμό κίνησης του simulator. Όπως ειπώθηκε η κίνηση στο χώρο θα πρέπει να είναι ομαλή με τα frames να διαδέχονται το ένα μετά το άλλο δημιουργώντας επαναλαμβανόμενες εικόνες συναρτήση του χρόνου και της απόστασης.
Ο χώρος κίνησης όπως δείχνει βρίσκεται στη δομή πίνακα path και ο χρόνος κίνησης μας (ο πραγματικός) στο path.duration όπως έχει υπολογιστεί στην ενότητα 6.3.3. Καθώς τρέχουν τα frames, θα πρέπει να προσαρμοστεί η κίνηση έτσι ώστε να φαίνεται ομαλή και ρεαλιστική.
Αυτό που γνωρίζουμε όπως αναφέρθηκε είναι το κάθε σημείο i στο path και το path[i].duration, το χρόνο που χρειάζεται για να φτάσουμε από το ένα σημείο στο άλλο. Συνεπώς ανάμεσα σε 2 διαδοχικά σημεία i και i+1 πρέπει να υπολογιστούν όλα τα ενδιάμεσα σημεία συναρτήση του δειγματολολιπτικού χρόνου κίνησης segmentTime.

Το segmentTime είναι συνάρτηση της ταχύτητας επί της σταθεράς TICK_SIM_MS που ορίζεται ίση με 66ms ύστερα από πειραματική διαδικασία. Διαιρόντας με το 1000 παίρνουμε το δειγματολιπτικό χρόνο σε sec.

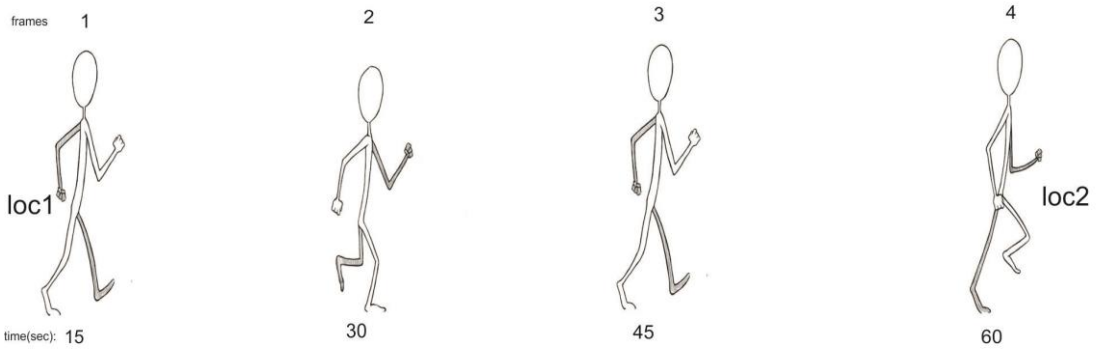
```
this.segmentTime_ += DDSimulator.TICK_SIM_MS * this.options.speed / 1000.0;
```

Γνωρίζοντας λοιπόν τον χρόνο διάρκειας δύο σημείων και έχοντας υπολογίσει το χρόνο δειγματολολειψίας, μπορούμε να υπολογίσουμε το επόμενο σημείο ανάμεσα στο path[i].loc κα στο path[i+1].loc. Έτσι π.χ αν δύο σημεία απέχουν μεταξύ τους ας πούμε 60 sec και το segmentTime είναι ανά 5 sec τότε καθε frame θα έχει διάρκεια σε πραγματικό χρόνο 5 sec.




Σχήμα 6.9: Κίνηση των frames ανά 5sec

Αντιθέτως αν το segmentTime είναι π.χ 15 sec, τότε το κάθε frame θα είχε διάρκεια 15 sec σε πραγματικό χρόνο και η κίνησή μας θα ήταν κάπως έτσι:



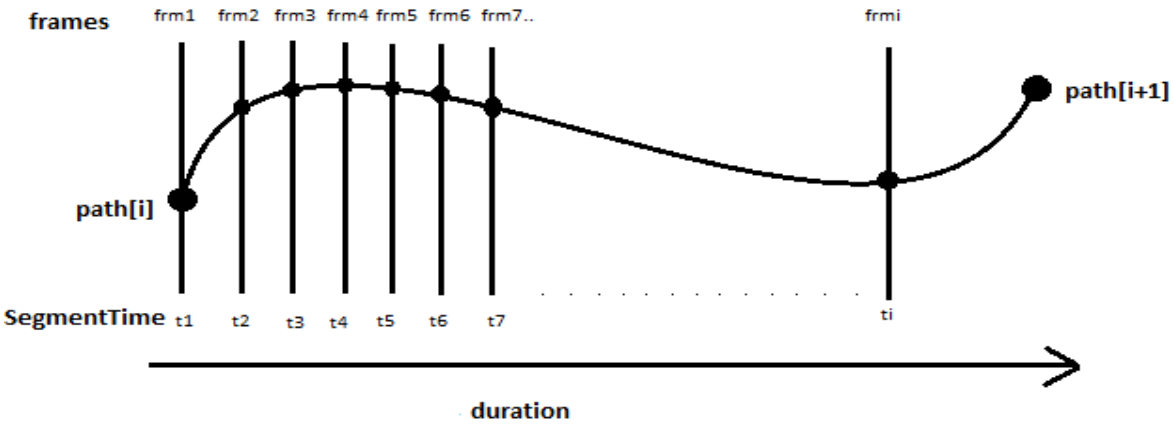

Σχήμα 6.10: Κίνηση των frames ανά 15sec

Το πρώτο ζητούμενο λοιπόν ήταν να βρεθεί εκείνος ο κατάλληλος χρόνος των frames έτσι ώστε η κίνησή μας να δείχνει όσο το δυνατό πιο ομαλή και φυσική χωρίς να υπάρχουν κενά στο χρόνο κίνησης αλλά και το αντίθετο να μην υπάρξει υπερφόρτωση από frames με αποτέλεσμα να κολλήσει η εφαρμογή. Όπως ειπώθηκε και πιο πάνω ύστερα από πειραματικές διαδικασίες ο σκοπός επιτεύχθει με αποτέλεσμα να πετύχουμε μια smooth κίνηση στο χώρο.

Ο υπολογισμός των ενδιάμεσων σημείων μεταξύ δύο location με βάση τη δειγματοληψία μας, δηλαδή με βάση το χρόνο κίνησης, υλοποιήθηκε με τη χρήση της Linear Interpolation equation formula

$$x_0 = a + t/n(c-a), y_0 = b + t/n(d-b) \;\; c(x_0, y_0)$$

Το (a,b) και (c,d) είναι οι συντεταγμένες των δύο σημείων path[i].loc path[i+1].loc αντίστοιχα, το t δηλώνει το segmentTime, ενώ το n είναι ο συνολικός χρόνος διάρκειας των δύο σημείων δηλαδή το path[i].duration.



Σχήμα 6.11: Υπογισμός σημείου με βάση το segmentTime

Με βάση την παραπάνω μαθηματική εξίσωση υλοποιήθηκε η συνάρτηση interpolateLoc() function, η οποία αναλαμβάνει την υλοποιήση της παραπάνω περιγραφής. Το επόμενο στάδιο είναι να διορθώνεται το heading της κάμερας και αυτό πραγματοποιείται μέσω της συνάρτησης getHeading() η οποία αναλύθηκε πιο πάνω.
Τέλος ο simulator θα πρέπει να κάνει drive την διαδρομή δηλαδή να ξεκινήσει την πλοήγηση. Έτσι υλοποιούμε τη function drive(), η οποία παίρνει σα παραμέτρους τις δύο παραπάνω συναρτήσεις.Ο κώδικας έχει ως εξής:

```
// moving frames per second
DDSimulator.prototype.tick_ = function() {

code
…
…
 // update the current location
  this.currentLoc = this.geHelpers_.interpolateLoc(
      this.path[this.currPathIndex].loc,
      this.path[this.currPathIndex + 1].loc,
      this.segmentTime_ / this.path[this.currPathIndex].duration);
 this.currentHeading=
this.geHelpers_.getHeading(this.path[this.currPathIndex].loc,
                                  this.path[this.currPathIndex + 1].loc);
  this.drive_(this.currentLoc, this.currentHeading);

}
```

Η συνάρτηση drive() ουσιαστικά παίρνει τιμές το location και το heading δηλαδή το σημείο και το που θα κοιτάει κάμερα και εκτελεί την πλοήγηση.

Το επόμενο βήμα είναι η ομαλή μετατόπιση της κάμερας από το ένα σημείο στο επόμενο και το κατάλληλο εύρος ανάλογα με την ταχύτητα κατά τη διάρκεια του simulation. Εδώ έχουμε την συνάρτηση moveToPointDriving(), η οποία υλοποιεί την παραπάνω διαδικασία και καλείται μέσω της συνάρτησης drive() και παίρνει ακριβώς τις ίδιες παραμέτρους. Αυτό φαίνεται στον παρακάτω κώδικα:

```
//simulation drive
DDSimulator.prototype.drive_ = function(loc, heading) {

  this.moveToPointDriving_(loc, heading);
}
```

### 6.4.10 Κίνηση Κάμερας

Τέλος όπως έχει αναφερθεί ο χρήστης μπορεί να παίξει με την κάμερα αλλάζοντας τόσο το εύρος της κάνοντας δηλαδή zoum in ή zoum out, όσο και το πού θα κοιτάει αριστερά ή δεξιά.
Οι συναρτήσεις moveHeading() και moveAltitude() παίρνουν σα παράμετρο τη τιμή(σε μοίρες) που δίνει ο χρήστης πατώντας το κουμπί και την προσθέτουν στο εκάστοτε σημείο που βρίσκεται η κάμερα. Μ' αυτό τον τρόπο προκαλούμε μετατόπιση της κάμερας manually και μπορούμε να κοιτάμε σε οποιοδήποτε σημείο του χώρου ανεξάρτητα από το σημείο που πλοηγείται.

```
//moveHeading():
function moveHeading(move)  {

if (!DS_simulator)
            return;
// simulator current Altitude plus UI button parameter
      DS_simulator.currentLook = DS_simulator.currentLook + move;

      DS_simulator.lookAuto = false;
}

// moveAltitude():
function moveAltitude(move)  {
      if (!DS_simulator)
            return;
// simulator current Altitude plus UI button parameter
      DS_simulator.currentAltitude = DS_simulator.currentAltitude + move;

}
```

# Κεφάλαιο 7

## ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΡΟΤΑΣΕΙΣ

Ανακεφαλαιώνοντας μπορεί να διατυπωθεί ο ισχυρισμός πως η διπλωματική εργασία που παρουσιάζεται εκπλήρωσε τον αρχικό της σκόπο. Κατά την ανάπτυξη της εφαρμογής, η συμμετοχή μου σε δικτυακούς τόπους συζητήσεων (forums) βοήθησε την εύρεση βέλτιστων αλγορίθμων και μεθόδων. Η υλοποίηση ενός τρισδιάστατου συστήματος πλοήγησης χρησιμοποιώντας τις Google Eath API και Maps API υπηρεσίες ολοκληρώθηκε με επιτυχία. Το 3D Goolge Map σύστημα πλοήγησης έχει την δυνατότητα να χρησιμοποιηθεί από τους περισσότερους browsers στο internet χρησιμοποιώντας το Google Earth Plugin στον υπολογηστή.

Η εγκατάσταση της Google Earth Plugin στον υπολογιστή μπορεί να είναι εύκολη και γρήγορη διαδικασία, έχει όμως κάποιες αδυναμίες και περιορισμούς. Η φόρτωση του plugin είναι μία χρονοβόρα διαδικασία και χρειάζεται αναμονή ωσότου το Google Earth ολοκληρώσει την επιλεγμένη διαδρομή αναζήτησης λόγω των πολλών δεδομένων που χρειάζονται να επεξεργαστούν. Επίσης το βασικό ζήτημα ανάγεται στο λειτουργικό σύστημα που χρειάζεται για να χρησιμοποιηθεί η παραπάνω εφαρμογή. Το Plugin του Goolge Earth API τρέχει σε Windows λειτουργικό γεγονός που καθιστά ανέφικτη την χρήση της εφαρμογής σε άλλα λειτουγικά συστήματα όπως Android, iOS κ.τ.λ. Για τους παραπάνω λογους η χρήση του 3D navigation system της Google σε ηλεκτρονικές συσκευές όπως τα smartphone, tablets, ipad κ.τ.λ είναι αδύνατη λόγω των περιορισμένων ακόμα δυνατοτήτων.

Τα τελευταία χρόνια αναπτύσσονται διεπαφές λογισμικού για την πρόσβαση σε υλικό γραφικών μέσα από ένα web browser όπως το WebGL, το οποίο έχει τη δυνατότητα να κάνει render 3D και 2D γραφικά σε οποιοδήποτε web browser χωρίς τη χρήση plugins. Το WebGL παρέχει ένα API με 3D στοιχεία, τα οποία ενσωματώνονται στην HTML5 δίνοντας στη διεπαφή 3D χαρακτηριστικά.

Με βάση τα παραπάνω, πιστεύουμε ότι είναι θέμα χρόνου η Google να αναπτύξει 3D γραφικά πάνω σε Google maps χωρίς τη χρήση του plugin. Με βάση αυτό, το σύστημα πλοήγησης θα έχει πλέον τη δυνατότητα να τρέχει σε κάθε λειτουργικό σύστημα οποιασδήποτε ηλεκτρονικής συσκευής.

# Βιβλιογραφία

## Papers:

V.Narayani,S.RajKumar(March2014).*International Journal of Innovative Technology and Exploring Engineering (IJITEE)ISSN: 2278-3075,Volume-3, Issue-10*,

Jovin J. Mwemezi, Youfang Huang(2011). *Optimal Facility Location on Spherical Surfaces: Algorithm and Application New York Science Journal,*

Kenneth Gade THE JOURNAL OF NAVIGATION (2010),63, 395–417. *The Royal Institute of Navigationdoi:10.1017/S0373463309990415. A Non-singular Horizontal Position Representation*

Hatem Hamad, Motaz Saad, and Ramzi Abed(January 2010). *International Arab Journal of e-Technology, Vol. 1, No. 3. Performance Evaluation of RESTful Web Services for Mobile Devices*

Ed Williams. *Aviation Formulary V1.46* http://williams.best.vwh.net/avform.htm The great circle distance d between two points with coordinates formula*(2011)/This is an electronic document.*

## Links

***Wikipedia:***

http://en.wikipedia.org/wiki/Spherical_coordinate_system ***(****ανάκτηση 4/52012)*

http://en.wikipedia.org/wiki/Great-circle_distance ***(****ανάκτηση 4/32012)*

http://en.wikipedia.org/wiki/Dot_product ***(****ανάκτηση 7/62012)*

http://en.wikipedia.org/wiki/Linear_interpolation ***(****ανάκτηση 7/32012)*

## Google Earth:

https://developers.google.com/earth/documentation/index (ανάκτηση 30/8/ 2012.)

https://developers.google.com/earth/documentation/placemarks (ανάκτηση 30/7/ 2012.)

https://developers.google.com/earth/documentation/events (ανάκτηση 30/5/ 2012.)

## The Google Directions API - Google Maps API Web Services

https://developers.google.com/maps/documentation/directions/ (ανάκτηση 3/6/2012.)

## Street View Service - Google Maps JavaScript API v3

https://developers.google.com/maps/documentation/javascript/streetview (ανάκτηση 3/6/2012.)

# ΠΑΡΑΡΤΗΜΑ

## index.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
           <meta charset="utf-8">
           <title>3D Navigator</title>
           <meta name="viewport" content="width=device-width, initial-scale=1.0">
           <meta name="description" content="">
           <meta name="author" content="">

           <!-- Le styles -->
           <link href="bootstrap/css/bootstrap.css" rel="stylesheet">

           <!--<link href="bootstrap/bootstrap-responsive.css" rel="stylesheet">-->
           <link href="css/default.css" type="text/css" rel="stylesheet">
           <link href="css/bootstrap.min.css" rel="stylesheet" media="screen">
           <!--<link  href="css/bootstrap-responsive.min.css" rel="stylesheet" media="all" />-->
           <script src="http://code.jquery.com/jquery.js"></script>
    </head>

    <body>



           <div class="navbar navbar-inverse navbar-fixed-top">
                  <div class="navbar-inner">
                         <div class="container-fluid" style="height:50px;">
                                <a class="brand" href="#" style="font-size:25px;" >3D GoogleEarth
Navigation</a>
                         </div>
                  </div>
           </div>


           <div class="container-fluid fill">
                  <div class="row-fluid fill">
                         <div class="span3 fill" >
                                <div class="well sidebar-nav">

                                       <div class="row-fluid fill">
                                              <div class="span12" style="padding:0 4px; background-
color:#EFF5FB;">
                                                     <form class="form-search">
                                                            <input type="text" id="from"
placeholder="Origin...">
                                                            <input type="text" id="to"
placeholder="Destination...">
                                                            <a id="go" type="btn" class="btn  btn-
info" value="Go" data-toggle="tooltip" data-trigger="hover" title="Find route" data-
placement="right" onclick="goDirections();"><i class="icon-globe"></i></a>
                                                     </form>
                                              </div>
                                       </div>

                                       <div class="row-fluid fill">
                                              <div class="span8" style="padding:0 4px;" >
```

```
                                                <p><strong>Navigation System
Controls:</strong></p>
                                                <a type="btn" id="str" class="btn btn btn-
success  btn-mini" value="Start" onclick="controlSimulator('start');"><i class="icon-
play"></i></a>
                                                <a type="btn" id="ps" class="btn btn-danger
btn-mini" value="pause" onclick="controlSimulator('pause');"><i class="icon-pause"></i></a>
                                                <a type="btn" id="res" class="btn btn-warning
btn-mini" value="Reset" onclick="controlSimulator('reset');"><i class="icon-refresh"></i></a>

                                                <b>Speed:</b> <strong><span id="speed-
indicator"></span></strong>
                                                <a type="btn" class="btn btn-mini  icon"
onclick="controlSimulator('slower');" value="-">
                                                      <i class="icon-minus"></i>
                                                </a>

                                                <a  type="btn" class="btn btn-mini"
onclick="controlSimulator('faster');" value="+">
                                                      <i class="icon-plus"></i>
                                                </a>
                                          </div>

                                          <div class="span4 fill">
                                                <p><strong>Camera Controls:<strong></p>
                                                <div>
                                                      <a type="btn" class="btn btn-mini
offset3" id="point1" value="Altitude-" onclick="moveAltitude(-10);"><i class="icon-zoom-
in"></i></a>
                                                </div>

                                                <div>
                                                      <a type="btn" class="btn btn-mini
offset1" id="pointX" value="Heading-" data-toggle="tooltip" data-trigger="hover" title="look
left" data-placement="left"  onclick="moveHeading(-10);"><i class=" icon-circle-arrow-
left"></i></a>
                                                      <a type="btn" class="btn btn-mini
offset1 " id="point2" value="Heading+" onclick="moveHeading(+10);"><i class=" icon-circle-arrow-
right"></i></a>
                                                </div>

                                                <div>
                                                      <a type="btn" class="btn btn-mini
offset3" id="point3" value="Altitude+" onclick="moveAltitude(+10);"><i class=" icon-zoom-
out"></i></a>
                                                </div>

                                          </div>

                                    </div></br>

                                    <div class="row-fluid fill">
                                          <div class="span12" >
                                                <form class="form-inline" style="padding:0 4px;
background-color:#EFF5FB;">
                                                      <a type="btn" class="btn btn-inverse
btn-small" onclick="showPano();" value="StreetView">StreetView</a>
```

```
                                                        <input type="radio" name="streetview"
id="Chkauto" value="auto" />auto
                                                        <input type="radio" name="streetview"
id="Chkcheckpoint"  value="checkpoint" />every checkpoint
                                                        <input type="radio" name="streetview"
id="Chkondemand" value="ondemand" checked />ondemand
                                                </form>

                                        </div>
                                  </div>




                                  <div id="routes" style="padding: 0 10px 0 15px
10px;height:220px;overflow:auto;">
                                        <table id="route-details" class="table table-hover" >

                                        <tbody>


                                        </tbody>

                                        </table>
                                  </div>

                            </div><!--/.well -->

                            <div class="well sidebar-nav">
                                  <div id="pano" class="fill"></div>
                            </div>

                      </div><!--/span3-->

                      <div class="span9 fill">

                            <div id="map3d"></div>

                      </div><!--/span9-->

                </div><!--/row-->
          </div><!--/.fluid-container-->

          <!-- javascript
          ================================================== -->
          <!-- Placed at the end of the document so the pages load faster -->

          <script
src="http://www.google.com/jsapi?key=ABQIAAAAWGhZim6UJLyxNTeb9UQgmBT2yXp_ZAY8_ufC3CFXhHIE1NvwkxT
Cf8E9OTYlrI2PSdn50JWIUg_6Ng&sensor=false"> </script>
          <script src="js/math3d.js"></script>
          <script src="js/simulator.js"></script>
          <script src="js/geplugin-helpers.js"></script>
          <script src="js/main.js" type="text/javascript"></script>
          <script src="js/direction.js" type="text/javascript"></script>
          <script src="js/extensions-0.2.1.js"></script>
          <script src="js/geo-0.2.js"></script>
          <script src="bootstrap/js/bootstrap.js"></script>
```

```
        <script type="text/javascript">
          var ge;
          var map;
          var gex;
          google.load("earth", "1",{"other_params":"sensor=false"});
          google.load("maps", "3",{"other_params":"sensor=false"});

          function init() {
                google.earth.createInstance('map3d', initCB, failureCB);

          }
          function initCB(instance) {
                ge = instance;
                ge.getWindow().setVisibility(true);
                ge.getNavigationControl().setVisibility(ge.VISIBILITY_AUTO);
                ge.getLayerRoot().enableLayerById(ge.LAYER_BUILDINGS, true);
                 ge.getLayerRoot().enableLayerById(ge.LAYER_ROADS, true);
                  ge.getLayerRoot().enableLayerById(ge.LAYER_TERRAIN, true);

               gex = new GEarthExtensions(ge);

          }
          function failureCB(errorCode) {
          }
          google.setOnLoadCallback(init);
        </script>
    </body>
</html>
```

**default.css**

```
html{height: 100%;}
body {
        padding-top: 60px;
        padding-bottom: 40px;
        height: 100%;
    width: 100%;
 -webkit-box-sizing: border-box; /* Safari/Chrome, other WebKit */
    -moz-box-sizing: border-box;    /* Firefox, other Gecko */
    box-sizing: border-box;         /* Opera/IE 8+ */

      }
      .sidebar-nav {
        padding: 9px 0;
      }

      @media (max-width: 980px) {
        /* Enable use of floated navbar text */
        .navbar-text.pull-right {
          float: none;
          padding-left: 5px;
          padding-right: 5px;
        }
      }

#map3d{
   width: 100%;
    height: 100%;
    min-height: 100%;

    display: block;
}

.fill {
    min-height: 100%;
    height: 100%;
 overflow:hidden;
}

#pano{
 height:300px;
 width:100%;
 min-height: 100%;


}


.itemlist
{
 background-color:#F2FBEF;
 color: #000000;
 margin: 1px 1px;
 cursor:pointer;
 border-bottom-style:solid;
 border-bottom-width:1px;
 border-bottom-color:#B3AEAB;
}
```

```
.selected
{
 background-color:#BCF5A9;
 color: #ffffff;
}
#start
{
 background-color:#CEF6F5;
 color: #ffffff;
 margin: 1px 1px;
 cursor:pointer;
 font-size :15px;
 font-weight :bold;
 border-bottom-style:solid;
 border-bottom-width:1px;
 border-bottom-color:#B3AEAB;
 border-top: 1px solid black;
}
#end
{
 background-color:#CEF6F5;
 color: #ffffff;
 margin: 1px 1px;
 cursor:pointer;
 font-size :15px;
 font-weight :bold;
}
```

**main.js**

```
$(document).ready(loadApp);

$(window).resize(function(){
 adjustSizes();
 });

 function adjustSizes(){


 }
 function loadApp(){

        adjustSizes();


        $('input[name="streetview"]').change(function(){
               if($(this).is(':checked'))  {
                      if($(this).val()==="auto")
                             $('input[id="Btnstreetview"]').attr('disabled','disabled');
                      else if($(this).val()==="checkpoint")
                             $('input[id="Btnstreetview"]').attr('disabled','disabled');
                      else if($(this).val()==="ondemand")
                             $('input[id="Btnstreetview"]').removeAttr('disabled');
               }


        });
 }


function moveTilt(move)  {
var camera = ge.getView().copyAsCamera(ge.ALTITUDE_RELATIVE_TO_GROUND);
camera.setTilt(camera.getTilt() + move );
// Update the view in Google Earth
ge.getView().setAbstractView(camera);
}

function moveHeading(move)  {
if (!DS_simulator)
        return;
 DS_simulator.currentLook = DS_simulator.currentLook + move;
 DS_simulator.lookAuto = false;
}

function moveAltitude(move)  {
 if (!DS_simulator)
        return;
 DS_simulator.currentAltitude = DS_simulator.currentAltitude + move;
}
```

## direction.js

```
var directions = null;
var steps = [];
var DS_simulator;
var path = [];
var pathidx = 0;
var lastTickTime =0;
var myPano = null;

/*
 * The goDirections CB function is fired when the 'Go!' button is pressed.
 *This CB function uses the Maps API's Directions class to get the route
 * and pull out the individual route steps into a path, which is rendered as a polyline.
 */
function goDirections() {
  $('#route-details').empty();
  $('#route-details').html(
      '<span class="loading">Loading directions...</span>');

    if(DS_simulator){
          controlSimulator('pause');
          DS_simulator = null;
    }
    var start = $('#from').val();
    var end = $('#to').val();

    var request = {
    origin:start,
    destination:end,
    travelMode: google.maps.TravelMode.DRIVING
    };

    directionsService = new google.maps.DirectionsService();
    directionsService.route(request, function(response, status) {
    if (status == google.maps.DirectionsStatus.OK) {

      directionsLoaded(response);
    }
  });
}

/*CB function:function to call when the event is fired(load event)*/
function directionsLoaded(directionResult) {
    $('#route-details').empty();
    gex.dom.clearFeatures();
    var route = directionResult.routes[0].legs[0];

    var numSteps = route.steps.length;
    buildPathStepArrays(directionResult);
    // build the path and step arrays from the google.maps.Directions route
    buildDirectionList(directionResult);
    placemarks = {};
    // create the starting point placemark and placemarks for every step
    placemarks['start'] = gex.dom.addPointPlacemark(
      new google.maps.LatLng(route.start_location.lat(),
```

```
                            route.start_location.lng()),
      {description: route.start_address});

    for (var i = 0; i < steps.length; i++) {
          var step = steps[i];
          placemarks['item'+i] = gex.dom.addPointPlacemark(
                step.loc,{description: step.desc});
    }

    // create the ending point placemark
    placemarks['end'] = gex.dom.addPointPlacemark(
      new google.maps.LatLng(route.end_location.lat(),
                            route.end_location.lng()),
      {description: route.end_address});

    // Creating the lineString route on the ground
    var lineStringKml = '<LineString><coordinates>\n';
    for (var i = 0; i < path.length; i++)
    lineStringKml +=
        path[i].loc.lng().toFixed(5).toString() + ',' +
        path[i].loc.lat().toFixed(5).toString() +
        ',10\n';

    lineStringKml += '</coordinates></LineString>';
    //parseKml requires a KML string, and returns a KmlFeature object.
    var routeLineString = ge.parseKml(lineStringKml);
    var linePlacemark = gex.dom.addPlacemark({
      lineString: {                                 // Draw me
        path: routeLineString,
        altitudeMode: ge.ALTITUDE_CLAMP_TO_GROUND,
        tessellate: true
      },
      style: {
        line: { color: '88ff0000', opacity: 0.5, width: 10 }
      }
    });
    flyToLatLng(steps[0].loc);

}

/* build the path and step arrays from the google.maps.Directions route */
function buildPathStepArrays(directionResult) {
  steps = [];
  path = [];
  var route = directionResult.routes[0].legs[0];
  var polyline = directionResult.routes[0].overview_path;
  var numPolylineVertices = polyline.length;
  var numSteps = route.steps.length;

  for (var i = 0; i < numSteps; i++) {
    var step = route.steps[i];

    steps.push({
      loc: step.start_location,
      desc: step.instructions,
      distanceHtml: step.instructions,
      pathIndex: path.length
    });
```

```
    var stepDistance = step.distance.value;
    for (var j = 0; j < step.path.length; j++) {
          if(i != numSteps - 1 && j==step.path.length-1)
          {
                if( j == step.path.length -1)
                      continue;
          }
      //var currpath = step.path[i];
      var loc = step.path[j];
      var distance = 0;
      if(i == numSteps - 1 && j==step.path.length-1)
      {
          distance = 0;
      }
      else{
          distance = hdistance(loc, step.path[j+1]);
          }

      path.push({
        loc: loc,
        step: i,
        distance: distance,

        // this segment's time duration is proportional to its length in
        // relation to the length of the step
        duration: step.duration.value * distance / stepDistance
      });
    }
  }
  }


 /* functions which creates the HTML elements for the left directions list*/
  function buildDirectionList(directionResult)
  {
    var route = directionResult.routes[0].legs[0];;
    //var numSteps = route.getNumSteps();
    var start = route.start_address;
    var end = route.end_address;
    $('#route-details').append('<tr id="start"><td>'+start+'</td></tr>');


    for (var i = 0; i < steps.length; i++) {
          var step = steps[i];
          $('#route-details').append('<tr id="item'+i+'" class="itemlist"><td>'+step.desc+' '
+step.distanceHtml+'</td></tr>');
    }
    $('#route-details').append('<tr id="end"><td>'+end+'</td></tr>');
    $('#start').click(function() {
          gex.util.flyToObject(placemarks['start']);

    });

    $('#end').click(function() {
          gex.util.flyToObject(placemarks['end']);

    });
```

```
    $('.itemlist').click(function() {
                var id = $(this).attr('id');
                //alert(id);
                var stepNum = parseInt(id.match(/item(\d+)/)[1]);
                pathidx = steps[stepNum].pathIndex;
                if (DS_simulator)
                      DS_simulator.currPathIndex = pathidx;
                $('.itemlist').removeClass('selected');
                $('#'+id).addClass('selected');
                flyToStep(stepNum);
          });
  }


function moveToPointDriving(loc, heading) {
    ge.getOptions().setFlyToSpeed(0.1);
  var oldLa = ge.getView().copyAsLookAt(
      ge.ALTITUDE_RELATIVE_TO_GROUND);
  var curHeading = oldLa.getHeading();
  var desiredHeading = heading;

  var curRange = oldLa.getRange();
  var desiredRange = Math.max(20.0, 1 * 10);

  var la = ge.createLookAt('');
  la.set(loc.lat(), loc.lng(),
      0, // altitude
      ge.ALTITUDE_RELATIVE_TO_GROUND,
      curHeading + getTurnToDirection(curHeading, desiredHeading),
      60, // tilt
      curRange + (desiredRange - curRange) * 0.1 // range (inverse of zoom)
      );
  ge.getView().setAbstractView(la);
}

  function flyToLatLng(loc) {
  var la = ge.createLookAt('');
  la.set(loc.lat(), loc.lng(),
      10, // altitude
      ge.ALTITUDE_RELATIVE_TO_GROUND,
      90, // heading
      0, // tilt
      200 // range (inverse of zoom)
      );
    ge.getView().setAbstractView(la);

}

function getTurnToDirection(heading1, heading2) {
  if (Math.abs((heading1) - (heading2)) < 1)
    return heading2 - heading1;

  return (fixAngle(heading2 - heading1) < 0) ? -1 : 1;
}

function flyToStep(stepNum) {
  var step = steps[stepNum];
```

```
  var la = ge.createLookAt('');
  la.set(step.loc.lat(), step.loc.lng(),
      0, // altitude
      ge.ALTITUDE_RELATIVE_TO_GROUND,
      getHeading(step.loc, path[step.pathIndex + 1].loc),
      60, // tilt
      50 // range (inverse of zoom)
      );
  ge.getView().setAbstractView(la);
}

function getHeading(loc1, loc2) {
  lat1 = deg2rad(loc1.lat());
  lon1 = deg2rad(loc1.lng());

  lat2 = deg2rad(loc2.lat());
  lon2 = deg2rad(loc2.lng());

  var heading = fixAngle(rad2deg(Math.atan2(
    Math.sin(lon2 - lon1) * Math.cos(lat2),
    Math.cos(lat1) * Math.sin(lat2) - Math.sin(lat1) * Math.cos(lat2) *
      Math.cos(lon2 - lon1))));

  return heading;
}

function deg2rad(d) {
  return d  / 180.0 * Math.PI;
}

function rad2deg(r) {
  return r  / 180.0 * Math.PI;
}
function fixAngle(a) {
  while (a < -180)
    a += 360;

  while (a > 180)
    a -= 360;

  return a;
}

function hdistance(loc1, loc2) {
  p1 = V3.latLonAltToCartesian([loc1.lat(), loc1.lng(),
    ge.getGlobe().getGroundAltitude(loc1.lat(), loc1.lng())]);
  p2 = V3.latLonAltToCartesian([loc2.lat(), loc2.lng(),
    ge.getGlobe().getGroundAltitude(loc2.lat(), loc2.lng())]);
  return V3.earthDistance(p1, p2);
}

function controlSimulator(command, opt_cb) {
  switch (command) {
    case 'reset':
          if (DS_simulator){
                DS_simulator.destroy();
                pathidx = 0;
```

```
      }
  DS_simulator = new DDSimulator(ge, path, pathidx, {

    on_tick: function() {
             if($('input[id="Chkauto"]').is(':checked'))  {
                    if(Math.abs(DS_simulator.totalTime - lastTickTime) > 6 ){
                           lastTickTime = DS_simulator.totalTime;
                           showPano();

                    }
             }
    },

    // when the simulator moves to a new step (specified as an integer
    // index in DS_path items), highlight that step in the directions
    // list
    on_changeStep: function(stepNum) {
             if($('input[id="Chkcheckpoint"]').is(':checked'))  {
                           showPano();
             }
             $('.itemlist').removeClass('selected');
             $('#item'+stepNum).addClass('selected');

             var rowPos = $('#item'+stepNum).position().top;
             // Get row position by index
             var ypos = $('#item'+stepNum).offset().top;
             // Go to row
             if(stepNum > 1){
                    $('#routes').animate({
                           scrollTop: $('#routes').scrollTop() + rowPos - 350}, 500);
                    }

    }
  });

  DS_updateSpeedIndicator();
  DS_simulator.initUI(opt_cb);
  break;

case 'start':
  if (!DS_simulator)
    controlSimulator('reset', function() {
      DS_simulator.start();
      if (opt_cb) opt_cb();
    });
  else {
    DS_simulator.start();
    if (opt_cb) opt_cb();
  }
  break;

case 'pause':
  if (DS_simulator)
    DS_simulator.stop();

  if (opt_cb) opt_cb();
  break;
```

```
    case 'resume':
      if (DS_simulator)
        DS_simulator.start();

      if (opt_cb) opt_cb();
      break;

    case 'slower':
      if (DS_simulator && DS_simulator.options.speed > 0.125) {
        DS_simulator.options.speed /= 2.0;
        DS_updateSpeedIndicator();
      }
      break;

    case 'faster':
      if (DS_simulator && DS_simulator.options.speed < 32.0) {
        DS_simulator.options.speed *= 2.0;
        DS_updateSpeedIndicator();
      }

  }
}

/**
 * Update the current simulation speed multiplier
 */
function DS_updateSpeedIndicator() {
  if (DS_simulator.options.speed < 1)
    $('#speed-indicator').text('1/' +
        Math.floor(1 / DS_simulator.options.speed) + 'x');
  else
    $('#speed-indicator').text(Math.floor(DS_simulator.options.speed) + 'x');
}

function showPano() {

    if (!DS_simulator)
           return;

    var panoramaOptions = {
    position: DS_simulator.currentLoc,
    pov: {
      heading: DS_simulator.currentHeading,
      pitch: 0
    }
    };


    if(!myPano)
          myPano = new
google.maps.StreetViewPanorama(document.getElementById("pano"),panoramaOptions);
    myPano.setPosition(DS_simulator.currentLoc);
    myPano.setPov({
           heading: DS_simulator.currentHeading,
            zoom: 1,
            pitch: 0
        });
    myPano.setVisible(true);   }
```

## simulator.js

```
DDSimulator.TICK_SIM_MS = 66;
DDSimulator.prototype.totalTime = 0;
DDSimulator.prototype.totalDistance = 0;
DDSimulator.prototype.currentSpeed = 0;
DDSimulator.prototype.currentLoc = null;
DDSimulator.prototype.currentHeading = 0;
DDSimulator.prototype.currentLook = -1;
DDSimulator.prototype.lookAuto = true;
DDSimulator.prototype.currentAltitude = 140;
DDSimulator.prototype.currPathIndex = 0;

function DDSimulator(ge, path, pathidx, opt_opts) {
  this.ge = ge;
  this.path = path;
  this.options = opt_opts || {};
  if (!this.options.speed)
    this.options.speed = 1.0;

  this.currentLoc = this.path[0].loc;

  // private vars
  this.geHelpers_ = new GEHelpers(ge);
  this.doTick_ = false;
  this.currPathIndex = pathidx;
  this.currentStep_ = -1;
  this.segmentTime_ = 0;
  this.segmentDistance_ = 0;
}

 // Initializes the simulator UI
DDSimulator.prototype.initUI = function(opt_cb) {
  var me = this;
  me.finishInitUI_(opt_cb);
}
// Completes the UI initialization
DDSimulator.prototype.finishInitUI_ = function( opt_cb) {
  this.drive_(this.path[0].loc,
      this.geHelpers_.getHeading(this.path[0].loc, this.path[1].loc));

  var me = this;
  this.tickListener = function() {
                        if (me.doTick_)
                          me.tick_();
                      };
  window.google.earth.addEventListener(this.ge, 'frameend', this.tickListener);
  if (opt_cb) opt_cb();
}

//Destroy the UI and detach from the Earth instance

DDSimulator.prototype.destroy = function() {
  this.stop();
  //this.ge.getFeatures().removeChild(this.modelPlacemark);
  window.google.earth.removeEventListener(this.ge, 'frameend',
```

```
                                        this.tickListener);
}

//Start/resume the simulation clock
DDSimulator.prototype.start = function() {
  if (this.doTick_)
    return;

  this.oldFlyToSpeed = this.ge.getOptions().getFlyToSpeed();
  this.ge.getOptions().setFlyToSpeed(this.ge.SPEED_TELEPORT);

  this.doTick_ = true;
  this.tick_();
}

/**
 * Stop/pause the simulation clock
 */
DDSimulator.prototype.stop = function() {
  if (!this.doTick_)
    return;
  this.ge.getOptions().setFlyToSpeed(this.oldFlyToSpeed);

  this.doTick_ = false;
}

//Position the simulator driving to the given location
DDSimulator.prototype.drive_ = function(loc, heading) {
  this.moveToPointDriving_(loc, heading);
}


DDSimulator.prototype.getTurnToDirection_ = function(heading1, heading2) {
  if (Math.abs((heading1) - (heading2)) < 1)
    return heading2 - heading1;
  return (this.geHelpers_.fixAngle(heading2 - heading1) < 0) ? -1 : 1;
}


DDSimulator.prototype.moveToPointDriving_ = function(loc, heading) {
  var oldLa = this.ge.getView().copyAsLookAt(
      this.ge.ALTITUDE_RELATIVE_TO_GROUND);
  var curHeading = oldLa.getHeading();
  var desiredHeading = heading;

  var curRange = oldLa.getRange();
  var desiredRange = Math.max(20.0, this.currentSpeed * 10);
  if(this.lookAuto)
    this.currentLook = curHeading + this.getTurnToDirection_(curHeading, desiredHeading);
  //if(this.currentLook == -1)


  var la = this.ge.createLookAt('');
  la.set(loc.lat(), loc.lng(),
      0, // altitude
      this.ge.ALTITUDE_RELATIVE_TO_GROUND,
      this.currentLook,
      60, // tilt
```

```
      this.currentAltitude

      );
  this.ge.getView().setAbstractView(la);
}

// simulate per tick
DDSimulator.prototype.tick_ = function() {
  if (this.currPathIndex >= this.path.length - 1) {
    this.doTick_ = false;
    return;
  }

  // update current route step and run callback
  if (this.path[this.currPathIndex].step != this.currentStep_) {
    this.lookAuto = true;
    this.currentStep_ = this.path[this.currPathIndex].step;
    if (this.options.on_changeStep)
      this.options.on_changeStep(this.currentStep_);
  }
  // move up TICK_SIM_MS milliseconds
  this.totalTime += DDSimulator.TICK_SIM_MS * this.options.speed / 1000.0;
  this.segmentTime_ += DDSimulator.TICK_SIM_MS * this.options.speed / 1000.0;

  var segmentDuration = this.path[this.currPathIndex].duration;

  if (!this.beforeSegmentDistance_)
    this.beforeSegmentDistance_ = 0.0;

  // move to next segment if we pass it in this tick
  while (this.currPathIndex < this.path.length - 1 &&
    this.segmentTime_ >= segmentDuration) {
    this.segmentTime_ -= segmentDuration;

    // adjust distances
    this.beforeSegmentDistance_ += this.path[this.currPathIndex].distance;
    this.segmentDistance_ = 0;

    // update new position in path array
    this.currPathIndex++;
    segmentDuration = this.path[this.currPathIndex].duration;
  }
  if (segmentDuration) {
    this.segmentDistance_ = this.path[this.currPathIndex].distance *
                            Math.min(1.0, this.segmentTime_ / segmentDuration);
    this.currentSpeed = this.path[this.currPathIndex].distance / segmentDuration;
  } else {
    this.segmentDistance_ = 0.0;
    this.currentSpeed = 0.0;
  }
  this.totalDistance = this.beforeSegmentDistance_ + this.segmentDistance_;
  if (this.currPathIndex >= this.path.length - 1) {
    this.doTick_ = false;
    return;
  }
  // update the current location
  this.currentLoc = this.geHelpers_.interpolateLoc(
      this.path[this.currPathIndex].loc,
```

```
      this.path[this.currPathIndex + 1].loc,
      this.segmentTime_ / this.path[this.currPathIndex].duration);
 this.currentHeading = this.geHelpers_.getHeading(this.path[this.currPathIndex].loc,
                                  this.path[this.currPathIndex + 1].loc);
  this.drive_(this.currentLoc, this.currentHeading);


  // fire the callback if one is provided
  if (this.options.on_tick)
    this.options.on_tick();
}
```