A framework for User-Assisted Sketch-Based Fitting of Geometric Primitives

Davide Portelli
VCL - ISTI, Pisa
davide.portelli@gmail.com

Fabio Ganovelli
VCL - ISTI, Pisa
fabio.ganovelli@isti.cnr.it

Marco Tarini
Univ. dell'Insubria, Varese
marco.tarini@isti.cnr.it

Paolo Cignoni
VCL - ISTI, Pisa
paolo.cignoni@isti.cnr.it

Matteo Dellepiane
VCL - ISTI, Pisa
matteo.dellepiane@isti.cnr.it

Roberto Scopigno
VCL - ISTI, Pisa
roberto.scopigno@isti.cnr.it

ABSTRACT

In this paper, we present a user-assisted sketch-based framework to extract hi-level primitives (e.g. columns or staircases) from scanned3D models of an architectural complex. The framework offers a unified level of representation of the hi-level primitives, so that new types of primitives can be easily added as plug-ins to the main engine. Primitives are fitted with a user-assisted procedure: the user suggests the approximate location of the primitive by means of simple mouse gestures, sketched over a rendering of the model. The viewpoint that was selected prior to the sketching is also taken in consideration as hints on the orientation and size of the primitive. The engine performs a GPU assisted fitting and the result is shown in real time to the user. Ad-hoc gestures cause the system to add and fit groups of primitive in one go (e.g. a column complex, or a sequence of windows).

Keywords: 3D segmentation, fitting geometric primitives

1 INTRODUCTION

Before the advent of scanning devices, 3D digital models of architectural buildings were mainly obtained via manual modeling. This operation is typically guided by 2D data, like sections and prospects. A modeler usually proceeds by decomposing the structure in a set of primitives, then “builds" the model by adding the primitives.

The increasing availability of 3D range scanning devices, the development of software increasingly efficient and user-friendly for the creation and manipulation of complex 3D digital models and the drop of the scanning technology costs, are the main reasons of the recent fast proliferation of scanning campaigns for the acquisition of the shape of real world objects. Along with other application fields, 3D range scanning [CM02] is increasingly used in architecture.

The result that can be obtained using 3D scanning, organized as clouds of points or as triangle meshes, is a far more accurate description of the actual shape of the building or the faćade then the one obtained with manual modeling, but it does does not carry any information on what the object or its parts are.

The possibility to decompose an architectural model in a set of higher level primitives (which are very often repeated on the same faćade) is extremely important for a number of possible applications: analysis, archival, comparison with other models. This would combine the flexibility of direct 3D modeling to the accuracy of 3D scanning. The primitive extraction can also be applied to different approaches aimed at recovering the 3D information of buildings [BSZF99, SB03].

In this paper we present a framework for a user-assisted extraction of geometric primitives. The intervention of the user is limited to a few sketches over a rendering of the low-level model. The sketches roughly define the size, orientation and position of the intended primitive. The approach is robust with respect to incomplete geometry, and is also capable automatically identifying and extract repeated instances of a single primitive. As a result, the user can decompose a complex 3D models in a few minutes, without the need of picking accurate positions, and obtain good results.

The next subsections will briefly review several state-of-the-art automatic and semi-automatic approaches for primitive fitting. Then, the proposed framework will be shown. A discussion on the obtained result will be presented before the conclusions.

2 RELATED WORK

The literature on reverse engineering from 3D data is vast. In this section we will only give a brief overview of the approches more closely related to our domain. We will subdivide the approaches in *segmentation* approaches and *fitting* approaches. In the first class we put the approaches for finding low level features, such as lines, planar regions or high curvature points in the 3D dataset. These methods do not aim to give the information about the nature of an object, instead they try to convert a raw geometric description (i.e. a point cloud or a triangles mesh) into a more abstract description. Usually these techniques rely on on discrete local curvature operators to detect features [OBS04, WB01, HHW05, CSAD04], the biggest challenge being making the algorithm robust to geometrical noise. Extracting features from an irregular 3D point cloud or from triangle mesh produced by 3D scanning is made difficult by the inherent ambiguities of the task as well as by the presence of geometrical noise, holes in the model, and other inconsistencies. Once basic geometric features such as lines and planes have been found, they can grouped to describe higher level structures. In [SWWK07] this is done by creating a graph of relations where sub parts of the graph define structural elements and arcs describe the constraint between elements. In the class of fitting approaches we place those methods which use parametrized description of higher level primitives and try to "place" them in subparts of raw data by means of minimizing an error function. The function being minimized can be defined ad hoc for a given type of primitive (e.g. planes, cones, cylinders) [MLM01, Ben02]. In the general case, however, it consists in some form of distance between the surface of the primitive being fitted and the real model. In [USF08] the authors give a GML parametric description for the model being fitted and the minimization performs the fitting using the given parameters. In [PMW*08] the case of repeated regular structure in manufactures or natural objects is studied, such as a series of windows or a snow flake. The approach uses a sequence of operation consisting of partitioning the object, finding a set of transformations between parts and clustering them to extract geometric relations in the model.

3 OUR FRAMEWORK

Our framework falls in the group of fitting approaches. Rather than trying a fully automatic approach, we aim at reducing user intervention down to few mouse gestures. The gestures are used to reduce the search domain the the minimization required by the fitting process, so to avoid the most computationally demanding phase which is often carried out with RANSAC based algorithms. Figure 1 shows the steps required for the user to identify and fit a set of 5 columns. The user selects a view of the 3D raw dataset by manipulating a mouse-controlled trackball. Then he perform a sign over the current rendering with the mouse, as shown in Figure 1-(a). With this information a column shape (in this case, a trunk of cone) is fitted over the 3D dataset – the surface shaded in red in Figure 1-(b); once the first column has been fitted the user may perform a second gesture to indicate that there is a series of similar columns, as shown Figure 1-(c); those columns are automatically fit (also see attached video).

Our main concern is to make the system easily extendable, so that the process of adding new types of primitives is easy and the system is not tied to a predefined set of primitive types. The fitting problem is approached as a generic minimization problem. All primitive types are defined likewise as a parametric *shape* function Sh, which takes as input a variable amount of intrinsic and extrinsic parameters, and returns in output a set of 3D points. More precisely, $Sh(x_1, \dots x_n, RT) = \{p_1, \dots, p_m\}$, where m is the number of produced samples on the surface, and n is the number of scalar *intrinsic parameters*, and RT is a roto-translation matrix, or *extrinsic parameters*, which specify the location in space of the shape. Specification of a primitive type also include an interval for each intrinsic parameter. Note that choosing to express the shape as a parametric point set does not allow to exploit non geometric information that we may know about the primitive. On the other hand it gives generalization of the primitive description and allow us to write a extendable framework.

While any primitive type has the same extrinsic parameters, the intrinsic parameters vary from type to type, both in number and in range of values.

For example the primitive type *Column* is defined by the shape function:

$$Column(r_{bottom}, r_{top}, len, RT)$$

where r_{bottom} and r_{top} are the two radii of trunked cone with length len.

The minimization problem can now be defined independently of the type of the primitive being fitted:

$$\begin{array}{ll} min & Err(Sh(x_1..x_n, RT), M) \\ & x_i \in Constr(i) \end{array} \qquad (1)$$

where Err is a measure of the difference between the primitive and the scanned model and $Constr(i)$ is the constraint defined for the parameter i (for example in the case of the column we have $0 < r_{bottom}, r_{top}, len$).

Figure 2 show a scheme of the whole process. The user select a shape and perform a mouse gesture so providing the input for the module that computes a first estimation of the parameters. Then the minimization process start by sampling the surface generated by the parameters, computing its distance from the model and updating the parameters to decrement the error, until a satisfactory fitting is found.

Being that our method relies also on user intervention, it may reminds to many user assisted techniques for segmentation of medical datasets for which a vast literature is available (see [PXP00] for a recent survey). However there are important distinctions both in finalities and in adopted strategies. The first difference is that for architectural manufactures we do not need a tool for supporting the recognition of a shape, as usually is for medical images, but only a tool for converting a raw description (a point cloud) in a structured one (union of architectural elements). Many techniques in medical segmentation are based on energy minimization methods but the exploitation of a known parameterization of the object to segment brings less advantages than in our case for the simple reason that human organs are much

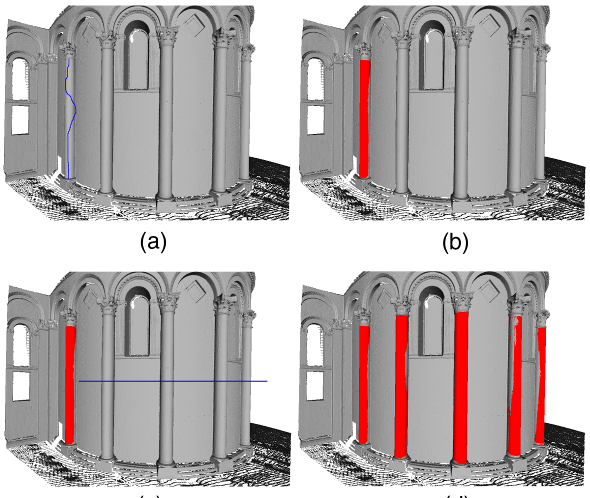


Figure 1: application example of fitting of 5 columns.

more difficult to parametrize than architectural buildings.

3.1 From gesture to parameters estimation.

The goal of using the mouse gesture is to reduce the search space for in the minimization process. However, in order to be effective, the gesture must be simple to do and not necessarily precise. The first task is to interpret a 2D mouse gesture in a selection of a 3D subpart of the original 3D data (mesh or point-cloud).

Figure 3.(a) shows the example of the column where the sign of the mouse is partly over the column (shaded in red) and partly over the background (shaded in green). In Figure 3 we see how the selected points are distributed in space.

We compute the distribution of the distance of these points from the viewer and use it to remove what we consider to be outliers (see Figure 3.(c)), in the assumption that the majority of points will be coherently on the part of the dataset that corresponds to the primitive being fitted. Then we take the bounding box of these points to infer an initial estimation of parameters for the shape. In the most general case, i.e. with no assumption neither on the type of dataset nor on the type of primitive, the only information that we could use from the bounding box is its volume, so we can solve a minimization problem:

$$\boldsymbol{min}\ ||Volume(Sh(X_0,\dots,x_n,RT) - Volume(BBox)||$$

and use the solution as the initial estimation for the problem 1. The computation of a solution is made less computationally intensive by taking in account the view transformation that was chosen by the user in order to have a suitable view of the intended feature:

- the view transformation selected by the user before he performs a mouse gesture is assumed to be such that the feature has a natural orientation (e.g. the column is not upside down in screen space);
- similarly, the intended instance of the primitive is oriented, in view-space, as facing the camera.

taking advantage of these reasonable assumptions, we infer a correspondence between the frame centered in the center of the bounding box and oriented with its sides, and the frame where the shape is defined for the initial to obtain the parameters estimation.

3.2 Minimization

At a first glance, we could take the function to minimize, referred as Err in the problem 1 as the sums of Euclidean distance between the primitive and the model. Unfortunately this is not enough, because we may have architectural elements which subparts are also instances of the same type of element. For example a portion of a plane is also a plane and a portion of a column is also shaped as a column. Of course this also depends on the definition of the primitive types. Consider for example how a column including a basement and the capital we would not have these ambiguities (however that primitive type could not be fitted, for example, over a 3D point cloud featuring a broken column, a case for which we would need an ad hoc primitive).

For these reasons, we aim at the *maximal* portion of dataset that matches with the primitive. Therefore we redefine our error function as:

$$Err(Sh,M) = \frac{1}{Area(Sh)} \sum_{j=0}^{j<k\ \lfloor Area(Sh) \rfloor} max(t, w_i\, D(s_i,M))^2 \tag{2}$$

where $s_i,\ i = 0\dots k$ is a sampling of the surface of the primitive, $D(s_i,M)$ is a measure of the distance from s_i to the model M, t is the minimum error that is assigned to each sample to smooth out the contribution due to the the noise of the scanned model and w_i is a $[0,1]$ weight associated with the i^{th} sample that is used to discard outliers that are created is the model misses portion of surface that are represented in the shape (e.g. a column with a missing piece). Essentially Err takes into account the distance between the primitive and the model **and** the area of the shape and decreases both if the distance decreases and if the area grows. Note that the distance measure is squared in order to express both parts of the fraction in the same scale, and that the number of samples is proportional to the area so that each sample accounts approximately for a constant area.

Computing $\mathbf{D(s_i,M)}$. We can define the distance function as the Euclidean distance to the closest point on M,

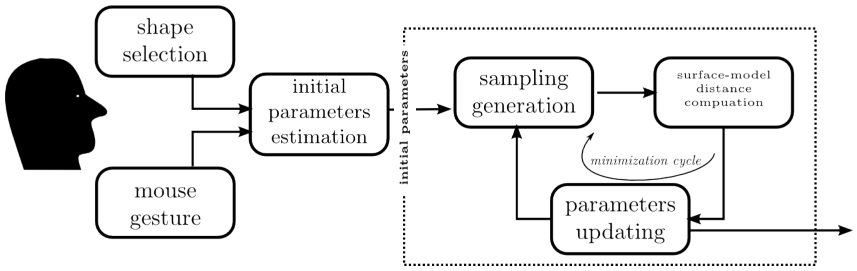


Figure 2: A scheme of the fitting framework

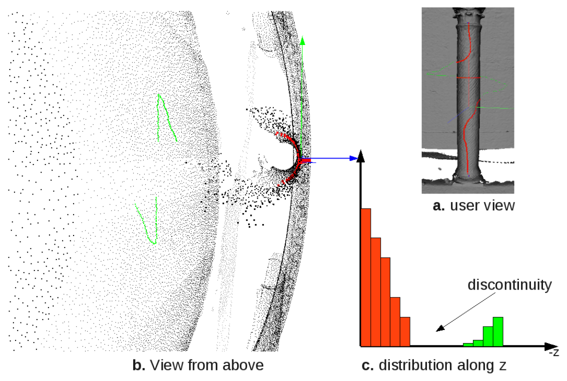


Figure 3: From mouse gesture to initial parameters.

just like in classic IPC algorithm [BM92] D. However, since we have an estimation of the normals both for the shape and for the model, we can achieve better results including the normals in the estimation and defining the distance as:

$$D(s_i,M) = min\ D(s_i,p_i),\ p_i \in M \tag{3}$$

where:

$$D(s_i,p_i) = E(s_i,p_i) + \frac{\alpha\ (1-\vec{n}_1\ \vec{n}_2)^{2\beta}}{E(s_i,p_i)+1} \tag{4}$$

the function D is simply the Euclidean distance E plus a positive bounded contribution En (the right part of the sum) which accounts for the normals in the distance computation. The expression is formulated so that the weight of the normal only comes into play where the two points are close to each other and the magnitude of their contribution is proportional to the angle between them. It can be easily seen that the maximum contribution (found when $E(s_i,p_i) = 0$ and $\vec{n}_1\ \vec{n}_2 = -1$) is $\alpha\ 2^{2\beta}$. We can set β to determine how fast the contribution of this term grows and α to relate the term to the density of the dataset. The value of α is important because the contribution of the term must be proportioned to the density of the sampling to affect the minimization. Typically a good choice is to set it to the average inter point distance. So if, say, $\beta = 2$ and the average inter point distance is 0.5, we will have a term that may increase the distance estimation from the Euclidean value at most by $0.5\ 2^{2\ 2} = 8$, when points with

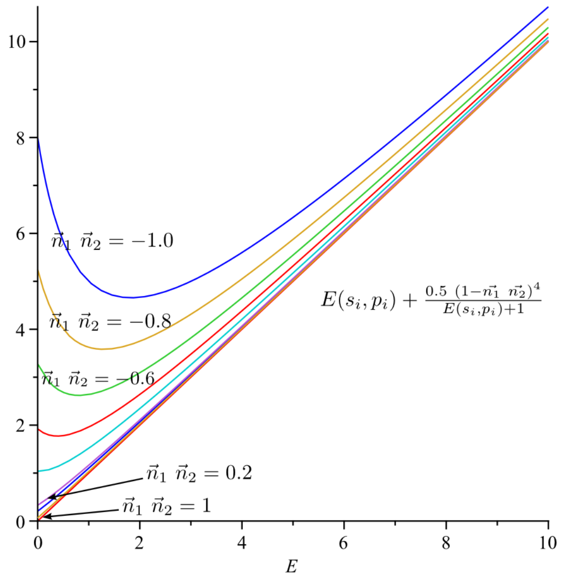


Figure 4: A plot of the distance function for $\beta = 2$, $\alpha = 0.5$ and several values of angle between the points' normal.

opposite normals coincide. Figure 4 shows a plot of En for different values of the product $\vec{n}_1\ \vec{n}_2$ and $\beta = 2$ where this behavior can be observed.

Al thought the distance function En is a $5D$ function, the closest point on M with respect to En can be found using only data $3D$ space indexing data structures for Euclidean distance by:

1. finding the closest point p_i with respect to the metric E

2. taking the closest point with respect to D among those which euclidean distance is less than $D(s_i,p_i)$.

It is easy to see that the algorithm returns the closest point w.r.t. D, because

$$E(s_i,p') > D(s_i,p_i) \rightarrow D(s_i,p') = E(s_i,p') + En(s_i,p') > D(s_i,p_i)$$

Minimization cycle. At this point we have defined both the parameters and the function to minimize and may apply any non linear minimization algorithm to find a hopefully optima solution. However, we exploit the knowledge of a closed form solution for the extrinsic parameters alone [BM92] and decompose the minimization cycle in three steps:

1 for each sample in the shape Sh, find the closest point in the model

2 find the rototranslation that minimizes the squared distances between all the pairs (only explicit parameters involved)

3 iterate a non linear minimization procedure (only implicit parameters involved)

- if $Err(Sh,M)$ is under a user selected threshold return, otherwise goto 1

We used both *Levemberg-Marquardt* [Lou09] and *Newuoa* [Pow08], with similar results.

4 EXTENDING TO MULTIPLE INSTANCES

When an architectural element has been fitted, it is likely that other similar elements (i.e. of the same type and size) are present. Examples are the columns, the steps of a stair of a series of window. Therefore we wanted to spare to the user to repeat the same mouse gesture for all the elements an simply make a single gesture which says *here there are other elements of this type*. As shown in Figure 1.(c), the gesture required is two mouse clicks to define a line segment. From this gesture the initial parameters for all the other columns are find and the minimization process just described is launched on each instance.

4.1 From gesture to parameters estimation.

Since we have fitted the first element, we already have the estimation of the initial implicit parameters for the other instances of the same type of element. The user may define a segment $(seg(t)_x, seg(t)_y$ to indicate where these other instances are, as shown in Figure 6. Therefore the information we need to extract is *how many* other elements there are and, for each one of them, an estimation of the extrinsic parameters. Furthermore we can exploit the fact the in architectural manufactures repeated elements usually differ by a translation but are oriented in the same way and reduce the missing extrinsic parameters to a translation.

In principle we could sample the segment and, for each sample, launch the optimization taking the projection of the sample onto the scene as a starting point for translation. Unfortunately the minimization process requires few seconds to complete and therefore we need to reduce the set of candidates translations.

We harness the rasterization process in order to quickly reduce the candidate translations. More precisely we exploit the z-fighting artifact. The z-fighting

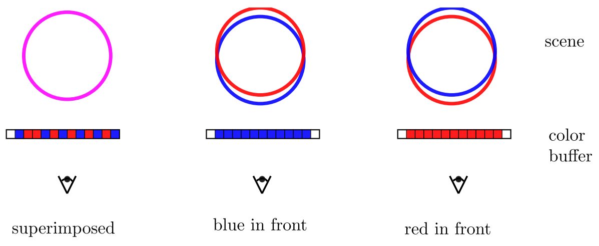


Figure 5: A schematic representation of z-fighting quantification

is the rendering artifact that happens when the depth values of the rasterization of different polygons falls in a range of values close or under the precision of the z-buffer, so that the pixel are evenly written by the conflicting polygons.

The idea is that if we consider the $3D$ point $(seg(t')_x, seg(t')_y, seg(t')_z)$ where $(seg(t')_x, seg(t')_y$ are 2D points belonging to the segment and $seg(t')_z$ is the projection on the model and render an instance of the shape translated by $seg(t')$ together with the scene, the presence of z-fighting indicates a superimposition of the rendered shape with the model, at least from the view used to draw the segment.

Normally the z-fighting is a symptom of a weakness of the geometric representation or of the rendering algorithm, therefore if not quantified but only, possibly, avoided. In our approach, however, the z-fighting is an estimation of matching between a shape and the model and therefore we are interested in quantifying it.

Figure 5.(a) shows a schematic example representing the section of a column in the model (shaded in blue) and a section of the shape being fitted (shaded in red). Since they are perfectly superimposed, we see part of the pixels red and part blue, in the proportion which is essentially random and cannot be used directly to quantify the superimposition. However, if we apply a small displacement of the shape towards the viewer we see that all the pixels are red and, vice versa, displacing the shape away from the viewer the pixels will all be blue. In other words, the more the shape and the model are superimposed, the more the two renderings with the displaced shape will be different. Therefore we quantify the z-fighting as:

$$Z_{fight}(Sh,M,T_i) = \frac{\|F_{Sh}(+\varepsilon) - F_{Sh}(-\varepsilon)\|}{F_{Sh}}$$

where $F_{Sh}(_{+/-}\ \varepsilon)$ is the number of fragments belonging to the shape when is displaced by $_{+/-}\ \varepsilon$ and F_{Sh} is the number of fragments of the shape Sh alone. The upper half of Figure 6 shows two examples of a fitted shape, a column and a step, and the segments defined by the user, while in the lower part are shown the plots obtained by setting t (the parameter of the segment with range $[0,1]$) as ascissa and (Sh,M,t), so $(Sh,M,0.5)$ is the value of the z-fighting when the shape is placed

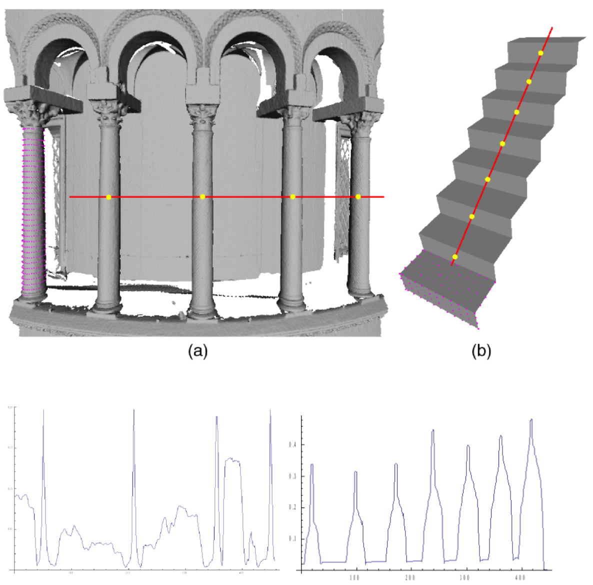


Figure 6: Example of estimation of extrinsic parameters from mouse gestures for a series of columns and a stairs.

Name	.	n.params.	meaning
Column		3	bot. rad., top rad., len.
SquareColumn		3	width, depth, leng.
Stap		3	width, depth, length
Arch		3	radius, angle, depth
Window		3	width, height,depth

Table 1: A few primitives defined to test the framework.

fig	n. pts	nI	nM	extr.(s)	intr.(s)	tot.
	2M	1	1	53	4.9	58.7
1	1.5M	5	1+1	132	18.5	1.51m
	309K	3	3	21.4	16.3	37.82
	122K	1	1	1.4	0.34	1.78
	226K	1	1	1.7	0.6	2.39
7(up)	730K	5	1+1	57.7	4.7	62.5
7(bt)	200K	4	4	12.4	0.43	12.8

Table 2: Time for fitting the primitives. **n. pts**: number of points of the model included in the user hint, **nI**: number of primitives fitted,**nM**: numbers of mouse gestures, **extr. intr.** time spent for minimization of extrinsic and intrinsic parameters, respectively, **tot.**: total time

on the projection of the middle point of the segment. Quantifying the z-fighting is very efficient because it requires only one rendering of the model and two renderings of the shape for each pixel of the segment, while the number of fragment for the displaced shape are counted by means of the hardware occlusion query.

Note that, being based on the rasterization, this technique is dependent on the window resolution, therefore it will be generally more effective with higer resolutions, simply because more translations are evaluated. It should be clear that the resolution to which we perform the zfighting computation can be different (higher) that the resolution used by the application for rendering.

5 DEFINING NEW PRIMITIVE TYPES

As stated in Section 3, our framework is not restricted to a given set of primitives but uses an abstraction layer the *sees* a primitive as a sampling of its surface dependent on a set of implicit parameters. Therefore a developer user may add new type of primitive by deriving from a base class Primitive and implementing two methods:

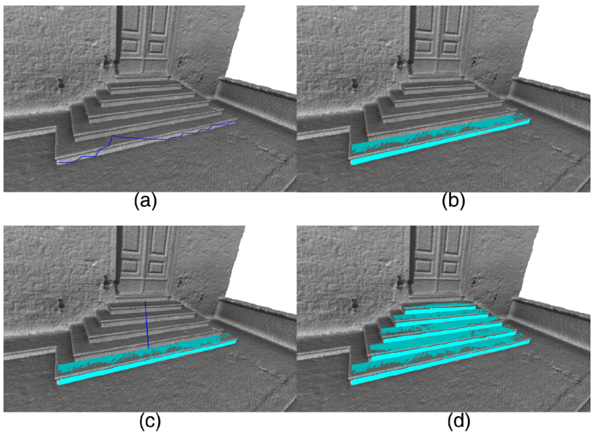
```
struct MyPrimitiveType: public Primitive{
 int N_params();// returns the number of the implicit parameters of the↩
        primitive

 points  Samples(float * params); // returns a sampling of the surface ↩
        with the passed parameters
 };
```

6 RESULTS AND DISCUSSION.

We tested our framework implementing a few types of primitives, summarized in Table 1.
We fitted the primitives to a scanned model of the Dome of Pisa and reported the timing for various runs in Table 2. Some of the runs are related to the figures we used in the paper, in which case a reference is reported in the first column of the table. The second columns reports the size of the portion of the model hinted by the user with the gesture and the third the number of primitives found with the run. The last three columns report the computation time. Note that the time for making the gestures are not reported in this table, since the experiments have been run only by an expert user and therefore not very meaningful. We took the number of mouse gestures as a measure of the user effort.

The table says that, thanks to the replication gesture, 5 architectural elements have been found with 2 gestures (second and sixth row), while where the replication is not used we need a mouse gesture per element, as for the arches refered in the last row. Conversely, indicating manually each and every element gives better starting points for the minimization and therefore the computation time are lower.

Since we performed minimization by alternating minimization of extrinsic parameters, for which we have a closed form solution, and extrinsic parameters, the time spent on each one is reported separately. The result may appear surprising at first, because the easiest side of the problem, i.e. finding the rototranslation between two sets of points, is actually the most expensive, in some cases almost by an order of magnitude. On the other hand, we must consider that the extrinsic step is performed many more times, in that we solve a mesh alignment problem for each iteration. It goes without saying that tweaking the thresholds of the minimization algorithms we may obtain different ratios between the two timings, we simply tuned their values to have robust fitting in reasonable time. The time for



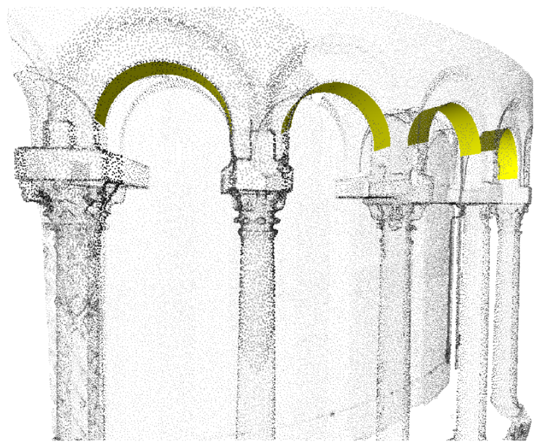


Figure 7: Above: example of selection of stair steps: (a) mouse gesture for the first step (b) fitting (c) mouse gesture for replicating the fitting (d) result. Below: final results for a set of arcs.

the analysis of the mouse gestures are not reported explicitly since they amount to few milliseconds both for the single primitives than for replication with the z-fighting computation.

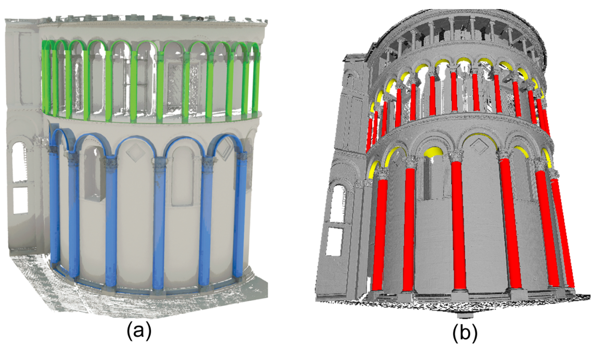


Figure 8: Fitting of columns and arches. (a) results from [USF08] (b) results of our framework on a similar model.

7 CONCLUSIONS

In this paper we presented a framework for user assisted fitting of geometric primitives on scanned architectural models. The main advantage of our framework is the generalized description of the primitive to fit that allows to include new type of primitive with minimal effort. We also devised efficient and practical solutions for enabling the user to hint the approximate position of the primitives, for improving the assessment of primitive models distance with a novel measure and for quantifying the superimposition of primitive and model by exploiting the rasterization hardware. Although requiring user assistance is in general a drawback, we made this choice motivated by two facts: 1) For a human it is very easy to indicate where an architectural component is, while is much more difficult to manually superimpose the CAD model of a component on the raw data; conversely, the analysis of raw data to locate architectural components is computationally time consuming while the minimization for finding the exact placement is an efficient process. 2) The process to digitize and entire building still take many man-hours and the reverse engineering is done once for all in a fraction of the time required for the rest of the scanning pipeline. In other words the little interaction used in this approach is hardly the bottleneck of the whole process.

From the work carried out so far, we can envisage at least two independent improvements.
The first one is to exploit more deeply the z-fight quantification to define a faster minimization algorithm only based on the rendering and therefore taking advantage of the rasterization hardware.
The second one is to derive the parametric primitive directly from a known model, that would allow a non-developer to define new type of primitives. The idea is that the user could provide a sample of the primitive as a geometric model (from a CAD or 3D scanning) and we should derive a parametric description of it, either automatically or providing a tool to do it. In this manner we could include complex shapes for which to find a parameterization is too complicated. With some approximation this would allow to include artifacts as statues when if they are copies of statues for which the digital counterpart is available.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the EG 7FP IP “3D-COFORM” project (2008-2012, n. 231809).

REFERENCES

[Ben02] BENKO P.: Constrained fitting in reverse engineering. *Computer Aided Geometric Design 19* (March 2002), 173–205.

[BM92] BESL P. J., MCKAY N. D.: A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and machine Intelligence 14*, 2 (Feb. 1992), 239–258.

[BSZF99] BAILLARD C., SCHMID C., ZISSERMAN A., FITZGIBBON A.: Automatic line matching and 3d reconstruction of buildings from multiple views. In *ISPRS Conference on Automatic Extraction of GIS Objects from Digital Imagery* (Munich, 1999), pp. 69–80.

[CM02] COLOMBO L., MARANA B.: 3d building models using laser scanning. *GIM - Geomatics Info Magazine 16*, 5 (2002), 32–35.

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. *ACM Transactions on Graphics 23* (August 2004), 905.

[HHW05] HILDEBRANDT K., HILTHIER K., WARDETZKY M.: Smooth feature lines on surface meshes. In *Symposium on Geometry Processing* (2005), pp. 85–90.

[Lou09] LOURAKIS M. I. A.: *Levenberg-Marquardt nonlinear least squares algorithms in C/C++*. Online, Apr. 2009.

[MLM01] MARSHALL D., LUKACS G., MARTIN R.: Robust segmentation of primitives from range data in the presence of geometric degeneracy. *IEEE Trans. Pattern Anal. Mach. Intell. 23*, 3 (2001), 304–314.

[OBS04] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph. 23*, 3 (2004), 609–612.

[PMW*08] PAULY M., MITRA N. J., WALLNER J., POTTMANN H., GUIBAS L.: Discovering structural regularity in 3D geometry. *ACM Transactions on Graphics 27*, 3 (2008), #43, 1–11.

[Pow08] POWELL M. J. D.: Developments of NEWUOA for minimization without derivatives. *IMA Journal of Numerical Analysis 28*, 4 (Oct. 2008), 649–664.

[PXP00] PHAM D. L., XU C., PRINCE J. L.: A survey of current methods in medical image segmentation. In *Annual Review of Biomedical Engineering*, vol. 2. 2000, pp. 315–338.

[SB03] SCHINDLER K., BAUER J.: A model-based method for building reconstruction. In *First IEEE International Workshop on Higher-Level Knowledge in 3D Modeling and Motion Analysis* (2003), pp. 74–82.

[SWWK07] SCHNABEL R., WAHL R., WESSEL R., KLEIN R.: *Shape Recognition in 3D Point Clouds*. Tech. Rep. CG-2007-1, Universität Bonn, May 2007.

[USF08] ULLRICH T., SETTGAST V., FELLNER D. W.: Semantic fitting and reconstruction. *JOCCH 1*, 2 (2008).

[WB01] WATANABE K., BELYAEV A. G.: Detection of salient curvature features on polygonal surfaces. *Comput. Graph. Forum 20*, 3 (2001).