A Grammar-Based GUI for Single View Reconstruction

Dakshita Khurana*
Indian Institute of Technology Delhi
Hauz Khas
New Delhi, India
dakshitakhurana@ucla.edu

Surabhi Sankhla
Indian Institute of Technology Delhi
Hauz Khas
New Delhi, India
surabhi.0391@gmail.com

Abhinav Shukla
Indian Institute of Technology Delhi
Hauz Khas
New Delhi, India
abhinav.shukla2@gmail.com

Richa Varshney
Indian Institute of Technology Delhi
Hauz Khas
New Delhi, India
richa28.varshney@gmail.com

Prem Kalra
Indian Institute of Technology Delhi
Hauz Khas
New Delhi, India
pkalra@cse.iitd.ernet.in

Subhashis Banerjee
Indian Institute of Technology Delhi
Hauz Khas
New Delhi, India
suban@cse.iitd.ac.in

ABSTRACT

This paper presents a novel grammar-based GUI with an integrated back-end engine that enables accurate single view reconstruction with minimum supplementary information from the user. Our grammar allows a dynamic symbolic capture of constraints, which can be solved together in the end to eliminate incremental errors. This approach focuses on the simultaneous reconstruction of connected faces minimizing least square error, as opposed to a piece-wise approach where error keeps building up at every stage. We demonstrate how our Graphical User Interface is usable even by a layman to generate 3D models with negligible error.

Keywords

constraints, grammar, GUI, single view reconstruction, programming semantic

1. INTRODUCTION

Since Horry *et al.*'s [4] "Tour into the Picture", single view reconstruction has emerged as an useful technique for piecewise planar 3D modelling of paintings and photographs. The technique is particularly applicable to scenes with regular geometric structures consisting primarily of blocks and faces, as can be found in many heritage monuments in India and elsewhere. Subsequently, several systems for single view reconstruction have been reported [1, 6, 7, 8, 9, 11] which use user provided constraints on vanishing points and lines, parallelism and orthogonality, coplanarity and incidence relationships to calibrate the camera and determine the scene geometry. Criminisi *et al.* show that the vanishing line of a reference plane and a vanishing direction is the minimal information necessary for camera calibration and interactive affine measurements in the image. Sturm and Maybank [11] use coplanarity of points and lines, and parallelism and orthogonality of planes and directions to derive constraints for single view reconstruction of sets of connected planes. The basic geometric techniques have also been extended for automatic reconstruction from single views in some situations [3].

Incremental computation of single view reconstruction from user provided constraints often results in error accumulation during the sequential computation steps. In this paper we address the problem of first capturing all user provided constraints through a graphical user interface in a symbolic representation and then compiling the symbolic information to derive a set of equations which can be solved in one go in a least square sense. One shot computation of a tightly coupled system ensures robustness and prevents error accumulation. We derive a set of constraints that can be captured symbolically using a simple data structure and programming semantic and describe a method for subsequent compilation of the symbolic constraints in to a set of equations. As an added benefit, we can also process the symbolically captured information to determine whether the constraints provided are adequate for the complete reconstruction. We present results, like those in Fig. 1, which demonstrate the efficacy of the method.

We start by implementing already known methods for camera calibration from two orthogonal planes ([1, 2, 5]). After reviewing the basic techniques for 3-D reconstruction, we demonstrate how our approach helps give tight constraints that generate an optimal solution. In what follows we describe all the rules, their semantics and reconstruction results.

2. CAMERA CALIBRATION

For calibration, we follow a method very similar to that used in [11]. The user is required to mark two orthogonal planes in the image which help register homographies from the world to the image plane. He may pick up any two orthogonal planes in the image for this purpose, as illustrated

*Corresponding author

ICVGIP '12, December 16-19, 2012, Mumbai, India

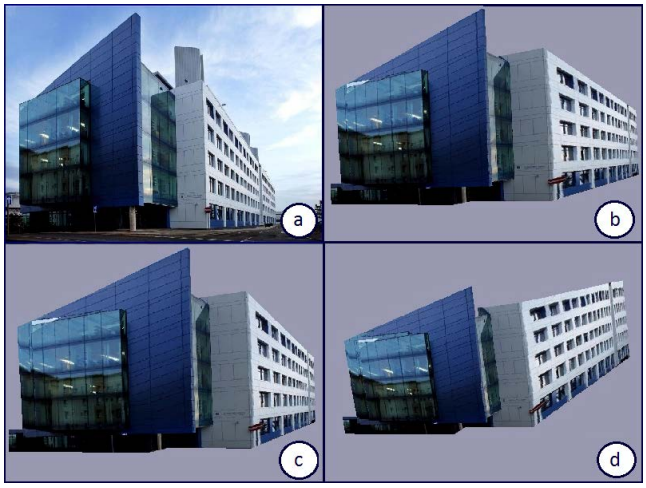

Figure 1: (a) An image of the Wellcome Trust Biocentre at the University of Dundee (b) and (c) side views and (d) top view of the reconstruction attempted using only this image.

in Figure 2. The Euclidean Projection matrix $\mathbf{P}_{euclid}$ and the camera matrix C are obtained using homographies from orthogonal planes in the world to the planes marked in the image.

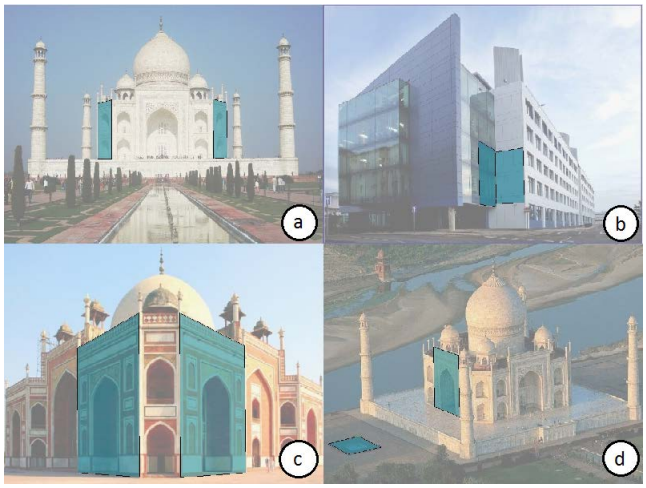

Figure 2: Camera Calibration in a variety of settings, to illustrate the use of various possible pairs of orthogonal planes in architectural and structured scenes. (a,d) The shaded areas represent the faces chosen for calibration in two different views of the magnificent Taj Mahal in India, (b) The WTB at Dundee University (c) and Humayun's Tomb in Delhi.

3. RECONSTRUCTION TECHNIQUES FOR INDIVIDUAL FACES

Typical architectural scenes consist of piecewise planar surfaces (referred to as faces). We present two ways to reconstruct such polygonal faces. The first method follows an incremental approach by reconstructing faces one after the other; the second simultaneously solves for all faces incorporating additional constraints like parallelism, perpendicularity, corresponding edges and equal heights. The second method builds upon the first and will be described in subsequent sections along with its benefits. Here we take a brief look at the first method of reconstructing individual faces one after the other.

Each face is reconstructed by specifying its edges and vanishing lines that help compute the orientation. Faces must be reconstructed with respect to a reference plane, which may be real or virtual. One can begin with faces that are adjacent to or lie on the coordinate planes. Then these reconstructed faces can be used as a reference and proceed to faces which are connected to these, continuing in a hierarchical fashion.

3.1 Frustum Constraints

Each face is assumed to be a polygon represented by a list of vertices:
$\mathbf{V_1} = (V_{1x}, V_{1y}, V_{1z})^T, \mathbf{V}_2 = (V_{2x}, V_{2y}, V_{2z})^T$, and so forth

The edges of the face in the image may be denoted as l_1, l_2, ...l_s. Each edge l_i is projected to obtain the planes containing the vertices of each face, thereby forming the frustum. This gives $2s$ frustum constraints:

$$(\mathbf{P}^T l_i)^T [V_{jx}, V_{jy}, V_{jz}, 1]^T = 0 \qquad (1)$$

where j represents all vertices lying on edge i

Frustum constraints are illustrated in Fig. 3.

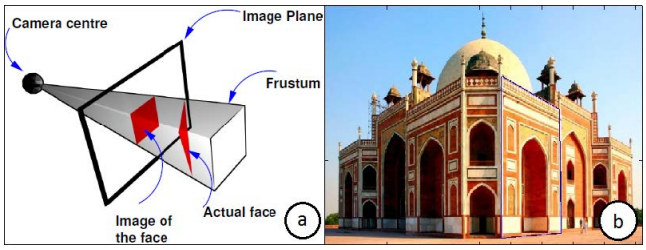

Figure 3: (a) Frustum constraints on a face can be computed using only the vertices or edges of the faces and therefore, marking just the outline of the face suffices. (b) An example of demarcated vertices/edges on an image of Humayun's Tomb.

3.2 Orientation Constraints

To fix the direction of the face, a set of two or more sets of parallel lines on the face, that help compute vanishing points and a vanishing line, is required. The vanishing line helps find the normal to the plane, thereby fixing the orientation of the face.

If the vanishing line l_v for a plane can be identified in the image, the normal $\mathbf{n}$ to the plane is obtained simply as:

$$\mathbf{n} = \mathbf{R}^T(\mathbf{K}^T l_v) \qquad (2)$$

where $\mathbf{R}$ and $\mathbf{K}$ denote the camera rotation and internal matrices respectively. Orientation constraint equations will typically look like:

$$(\mathbf{R}^T\mathbf{K}^T l_v)^T(\mathbf{V_i} - \mathbf{V_{i-1}}) = 0, \forall i = \{2, 3 \ldots s\} \qquad (3)$$

In Fig. 4, orientation constraints for the blue face will be:

$$(\mathbf{R}^T\mathbf{K}^T l_v)^T(\mathbf{V_2} - \mathbf{V_1}) = 0 \quad (4)$$

$$(\mathbf{R}^T\mathbf{K}^T l_v)^T(\mathbf{V_3} - \mathbf{V_2}) = 0 \quad (5)$$

$$(\mathbf{R}^T\mathbf{K}^T l_v)^T(\mathbf{V_4} - \mathbf{V_3}) = 0 \quad (6)$$

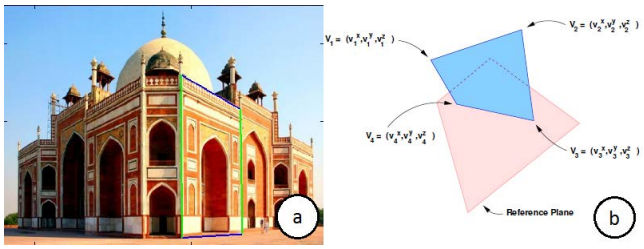

Figure 4: Orientation constraints can be computed using two pairs of parallel lines in the direction of the plane. (a) Two pairs of parallel lines on the face in question are visible. (b)Incidence constraints arise from binding a face vertex (V_3^x of the blue face) to lie on the pink face.

3.3 Incidence Constraints

The face under consideration lies in the set of parallel faces that result from the intersection of the family of planes along the given orientation and bounded by the same frustum. To completely localize the face, additional constraints on its position with respect to a previously reconstructed face, are specified.

Incidence constraints are derived from the condition that one face lies on another plane whose equation is known. For a vertex $\mathbf{V}$ incident on a plane with equation

$$[\mathbf{N}^T, \mathbf{d}] = [\mathbf{N}_{rx}^T, \mathbf{N}_{ry}^T, \mathbf{N}_{rz}^T, \mathbf{d}]$$

The incidence constraint can be written as:

$$[\mathbf{N}^T, \mathbf{d}][\mathbf{V}_3^T, 1]^T = 0 \quad (7)$$

3.4 The combined matrix

We collect the frustum and orientation constraints of each face, together with the interlinked incidence constraints, into a single matrix. Let $(\mathbf{R}^T\mathbf{K}^T l_v)^T$ be represented by $\mathbf{b}$. Then the matrix for a single (quadrilateral) face typically looks like:

$$\begin{pmatrix} l_1^T\mathbf{P} & \mathbf{0} & \mathbf{0} & \mathbf{0} & l_1^T\mathbf{p_4} \\ \mathbf{0} & l_1^T\mathbf{P} & \mathbf{0} & \mathbf{0} & l_1^T\mathbf{p_4} \\ \mathbf{0} & l_2^T\mathbf{P} & \mathbf{0} & \mathbf{0} & l_2^T\mathbf{p_4} \\ \mathbf{0} & \mathbf{0} & l_2^T\mathbf{P} & \mathbf{0} & l_2^T\mathbf{p_4} \\ \mathbf{0} & \mathbf{0} & l_3^T\mathbf{P} & \mathbf{0} & l_3^T\mathbf{p_4} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & l_3^T\mathbf{P} & l_3^T\mathbf{p_4} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & l_4^T\mathbf{P} & l_4^T\mathbf{p_4} \\ l_4^T\mathbf{P} & \mathbf{0} & \mathbf{0} & \mathbf{0} & l_4^T\mathbf{p_4} \\ -\mathbf{b} & \mathbf{b} & \mathbf{0} & \mathbf{0} & 0 \\ \mathbf{0} & -\mathbf{b} & \mathbf{b} & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{0} & -\mathbf{b} & \mathbf{b} & 0 \\ \mathbf{0} & \mathbf{0} & (\mathbf{N}^T) & \mathbf{0} & -d \end{pmatrix} \begin{pmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \\ \mathbf{V}_4 \\ 1 \end{pmatrix} = 0 \quad (8)$$

This is of the form $\mathbf{AX} = 0$, with 12 equations for 12 unknowns, and can be directly solved to obtain $\mathbf{X}$ provided the equation $\mathbf{N}$ of the reference plane is known, yielding the 3D co-ordinates of the vertices $\mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3$ and $\mathbf{V}_4$.

However, error is likely to creep in while reconstructing each face separately [10]. Since such a process calls for a tree of dependencies to be created and updated with progressive reconstruction, error tends to accumulate.

For example, if the height of a plane can be assumed computed to within an error bound of 10 %, plane A will have at the most 10 % error in height. However, plane B is reconstructed using a plane at the height of plane A as its reference, and hence, when the height of plane B from the ground is computed, the error bound will increase to 20 %. Error will keep increasing in this fashion down the hierarchy tree, resulting in huge error in connected faces that are dependent on many other faces. Such error can be minimized by following an approach which allows the simultaneous reconstruction of connected faces.

4. SIMULTANEOUS RECONSTRUCTION

4.1 Dependencies Between Faces

When solving simultaneously for a set of connected faces, the user no longer knows the equation of the reference plane. We modify the incidence constraints for such a face so that the new incidence constraints for this face includes the condition that a point on the first plane lies on the second. If the equation of the pink face in 4(b) is unknown, the incidence constraint for the blue face will be:

$$(\mathbf{R}^T\mathbf{K}^T l_{v2})^T(V_{31} - V_{32}) = 0 \quad (9)$$

It is important to note that one universal reference plane (say, the ground plane) will always be required for a set of connected faces.

4.2 Equality Constraints

Faces that are joined at one vertex can be constrained to have a common vertex in three dimensions. The incidence constraints for one of these faces can be replaced by a dependency matrix that consists of three linear equations, one for each co-ordinate of the corresponding vertices. For a pair of vertices V_1 and V_2, these equations are:

$$V_{1x} = V_{2x}, V_{1y} = V_{2y}, V_{1z} = V_{3z} \quad (10)$$

Common edges between adjacent faces are regular features of structured monuments. Equality constraints can be used to obtain perfectly matching edges in 3D, by specifying a correspondence between both end points of the edge; as is illustrated in Fig. 5.

4.3 Completeness of the constraints

To reconstruct an s-sided polygon, a minimum of $2s$ frustum constraints (as face vertices), $s-1$ orientation constraints (two vanishing directions) and at least one incidence constraint (a plane and a point of incidence) is required. If properly indicated, these constraints will necessarily result in a full-rank consistent matrix with $3s$ equations for $3s$ unknowns, that can be solved to obtain the 3D co-ordinates of each vertex. This enables the GUI to diagnose depending upon the number of constraints for each face, whether all faces are reconstructible and provide an input back to the user by checking the rank of the system of equations corresponding to each face.

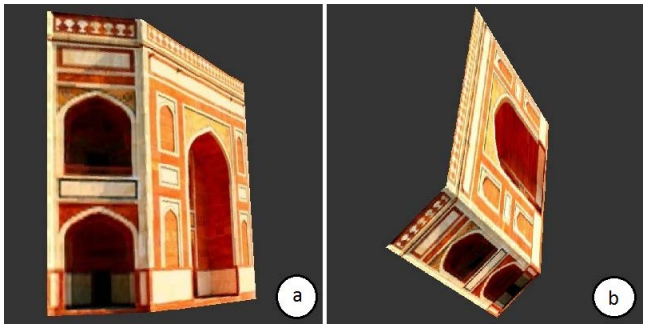

Figure 5: (a, b) The matching edges method enforces the constraint of corresponding edges between adjacent faces, giving perfect results.

While simultaneously reconstructing a set of connected faces, frustum constraints are mandatory for each face. Dependencies between faces can, however, induce new constraints that are tighter than and may replace the orientation or incidence constraints. Dependency equations are added to the matrix (8) created above, and the solution is obtained so as to reduce least squares error. Clearly, the more the dependencies and constraints marked, the better the solution.

For example, multiple incidence constraints can be specified to accurately fix the position of a face. Specifying perfectly matching edges between adjacent faces gives constraints that are tighter than, and enable the user to do away with, incidence constraints for one of the faces. A vanishing direction could be fixed once and for all by specifying a pair of vanishing lines to improve accuracy. This is extremely useful, for example, in reconstructing a set of vertical planes, since the vertical vanishing direction needs to be specified once and the same vanishing point (or pair of parallel lines) can be used henceforth for all other planes.

5. A DATA STRUCTURE FOR SYMBOLIC REPRESENTATION OF USER CONSTRAINTS

To reconstruct faces (connected or unconnected) the user specifies, for each face:

- **Frustum Constraints-** *Mark the vertices of each face in a cyclic order.*
- **Orientation Constraints-** *Give two pairs of parallel lines in the plane of the face.*
- **Incidence Constraints-** *Give at least one vertex on the face along with a plane of incidence for that vertex.*
- **Other Constraints-** *Specify some vertices or edges of one face which correspond with some vertices or edges of the second.*

5.1 Points and Lines: Basic Data Structures

- **Points**
  A point is a set of (x,y) co-ordinates on the two dimensional image plane, defined by user-clicks and incrementally stored in a random access dynamic list. The list of points is the ordered set $\mathbf{v} = \{v_i | i = 1, 2, ...n\}$ of $v_i = (x_i, y_i)$ co-ordinates for every 2D point v_i in the image.
- **Lines**
  A line is a user-defined pair of pointers to two *'Point'* values (endpoints of the line in the image) in the list of Points. Each line is stored incrementally in another dynamic random access list. The list of lines is represented by $\mathbf{l} = \{l_i | i = 1, 2...m\}$, where $l_i = (v_{y_i}, v_{z_i})$ is the representation of a line as a pair of its end-points.

5.2 Faces: Frustum Constraints

A face in three dimensions is defined by a cyclic ordering of its vertices. Frustum constraints are obtained by specifying the 2D 'Points' corresponding to each vertex of the face on the image. Each 'Face' is stored incrementally in a dynamic list represented by $\mathbf{F} = \{f_i | i = 1, 2...p\}$, where $f_i = (v_{i_k} | k = 1, 2, ...s)$ is the representation of the s-sided polygon as a cyclic ordering of its vertices on the image.

It should be noted that the user does not need to explicitly specify frustum constraints- marking out the vertices of every face gives sufficient information to extract these constraints.

5.3 Faces: Orientation Constraints

Orientation constraints require a vanishing direction for every plane, which can be obtained using a corresponding vanishing line.

A vanishing point lies at the intersection of a set of parallel lines on the plane of the face, and the vanishing line of a face connects these vanishing points. The vanishing line can therefore be obtained by getting two vanishing points using parallel lines in any two directions on the plane of the face.

The user must pick two or more pairs of parallel lines in two distinct directions along the plane of the face. Two pairs of Pointers to two sets of *'lines'* are stored incrementally in a new dynamic list (random access not necessary); $\mathbf{O} = \{O_i | i = 1, 2...p\}$ where $O_i = ((l_{i_j} | j = 1, 2, ...r), (l_{i_k} | k = 1, 2, ...s)$ represents two sets of parallel lines in two different directions along face f_i. Thus, the list of orientation constraints has one entry corresponding to each face, indexed by face.

The user can optionally indicate that the face under consideration is a rectangle, in which case the orientation constraints are formed automatically. The user may not provide any orientation constraints and give extra incidence constraints to make up for the loss in rank. In such a case, an empty character is stored at the corresponding position in the orientation constraint matrix.

5.4 Faces: Incidence Constraints and Other Dependencies

Incidence constraints require the user to specify a point and a face of incidence. These constraints are stored as sets of w (pointer to 'Point', pointer to 'Face') pairs in a new dynamic list, $\mathbf{I} = \{I_i | i = 1, 2...p\}$, where $I_{i_j} = (k_j, f_{l_j})$ for $j = 1, 2...w$ with j keeping a count on the members of the set, k pointing to the number of this point in the cycling ordering of points on face i, and $f_l \in \mathbf{F}$ being the face of incidence. The user may not provide any incidence constraints at all, in which case an empty character must be stored at the corresponding position.

A tighter version of incidence constraints is the dependency list $\mathbf{D} = \{D_i | i = 1, 2....q\}$ and $D_i = ((f_x, j), (f_y, k))$

representing corresponding points between image-vertices j and k of faces f_x and f_y.

6. THE BACK-END ENGINE

6.1 The Compiler

Once all constraints have been specified, the user sends for reconstruction by pressing the 'Build' button. The crux is to solve it all 'together', so a single matrix is built out of all constraints given so far.

To build this final matrix, the compiler parses these constraint lists one by one.

- **Frustum Constraints:**
  The compiler begins by parsing the list of faces **F**, accessing the 2D co-ordinates of face i for $i = 1, 2...p$. $f_i[k] = (v_{i_k}) = (x_{i_k}, y_{i_k})$ gives the co-ordinates of the k^{th} vertex of face i. These are used directly to construct equations for edges l_1, l_2, l_3, l_4 ... to obtain the frustum constraints of each face via eqn 1.
  From frustum constraints, $2s_i$ equations are derived for each face

- **Orientation Constraints:**
  From the list **O**, equations of sets of lines in the image, that map to parallel lines in 3D, can be formed by extracting the values of their end-points from **L**. $O_i[j]$ points to the j^{th} parallel line for the i^{th} face (similarly for the k^{th} parallel line), which is an indexed object in the list **L**. $O_i[j] = l_{i_j} = (v_{i_{j_1}}, v_{i_{j_2}})$, thus the equation of this line can be obtained using its end points $(v_{i_{j_1}}, v_{i_{j_2}})$, and the intersection of two or more such lines mapping to parallel lines in 3D gives a vanishing point in that direction. Two or more such vanishing points yield a vanishing line, which is used to obtain an equation to the normal of the plane via eqn 2. Once obtained, the equation to the normal directly gives incidence constraints via eqns 4, 5 and 6.

  Provided vanishing lines are known, $s_i - 1$ equations are derived for each face.

- **Incidence Constraints:**
  I is parsed, and for the i^{th} face, $I_{i_j} = (k, f_l)$ directly gives the j^{th} vertex of incidence as $v_{i_k} = F_i[k]$, and the corresponding face of incidence as f_l. Eqn 7 is then applied to this set of (vertex, plane of incidence) to obtain incidence constraints for the plane demarcated by that vertex.

  Using incidence constraints, typically at least 1 equation is derived for each face.

  The frustum, orientation and incidence constraints are appended to generate the matrix of equation 8 for every face. All these matrices are appended together into a single large system of equations.

- **Other Constraints - Matching vertices and edges:**
  The list containing vertices of two planes that map to the same point in three dimensions as well, is parsed to give $D_i = ((f_x, j), (f_y, k))$. This implies that v_{x_j} and v_{y_k} must be equated to to create a dependency matrix via eqn 10. All dependency matrices thus created are appended at the end of the entire system of equations to generate one large matrix that must be solved to yield the 3D co-ordinates of every point.

6.2 Solving the constraints

We move from creating separate constraint matrices and solving them for every face, to an approach that couples all constraints together with the dependencies into a single system.

Constraints for all faces are bundled together to create a huge matrix that will solve to yield each vertex. All special dependencies, such as perfectly corresponding vertices or edges, are added onto this matrix. Like the previous case of reconstructing face-by-face, the matrix is of the form $\mathbf{AX} = \mathbf{0}$ where **A** is the constraint-cum-dependency matrix, **X** is a column matrix comprising of three dimensional co-ordinates of every face inserted one after the other. The last element of the vector **X** is 1 (used for scaling).

As an example, suppose the number of faces being reconstructed is n, and let s_i denote the number of vertices of face i. Then there are a total of $\sum_{i=1}^{n} s_i$ vertices whose three dimensional co-ordinates must be computed. The matrix X consists of the required x, y and z co-ordinates of each vertex one after the other, so in all there are 3 x $\sum_{i=1}^{n} s_i$ entries in matrix X, excluding the last entry which is fixed at 1.

In order to compute the null space **X**, the minimum number of row entries in matrix A must be $3\sum_{i=1}^{n} s_i$ [10], with a minimum of $3s_i$ equations for face i as was shown above. Then the value of X comes from a direct computation of the null space of matrix **A**.

The advantage over piece-wise reconstruction lies in that is scheme is designed to capture and solve dependencies well. If the user has specified additional dependencies, there will be more than $3\sum_{i=1}^{n} s_i$ entries in **A**. In such a case, the null space does not exist, unless in the highly improbable situation of the rank of the constraint-cum-dependency matrix reducing to $3\sum_{i=1}^{n} s_i$. For a rank greater than $3\sum_{i=1}^{n} s_i$, we proceed to find the $3\sum_{i=1}^{n} s_i$ required values by using least squares optimization on the matrix **A**.

7. RECONSTRUCTION RESULTS

7.1 Mapping Textures

The least squares optimizer yields three-dimensional co-ordinates of each of the vertices of every face specified. Textures extracted from the image itself are processed for perspective correction to avoid undesirable effects obtained when non rectangular textures are mapped onto rectangular planes. Perspective correction is done by matching barycentric coordinates of the texture to the plane. Textures thus obtained are applied on the corresponding reconstructed faces to get a realistic effect.

7.2 Experimental Verification

We conclude with a few results we obtained on experimenting with the GUI and the data structure, which clearly demonstrate that the solution obtained improves as additional constraints are introduced (Fig. 6, 7). Fig. 6 (a) illustrates result of reconstructing a part of the Taj Mahal. Fig. 6 (b) is a simple reconstruction of all rectangular faces demarcated by vertices, while Fig. 6 (c) shows a significant improvement in results achieved by the added constraint of having fixed a vanishing direction for all three planes. Fig. 6 (d) shows almost perfect reconstruction obtained by giv-

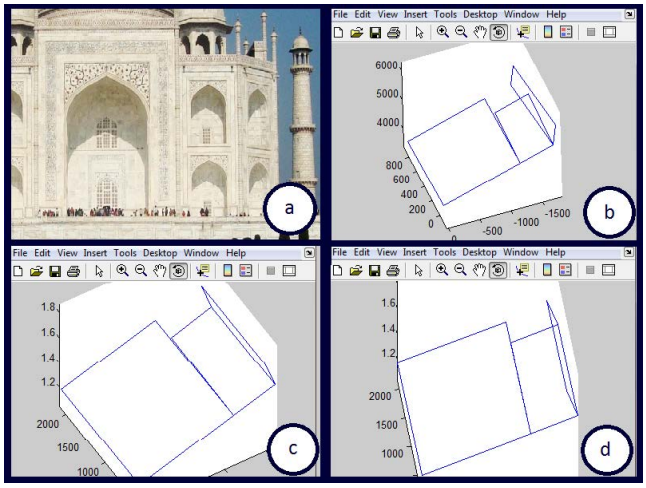

Figure 6: (a) Wireframe models - the result of reconstructing a part of the Taj Mahal. (b) If no common vanishing constraints are specified for faces, the orientation of each rectangular face must be computed using its edges. Slight errors in marking edges often lead to huge reconstruction errors, as can be observed. (c) Most of these errors were eliminated by fixing a vanishing direction once and for all, and used the same pair of parallel lines to give vanishing constraints for every face lying in that direction. (d) Additional constraints such as common edges and fixing the incidence of points on adjacent planes helped get extremely accurate reconstruction results.

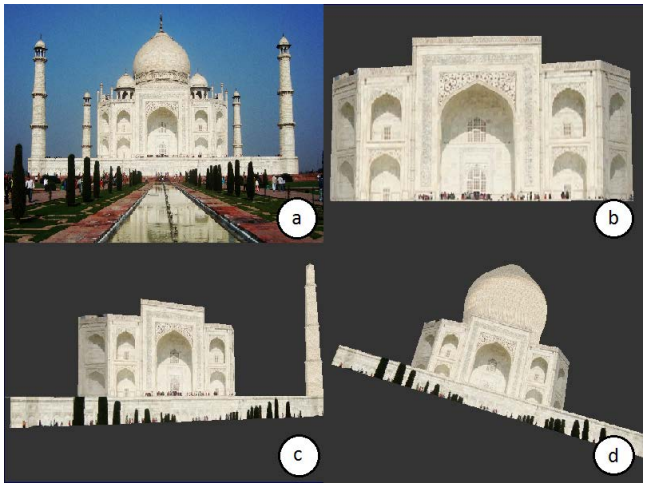

Figure 7: Reconstruction results we obtained for the Wellcome Trust Building at Dundee have already been displayed in the introduction to give a generic idea of our work. (a, b, c, d) We used this method to render several other buildings, among them the majestic Taj Mahal and obtained aesthetic and accurate results.

ing additional constraints like common edges or fixing the incidence of points on adjacent planes.

8. CONCLUSION

The major contribution of the paper is our data structure, at the back end of the GUI that allows a symbolic capture of all constraints in a robust and dynamic manner. Modified constraints are aggregated in this structure in such a way that the location of a particular point on the image can be changed at any stage of the input process and it automatically reflects throughout all other input (lines, faces, etc) containing that point.

We perform a symbolic capture of user constraints which are solved together as one hits the build button. This helps avoid the propagation of incremental error by solving the entire system of equations in one go. An easy method of user input with high accuracy guarantees is the USP of our GUI. Camera calibration, however, could not be added to our optimization matrix and had to be done a priori in order to obtain a linear system of equations for reconstruction. Once the solution to this linear system is obtained, it can be used as a starting point for the iterative computation of a solution to the non-linear system consisting of camera calibration and reconstruction equations.

9. REFERENCES

[1] A. Criminisi, I. Reid, and A. Zisserman. Single view metrology. 40(2):123–148, 2000.

[2] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521623049, 2000.

[3] D. Hoiem, A. A. Efros, and M. Hebert. Automatic photo pop-up. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 577–584, 2005.

[4] Y. Horry, K.-I. Anjyo, and K. Arai. Tour into the picture: using a spidery mesh interface to make animation from a single image. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 225–232, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[5] A. Kushal, V. Bansal, and S. Banerjee. A simple method for interactive 3d reconstruction and camera calibration from a single view. In *Proc. Indian Conference in Computer Vision, Graphics and Image Processing*, 2002.

[6] A. Kushal, G. Chanda, K. Srivastava, M. Gupta, S. Sanyal, T. Sriram, P. Kalra, and S. Banerjee. Multilevel modelling and rendering of architectural scenes. In *Proc. EUROGRAPHICS*, 2003.

[7] D. Liebowitz, A. Criminisi, and A. Zisserman. Creating architectural models from images. In *Annual Conference of the European Association for Computer Graphics (Eurographics)*, volume 18, pages 39–50, 1999.

[8] M. Lourakis and A. Argyros. Enforcing scene constraints in single view reconstruction. In *Proceedings of the EuroGraphics 2007 conference*, 2007.

[9] P. Müller, G. Zeng, P. Wonka, and L. Van Gool. Image-based procedural modeling of facades. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, 2007.

[10] S. Sanyal. Interactive image-based modeling and walkthrough planning, 2006. Doctoral Thesis,

submitted to IIT Delhi.

[11] P. F. Sturm and S. J. Maybank. A method for interactive 3d reconstruction of piecewise planar objects from single images. In *In Proc. BMVC*, pages 265–274, 1999.