Elaborazione di un point cloud e triangolazione con algoritmo Greedy Projection Triangulation

Davide Pala

Anno Accademico 2014/2015

POLITECNICO
DI TORINO

Indice

1 Introduzione

1.1 Team DIANA

Il team Diana è un gruppo studentesco del Politecnico di Torino che si occupa di robotica spaziale, nato per rappresentare l'ateneo all'interno del progetto AMALIA[1]. L'attività del team è dedita principalmente allo sviluppo di un rover lunare in tutte le sue funzionalità. Il rover è dotato, allo stato attuale, di un sistema di acquisizione di immagini composto da due camere Black-and-White di Point Gray[2] e da un sensore Mesa SR4500 ToF camera[3]. Questa strumentazione permette al sistema di effettuare delle riprese in cui vengono acquisiti dei Point Cloud, insiemi di punti dello spazio tridimensionale che descrivono la scena ripresa.

1.2 Obiettivi e descrizione del progetto

Lo scopo del programma è generare una mesh poligonale effettuando la triangolazione di un point cloud. Il programma acquisice il nome del file contenente il point cloud e il nome del file di output da linea di comando, i punti vengono letti dal file e processati da una serie di algoritmi di filtraggio. Successivamente viene applicato lo smoothing delle superfici, durante il quale viene effettuato anche il calcolo delle normali, in ultimo l'algoritmo di ricostruzione dei triangoli elabora il point cloud, la mesh così generata viene quindi scritta sul file di output. Il software è scritto in C++, per lo sviluppo è stato utilizzato l'IDE Qt Creator[4], mentre per le strutture dati e gli algoritmi per il processing del point cloud è stata utilizzata la libreria pcl[5].

[1] http://www.amalia-teamitalia.it/
[2] http://team-diana.github.io/en/#!pages/blackfly_bw_poe_gige_hardware.md
[3] http://team-diana.github.io/en/#!pages/sr4500.md
[4] http://wiki.qt.io/Qt_Creator
[5] http://pointclouds.org/about/

2 Il programma

2.1 Approccio Object Oriented

Il programma è stato sviluppato seguendo l'approccio della programmazione orientata agli oggetti: una facade class, PointCloudTriangulation, espone al main le funzioni per la lettura e l'elaborazione del point cloud e per la gestione dei filtri, che sono immagazinati internamente con una mappa che associa una stringa, il nome del filtro, ad un puntatore generico a PointCloudFilter. La classe espone anche le funzioni per effettuare la triangolazione e per salvarne il risultato su file, sono inoltre definiti i valori di default dei parametri della triangolazione.

```
1  #define PARAM_SEARCH_RADIUS 0
2  #define PARAM_MULTIPLIER 1
3  #define PARAM_MAX_NEAREST_NEIGHBORS 2
4  #define PARAM_MAX_SURFACE_ANGLE 3
5  #define PARAM_MIN_ANGLE 4
6  #define PARAM_MAX_ANGLE 5
7  #define PARAM_NORMAL_CONSISTENCY 6
8
9  #define DEFAULT_SEARCH_RADIUS 0.025
10 #define DEFAULT_MULTIPLIER 2.5
11 #define DEFAULT_MAX_NEAREST_NEIGHBORS 100
12 #define DEFAULT_MAX_SURFACE_ANGLE M_PI/4
13 #define DEFAULT_MIN_ANGLE M_PI/18  //10
14 #define DEFAULT_MAX_ANGLE 2*M_PI/3 //120
15 #define DEFAULT_NORMAL_CONSISTENCY true
16
17 class PointCloudTriangulation
18 {
19 private:
20     pcl::PointCloud<pcl::PointXYZ>::Ptr cloud;
21     pcl::PointCloud<pcl::PointNormal>::Ptr normalCloud;
22     pcl::GreedyProjectionTriangulation<pcl::PointNormal> gp3;
23     pcl::PolygonMesh triangles;
24     std::map<std::string, PointCloudFilter* > algorithms;
25
26 public:
27     PointCloudTriangulation(pcl::PointCloud<pcl::PointNormal>::Ptr);
28     PointCloudTriangulation(std::string);
29     PointCloudTriangulation();
30
31     void addFilter(std::string, PointCloudFilter*);
32     void applyFilter(std::string);
33     void loadCloudFromFile(std::string);
34     void surfaceSmoothing();
35     void reconstruct(); //apply greedy projection triangulation
36     void saveTriangulation(std::string);
37     PointCloudFilter* getFilter(std::string);
38     pcl::PointCloud<pcl::PointNormal>::Ptr getCloud();
39     pcl::PolygonMesh getTriangulation();
40 };
```

Figura 1: la classe PointCloudTriangulation

PointCloudFilter è una classe puramente virtuale realizzata come interfaccia per la gestione dei filtri, dichiara soltanto 3 metodi utili per applicare l'algoritmo e settarne o leggerne i parametri. I parametri sono identificati da delle costanti numeriche e sono trattati come dei double. L'effettiva implementazione dei filtri è realizzata con 2 sottoclassi di PointCloudFilter: PCVoxelGridFilter e PCStatisticalOutlierRemoval.

```
1 class PointCloudFilter
2 {
3 public:
4     virtual void apply(pcl::PointCloud<pcl::PointNormal>::Ptr) = 0;
5     virtual void setParameter(int, double) = 0;
6     virtual double getParameter(int) = 0;
7 };
```

Figura 2: la classe PointCloudFilter

2.2 Il main

Grazie all'utilizzo della facade class e il main del programma risulta particolarmente conciso. Dopo il controllo degli argomenti da linea di comando, viene allocata un'istanza della facade class, triangulation, quindi si passa alla lettura del file contenente il point cloud. Due puntatori generici a PointCloudFilter vengono allocati come oggetti di tipo PCVoxelGridFilter e PCStatisticalOutlierRemoval e aggiunti a all'oggetto triangulation assegnandogli un identificatore di tipo string. I due filtri sono quindi applicati sul point cloud e, dopo il surface smoothing, $triangulation \rightarrow reconstruct()$ genera la mesh poligonale, che viene quindi salvata sul file di output.

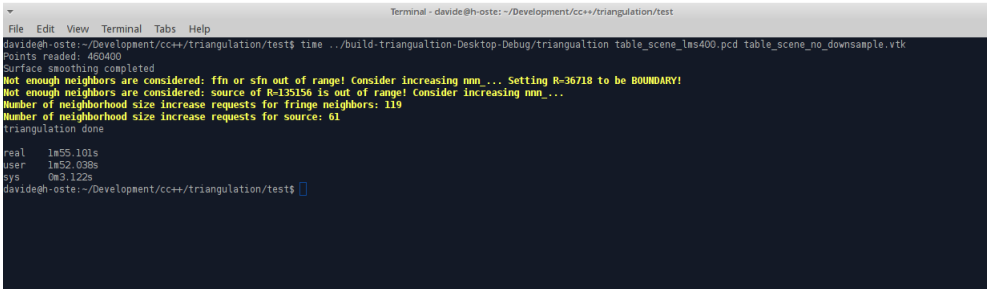
```
1  int main(int argc,char **argv)
2  {
3      help(argc, argv); // check command line args
4
5      PointCloudTriangulation *triangulation = new PointCloudTriangulation();
6
7      triangulation->loadCloudFromFile(argv[1]); // read cloud from pcd file
8
9      PointCloudFilter* voxel_grid_filter = new PCVoxelGridFilter();
10     PointCloudFilter* gaussian_noise_filter = new PCStatisticalOutlierRemoval();
11
12     triangulation->addFilter("voxel grid filter", voxel_grid_filter);
13     triangulation->addFilter("gaussian noise filter", gaussian_noise_filter);
14
15     triangulation->applyFilter("voxel grid filter");
16     triangulation->applyFilter("gaussian noise filter");
17
18     triangulation->surfaceSmoothing();
19     triangulation->reconstruct();
20
21     triangulation->saveTriangulation(argv[2]);
22
23     return 0;
24 }
```

Figura 3: il main del programmga

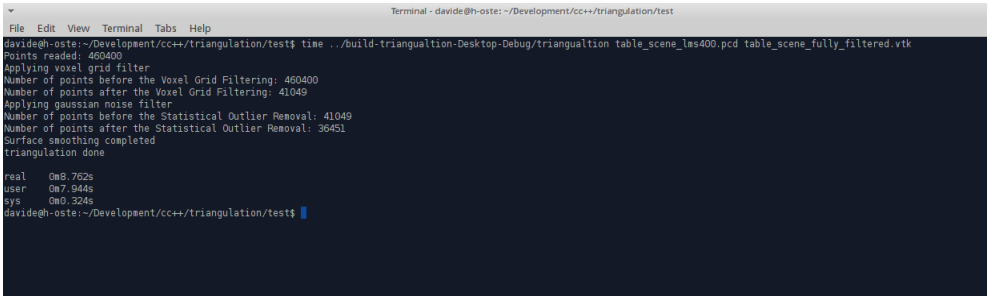
2.3 I filtri

2.3.1 Voxel Grid Filter

Il tempo di calcolo è stato il primo problema incontrato durante lo sviluppo; infatti il tempo necessario per l'esecuzione di surface smoothin e triangolazione su un point cloud di 460400 punti è di circa 2 minuti, mentre applicando tutte le elaborazioni dopo il downsample, il tempo di esecuzione è sceso al di sotto dei 10 secondi, spesi perlopiù nella fase di lettura del file.



(a)



(b)

Figura 4: confronto fra i tempi di esecuzione con e senza downsample

Per realizzare il downsample del point cloud è stato utilizzato l'algoritmo di voxel grid filtering. Questo algoritmo funziona suddividendo lo spazio in una griglia di voxel, piccole unità di volume, tutti i punti all'interno di un voxel vengono approssimati dal loro baricentro. Il filtro ha tre parametri fondamentali che sono la lunghezza degli spigoli del voxel lungo i 3 assi cartesiani.

```
#define DEFAULT_LEAF_SIZE 0.01f
#define PARAM_LEAF_SIZE_X 0 //"leaf size x"
#define PARAM_LEAF_SIZE_Y 1 //"leaf size y"
#define PARAM_LEAF_SIZE_Z 2 //"leaf size z"

class PCVoxelGridFilter : public PointCloudFilter
{
public:
    PCVoxelGridFilter();
    void apply(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud);
    //void setParameter(int, std::string);
    void setParameter(int, double);
    double getParameter(int);
    bool isValidParam(int);
private:
    pcl::VoxelGrid< pcl::PointXYZ> voxel_grid_filter;
    std::map<int, double> params;
};
```

Figura 5: la classe PCVoxelGridFilter

La classe che realizza il filtro è PCVoxelGridFilter ed implementa l'interfaccia PointCloudFilter, il metodo *apply*() imposta i parametri ed applica il filtro, come dimensione di default per i lati del voxel è stata scelta 0.01, che dovrebbe corrispondere a cubi di spigolo un centimetro, con questa configurazione il numero di punti del point cloud di test viene ridotto da 460400 a 41049, un fattore di circa 11.

```
void PCVoxelGridFilter::apply(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud)
{
    std::cout<<"Number of points before the Voxel Grid Filtering: "<<cloud->width*cloud
        ->height<<std::endl;
    voxel_grid_filter.setInputCloud(cloud);
    voxel_grid_filter.setLeafSize((float)params[PARAM_LEAF_SIZE_X],
                                  (float)params[PARAM_LEAF_SIZE_Y],
                                  (float)params[PARAM_LEAF_SIZE_Z]);
    voxel_grid_filter.filter(*cloud);
    std::cout<<"Number of points after the Voxel Grid Filtering: "<<cloud->width*cloud->
        height<<std::endl;
}
```

Figura 6: il metodo *apply*() della classe PCVoxelGridFilter

2.3.2 Statistical Outlier Removal

Il secondo stadio di filtraggio si occupa di attenuare e rimuovere il rumore del cloud, infatti, durante la cattura, i sensori rilevano punti errati, spesso a causa di riflessi o errori di misura, questi errori complicano poi le elaborazioni successive e rischiano di condurre a risultati sbagliati, per esempio nel calcolo delle normali.

Anche in questo caso la classe che implementa il filtro, PCStatisticalOutlierRemoval, è una sottoclasse di PointCloudFilter e implementa quindi i soliti metodi *getParameter*(), *setParameter*() ed *apply*().

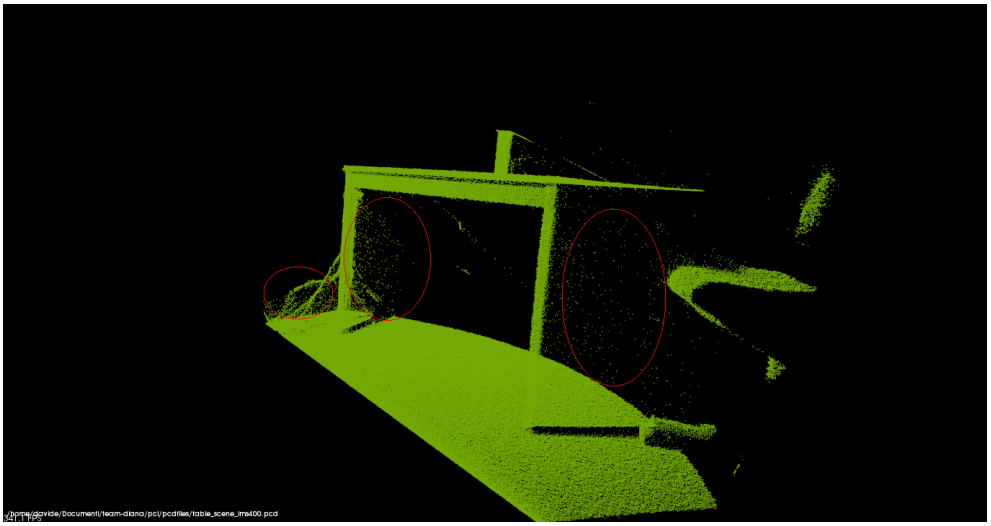
```
1  //number of point to use for mean distance estimation
2  #define PARAM_MEAN_K 0 //"mean k"
3  #define PARAM_STD_DEV_MUL_TH 1 //"standard deviation multipilier threashold"
4
5  // default values
6  #define DEFAULT_MEAN_K 50
7  #define DEFAULT_STD_DEV_MUL_TH 0.1
8
9  class PCStatisticalOutlierRemoval : public PointCloudFilter
10 {
11 public:
12
13     PCStatisticalOutlierRemoval();
14     void apply(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud);
15     void setParameter(int, double);
16     double getParameter(int);
17
18 private:
19     std::map<int, double> parameters;
20     pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor;
21
22 };
```

Figura 7: la classe PCStatisticalOutlierRemoval

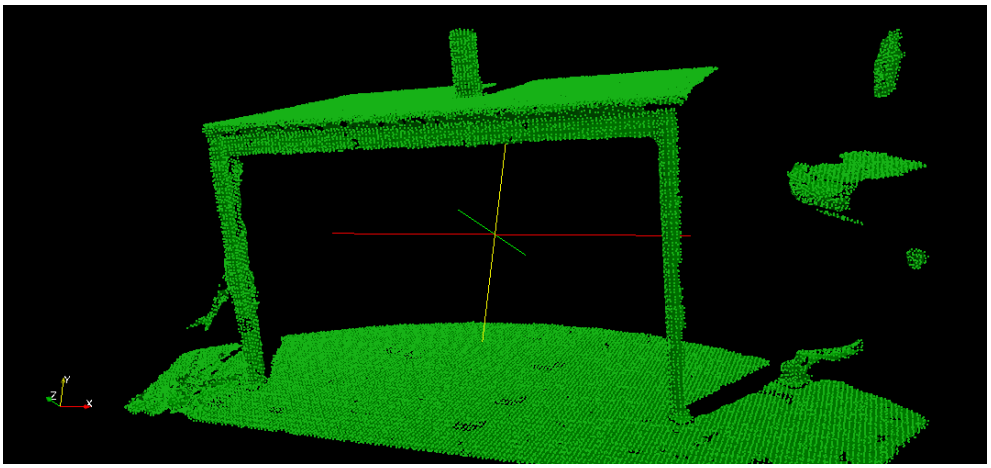
Il principio di funzionamento dell'algoritmo si basa sul valutare punto per punto la distanza media da tutti i suoi n vicini, la distribuzione risultante viene considerata come una gaussiana con una media e una deviazione standard σ, tutti i punti appartenenti al vicinato la cui distanza supera il valore $m \times \sigma$ sono considerati outlier e rimossi dall'insieme. I parametri del filtro sono il numero n di vicini da considerare e il moltiplicatore m della deviazione standard, i valori scelti come default sono $n = 50$ e $m = 1.0$, il filtro rimuove quindi tutti i punti la cui distanza supera un σ.

```
1 void PCStatisticalOutlierRemoval::apply(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud){
2     std::cout<<"Number of points before the Statistical Outlier Removal: "<<cloud->width
          *cloud->height<<std::endl;
3     sor.setInputCloud (cloud);
4     sor.setMeanK ((int)parameters[PARAM_MEAN_K]);
5     sor.setStddevMulThresh ((float)parameters[PARAM_STD_DEV_MUL_TH]);
6     sor.filter (*cloud);
7     std::cout<<"Number of points after the Statistical Outlier Removal: "<<cloud->width*
          cloud->height<<std::endl;
8 }
```

Figura 8: il metodo *apply*() della classe PCStatisticalOutlierRemoval



(a)



(b)

Figura 9: il point cloud non filtrato (a) e la mesh (b)

2.4 Surface Smoothing

L'algoritmo di surface smoothing mooving least squares si occupa di fare una ricostruzione delle superfici basata su un resample del cloud e sul'iterpolazione polinomiale, questa elaborazione dovrebbe contibuire a ridurre il rumore e i piccoli difetti del data set, inoltre l'algoritmo di triangolazione ha un risultato migliore se effettuato su superfici regolari. Il parametro fondamentale di questo algoritmo è il raggio di ricerca, il raggio della sfera usata per cercare i vicini su cui effettuare il fit polinomiale, per i restanti parametri è stato lasciato il valore di default specificato dalla libreria.

```
1  void PointCloudTriangulation::surfaceSmoothing(){
2      pcl::search::KdTree<pcl::PointXYZ>::Ptr tree(new pcl::search::KdTree<pcl::PointXYZ
           >());
3      pcl::MovingLeastSquares<pcl::PointXYZ, pcl::PointNormal> mls;
4
5      mls.setComputeNormals(true);
6
7      mls.setInputCloud(cloud);
8      mls.setPolynomialFit(true);
9      mls.setSearchMethod(tree);
10     mls.setSearchRadius(0.03);
11
12     mls.process(*normalCloud);
13     std::cout<<"Surface smoothing compleated"<<std::endl;
14 }
```

Figura 10: $surfaceSmoothing()$ nella classe PointCloudTriangulation

2.4.1 Normal Estimation

L'algoritmo di surface smoothing si occupa anche di effettuare il calcolo delle normali, nelle fasi iniziali dello sviluppo per il surface smoothing era stato utilizzato lo stesso approccio adottato con i filtri, le normali venivano quindi calcolate immediatamente dopo la lettura dei dati dal file, questo per avere una coerenza sui tipi di dato su cui operano i filtri, ciò comportava un tempo di elaborazione più lungo e una maggiore complessita del codice che doveva gestire anche i parametri dell'algoritmo di normal estimation.

2.5 Greedy Projection Triangulation

Il core del programma consiste nella generazione una mesh poligonale a partire da un point cloud con le normali. Per lo scopo viene utilizzato l'algoritmo di greedy projection triangulation, questo metodo si basa sulla triangolazione locale dei vicini di un punto, che vengono proiettati lungo la normale del punto stesso, i punti non ancora connessi vengono collegati. Il metodo ha numerosi parametri: maximum nearest neighbors specifica il numero massimo di vicini, il multiplier specifica la massima distanza per la quale un punto può essere considerato un vicino, relativa al punto a distanza minima, il search radius è il raggio di ricerca per i vicini e limita la dimensione massima dei lati dei triangoli, minimum angle e maximum angle definiscono il massimo e il minimo valore per gli angoli all'interno dei triangoli, maximum surface angle specifica il massimo angolo di deviazione tra la normale di un punto e quella di un suo vicino affinché questo possa essere considerato per la triangolazione.

```
1  void PointCloudTriangulation::reconstruct(){
2      // Create search tree*
3      pcl::search::KdTree<pcl::PointNormal>::Ptr tree (new pcl::search::KdTree<pcl::
           PointNormal>);
4      tree->setInputCloud (normalCloud);
5
6      // Set the maximum distance between connected points (maximum edge length)
7      gp3.setSearchRadius (DEFAULT_SEARCH_RADIUS);
8
9      // Set typical values for the parameters
10     gp3.setMu (DEFAULT_MULTIPLIER); // 2.5
11     gp3.setMaximumNearestNeighbors (DEFAULT_MAX_NEAREST_NEIGHBORS); // 100
12     gp3.setMaximumSurfaceAngle(DEFAULT_MAX_SURFACE_ANGLE); // 45 degrees
13     gp3.setMinimumAngle(DEFAULT_MIN_ANGLE); // 10 degrees
14     gp3.setMaximumAngle(DEFAULT_MAX_ANGLE); // 120 degrees
15     gp3.setNormalConsistency(true);
16
17     // Get result
18     gp3.setInputCloud (normalCloud);
19     gp3.setSearchMethod (tree);
20     gp3.reconstruct (triangles);
21     std::cout<<"triangulation done"<<std::endl;
22 }
```

Figura 11: il metodo *reconstruct*() della classe PointCloudTriangulation

3 Considerazioni e Sviluppi futuri

Questo lavoro ha richiesto molto tempo, sopratutto per la ricerca e la comprensione degli algoritmi e dei loro parametri, tuttavia non è da intendersi come un progetto concluso: sono possibili numerosi sviluppi e modifiche e il software è in continuo aggiornamento, è inoltre necessario efettuare dei test per affinare i valori dei parametri e per adattarli ai dati acquisiti dal rover.

3.1 Sviluppi futuri

La prima necessità che emerge dall'utilizzo del programma è quella di una più facile gestione dei parametri, infatti, seppure l'approccio object oriented ne facilita l'utilizzo, la configurazione dei valori è statica e bisogna ricompilare il programma ogni volta che si effettua una modifica, per questo motivo una configurazione con lettura da file sembra la via più efficace. Per un eventuale file di configurazione stiamo valutando, al momento, due formati: xml e YALM[6].

Il passo successivo dello sviluppo sarà quello dell'integrazione con ROS[7] e della creazione di un nodo in grado di ricevere i point cloud rilevati dal rover ed elaborarli, enventualmente in tempo reale, con la possibilità di effettuare tutta la catena per la triangolazione, o magari di effettuare solo un live filtering in modo da pulire i dati per altre applicazioni.

3.2 Conclusioni

Questo lavoro, e in generale tutta l'esperienza con il team DIANA, mi ha permesso di esplorare, anche se superficialmente, il mondo della computer vision, ho acquisito inoltre una maggiore familiarità con il linguaggio C++, con le librerie PCL e tutti gli strumenti di sviluppo, è stato anche un'ottima occasione per poter applicare le conoscenze apprese durante i corsi seguiti in questi anni per la soluzione di problemi pratici.

[6] http://yaml.org/
[7] http://www.ros.org/