Linear Programming Concept Visualization

[*]Jean Pierre Charalambos and Ebroul Izquierdo [**]

[*] *Image Processing and Computer Graphics Research Lab, Dept. of Computer Sciences, National University, Colombia*

[**] *Department of Electronic Engineering Queen Mary, University of London London E1 4NS, United Kingdom*

Abstract

A visualization scheme as a tool to find the solution of any 3D-linear programming problem is introduced. The presented approach is highly suitable to draw and visualize interactively the feasible region of a given 3D-linear programming problem. It can be used for a better understanding of the solution process when different methods devoted to solve the underlying problem are applied, e.g., the simplex method. The proposed technique comprises sensitive analysis during the solution process and interactive visualization of the feasible region. The analysis leading to the introduced schema also shows that the design of an appropriate and simple vertex representation is crucial to manage any order of degeneracy. To deal with this paradigm and to some extent to formalize it, the concept of adjacency invariance is introduced. Several experiments have been conducted to test and assess the performance of the introduced concepts and techniques.

1. Introduction

Whenever we try to communicate ideas or to find the solution to problems arising from science and technology a major challenge has to be tackled: to find the most appropriate way to describe and express either the solution process or the solution itself. For instance, using conventional oral communication varying emphasis may be put on relevant words to stress key concepts or to give a better image of the idea we want to put across. Nevertheless, when solving complex mathematical problems there are concepts that cannot be easily expressed in words or demonstrated using simple formulas. In those cases images and graphics are much better to present and describe the underlying concepts. This fact provided the origin of the popular sentence "a picture is worth a thousand words". The rapid development of, and increasing accessibility to, computer graphics programs and their inherent visualization tools allow us to express and present highly complex problems and solution schemes in a simple manner.

In this paper we are interested in the visualization of a specific class of problems and their solution process: linear programming. In this case it has been established empirically that both the visualization of the feasible region and the path followed by the algorithm used to find an optimal vertex are essential to understand the solution process and the solution to the problem itself. In Fig. 1 a feasible region of a 3D linear programming task is outlined. In this context and due to the hierarchical nature of algorithmic solutions to linear programming problems, visualization is extremely important as a tool to help researchers develop and test their techniques.

A visualization scheme that is highly suitable for interactively drawing and visualizing the feasible region of a given 3D-linear programming task is introduced in this paper. It can be used for the better understanding of the solution process when different methods devoted to solving the underlying problem are applied, e.g., the simplex method. The analysis leading to the presented schema comprises sensitive analysis during the solution process, interactive visualization of the feasible region, and experimental testing of intermediate results. It is shown that the design of an appropriate and simple vertex representation is crucial to manage any order of degeneracy. To deal with this paradigm and to some extent to formalize it, the concept of *adjacency invariance* is also introduced. The envisaged approach focuses on the development of an interactive and highly accessible visualization technique as a helping tool for the better understanding of 3D-linear programming problems. If the feasible region of a 3D-linear programming problem is completely bounded, it is given by a convex polyhedra. Thus, visualization of convex polyhedra intersection and the feasible region of a 3D-linear programming problem can be seen as equivalent tasks. Although several works dealing with highly efficient methods for finding convex polyhedra intersection can be found in the literature, most of them describe either analytic solutions or they are only concerned with the algorithmic complexity. Among several papers dealing with efficient algorithms for intersecting two convex polyhedra in the 3D-space, the following can be highlighted: The first non-trivial algorithm to solve the problem was presented by Muller and Preparata [1] in the late seventies. It was also the first technique to achieve this task in $O(n \log n)$ time. Here n is a resonable measure of the combined complexity of the two polytopes. Usually, the number of halfspaces defining the polytopes is taken as measure for the complexity. This work relies on the dual relationship between polytope intersection and convex

union. Other important techniques include Hertel *et al.* [2], in which the planar sweep strategy is used and also achieves $O(n\log n)$ time. Yet another $O(n\log n)$ algorithm follows by direct application of the hierarchical representations of 3-polyhedra introduced by Dobkin *et al.* [3]. Other important representation of 3-D polyhedra, known as the doubly-connected-edge-list can be found in Preparata and Shamos [4]. Chazelle's work [5] introduced the first optimal approach to determine the intersection of two polyhedra in linear time $O(n)$ time. Basically, this method is an improved version of the approach presented in Dobkin *et al* [3]. It uses the dual transforms previously introduced by Muller and Preparata. Finally, inspired in Chazelle's work and in contrast to it, Martin's optimal algorithm [6] solves the problem in the primal space. Any of the approaches for finding the intersection of two convex polyhedra mentioned above, can be extended to a generic method to find feasible regions of 3D-linear programming problems.

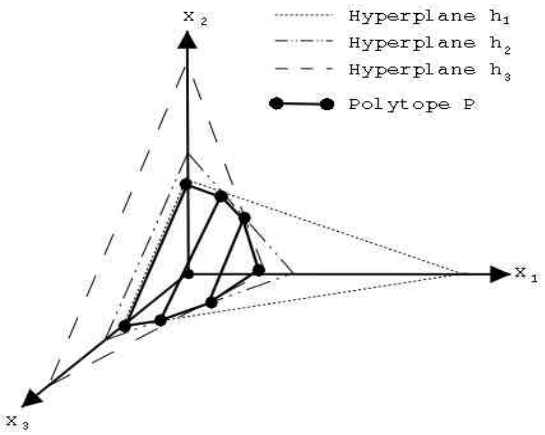

Fig. 1: Feasible Region of a 3-Linear Program.

As mentioned before all these works focus on the complexity issue by defining the algorithmic complexity in terms of the number of facets of the considered polyhedra. In contrast, in this paper the problem is regarded from a visualization point of view. We argue that linear optimization problems arising from practical applications are constrained by a small number of inequalities. Since these constraints basically define the feasible regions, their cardinality also defines the computational cost. Consequently, in our case visualization of the solution process and possible alternatives is essentially more important than algorithmic complexity in the sense that it is described in previous works from the literature. In addressing this specific problem, this paper makes two major contributions: firstly, the solution process can be visualized dynamically and interactively using a straightforward implementation and representation of the addressed problem. We are aware of the fact that the same goal can be achieved using more elaborated and computationally efficient techniques but at the cost of losing simplicity. Secondly, it is also straightforward to conduct an experimental testing of intermediate results. In addition to that the *adjacency invariant* concept is introduced. Using this concept any order of degeneracy can be easily represented. It can then be used, for instance, to outline the path followed by the simplex method when the optimal solution is sought.

The paper is organized as follows. In Section 2 the novel interactive visualization technique to display and find the feasible region of any 3D-linear programming problem is described. Section 3 deals with the *adjacency invariant* concept. Selected results from computer simulations are reported in section 4 and the paper closes with a summary and conclusions in section 5.

2. Visualization of the Feasible Region of Linear Programing Problem

2.1 Notation

In this paper we follow the notation used in [7]. Let p be a nonzero vector in E^d and k a scalar. The inequality $px \le k$ denotes a *constraint.* A *hyperplane* h is a set of the form $\{x : px = k\}$ that divides E^d into two complementary regions, called halfspaces. Hence a *halfspace* is a collection of points of the form $\{x : px \le k\}$ and is denoted by h^-, or a collection of the form $\{x : px \ge k\}$ denoted by h^+. The set of all halfspaces defining a given linear program is denoted by H. A *convex polyhedron* (or simply *polyhedron*) or *polyhedral set* X is defined as the intersection of a finite number of such halfspaces. A *polytope* P is a bounded polyhedra. If $x' \in X$ and $px' = k$ we say the constraint $px \le k$ is *tight or binding or active* at x'. The hyperplanes associated with the defining halfspaces of X are referred as *defining hyperplanes* of X. A point x of X in E^d, is called a *vertex* or *extreme point*, if x lies on some d lineraly independent defining hyperplanes of X. In E^d if more than d defined hyperplanes pass through a vertex, then that vertex is called a *degenerate extreme point* or a *degenerate vertex* and the excess number of planes over d is called its *order of degeneracy*. We say that hyperplane h *supports* a polytope P if $P \subseteq h^-$ and $h \cap P \neq 0$. If h supports P, then $h \cap P$ is called a *face* of P. One dimensional faces are called *edges* and $d-1$ dimensional faces are called *facets*. The set of all vertices of a polytope

P is denoted by $V(P)$. Analogously, the set of all edges and facets of a polytope P, are denoted $E(P)$ and $F(P)$ respectively.

2.2 Feasible Region

First we need to identify and draw suitable boundaries for the feasible region. This is done by applying the following constraints:
1) $-x_1 \le 0$, 2) $-x_2 \le 0$, 3) $-x_3 \le 0$, 4) $x_1 \le c_1$, 5) $x_2 \le c_2$, 6) $x_3 \le c_3$, where c_1, c_2 and c_3 are suitable constants defined by the user. As these halfspaces define a totally bounded region in the 3D space, the next processing step consists of finding the polytope contained in this region and specified by the n halfspaces defined by the problem constraints. Two possible algorithms to build the polytope generated by the intersection of n halfspaces are estudied.

1) Algorithm INTERSECT (H)***:***

Input: A set H of n half-spaces in E^3
Output: A polytope P or the empty space
if card $(H) = 1$
  $C \leftarrow$ the unique halfspace $h^- \in H$
else
  split H into sets H_1 and H_2 of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$
  $C_1 \leftarrow INTERSECT\,(H_1)$
  $C_2 \leftarrow INTERSECT\,(H_2)$
  $C \leftarrow Intersect\_Convex\_Regions\,(C_1, C_2)$
end

2) Algorithm POLYTOPE (H)***:***

Input: A set H of n half-spaces in E^3
Output: A polytope P or the empty space
for (each halfspace in H)
  for (each facet f in P)
    for (each border in f)
      Find_New_Points
      Update_Facet
    end
  end
  Create_New_Facet % in P
  Render (Feasible Region) % interactively
end

The first algorithm is optimal in terms of asymptotic efficiency. It is an extension of the technique described by de Berg *et al.* [8] to solve 2D linear programming problems. Here the divide-and-conquer strategy is used to recursively solve the problem. To find the intersection of n halfspaces, first the intersection of only two of them is sought. Then new intersections with each one of the remaining halfspaces are built recursively. The function *Intersect_Convex_Regions* finds the intersections of any two polytopes. Any optimal technique from the literature can be used. For instance, Chazelle's technique [5], which solves the problem in the dual space, or Martin's algorithm [6], which solves it in the primal space. Both techniques are optimal with complexity $O(n)$. The recurrence has complexity $O(n \log n)$, which is also optimal.

By applying Euler's formula it can be seeing that the number of edges per facet is constant in the second algorithm. Thus, the algorithm complexity is $O(n^2)$.

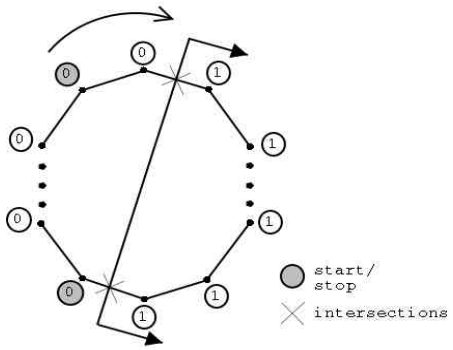

Fig. 2: Sequential search performed for finding new intersections.

As mentioned before linear optimization problems arising from practical applications are constrained by a small number of inequalities. Thus, we are basically concerned with low-scale problems in which the efficiency difference between the two algorithms described before is irrelevant. Since algorithm *POLYTOPE* give us better means to adopt the *adjacency invariant*, we will use this algorithm as a basis for the following discussions. Notice that on the one hand complexity does not play any relevant role and on the other hand the *adjacency invariant* is independent of the selected approach. Consequently, adopting algorithm *POLYTOPE* in the sequel does not mean any loss of generality. Let us begin by making some remarks with respect to our implementation of algorithm *POLYTOPE*:

1- Function *Find_New_Points* finds the world coordinates of the new vertices that emerge from the intersection between the *i*th halfspace with the polytope defined by the first $i-1$ halfspaces. This process is started by generating the polytope defined by equations 1-6. Thereafter, we perform a sequential search through adjacent borders of a given facet to find the intersections. In order to avoid the drastic approach of searching through all of them, when

traversing the borders of a facet a binary value is assigned to each vertex according to the rule: if the world coordinates of the current vertex is a point in the *i*th halfspace then the binary value 1 is assigned, otherwise it get assigned the binary value 0. Since it cannot be more than two new vertex intersecting a given facet, we know exactly when to stop the searching process. See Fig. 2.

2- The two functions *Create_New_Facet* and *Update_Facet* have been implemented specifically to preserve the *adjacent invariant* of both the modified facets and the new facet emerging at the current iteration.

3. Adjacency Invariant

To efficiently manage any order of degeneracy that could arise running any region visualization algorithm, we need to adopt an efficient facet neighborhood representation for every vertex v at each facet of P. For this we state the following invariant: Let a and b be the two facets associated to each vertex v of a given facet f that collide with facet f at vertex v. If the order of degeneracy of vertex v is more or equal than 1, then facets, a and b, are those adjacent to the left and right of facet f. Thus, facets a and b, are the facets forming the borders with facet f meeting at v. See Fig. 3.

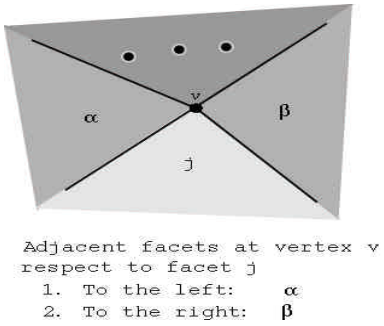

Fig. 3: Adjacency Invariant.

Observe that using this invariant the following processes can be run in constant time $O(1)$:

1. The estimation of two adjacent facets meeting at vertex v with facet f.
2. The estimations of all vertices of each facet colliding at vertex v with facet f.
3. The estimation of the facet f' forming a border with facet f.

The processes described in 1 and 2 are particularly useful for sensitive analysis in linear programming techniques aiming at iterative path following towards the optimum value, e.g., the simplex method. A detailed description of these processes is given in the next section. The last process is useful to determine the three equations to be solved when the world coordinates of a new vertex v'' emerging from edge intersections are sought, see Fig. 4.

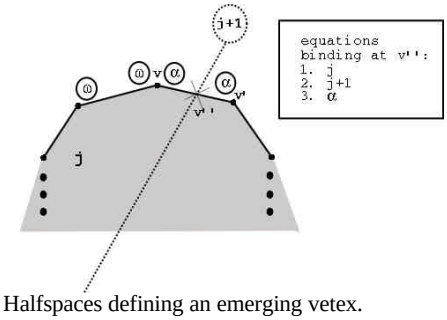

Fig. 4: Halfspaces defining an emerging vetex.

To show how to keep the *adjacency invariant* of facets 1,2,..., $j-1$ and the *j*th facet at iteration j in constant time we considered all intersection possibilities between the *j*th plane corresponding to the *j*th halfspace and facet i, with $0 \le i \quad j-1$. These possibilities are outlined in Tab. 1.

Cases	Intersection Type
1	No intersection
2	Intersection in 1 point (for a vertex of facet i
3	Intersection with two adjacent vertex
4	-adjacent vertex
5	
6	Intersection with two new points

Different cases for the intersection between a halfspace and a facet of the polytope

Let us show how to keep the of facets ,…, $j-$, depending on these six cases. Thereafter, it will be *invariant* of emerging facets.

Case 1: As the plane does not intersect the facet i, it remains unchanged or it will be discarded. This case is depicted in Fig. 5a.

Case 2: The plane intersects facet *i* at vertex v, thus, it remains unchanged or it will be discarded. Fig. 5b illustrates this case.

Case 3: The plane intersects facet i at the adjacent v and v $j-1$. If the halfspace covers facet i, it j is taken instead of a at v and v' to represent the out side facet of the edge. If the halfspace does not cover the facet i, it is discarded. This case is depicted in Fig. 5c.

Case 4: The plane intersects facet i at two non-adjacent vertex v and v' that from iteration $j-1$ respectively have w - a and m - l as their associated facets. vv' is a new edge in facet i. Two cases are considered: 1) j is taken instead of w - m or a - l to represent the out side facet of edge vv' (depending of the region covered by the halfspace). 2) The corresponding regions are preserved. Fig. 5d illustrates the case.

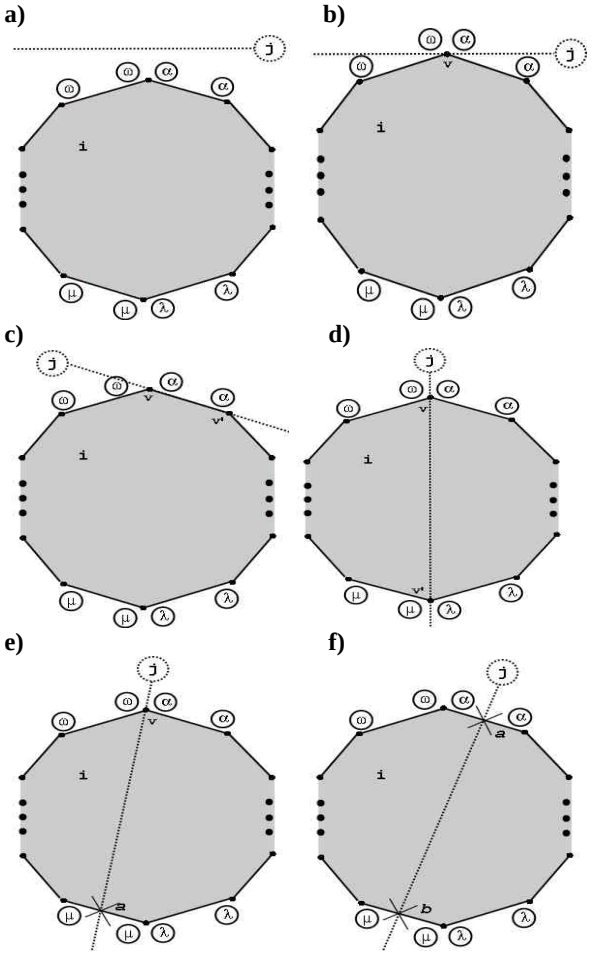

Fig. 5: The six cases considered in Tab. 1

Case 5: The plane intersects facet i at vertex v and at point a of an edge. From iteration $j-1$ and the application of the *adjacency invariant* two cases are derived: 1) vertex v has w - a as its associated facets. 2) m is the outside facet at the intersected edge. Taking v' as a, vv' is now a new edge in facet i. Three steps are considered: 1) j is taken as w or a to represent the out side facet of edge vv' at vertex v, depending of the region covered by the halfspace. 2) Facets m and j are associated at vertex v'. 3) The corresponding regions are preserved. Fig. 5e illustrates the case.

Case 6: The plane intersects facet i at points a and b of two of his edges. From iteration $j-1$, a and m are the outside facets at the intersected edges to which a and b belong. Taking v as a and v' for b, vv' is now a new edge in facet i. Thus, 1) Facets a and j are associated at vertex v. 2) Facets m and j are associated at vertex v'. 3) The correspondent regions are preserved. Fig. 5f illustrates the case.

As the average number of vertex per facet is constant, cases 1 to 6 can clearly be achieved in constant time $O(1)$. Specifically, this can be done using the function *Update_Facet* in algorithm *POLYTOPE*.

Now we show how to keep the *invariant* for emerging facets. For this let us assume that at iteration j a new facet j emerges and that vertex v belongs to facet j. If vertex v emerges in the middle of an edge, then the order of degeneracy at v is zero. Consequently, the two facets that define the edge are associated to vertex v. If vertex v corresponds to an existent vertex, then facet j will be the inside facet of two borders meeting at v. Thus, the two facets in which j define these borders are necessarily their associated outside facets at vertex v. Fig. 6 illustrates this degeneracy case.

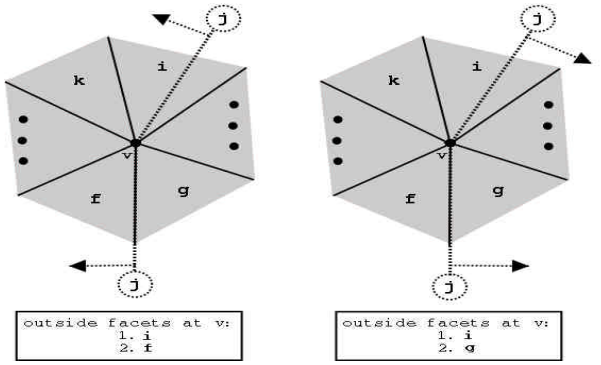

Fig. 6: How to keep the *adjacency invariant* in emerging facets.

As the average of the number of vertex per facet is constant and this happen independently of new emerging facets, the invariant can clearly be kept in constant time $O(1)$. Observe that this task is achieved using the function *Create_New_Facet* in algorithm *POLYTOPE*.

Finally, it can easily be proven by induction on the number of facets defining a polytope that the invariant remains valid.

4. Interactive Visualization

Once an algorithm to draw the feasible region of a 3-linear program is available, sensitive analysis visualization and testing can be conducted. These are crucial concepts for the visualization of linear programming tasks. In this section we describe how to obtain a suitable simulation of the path followed by the simplex method when an optimal vertex is sought.

4.1. Path Following

To simulate the path following by the simplex method when an optimal vertex is sought, the next algorithm is used:

Algorithm SIMPLEXPATH (P,c)***:***

Input: Polytope P and Objective Function c

Output: Optimal Vertex v^*

$v^* \leftarrow v$ % v is any vertex of P

while (v^* is not optimal)

Render_Path

Select_Edge % one of all possible from v^*

$v^* \leftarrow v'$ % v' is the vertex at the extreme of the selected edge

end

Observe that by assigning $v^* \leftarrow v$ we can show how the build a first basic feasible solution. Function *Render_Path* can be implemented by traversing a set of rendered points in the surface of the polytope or by traversing a set of arrows pointing out from one vertex to another. The implementation of the function *Select_Edge* is straightforward if the *adjacency invariant* is used.

4.2. Sensitive Analysis

As stated by Taha [9], sensitive analysis (or dynamic analysis) on a linear program allows us to check how the feasible region and the optimal solution evolve, when some of the parameters vary. This process can be performed dynamically, i.e., it is not necessary to start the whole process each time a parameter is changed. Relevant variations are perceived when coefficients of the objective function or the right-hand side of the constraints are modified. Performing a controlled change over any of these coefficients lead to significant changes of the optimal vertex and the feasible region, respectively. Moreover if the considered constraint represents a resource then if it is binding at optimal vertex v^*, we say the resource it represents is scarce and we are to determine its maximum increase, contrarily if it is not binding, we say the resource is abundant and we are to determine its maximum decrease.

We are particularly interested in sensitive analisys visualization of a 3-linear program from a resource analysis point of view. For this we use the following algorithm:

Algortihm DYNAMICRESOURCES (H,v^*)***:***

Input: Set H of m half-spaces in E^3 representing the constraints and optimal Vertex v^*

Output: A polytope P' that represents the new feasible region according to the changed resources

Classify_Resources

for (each abundant resource)

Determine_Decrease

Render (Feasible Region) %Updating the original feasible region.

end

for (each scarce resource)

Determine_Increase

Render (New Region) % In order to augment the original feasible region.

end

The function *Classify_Resources* can be implemented as follows:

1. Discard constraints that are not resources.
2. From the optimal vertex v^* select the scarce resources by determining the corresponding adjacent facets at v^*.

The role of the two functions *Determine_Decrease* and *Determine_Increase* are to find the maximum decrease and increase of abundant and scarce resources respectively. This is done by finding the new vertex where the corresponding facets are intersected. In the first case, i.e., vertex v^* is the optimal one. In Respect to the latter we need first to distinguish two cases:

Case 1: The scarce resource increases but it does not intersect any halfspaces whose corresponding planes do not support the original polytope. Fig. 7a illustrates this case.

Case 2: The scarce resource increases but it intersects some halfspaces whose corresponding planes do not support the original polytope. Figure 7b illustrates the case.

Observe however that in both cases the region increases forming a new polytope P' whose set of facets $F(P')$ must be a subset of:

1. The facet corresponding to the given scarce resource.
2. The facets adjacent to the facet corresponding to the given scarce resource at optimal vertex v'.
3. The emerging facets corresponding to some of the halfspaces that do not belong to the original polytope.

In this case a new set of halfspaces H' should be taken to estimate P'. This set is chosen from the following options:

1. The halfspace complementary to the halfspace corresponding to the given scarce resource.
2. The halfspaces corresponding to all facets in ii)
3. All halfspaces not contained in P.

Observe that the *adjacency invariant* plays a crucial role in this simple strategy to estimate H'. Furthermore, it is important to stress that the input of the function *Render*() in algorithm *DYNAMICRESOURCES* is a set of halfspaces and its output is a polytope. Thus, any of the algorithms described in section 2 can be used as *Render*() function.

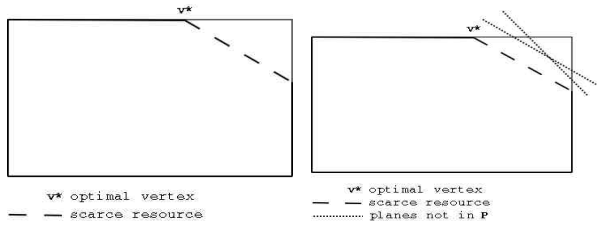

Fig. 7. Scarce resource analysis. a) Case 1, b) case 2.

5. Summary and Further Study

A comprehensive study and a scheme to visualize the feasible region of a given 3D-linear programming task has been presented. The visualizations scheme can be used for the better understanding of the solution process when different methods devoted to solving the underlying problem are applied, e.g., the simplex method. The performed analysis comprises sensitive analysis, interactive visualization of the feasible region and experimental testing of intermediate results. It is shown that the design of an appropriate and simple vertex representation is crucial to manage any order of degeneracy. To deal with this paradigm and to some extent to formalize it, the concept of *adjacency invariance* is also introduced. The envisaged approach focuses on the development of an interactive and highly accessible visualization technique as a helping tool for the better understanding of 3D-linear programming problems. A system comprisisng the most relevant algorithms presented in this paper has been implemented, see Fig. 8. This system has been designed to support teaching models based on visualization.

The presented study can be expanded in three different directions: Testing experimental gradient solution algorithms. Developing objective coefficient analysis. Inserting the system in a closed virtual environment, e.g., in an electronic book. This last objective will significantly help to improve the quality of teaching and to understand basic mathematic concepts.

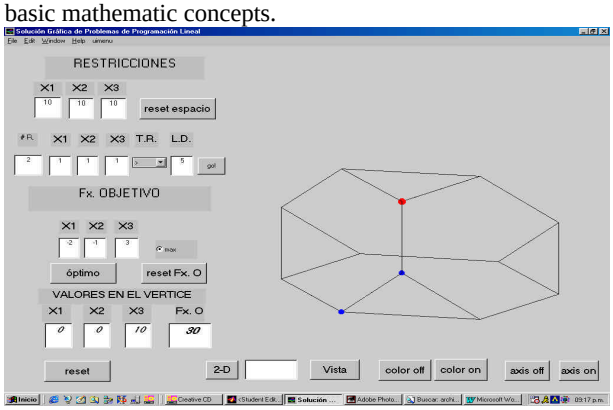

Fig. 8: A screenshot of the implemented system.

6. References

[1] D. Muller and F. Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7:217-236, 1978.

[2] S. Hertel, M. Mäntylä, K.. Mehlhorn y J. Nievergelt. Space sweep solves intersection of convex polyhedra. *Acta Informatica*, 21(5):501-519, 1984.

[3] D. P. Dobkin y D. Kirkpatrick. Fast detection of polyhedral intersection. Theoretical Computer Science, 27:241-253, 1983.

[4] F.P. Preparata y M.I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[5] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. Technical Report, Department of Computer Science, Princeton University, February 1989.

[6] A. K. Martin. A Simple Primal Algorithm for Intersecting 3-Polyhedra in Linear Time. Technical Report, Department of Computer Science, University of British Columbia, July 1991.

[7] Bazaara, J. Jarvis y H. Sherali. *Linear Programming and Network Flows.* John Wiley and Sons. 1990.

[8] M. De Berg, M. Van Kreveld, M. Overmars y O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 1997.

[9] H. Taha. *Operations Research: An Introduction.* Prentice-Hall. 1997.