ΥΠΟΛΟΓΙΣΤΙΚΗ ΓΕΩΜΕΤΡΙΑ

Απαλλακτική Εργασία εαρινού εξαμήνου 2018

Κίνηση Ρομπότ σε χώρο με εμπόδια

Γιαλίτσης Νικόλαος
1115201400027
Υπολογιστική Γεωμετρία 2018
καθ.Εμίρης Ιωάννης

Παραδοτέα

- **Source Code** : motion_planning.c
- **zip Αρχείο** με βίντεο demos εκτέλεσης διαφόρων περιπτώσεων κίνησης
- **παρουσίαση** power point
- **pdf** παρουσίασης

Μεταγλώττιση

1. cgal_create_CMakeLists -s robot
2. cmake .
3. make

Ενδεικτική Εκτέλεση

./robot -size 100 -obstacles 2 -sleep 3 -freedom 0

Περιγραφή Προβλήματος:

- Ν πολύγωνα που θεωρούνται εμπόδια.
- χρήση ρουτίνων CGAL για την αναπαράσταση του ελεύθερου χώρου.
- 2 δεδομένα σημεία στο επίπεδο
- το πρόγραμμά αποφασίζει αν είναι δυνατή η κίνηση από το ένα σημείο στο άλλο ή όχι δηλ. αν υπάρχει μονοπάτι που συνδέει τα 2 σημεία χωρίς να τέμνει το εσωτερικό των εμποδίων.

Μεθοδολογία

Για την Αναπαράσταση του ελεύθερου χώρου :

Υπολογίζω μια διαμέριση του ελεύθερου χώρο μέσω υποδιαίρεσης σε τρίγωνα με τριγωνοποίηση Delaunay , εισάγοντας τα εμπόδια και το εξωτερικό περίγραμμα ως περιορισμούς . Έχω πρόσβαση στο γράφο των τριγώνων μέσω της δομής της τριγωνοποίησης της CGAL.

```
typedef CGAL::Constrained_Delaunay_triangulation_2<K, TDS, Itag>  CDT;
```

```
Polygon_2 Bounding_box;
Bounding_box.push_back(Point(0,0));
Bounding_box.push_back(Point(box_size-1,0));
Bounding_box.push_back(Point(box_size-1,box_size-1));
Bounding_box.push_back(Point(0,box_size-1));

CDT cdt;
cdt.insert_constraint(Bounding_box.vertices_begin(),Bounding_box.vertices_end(),true);
```

Μεθοδολογία

Για την κίνηση ρομπότ:

Εντοπίζω το τρίγωνο της αρχικής και της τελικής θέσης του ρομπότ, αναζητώ μονοπάτι στον γράφο των τριγώνων που συνδέει την αρχική και τελική θέση. Για τον υπολογισμό του μονοπατιού πραγματοποιώ Depth First Αναζήτηση με τη χρήση στοίβας.

```
std::cout << "Locate starting point..." << std::endl;
CDT::Face_handle start_face = cdt.locate(StartPoint,cdt.all_faces_begin());
```

```
while(! stack.empty()){
    CDT::Face_handle fh = stack.back();
    stack.pop_back();
    if(!output.empty())path = output.back();
    path.push_back(fh);
    output.push_back(path);
```

Βελτιστοποίηση

Σε περίπτωση που υπάρχει μονοπάτι, υπολογίζεται εκείνο το μονοπάτι που μεγιστοποιεί την ελάχιστη απόσταση από τα εμπόδια,καθώς κατά τη διάρκεια της αναζήτησης αποθηκεύεται το άθροισμα των αποστάσεων των κέντρων των εμποδίων από την τρέχουσα θέση για κάθε μονοπάτι. Στο τέλος του υπολογισμού, επιστρέφεται το μονοπάτι με το μικρότερο άθροισμα αποστάσεων.

```
        Point obstacle_center = CGAL::centroid(obstacle_vertices.begin(),obstacle_vertices.end());
        current_dist += CGAL::squared_distance(t,obstacle_center);
    }
}
if(current_dist <= best_dist){
    best_dist = current_dist;
    best_path.clear();
    std::copy(final_path.begin(),final_path.end(),std::back_inserter(best_path));
}
```

Κατασκευή Εμποδίων

```
//Dimiourgise to polugono me ta tuxaia simeia
CGAL::random_polygon_2(4, std::back_inserter(Obstacle),
                       positive_point_set.begin());


assert(Obstacle.vertices_num() == 4);

//Elegxos prosanatolismou polugonou
if(Obstacle.orientation() == CGAL::CLOCKWISE)
    Obstacle.reverse_orientation();

if(Obstacle.orientation() != CGAL::COUNTERCLOCKWISE){
    cout << "Invalid orientation"<<std::endl;
    return -1;
}
//Elegxos kurtotitas empodiou
std::cout << "The polygon is " <<
(Obstacle.is_convex() ? "" : "not ") << "convex." << std::endl;
```

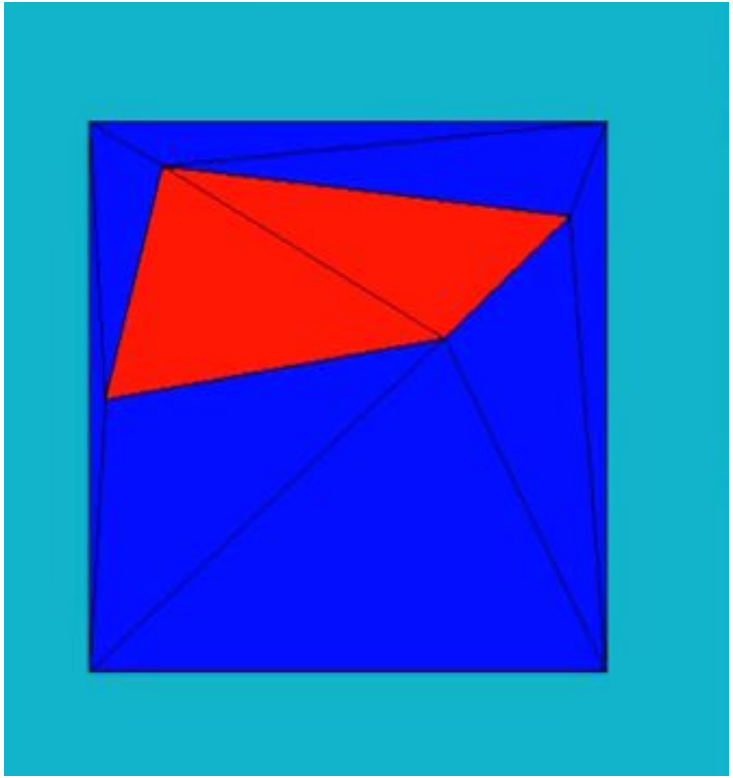
Διαχείριση Ελεύθερου Χώρου

```
struct FaceInfo2
{

  int nesting_level;
  bool visited;
  bool in_domain(){
    return nesting_level%2 == 1;
  }
    FaceInfo2(){
        visited = false;
     }
};
```

Για τον διαχωρισμό ελεύθερου χώρου και εμποδίων χρησιμοποιώ τη δομή FaceInfo2 και τη συνάρτηση mark_domains. Ξεκινώντας από το εξωτερικό του Bounding Box με nesting level 0 απαριθμώ όλα τα τρίγωνα που γειτονεύουν από το infinite_face της τριγωνοποίησης. Στη συνέχεια, με nesting level 1 αυτά που γειτονεύουν στα τρίγωνα με nesting level 0 και συνεχίζω με τον ίδιο τρόπο. Το ρομπότ θα μπορεί να κινηθεί μόνο στα τρίγωνα με nesting level 1 .

Δημιουργία διαγράμματος κίνησης

```
if(fh->info().nesting_level == -1){
  fh->info().nesting_level = index;
  //std::cout<<"INDEX "<<index<<std::endl;
  Triangle t = FaceToTriangle(fh);
  if(fh->info().nesting_level == 1){
    indomain << CGAL::BLUE;
    indomain << t;
  }
  else if(fh->info().nesting_level > 1){
    indomain << CGAL::RED;
    indomain << t;
  }
```

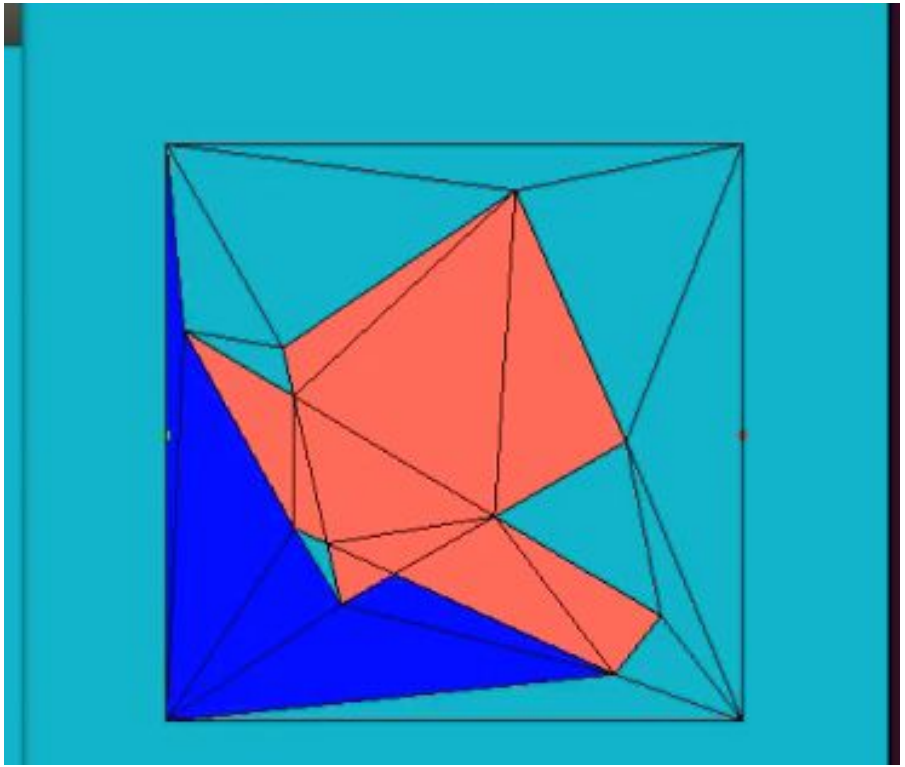


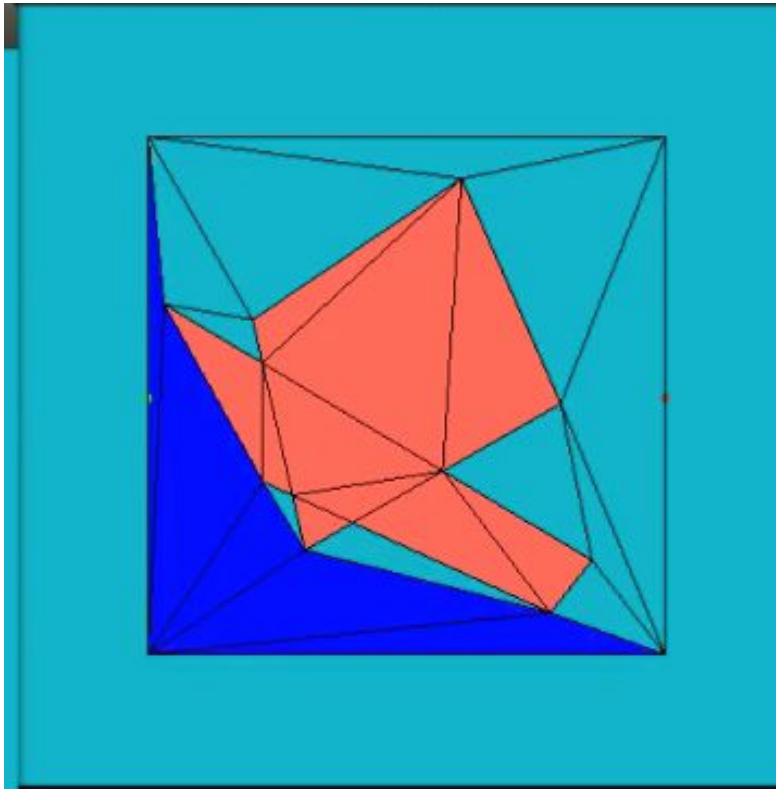
Επανάκαμψη από αδιέξοδα

Αδιέξοδο έχω όταν όλοι οι γείτονες του τρέχοντος τριγώνου είναι είτε εμπόδια είτε τα έχουμε ήδη επισκεφτεί σε προηγούμενο βήμα της αναζήτησης.

Τότε αφαιρούμε από το output , το οποίο είναι μια λίστα από υπομονοπάτια, τόσα στοιχεία όσα χρειάζονται(pop) έτσι ώστε το μονοπάτι να συμφωνεί με τη τρέχουσα στοίβα δηλαδή το καινούριο στοιχείο της στοίβας να προέρχεται από το τελευταίο τρίγωνο του μονοπατιού.

```
if(dead_end){
    std::cout<<"\tDead end\n"<<std::endl;
    output.pop_back();
    while(true){
        if(output.empty() || stack.empty())break;
        CDT::Face_handle  c = output.back().back();
        bool reset = true;
        for(int x = 0;  x < 3; x++){
            CDT::Face_handle n = c ->neighbor(x);
            if (n == stack.back()){
                reset = false;
                break;
            }



        }
        if(!reset)break;
        else {
            output.pop_back();
        }
    }
```





Οπτικοποίηση:

Για την οπτικοποίηση του περιβάλλοντος και της κίνησης του ρομπότ χρησιμοποιώ τη διεπαφή που προσφέρει η CGAL με το Geomview. Με το Geomview ορίζω ένα παράθυρο στο οποίο μπορώ να εισάγω γεωμετρικά σχήματα σχήματα και σχήματα.

```
CGAL::Geomview_stream gv(CGAL::Bbox_3(0,0, 0,box_size-1,box_size-1,0));
gv.clear();
gv.set_line_width(5);
gv.set_bg_color(CGAL::Color(0, 200, 200));
gv.set_wired(true);

gv.clear();
gv << CGAL::GREEN;
gv << StartPoint;
gv << CGAL::RED;
gv << EndPoint;

PaintObstacles(obstacles_vector,gv);
gv << ct;
gv << CGAL::YELLOW;
```

Παράθυρα geomview και Χρώματα

Κατά την εκτέλεση ανοίγουν δύο παράθυρα οπτικοποίησης geomview .

- Το πρώτο παράθυρο απεικονίζει με μπλέ τον ελεύθερο χώρο δηλαδή τον χώρο κίνησης του ρομπότ ενώ το κόκκινο τα εμπόδια. **Πρέπει να κλείσετε ή να μετακινήσετε χειροκίνητα το παράθυρο αυτό έτσι ώστε να παρακολουθήσετε το επόμενο παράθυρο που απεικονίζει τα βήματα κίνησης του ρομπότ.**
- Το δεύτερο παράθυρο απεικονίζει με μπλε το μονοπάτι στο οποίο βρίσκεται το ρομπότ και με κόκκινο τα εμπόδια. Όταν το μονοπάτι είναι μονοπάτι στόχου τότε χρωματίζεται με κίτρινο και το ρομπότ αναζητά το επόμενο μονοπάτι. Στο τέλος , χρωματίζεται με πορτοκαλί το καλύτερο μονοπάτι και φαίνεται με πράσινο η γραμμική τελική διαδρομή του ρομπότ.

Βασικές Δομές Αναζήτησης (1)

```
std::list<CDT::Face_handle> stack;
std::list<CDT::Face_handle> path;
```

stack: Στοίβα στην οποία για κάθε τρίγωνο εισάγω τους διαθέσιμους γείτονές του κατα Depth First πολιτική

path: Τα faces που βγαίνουν από τη στοίβα τοποθετούνται στο τέλος του path , το οποίο είναι στην ουσία μία ουρά τριγώνων.

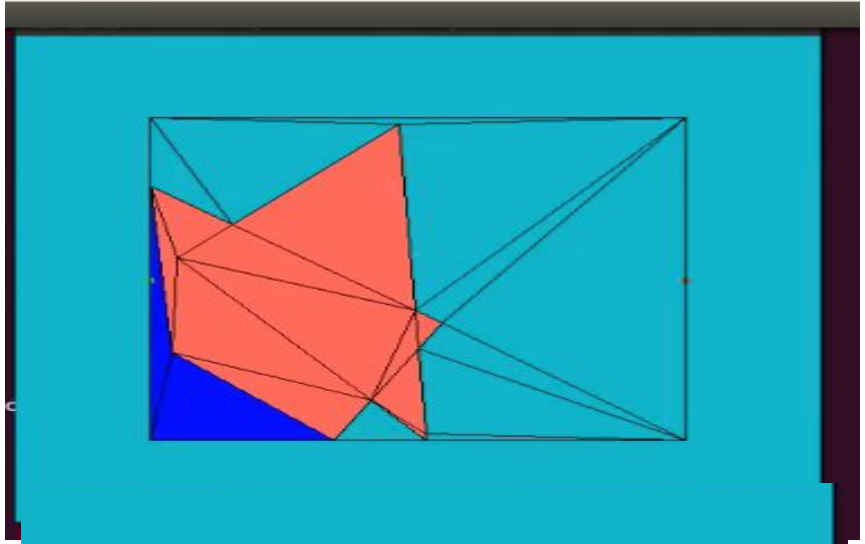
Βασικές Δομές Αναζήτησης (2)

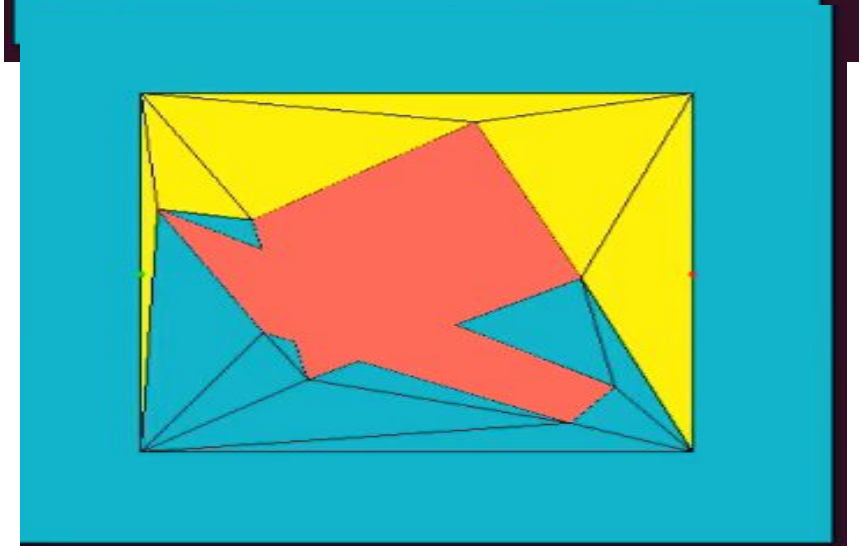
```
std::list<CDT::Face_handle> best_path;

std::list<std::list<CDT::Face_handle> > output;
bool path_exists = false;
```

Το **output** είναι μια λίστα από μονοπάτια , σε κάθε επανάληψη προστίθεται ένα νέο μονοπάτι το οποίο συντίθεται από το προηγούμενο συν το νέο face που γίνεται pop από τη στοίβα.

Το **best_path** είναι το μονοπάτι το οποίο οδηγεί στον στόχο αλλά και δίνει την μικρότερη απόσταση από τα εμπόδια.

Η μεταβλητή **path_exists** δηλώνει αν βρέθηκε ή όχι μονοπάτι κατά την αναζήτηση





```
Dead end

NO VIABLE PATH FOUND
Enter a key to finish
```

```
Push
PATH FOUND
```

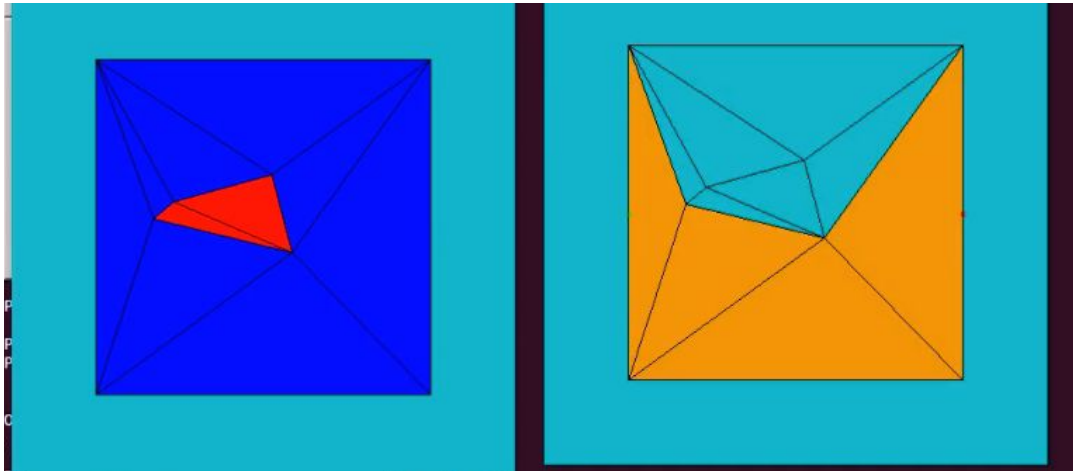
Δημιουργία Ευθύγραμμου Μονοπατιού Ρομπότ

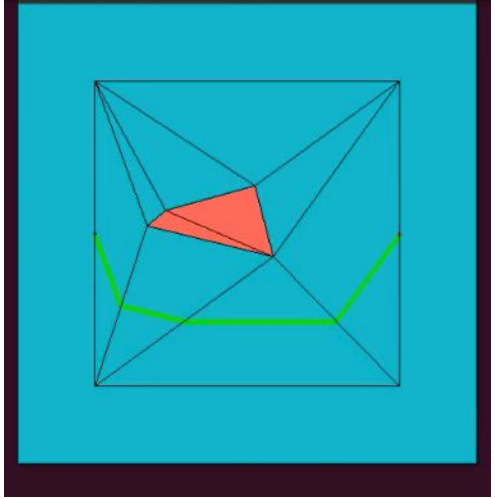
Αφού έχω υπολογίσει , αν υπάρχει , το καλύτερο μονοπάτι τριγώνων δημιουργώ το τελικό γραμμικό μονοπάτι που θα πρέπει να ακολουθήσει το ρομπότ ενώνοντας τα μέσα των πλευρών των γειτονικών τριγώνων.

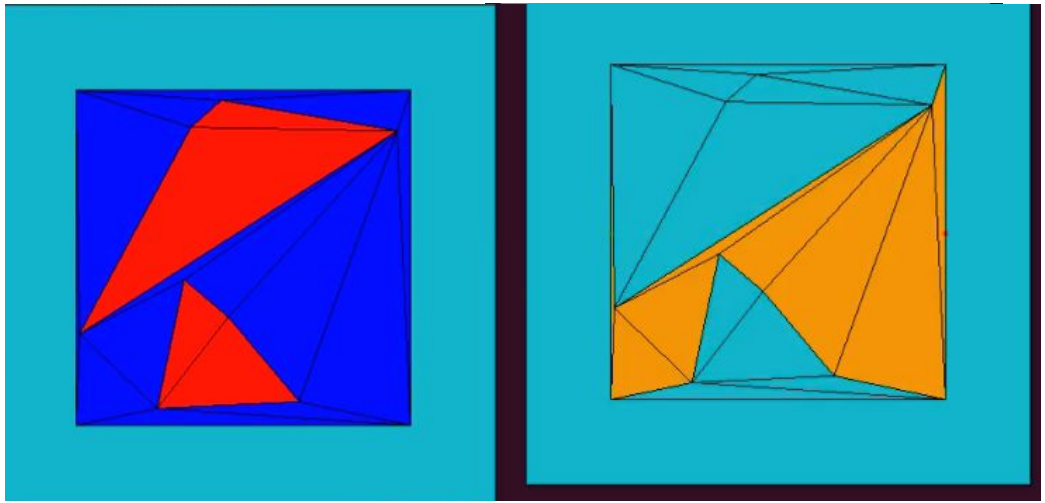
```
Triangle next_t = FaceToTriangle(*it);
it--;


CGAL::cpp11::result_of<Intersect_2(Triangle,Triangle)>::type result = CGAL::intersection(t,next_t);
if(!result){
    break;
    cout<<"No intersection"<<endl;
}
```

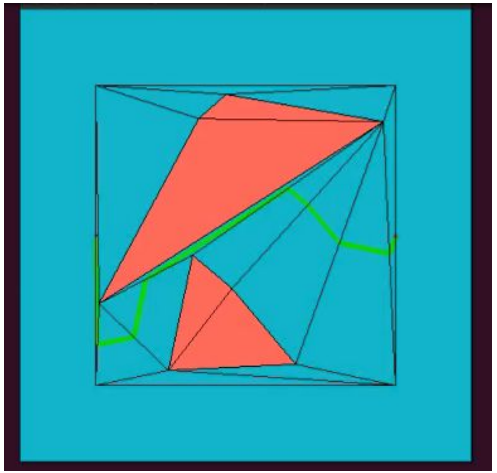
```
if(!result){
    break;
    cout<<"No intersection"<<endl;
}

std::vector<Point>* lv = boost::get<std::vector<Point> >(&*result);
Point* lp = boost::get<Point>(&*result);
Segment* ls = boost::get<Segment>(&*result);
Point midpoint;
if(lv)
    midpoint = CGAL::midpoint(lv->front(),lv->back());
else if(lp){
    midpoint = *lp;
}
else if(ls){
    midpoint = CGAL::midpoint(ls->source(),ls->target());
}
else
    cout<<"Unexpected return value"<<endl;

Segment connect(last_point,midpoint);
gv << CGAL::GREEN;
gv << connect;
last_point = midpoint;
```

Η τομή δύο τριγώνων μπορεί να είναι τρίγωνο,σημείο, ευθύγραμμο τμήμα ή σύνολο σημειών. Αντιμετωπίζω την κάθε περίπτωση ξεχωριστά









Ορίσματα Εκτέλεσης

- **size**: μέγεθος bounding box - τετράγωνο size*size
- **obstacles** : αριθμός εμποδίων
- **sleep_time** : χρόνος καθυστέρησης μεταξύ των βημάτων της αναζήτησης μονοπατιού
- **freedom** : επειδή η περίπτωση τα εμπόδια να εμποδίζουν το πέρασμα του ρομπότ εντελώς είναι αρκετά σπάνια, αυτή η μεταβλητή ωθεί εμπόδια που δεν θα ακουμπούσαν το bounding box και απέχουν freedom απόσταση από αυτό, πάνω του,έτσι ώστε να μην υπάρχει κενό ενδιάμεσα.

Ευχαριστώ για την προσοχή σας