Reconstruction with Interval Constraints Propagation

Michela Farenzena, Andrea Fusiello
Univ. of Verona (IT), Dip. di Informatica
farenzena@sci.univr.it,andrea.fusiello@univr.it

Agostino Dovier
Univ. of Udine (IT), DIMI.
dovier@dimi.uniud.it

Abstract

In this paper we demonstrate how Interval Analysis and Constraint Logic Programming can be used to obtain an accurate geometric model of a scene that rigorously takes into account the propagation of data errors and roundoff. Image points are represented as small rectangles: As a result, the output of the n-views triangulation is not a single point in space, but a polyhedron that contains all the possible solutions. Interval Analysis is used to bound this polyhedron with a box. Geometrical constraints such as orthogonality, parallelism, and coplanarity are subsequently enforced in order to reduce the size of those boxes, using Constraint Logic Programming. Experiments with real calibrated images illustrate the approach.

1. Introduction

The first and most important stage of model reconstruction consists in recovering the coordinates of points in three-dimensional (3D) space given their images in two or more cameras. In the absence of errors, this problem is trivial, involving only finding the intersection of rays in the space (hereby it is called *triangulation*). If data are perturbed, however, the rays corresponding to back-projections of image points will not intersect, and obtaining the 3D coordinates of the reconstructed point becomes not trivial at all, as witnessed by the renewed interest aroused by this issue [7, 10, 16].

In [8] the problem is solved for the case of two views, taking advantage of the epipolar constraint and involving the solution of a sixth-degree polynomial. However, the method is not generalizable to more than two views. Recently, an optimal solution is found even for the case of three views [16], but again the method is not generalizable. For the generic n views case, a simple algebraic method exists [5], but the value being minimized has no geometric meaning, so the method is not reliable: a minimization of a suitable (non-linear) cost function, like the re-projection error in the image plane, should be performed to achieve better accuracy [5, 19]. An alternative approach is to find the closest point in 3D space to the rays back-projected from the image points. In the case of two views, this is the midpoint of the common perpendicular to the two rays. However, it is known that this method fails badly in the case where the rays are almost parallel, corresponding to a point near infinity, since in this case the computed point will be close to the point half-way between the two camera centers. In [7] a L_∞ minimization of the re-projection error is explored. Using the L_∞ cost function is significantly simpler, and computationally faster, than the L_2 cost, but the method is extremely not robust. A further development of this work is [10, 12], where the idea of using L_∞ norm is generalized to a class of 3D reconstruction problems, and an efficient algorithm based on standard optimization techniques is presented. Despite these improvements, however, the problem of robustness remains unsolved.

In our approach, instead of aiming at one "best" solution, as customary, we describe the set of *all* the possible solutions, given a bounded error affecting the image points. In practice, image points are modeled as 2D intervals, and the *solution set* is defined as the set of all the 3D points that can be obtained as the intersection of two conjugate points contained in the 2D intervals. This solution set is a 3D polyhedron representing the best piece of information about the localization of the error-free 3D point one can deduce from the bounded image correspondences. Interval Analysis (IA) is used to obtain a box that rigorously encloses the polyhedron.

According to the IA paradigm, we do not model a probability distribution inside the intervals, therefore there is not a preferred solution in the solution set. Nevertheless, a pointwise solution is needed to make the 3D model usable for other applications. For want of more information, one could choose at random a point inside the intervals. Yet, in many cases, geometric constraints such as orthogonality, parallelism and coplanarity can be imposed on the reconstructed structure. The idea is to formalize this as a Constraint Satisfaction Problem, to use Constraint Logic Programming (CLP) to narrow the intervals as much as possible and then to pick one solution, i.e., to select a point inside each interval such that it satisfies all constraints.

This work builds on [4], where interval-based triangula-

tion is described. The novel contribution of this paper is the use of CLP(Intervals) to impose geometrical constraints on an interval reconstruction. The closest work we are aware of is [17], where IA is applied to the 3D reconstruction problem as well. The focus of the paper, however, is on modeling the interval width that has to be associated to a pixel. Triangulation is cast as the solution of an interval linear system of equations (which usually cause a gross overestimation) and epipolar constraint is taken into account by formulating a Constraints Satisfaction Problem. The two views case only is considered there (our method is for the generic n views case) and there is no attempt to shrink the interval-triangulation solution to arrive at a pointwise solution.

2. Problem formulation

Let P_i, $i = 1, \ldots, n$ be a sequence of n known cameras and m_i be the image of some unknown point M in 3D space, both expressed in homogeneous coordinates. Thus, we write $m_i \simeq \mathrm{P}_i M$, where $\simeq$ denotes equality up to a scale factor. The problem of computing the point M given the camera matrices P_i and the image points m_i is known as the *triangulation* problem. In the absence of errors, this problem is trivial, involving only finding the intersection point of rays in the space. When data are perturbed by errors, however, the rays corresponding to back-projections of the image points do not intersect in a common point, therefore only an approximate solution can be obtained. This approximation can be circumvented if one refrains from searching for *one* solution and compute instead a *set* of solutions (defined in terms of the error affecting the image points) that contains the error-free solution.

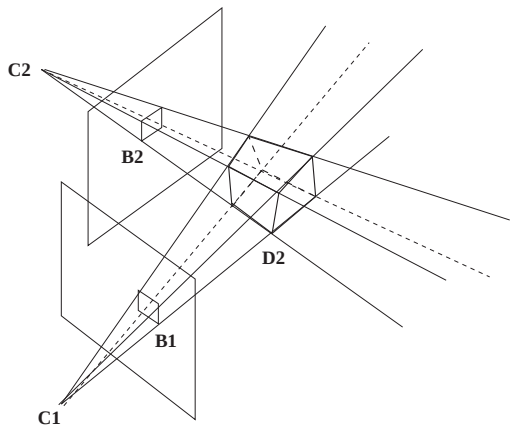


Figure 1. Interval-based triangulation.

In the case of two views, assuming that point location errors are bounded by rectangles B_1 and B_2 in the two images respectively, the *solution set* of triangulation is a diamond-shaped polyhedron D_2 as in Fig. 1. Geometrically, D_2 is obtained by intersecting the two semi-infinite pyramids defined by the two rectangles B_1 and B_2 and the respective camera centers.

In the general case of n views, the solution set is defined as the polyhedron formed by the intersection of the n semi-infinite pyramids generated by the intervals $B_1, \ldots B_n$. Analytically, this region is defined as the set

$$D_n = \{M \,:\, \forall i = 1, \ldots, n \, \exists m_i \in B_i \text{ s.t. } m_i \simeq \mathrm{P}_i M\}.$$

In the following section we will show how the solution set can be enclosed with an axis-aligned box using Interval Analysis.

These 3D polyhedra are the best piece of information about the localization of the error-free 3D point one can deduce from the bounded correspondences. In order to narrow the solution set we must include additional information. Thus, we seek for geometric constraints between scene primitives that are able to reduce the solution set. If we add a sufficient set of constraints we will be able to isolate, ideally, a single solution.

The constraints considered in this paper are geometrical constraints, such as orthogonality, coplanarity and parallelism between lines, that can be deduced automatically from the 3D model obtained by the interval-based triangulation. Eventually, a pointwise solution is derived by constraints propagation, as we will discuss in Section 6.

3. Interval Analysis

IA [15] is an approach to solve numerical problems by performing computations on sets of reals rather than on floating point approximations to reals. There are two principal advantages of IA over classical numerical analysis. The first is that the input errors and the roundoff errors are automatically incorporated into the result interval. Thus, interval evaluation can be viewed as automatically performing both a calculation and an error analysis. The second is that IA allows one to compute provably correct upper and lower bounds on the range of a function over an interval, and this proves useful in the construction of verifiable *constraint solvers*, which return intervals that are guaranteed to contain all the real solutions.

In the sequel of this section we shall follow the notation used in [14], where intervals are denoted by boldface. Underscores and overscores will represent respectively lower and upper bounds of intervals. The midpoint of an interval $\boldsymbol{x}$ is denoted by $\mathrm{mid}(\boldsymbol{x})$. $\mathbb{IR}$ and $\mathbb{IR}^n$ stand respectively for the set of real intervals and the set of real interval vectors of dimension n. If $f(x)$ is a function defined over an interval $\boldsymbol{x}$ then $\mathrm{range}(f, \boldsymbol{x}) = \{y : \exists x \in \boldsymbol{x}, y = f(x)\}$.

If $\boldsymbol{x} = [\underline{x}, \overline{x}]$ and $\boldsymbol{y} = [\underline{y}, \overline{y}]$, a binary operation between $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined in interval arithmetic as:

$$\boldsymbol{x} \circ \boldsymbol{y} = \{x \circ y \mid x \in \boldsymbol{x} \wedge y \in \boldsymbol{y}\}\,, \quad \text{for all binary operator } \circ \in \{+, -, \times, \div\}\,. \quad (1)$$

Operationally, interval operations are defined by the min-max formula:

$$\boldsymbol{x} \circ \boldsymbol{y} = \big[\min\left\{\underline{x} \circ \underline{y}, \underline{x} \circ \overline{y}, \overline{x} \circ \underline{y}, \overline{x} \circ \overline{y}\right\}, \max\left\{\underline{x} \circ \underline{y}, \underline{x} \circ \overline{y}, \overline{x} \circ \underline{y}, \overline{x} \circ \overline{y}\right\}\big] \quad (2)$$

Thus, the ranges of the four elementary interval operations are exactly the ranges of the corresponding real operations.

Here, interval division $\boldsymbol{x}/\boldsymbol{y}$ is undefined when $0 \in \boldsymbol{y}$. However, in the *extended arithmetic* [13], division by zero is included, resulting the interval $[-\infty, \infty]$ in the worst case.

In general, interval computation cannot produce the exact range of a function, but only approximated it, providing a guaranteed overestimation.

Definition 1 (Interval extension) *[13] A function* $\boldsymbol{f} : \mathbb{IR} \rightarrow \mathbb{IR}$ *is said to be an* interval extension *of* $f : \mathbb{R} \rightarrow \mathbb{R}$ *provided that* $\mathrm{range}(f, \boldsymbol{x}) \subseteq \boldsymbol{f}(\boldsymbol{x})$ *for all intervals* $\boldsymbol{x} \subset \mathbb{IR}$ *within the domain of* $\boldsymbol{f}$.

This property is particularly suited for error propagation: If $\boldsymbol{x}$ bounds the input error on the variable x, $\boldsymbol{f}(\boldsymbol{x})$ bounds the output error. Therefore, if the exact value is contained in interval data, the exact value will be contained in the interval result. This approach is different from the established techniques for error propagation [18, 5, 6, 11], mainly based on statistical analysis: a statistical distribution of the error need not to be assumed, and the result is mathematically guaranteed to contain the exact value. On the contrary, covariance propagation assumes Gaussian distribution (otherwise higher order moments should have been considered), and is correct only under linear transformations.

Operationally, a straightforward interval extension is defined as follows:

Definition 2 (Natural interval extension) *Let us consider a function* f *computable as an arithmetic expression* f*, composed of a finite sequence of operations applied to constants, argument variables or intermediate results. A* natural interval extension *of such a function, denoted by* $\mathbf{f}(\boldsymbol{x})$*, is obtained by replacing variables with intervals and executing all arithmetic operations according to the rules (2).*

Similar definitions apply for interval vectors (or boxes) in $\mathbb{IR}^n$.

4. Interval-based triangulation

Given the camera matrices P_1 and P_2, let m_1 and m_2 be two corresponding points. It follows that m_2 lies on the epipolar line of m_1 and so the two optical rays back-projected from image points m_1 and m_2 lie in a common epipolar plane. As they lie in the same plane, they will intersect at some point. This point is the reconstructed 3D scene point M. The optical ray of m_1 is given by

$$M = \begin{pmatrix} -\mathrm{P}_{3\times3,1}^{-1} p_{4,1} \\ 1 \end{pmatrix} + \lambda \begin{pmatrix} \mathrm{P}_{3\times3,1}^{-1} m_1 \\ 0 \end{pmatrix}, \quad \lambda \in \mathbb{R}, \quad (3)$$

where $\mathrm{P}_{3\times3,1}$ is the matrix composed by the first three rows and first three columns of P_1, and $p_{4,1}$ is the fourth column of P_1. The epipolar line corresponding to m_1 represents the projection of the optical ray of m_1 onto the image plane of the second camera:

$$\kappa m_2 = e_2 + \lambda m_1' \quad (4)$$

where $e_2 = \mathrm{P}_2 \begin{pmatrix} -\mathrm{P}_{3\times3,1}^{-1} p_{4,1} \\ 1 \end{pmatrix}$, $m_1' = \mathrm{P}_{3\times3,2}\mathrm{P}_{3\times3,1}^{-1} m_1$ and k is the depth of M with respect to the second camera.

Analytically, the reconstructed 3D point M can be found by first solving for parameters λ in Eq. (4), and then inserting it into Eq. (3).

As the 3-vectors m_2, m_1', e_2, are coplanar, the scalars κ and $-\lambda$ are the so-called *baricentric coordinates* of e_2, and they can be easily computed using the cross product [11] (a similar formula holds for κ):

$$\lambda = \frac{(e_2 \times m_2) \cdot (m_2 \times m_1')}{||m_2 \times m_1'||^2} \quad (5)$$

After doing all the substitutions, we can write a closed form expression that relates the reconstructed point to the two conjugate image points:

$$M = \mathsf{f}(m_1, m_2) \quad (6)$$

Formula (6) represents the geometric operation of intersecting rays in 3D space and – being a simple arithmetic expression – is particularly well suited for applying IA. If we let m_1 and m_2 vary in B_1 and B_2 respectively, then $\mathrm{range}(\mathsf{f}, B_1 \times B_2)$ describes the solution set D_2 (Fig. 1). Interval Analysis gives us a way to compute an axis-aligned bounding box containing D_2 by simply evaluating $\mathbf{f}(\boldsymbol{m}_1, \boldsymbol{m}_2)$, the natural interval extension of f, with $B_1 = \boldsymbol{m}_1$ and $B_2 = \boldsymbol{m}_2$. IA guarantees that if the conjugate intervals $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ contain the exact point correspondences, then the interval result contains the exact (i.e. error-free) 3D reconstructed point.

It may be worth noting that the result is not to be interpreted in a probabilistic or fuzzy way: no assumption is made on error statistical distribution (only that the error is bounded), hence no point inside the resulting 3D interval is more probable or more important than others.

This approach is easily extendible to the general case. As defined in Section 2, the solution set of triangulation is the 3D polyhedron formed by the intersection of the semi-infinite pyramids generated by back-projecting in space the intervals $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_n$ (Fig. 2). Thanks to the associativity of intersection, D_n can be obtained by first intersecting

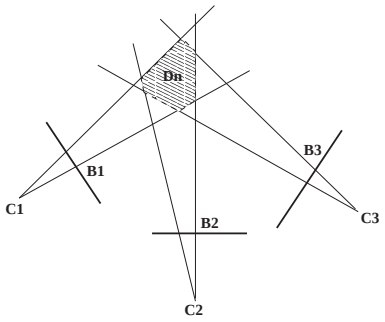


Figure 2. n-views interval-based triangulation.

pairs of such pyramids and then intersecting the results. Let $D_2^{i,j}$ be the solution set of the triangulation between view i and view j. Then:

$$D_n = \bigcap_{\substack{i=1,\ldots,n \\ j=i+1,\ldots,n}} D_2^{i,j}. \quad (7)$$

An enclosure of the solution set D_n is obtained by intersecting the $n(n-1)/2$ enclosures of $D_2^{i,j}$ computed with the method described above. Since each enclosure contains the respective solution set $D_2^{i,j}$, their intersection will contain D_n. Similarly, as the error-free solution is contained in each $D_2^{i,j}$, then it must be contained in D_n as well.

As robustness is concerned, our method allows to detect outliers, because the intersection of pyramids is likely to be empty in the case of outlying points. In particular this will happen whenever the actual error is larger than the error modelled by the interval. This is analogous to setting a rejection threshold in M-estimators or to the consensus threshold in RANSAC.

5. Constraint Logic Programming

CLP [9, 1] is a declarative programming paradigm particularly well suited for encoding minimization problems in various domains. It is the natural merger of the two declarative AI paradigms known as Constraint Solving and Logic Programming.

Generally, a Constraint Satisfaction Problem is modeled by assigning admissible intervals of values and constraints to each variable. A built-in constraint solver then contracts these intervals preserving the solution set. The output of the constraint solver is an interval associated to each variable of the problem, such that any value of the variables in their intervals is a solution to the initial problem (if solutions exist).

One of the peculiar features of CLP is the independence of the problem modeling and of the search strategy. Problem modeling is based on traditional declarative programs in which one can use the built-in notion of constraint. Constraints are in general first-order formulas concerning variables that can assume values in some domains. The scheme is general, and various possible constraints and domains can be used. The problem considered in this paper is naturally encoded using constraints over *interval of reals*. We denote this instance of the CLP scheme as CLP(Intervals). This language can be used in practice using the library `ic` of the ECLiPSe 5.8 system [3].

We briefly introduce this programming paradigm with a simple example. Let us consider three variables A, B, and C that we know can assume values in the following intervals (the syntax, rather intuitive, is the same as in ECLiPSe):

```
A :: 1.0 .. 4.0, B :: 0.0 .. 2.0, C :: -2.5 .. 3.5
```

Assume moreover that we know the following two constraints hold on those variables

```
A + 0.1 < B, B + 0.1 < C
```

The constraint solver of CLP(Interval) performs a form of reasoning, known as *constraint propagation* and return the new intervals:

```
A = A1.0 .. 1.9000000000000001
B = B1.0999999999999999 .. 2.0
C = C1.1999999999999993 .. 3.5
```

for the three variables. Let us observe that the solution set is unchanged.

The built-in predicate locate looks for real (interval) solutions to a set of constraints. This predicate requires a precision parameter that specifies the width of the interval solution. For instance, locate([A,B,C],0.01) returns:

```
A = A1.0 .. 1.005027077049309
B = B1.0999999999999999 .. 1.1100774675439455
C = C1.1999999999999993 .. 1.2100774675439454
```

Solutions that minimize/maximize a given function of the variables can also be found by the solver. In this case a search tree is internally generated and pruned using the constraints. Moreover, search strategies can be then selected among the built-in available (such as branch and bound) or programmed at this level. Let us observe that while constraint propagation is executed by polynomial time algorithms, solution search can be computationally more expensive. In particular, computation time depends on the precision chosen.

6. Interval Constraints Propagation

The interval-based triangulation yields boxes in 3D space, each representing the error-free 3D point bounded by its error. Given the connectivity information between these points, geometric constraints can be deduced, such as orthogonality, parallelism, and coplanarity. They were obtained automatically using very simple heuristics: If two such primitives in a pointwise reconstruction nearly satisfy the constraint, then the constraint is added to the list.

As previously said, this problem is suitable for being formalised as a constraint satisfaction problem: Each reconstructed box is associated to an interval variable, and these variables must satisfy the aforementioned geometric constraints. The main ECLiPSe predicate is the following:

```
best_pos(V,Rcts,Pars,SameL,SameP,SameA,Dom,Res) :-
    set_intervals(V,Dom,Res,VList),      % 1
    set_orthogonal(Rcts,Res),            % 2
    set_parallel(Pars,Res),              % 3
    set_coplanar(SameP,Res),             % 4
    epsilon(Eps), locate(VList,Eps,lin). % lin/log
```

The predicates set_intervals, set_orthogonal, etc., impose the constraints corresponding to their names. They are defined using some recursion to address the desired variables; then the constraints on the variables are added. set_intervals fixes the known input domain Inf..Sup to the triple of variables representing the point A:

```
Ax :: Infx..Supx, Ay :: Infy..Supy, Az :: Infz..Supz.
```

set_orthogonal imposes that $\widehat{ABC}$ is a right angle:

```
ic:( Cos =:= (Ax-Bx)*(Cx-Bx)+(Ay-By)*(Cy-By)+
   (Az-Bz)*(Cz-Bz) ), ic:( abs(Cos) < Delta ).
```

where Delta is a precision parameter, set to 10^{-10}. The fact that AB and CD are parallel segments is imposed by set_parallel as follows:

```
ic:( Vetx =:=(Ay-By)*(Cz-Dz)-(Az-Bz)*(Cy-Dy) ),
ic:( Vety =:= (Az-Bz)*(Cx-Dx)-(Ax-Bx)*(Cz-Dz) ),
ic:( Vetz =:= (Ax-Bx)*(Cy-Dy)-(Ay-By)*(Cx-Dx) ),
ic:( abs(Vetx)< Delta), ic:(abs(Vety) < Delta),
ic:( abs(Vetz) < Delta).
```

The constraints encodings in the other predicates are similar and omitted due to lack of space.

Finding the solution of this problem is committed to the ECLiPSe constraint solver, using the built-in predicate locate (cf. Sec. 5) with precision parameter Eps.

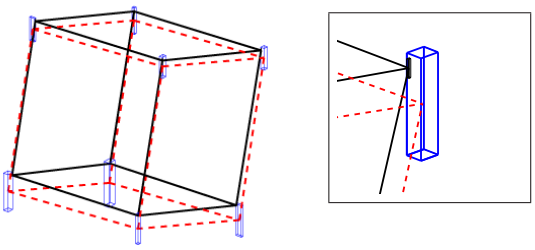


Figure 3. Ground truth structure (dashed line) and pointwise solution (solid line). The bigger boxes are the result of interval-based triangulation, whereas the small ones (see the detail on the right) are the result obtained after constraints propagation.

The solution achieved by ECLiPSe is composed by intervals, though very thin. We finally attain a point solution taking a random point within each of these intervals because, as already outlined, there is not a preferred point inside them. Since computation time grows as the precision parameter Eps gets smaller, the level of precision attainable in a reasonable time strictly depends on the size of the problem's instance, namely the number of variables and the number of constraints.

7. Experimental results

We tested the overall technique (interval-based triangulation and constraint propagation) on real calibrated sequences. We report here the results relative to the *Tribuna* and *Castle* sequences (Fig. 4), consisting of five frames each. Intrinsic parameters were obtained from calibration, extrinsic parameters were recovered from the factorization of the essential matrices. We assumed a 2-pixel wide interval as a reasonable bound for feature points. Calibration error – provided by the Calibration Toolbox [2] – was also taken into account by transforming the intrinsic parameters matrix A in an interval matrix $\boldsymbol{A}$. Then, normalized image coordinates were computed in interval arithmetic as: $\boldsymbol{m} \leftarrow \boldsymbol{A}^{-1}\boldsymbol{m}$. In this way the error derived from calibration is summed up with the point localization error, and the resulting 3D box will take into account both. The average side length of the 3D boxes obtained by interval-based triangulation is about 10 cm for *Tribuna* and 70 cm for *Castle*.

The number and the type of constraints considered for the two examples are detailed in Table 1. The 3D intervals and the constraints are formalized in the constraint satisfaction problem as described in Sec. 6 and constraints propagation is applied. This permits us to shrink these boxes up to an average side length of about 5 mm and 10 cm respectively. The whole process took a few seconds on a PowerPC G4 1.33 GHz machine.

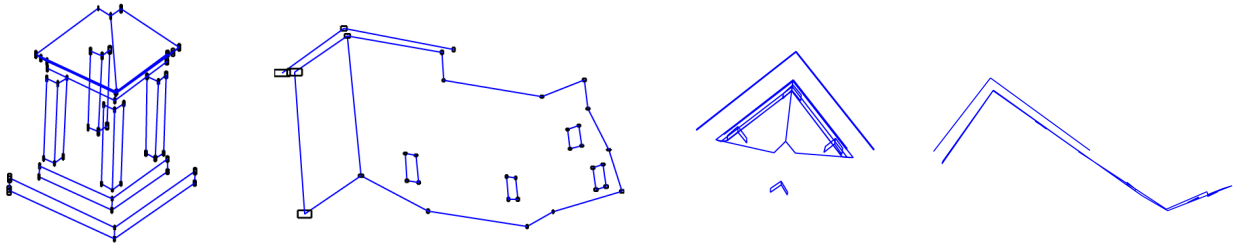
Constraints	*Tribuna*	*Castle*
Orthogonality	36	16
Parallelism	34	22
Coplanarity	4	33

Table 1. The constraints considered in the propagation step.

Finally, a pointwise solution is obtained taking a point at random in each box (there are 53 points in *Tribuna* and 33 points in *Castle*). The top views are depicted in Fig. 4 and the shaded views in Fig. 5 highlight the faithfulness of the structure.

8. Conclusions

In this paper we presented a new approach to scene modelling from many calibrated views based on Interval Analysis and Constraint Logic Programming. We demonstrate how IA and CLP can be used to obtain an accurate geometric model of a scene that rigorously takes into account the propagation of data errors and roundoff.



(a) (b) (c) (d)

Figure 4. Interval-based triangulation of *Tribuna* (a) and *Castle* (b). To better visualize the 3D structure, the segments joining the midpoints of the intervals have been drawn. Top views of the 3D models after constraint propagation, (c) and (d).

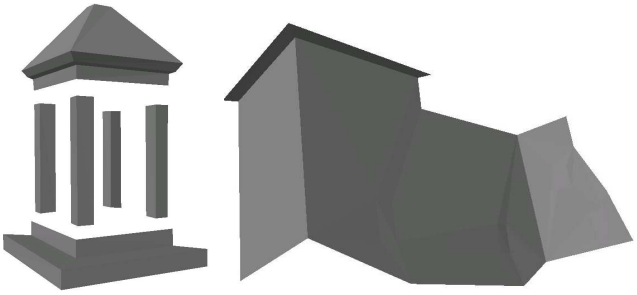


Figure 5. A shaded view of the final 3D model of *Tribuna* (left) and *Castle* (right).

The output of triangulation is no more a single 3D point, but a 3D box that contains all the possible solutions given a bounded perturbation of the conjugate points.

The width of the solution boxes is then reduced by propagating geometrical constraints using CLP. The final solution is composed by points that i) are inside the original boxes and ii) satisfy the given constraints. Experiments show that in a reasonable time we can achieve a (practically) pointwise solution.

Acknowledgements

Arrigo Benedetti co-authored some papers in the past and introduced the authors to IA. This work have been supported by the LIMA3D project (PRIN).

References

[1] K. R. Apt. *Principles of Constraint Programming*. Cambridge, 2003.

[2] J.-Y. Bouguet. MATLAB calibration toolbox.

[3] A. M. Cheadle, W. Harvey, A. J. Sadler, J. Schimpf, K. Shen, and M. G. Wallace. ECLiPSe: An Introduction. Technical Report 03–1, IC-Parc, Imperial College London, 2003.

[4] M. Farenzena and A. Fusiello. Rigorous computing in computer vision. In *Video, Vision and Graphics*, pages 101–108, Edimburgh,UK, 2005.

[5] O. Faugeras. *Three-Dimensional Computer vision: a geometric viewpoint*. MIT Press, Cambridge, MA, 1993.

[6] R. M. Haralick. Propagating covariance in computer vision. In *Workshop on Performance Characteristics of Vision Algorithms*, pages 1–12, Cambridge,UK, 1996.

[7] R. Hartley and F. Schaffalitzky. L_∞ minimization in geometric reconstruction problems. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 504–509, Washington, D.C., USA, 2004.

[8] R. I. Hartley and P. Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2):146–157, November 1997.

[9] J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19–20:503–581, 1994.

[10] F. Kahl. Multiple view geometry and the l_∞-norm. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 510–517, Beijing, China, 2005.

[11] K. Kanatani. *Geometric Computation for Geometric Vision*. Oxford University Press, 1993.

[12] Q. Ke and T. Kanade. Quasiconvex optimization for robust geometric reconstruction. In *IEEE International Conference on Computer Vision (ICCV 2005)*, October 2005.

[13] R. B. Kearfott. *Rigorous Global Search: Continuos Problems*. Kluwer, 1996.

[14] R. B. Kearfott, M. T. Nakao, A. Neumaier, S. M. Rump, S. P. Shary, and P. van Hentenryck. Standardized notation in interval analysis. Submitted to Reliable Computing.

[15] R. E. Moore. *Interval Analysis*. Prentice-Hall, 1966.

[16] H. Stewenius, F. Schaffalitzky, and D. Nister. How hard is 3-view triagulation really? In *Proceedings of the IEEE International Conference on Computer Vision*, pages 510–517, Beijing, China, 2005.

[17] B. Telle, M. Aldon, and N. Ramdani. Guaranteed 3d visual sensing based on interval analysis. In *IEEE International Conference on Intelligent Robots and Systems*, pages 1566–1571, Las Vegas, USA, 2003.

[18] J. Weng, T. S. Huang, and N. Ahuja. Motion and structure from two perspective views: Algorithms, error analysis, and error estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(5):451–476, 1989.

[19] Z. Zhang. Determining the epipolar geometry and its uncertainty: A review. *International Journal of Computer Vision*, 27(2):161–195, March/April 1998.