Project: "Recognition of digital plane"

Introduction

The objective of this project is to implement a linear programming based algorithm for the recognition of digital planes.

We expect from you:

- A short report with answers to the "formal" questions and a description of your implementation choices and results.
- A C++ project (`CMakeLists.txt` plus several **commented** cpp program files).

1 Digital plane

In the sequel, let us consider the following definition: a digital set $Z \subset \mathbb{Z}^3$ is a *naive digital plane* if and only if there exists a normal vector $N(a, b, c) \in \mathbb{Z}^3$ and a bound $\mu \in \mathbb{Z}$ such that:

$$\forall z \in Z,\ \mu \leq N \cdot z < \mu + \max\left(|a|, |b|, |c|\right) \tag{1}$$

(where $\cdot$ denote the scalar product).

We assume now that $0 \leq a \leq b < c$.

Question 1 *Show that in such case, any naive digital plane P is functional, i.e. for each pair $(z_1, z_2) \in \mathbb{Z}^2$, there is one and only one point z of coordinates (z_1, z_2, z_3) that belongs to P.*

Question 2 *Show that (1) implies that:*

$$\forall z \in Z, \begin{cases} N \cdot z \leq \mu \\ N \cdot (z + (0, 0, 1)) \geq \mu \end{cases} \tag{2}$$

Even if the converse is not true due to the large inequalities, we say that a digital set Z is a digital plane if the inequality set (2) is verified.

Question 3 *There exists a unique (Euclidean) plane passing by three digital points. Show that we can test whether another digital point lies BELOW, ON or ABOVE such a plane with integer-only computations and without explicitly computing its center and radius. You may have a look at "orientation test" or "which-side test", broadly used in computational geometry.*

Question 4 *Implement a function that checks whether two given digital sets are separated by a given plane passing by three digital points or not.*

2 Linear programming

Let us now consider algorithm 1 (which uses routine 2). It is a randomized and recursive algorithm that checks whether two point sets are separable by a plane in expected linear-time (see [Sei91] or [dBCvKO00][chapter4]).

The union of the bottom point set, denoted by S^-, and the top point set, denoted by S^+, is merely denoted by S. All the points of S are numbered from 1 to $|S|$, the size of S. The idea consists in maintaining a separating plane while iterating over the points $s_i \in S$ for i from 1 to $|S|$. For each point s_i, three cases may occur:

- if it belongs to S^- (resp. S^+) and it is located BELOW (resp. ABOVE) or ON the current separating plane, there is nothing to do.
- Otherwise (lines 6-9 of algorithm 2):
  1. Either the two input sets are not separable by a plane at all,
  2. or there exists a separating plane passing by s_i.

In the aim of deciding between these last two alternatives, the set of possible separating planes is restricted to planes passing by s_i and the same algorithm is recursively called from 1 to i (line 9 of algorithm 2). At each recursive call, the set of possible separating planes is restricted so that the base case involves a unique plane passing by three given points and consists in checking whether it separates S^- from S^+ or not (lines 11-17 of algorithm 2).

Algorithm 1: isDigitalPlane(Z, p_1, p_2, p_3)

Input: $Z \subset \mathbb{Z}^3$, the digital set
$p_1, p_2, p_3 \in \mathbb{Z}^2$, three points characterizing a plane
Result: “true” if Z is a digital plane, “false” otherwise
Output: p_1, p_2, p_3, three points caracterizing a separating plane if “true”
`// initialisation step`
Construct the set $S^- = Z$ and the set S^+ a copy of Z translated by $(0, 0, 1)$;
Construct the set of $S = S^- \cup S^+$;
Randomly permute the points of S ;
`// points of` S `are numbered from 1 to` $|S|$`,` $|S|$ `is the size of the set`
Initialize p_1, p_2, p_3 with three points of S ; `// we assume here that` $|S| > 3$
`// recursive step`
return areLinearlySeparable(S^-, S^+, S, $|S|$, p_1, p_2, p_3, 3) ;

Question 5 *Implement algorithms 1 and 2. Provide test files.*

3 Experiments

Question 6 *Perform a running time analysis of your recognition function.*

- *Implement a function that constructs a rectangular piece of side s of a naive digital plane.*
- *Output the running time of your recognition function for pieces of increasing size.*
- *Plot the graph of the running times against the size of the pieces.*
- *Do you observe the expected linear-time complexity ?*

4 Extra works

Question 7 *Modify your recognition procedure in order to have an on-line algorithm, which takes input points one by one and updates the current separating plane on the fly. What is the time complexity of your procedure ?*

Question 8 *Use your on-line procedure to partition a digital surface into pieces of digital planes.*

Algorithm 2: areLinearlySeparable(S^-, S^+, S, n, p_1, p_2, p_3, k)

Input: $S^-, S^+ \subset \mathbb{Z}^2$, the bottom and top point sets, $S = S^- \cup S^+$
n, number of points of S to process ($1 \leq n \leq |S|$)
$p_1, p_2, p_3 \in \mathbb{Z}^2$, three points characterizing a plane
k, number of variable points among $\{p_1, p_2, p_3\}$ ($0 \leq k \leq 3$)
Result: "true" if S^- and S^+ are separable by a plane, "false" otherwise
Output: p_1, p_2, p_3, three points caracterizing a separating plane if "true"

areSeparable $\leftarrow$ TRUE ;
if $k > 0$ **then**
  for l from 1 to k, initialize p_l with a point of S ;
  $i \leftarrow 1$;
  while areSeparable *and* $i < n$ **do**
    if *(*$s_i \in S^-$ *and* s_i *is strictly ABOVE the plane passing by* p_1, p_2, p_3*)*
    or ($s_i \in S^+$ *and* s_i *is strictly BELOW the plane passing by* p_1, p_2, p_3*)* **then**
      $p_k \leftarrow s_i$;
      areSeparable $\leftarrow$ areLinearlySeparable(S^-, S^+, i, p_1, p_2, p_3, $k-1$) ;
    $i \leftarrow i + 1$;
else
  $i \leftarrow 1$;
  while areSeparable *and* $i < n$ **do**
    if *(*$s_i \in S^-$ *and* s_i *is strictly ABOVE the plane passing by* p_1, p_2, p_3*)*
    or ($s_i \in S^+$ *and* s_i *is strictly BELOW the plane passing by* p_1, p_2, p_3*)* **then**
      areSeparable $\leftarrow$ FALSE ;
    $i \leftarrow i + 1$;
return areSeparable ;

References

[dBCvKO00] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: algorithms and applications.* Springer, 2000.

[Sei91] Raimund Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6(1):423–434, 1991.