A Public System for Image Based 3D Model Generation

Conference Paper *in* Lecture Notes in Computer Science · October 2011

DOI: 10.1007/978-3-642-24136-9_23 · Source: DBLP

2 authors:

David Tingdahl
KU Leuven
7 PUBLICATIONS **65** CITATIONS
SEE PROFILE

Luc Van Gool
ETH Zurich
1,262 PUBLICATIONS **181,578** CITATIONS
SEE PROFILE

A Public System for Image Based 3D Model Generation

D. Tingdahl and L. Van Gool

K.U.Leuven, ESAT-PSI

Abstract. This paper presents a service that creates complete and realistic 3D models out of a set of photographs taken with a consumer camera. In contrast to previous systems which produce sparse point clouds or individual depth maps, our system automatically generates textured and dense models that requires little or no post-processing before being used in a practical context. Our reconstruction pipeline features novelties such as automatic camera parameter retrieval from the web and a new intelligent view selection algorithm. The system is available to the public as a free-to-use web service. Results are made available both as a full-resolution model and as a low-resolution version that can be viewed directly in the web browser using WebGL. The system has already been deployed on-line on `http://www.arc3d.be` and currently sees a traffic of around 20 jobs per day.

1 Introduction

Recent advances in electronics and rendering software have brought us to a point where computerised 3D models are a part of peoples everyday life. Indeed, the average home is filling up with 3D capable products, such as 3D-TVs, smart phones, computers, video games, etc. We also see an increased use of 3D on the web, where standards such as WebGL [1] and XML3D [2] are becoming a part of the next-generation web browsers.

However, while the *usage* of 3D content has seen an increase and has mobilised the public, the same cannot be said about the *creation* of 3D content. Most of the 3D content we see in movies and games are still modelled using expensive scanners and manual labour. The increased amount of 3D capable products are of limited use to us if there are no easy ways for the public to create 3D content. We identify a wide gap between users and producers of 3D content, a gap we make an attempt to bridge in this paper by introducing a public, easy-to-use 3D reconstruction pipeline that outputs textured and dense 3D models.

1.1 Previous Work

To this date, there do exist a number of public methods for 3D reconstruction. ARC3D [3] is the "father" of 3D reconstruction services and is also the predecessor of the system presented in this paper. The original ARC3D has been

on-line since 2005 and produces depth maps that can be (manually) merged into a complete model. Two similar services are Microsoft's well-known PhotoSynth [4] and the more recent 3dTubeMe [5]. Both solutions generate a sparse point cloud which gives an overview of scene geometry but does not provide enough detail for photo-realistic visualisation. There do exist previous methods for dense reconstruction, but they are limited to advanced users. The Bundler software [6] is an open source package for reconstructing a sparse 3D point cloud. It can be combined with the PMVS software [7] to create dense point clouds. This method is intended for computer vision professionals rather than for the general public, and is thus non-trivial to install and to operate. In contrast, our solution generates a complete and textured 3D mesh, is fully automatic and is easy to use. The user simply uploads images to our web service which takes care of the rest.

All systems mentioned above are based on approximately the same structure from motion fundamentals [8, 9]. Local features are extracted and matched between the images. The matches are used to compute relative poses between image pairs and to triangulate a sparse reconstruction. The sparse reconstruction is then optionally upgraded to dense by a multi-view stereo algorithm. The method assumes that the internal parameters of the cameras are known, which is typically not the case if an arbitrary camera is used. However, there exist two practically usable methods for computing them. The classical methods start with a projective reconstruction which is upgraded to euclidean by finding the absolute quadric or its dual [10, 11]. See also [8] for a more intuitive, euclidean interpretation of these formulas. The euclidean upgrade is analogous to recovering the camera intrinsic parameters and can thus be used to calibrate the cameras. This method provides good result if the camera motion is general, but may break down in cases of turntable motion and planar scenes. More recently, it has been shown that EXIF meta data can be used to approximate a calibration matrix [6]. Since EXIF data my be inaccurate, one have to rely on a subsequent bundle adjustment. This method increases robustness to degenerate camera motion at the trade-off of a potential accuracy loss.

Our method does also consist of a sparse reconstruction pipeline with the subsequent upgrade to dense (Section 2). We improve the EXIF based precalibration by introducing an algorithm that searches the web for missing camera parameters (Section 3). This leads to a higher success chance for any given camera. At the end of the dense reconstruction stage, we introduce a new step that first selects suitable views for remeshing (Section 4) and then computes a textured mesh out of the depth maps of these views (Section 5). To the better of our knowledge, there are no public 3D reconstruction pipelines available that provide this functionality.

2 Reconstruction Pipeline

Our 3D reconstruction pipeline computes depth maps from a set of input images and is briefly discussed in this chapter.

Precalibration. We extract information from the EXIF meta data of the input images to create an initial calibration matrix K_{pre} for each camera. We automatically search the Web for missing information. See Section 3 for more details.

Feature Extraction. SURF features [12] are extracted from each input image. In order to limit the computational time of further steps in the pipeline, we limit the number of features to 10 000 per image.

Epipolar Geometry Computation. The SURF features are matched between all image pairs by comparing the euclidean distance between their descriptor vectors. For pairs where both cameras have an estimated K_{pre}, we compute an essential matrix $E_{i,j}$, using the 5-point algorithm [13] in a RANSAC scheme. For pairs without K_{pre}, we compute a Fundamental matrix using the 7-point algorithm [9].

Optional step: Self calibration. If there are less than two images with K_{pre}, we employ the self calibration method described in [3]. This method breaks down if K_{pre} varies throughout the image sequence or if the motion and structure is not general enough.

Sparse Reconstruction. The sparse reconstruction begins with selecting an initial view pair. To ensure a large enough baseline, we examine how well the matches between the images are related by the infinite homography [8]. The scene is reconstructed by adding views sequentially. For each new view, we resection its camera using the 3-point algorithm [14] and triangulate all matches with already reconstructed views. A bundle adjustment is then performed to minimise the reprojection errors. We optimise for the six pose parameters, focal length and two radial distortion coefficients.

Dense Reconstruction. The sparse reconstruction is used to initialise a dense reconstruction, where the aim is to reconstruct each single pixel of the input images. Using the computed cameras, we rectify the images and employ a dense stereo algorithm that uses dynamic programming to find the best disparity value for each pixel [15]. The stereo pairs are then linked together into a chain, and the optimal depth for each pixel is tracked using a Kalman filter. The result is a dense depth map for each view, where each pixel tells us its distance from the camera. In addition, the method also produces quality maps, which tells us in how many views each pixel was successfully matched. The depth maps are used as input to the mesh generation step which is detailed in Section 5.

3 Camera Parameter Retrieval

Most modern Structure from Motion pipelines make use of EXIF meta data for pre-calibration of cameras [5, 4, 6]. The EXIF data typically contains the focal

length f_e of the camera lens, given in millimetres. This can be used to obtain an estimate of the focal length in pixels: $\hat{f} = \frac{f_e w_i}{w_s}$. Here, w_i and w_s are the widths of the image (in pixels) and camera sensor (in millimeters) respectively. Assuming zero skew, the principal point at the origin and unity aspect ratio (square pixels) gives us a pre-calibration matrix of the following form:

$$K_{pre} = \begin{bmatrix} \hat{f} & 0 & \frac{w_i}{2} \\ 0 & \hat{f} & \frac{h_i}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

3.1 Obtaining the Sensor Width

While almost every camera stores f_e in the EXIF tag, many low-end consumer cameras do not store w_s. However, the information can usually be found in the technical specifications of the camera. To this end, we maintain a database of sensor widths for different camera models, indexed by the EXIF model name.

Whenever an unknown camera model is encountered, our software automatically searches the web for w_s. If the camera model is not very obscure, there is likely to be a review or product description web page that contains the required information. In practice, we perform the following:

Google search. The *model* and *make* tags of the EXIF data are combined to generate a Google search query. To better focus the search, we append the words *camera*, *specification*, *specs* and *sensor* to the query. We also add the string *dpreview* to favor web pages from the Digital Photography Website[1], which contains an extensive amount of camera reviews and specifications.

Parse for sensor sizes. The first five resulting HTML documents from Google are parsed for candidate sensor sizes. Sensor sizes can be given in two different formats: *width* x *height* in millimetres or as an imperial fraction, such as $1/1.8''$. We use a regular expressions based matcher to obtain all such potential sensor sizes in the document.

Reject invalid dimensions. The imperial fractions do not measure the diagonal of the sensor, but rather the diagonal of an imagined vidicon tube [16] centred on the sensor. Since these fractions are well defined, we use a look-up table to convert them to millimeters of width and height. A fraction that is not found in the look-up table is discarded. We also discard any measurements outside the interval $[2.0, 40.0]$ millimeters and we currently limit ourselves to dimensions where the aspect ratio is close to 4:3. This accounts for a large range of image sensors, from low end compacts to professional DSLRs. If there are still more than one sensor size left, we select the one which is closest to the word *sensor* in the document.

[1] `http://www.dpreview.com`

3.2 Retrieval Results

Uploaded content. We first evaluated the retrieval system using data uploaded by users during the course of two months. Out of 23 950 uploaded images, we removed all images which **(1)** already had a sensor width tag, **(2)** were lacking a focal length tag or **(3)** were lacking a model tag. This left us with 8 487 images that was used as input to the algorithm. These images were found to have been taken with 114 different camera models.

We evaluated the performance using two methods: per camera and per image. In the per-camera evaluation we simply search for each of the 114 camera models. In the per-image evaluation we weight the per-camera result with the number of images taken with each camera. This is to reflect the fact that some camera models are more probable than others. We use the term *retrieved* for camera models that returned any value and *correct* when the returned value was correct (verified manually). The results were as follows:

	Nr input	**Retrieved**	**Correct**	**Precision**	**Recall**
Per camera	114	76	73	96%	64%
Per image	8487	7622	7450	98%	88%

The precision is of most importance here, as a faulty parameter might harm the reconstruction algorithm. We achieve a high value both for the per-camera and for the per-image evaluation. The recall value is seen to be remarkably higher in the per-image evaluation result. This reflects the fact that most people tend to use well-known cameras which are likely to have a review on the Web. Failure cases include cellphone cameras (for which the camera specs are typically not listed on-line), old or unfamiliar camera models, and Google responses leading to web pages where several cameras are compared.

While this is indeed not fool-proof, it still greatly increases the chance of success for reconstruction systems relying on EXIF data for pre-calibration.

Bundler package. We also evaluated the tool using the camera list included in version 0.4 of the Bundler software package[2]. The list contains 268 different camera models with sensor widths. For each camera, we retrieved a value from the web, w_w, and compared it to the corresponding value from Bundler, w_b. We computed the relative error between the two as $\epsilon = (|w_w - w_b|)/w_b$, and consider the measurement to be correct if $\epsilon < 5\%$. This is motivated since the conversion from imperial fractions to millimeters is non-standard and may produce slightly different results. Out of the 268 models, we managed to retrieve a sensor width for 249 cameras. 208 of them were within the error margin. This gave us the following result: **Precision:** 84%, **Recall:**78%.

These rates are indeed inferior to the results obtained from the uploaded images. However, we note that most of the cameras on the Bundler list originate from the year of 2005 and back and does not reflect the camera models people are actually using today. The quality of the Bundler list can also be questioned, as we have found several of the values on the list to be incorrect.

[2] `http://phototour.cs.washington.edu/bundler/`

4 View Selection for Mesh Generation

Reconstructed depth maps typically have large overlaps since each scene point has to be visible in several images for 3D reconstruction to be possible. While overlap is essential in 3D reconstruction, too much redundancy will cause an unnecessary computational load for a mesh generation algorithm. Moreover, poorly taken images (for example with motion blur) may produce low quality depth maps that hurt more than help in the mesh generation process. We therefore employ a view selection algorithm that from a set of views, V_{in}, selects the best suited views for mesh generation, V_{out}.

In the literature, we find two recent publications on view selection for 3D reconstruction [17, 18]. However, both of these methods are concerned with the selection of views *before* dense reconstruction and not *after* as in our case. Here, we are not interested to the same degree in the overlap between images.

We identify three criteria for our view selection algorithm: **(1)** The number of views for re-meshing should be limited, **(2)** the views should cover the whole scene with as little overlap as possible, and **(3)** we want to prioritise depth maps of high quality, according to some measure. While **(1)** is simply an upper limit of m such that $|V_{out}| \leq m$, the other criteria require some more attention.

Coverage. We add a view to V_{out} only if it does not overlap too much with views already added. Scene elements common between views are used to determine the degree of overlap. Rather than using the dense depth maps, we avoid heavy computations by using the sparsely reconstructed 3D points to determine scene overlap. As in [18], we argue that the coverage of the sparse reconstruction is approximately the same as for the dense one. We also use their definition of a covered 3D point; a point is covered if it is seen in one or more selected views. To quantify the amount of overlap between views, we could count the number of uncovered points in each new view. However, using this number directly would bias towards views with a larger number of 3D points. This is undesirable, as the sparse 3D points are generally not evenly distributed and areas of high point densities would thus be favoured. Instead, we compute the *coverage ratio* as the ratio between the uncovered and the total number of points seen in a view. The coverage ratio ranges between 1 when all points are uncovered and 0 when no new scene content is present in the view. This is irrespective of the absolute number of 3D points.

Quality measure. Our multiview stereo algorithm generates a quality map for each depth map. The quality map shows in how many other views each pixel was seen. We use the average value of the quality map to assign a quality measure q_i to each depth map.

4.1 Algorithm

Denoting the set of sparsely reconstructed 3D points as M, and $M_V \subset M$ as the 3D points seen in view V, we iterate the following steps:

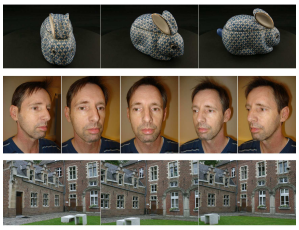


Fig. 1: Example of input images for the bunny, face and corner sequences.

		Cams	Faces	Time
Bunny	V_{in}	88	616k	45m
	V_{out}	22	607k	15m
Face	V_{in}	18	475k	9.7m
	V_{out}	6	457k	2.7m
Corner	V_{in}	8	399k	4.2m
	V_{out}	2	380k	1.8m

Table 1: Results from view selection algorithm. V_{in} is the original set of views and V_{out} is the selected set.

1. From all views we have not already checked or added, select the one with the highest quality. V_c is the set of checked views.

$$\hat{V} = \max_q(V_{in} \backslash (V_c \cup V_{out}))$$

2. Compute the coverage ratio:

$$c = \frac{|M_{\hat{V}} \backslash M_{V_{out}}|}{|M_{\hat{V}}|}$$

3. Add the view if the uncovered part was large enough:

$$V_{out} = \begin{cases} V_{out} \cup \hat{V}, & c \geq \tau \\ V_{out}, & c < \tau \end{cases}$$

4. Keep track of the views we have checked:

$$V_c = V_c \cup \hat{V}$$

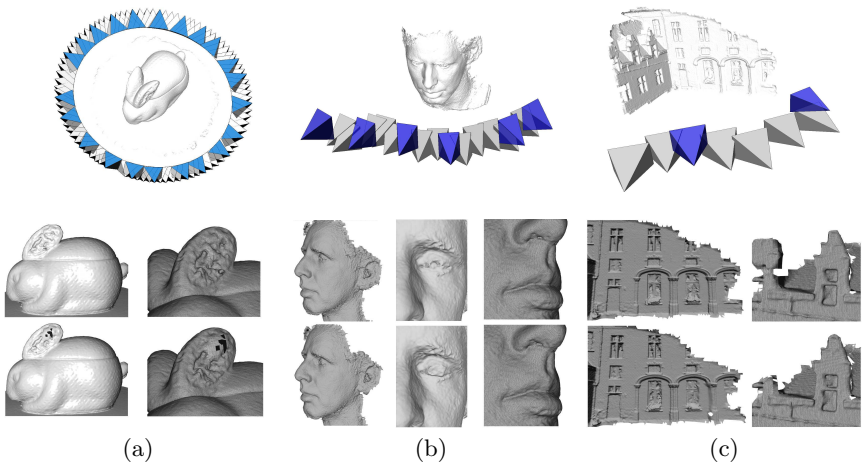
5. If $V_c = V_{in}$ all views are checked. In that case we reduce τ and start over.

Stop criteria. The iteration stops if any of the following criteria is fulfilled:

1. Max allowed views reached: $|V_{out}| = m$
2. All views added: $V_{out} = V_{in}$
3. All 3D points covered: $M_{V_{out}} = M$

4.2 Results

We evaluated the view selection algorithm on three image sequences which can be seen in Figure 1. For each set, we computed two models, one out of all views (V_{in}) and one out of the selected views (V_{out}). We compared the performance in terms of computational speed and model completeness, measured as the number



(a) (b) (c)

Fig. 2: Results of view selection algorithm for (a) bunny, (b) face and (c) corner. Top: the cameras selected for mesh generation are highlighted with a darker colour. Middle: mesh resulting from using all cameras. Bottom: result from using only the selected cameras.

of faces in the resulting mesh. It should be mentioned that the view selection algorithm itself is very fast and its computation time is negligible here.

For all experiments, we used $m = 30$ and $\tau = 0.7$ with a decrement of 0.1. The numerical results are presented in Table 1. The resulting camera poses as well as comparative views of the resulting meshes, with and without applying view selection, are shown in Figure 4.1. The visual difference is negligible for most parts of the meshes. Discrepancies tend to occur in poorly reconstructed areas, such as in the ears of the bunny and in the eye of the face. The eye is actually slightly better reconstructed from V_{out}, this is likely due to the inclusion of the quality measure. In the corner sequence, we can see that areas on the edge of the model which are not covered well by the sparse reconstruction are lost. The marginal loss of geometry is however well motivated by the large improvement in computational speed.

5 Mesh Generation

This section describes how we create a textured mesh out of a number of depth maps of a scene. Given a set of suitable views from the view selection algorithm in Section 4, we clean the depth maps and create a mesh using a state-of-the-art remeshing algorithm. Finally, we use the input images to texturise the model and create a WebGL-ready version by downsampling the mesh.

(a) (b)

Fig. 3: (a) Creating a mesh out of depth maps. Top left: Original noisy depth map. Top right: The depth map is cleaned and back projected into a range grid (Bottom left). All range grids are combined and remeshed (bottom right). (b) The mesh is textured to produce the final result.

5.1 Depthmaps to Model

The depth maps belonging to V_{out} are combined into a complete 3D model by applying the following steps, illustrated in Figure 3:

Filtering of depth maps. Since it is more efficient to work with 2D images than with 3D meshes, we perform as much of the filtering as possible in image space. By using the corresponding quality map, we first remove all pixels in the depth map not seen in at least three views. We then find the largest connected component and remove all pixels not belonging to the hull of this component. Finally, an erosion is performed to get rid of possibly noisy border pixels.

Depth map to 3D. Each depth map is back projected into a 3D range grid. A rough mesh is obtained by connecting neighbouring vertices in the grid. We measure the angle between each triangle normal and the optical axis of the camera and remove all triangles where the angle is greater than 80°. We also remove sliver triangles, where the length of the smallest side is less than 2 times the length of the largest. Finally we get rid of all connected components having a face count of less than 10% of the largest component.

Poisson reconstruction. The vertices from all depth maps are merged and used as input to the Poisson reconstruction algorithm [19]. This method casts the remeshing into a spatial Poisson problem and produces a watertight, triangulated surface. The Poisson reconstruction algorithm assumes a complete and closed surface, and covers areas of low point density with large triangles. To get rid of these "invented" surfaces we remove all triangles with a side larger than five times the mean value of the mesh.

Fig. 4: Resulting model reconstructed from a DSLR camera.

Applying texture. We compute texture coordinates for each face by determining in which cameras the face is seen. To handle occlusions, we examine if the rays between the camera centre and face vertices pass through any other face of the model. Using an octree data structure and employing line and frustum culling makes this computationally feasible. If the face is seen in more than one view, we project the triangle into each image and select the one with the largest projection area. This favors fronto-parallel, high resolution and close-up images.

Downscaling of the model. The remeshed model is large and may contain hundreds of thousands of faces. While such a high detail is desirable in some contexts, the amount of data makes the mesh unsuitable for on-line viewing. To this end, we apply a subdivision algorithm [20] to create a low-resolution version of the mesh. Since the texture remains at full resolution, the viewing experience is not seriously degraded. We set the maximum number of faces to 10 000 which results in an uncompressed size of around 1MB.

5.2 Results

All models shown in this paper are the direct output of our reconstruction pipeline and have not been modified in any way. It should be noted that the input images have been taken with the purpose of creating 3D content. This does not require any particular skills; it simply means that one moves around the object while continuously taking images with a large overlap.

Figures 3 and 4 display models obtained with semi-professional DSLR cameras under good lighting conditions. Both geometry and texture are of superior

Fig. 5: Reconstruction results. (a) Vase reconstructed from 65 images. (b) Statue reconstructed from 26 images. (c) Further reconstruction results.

quality. Figure 5 shows results obtained from consumer cameras. The vase was in (a) was captured without a tripod in a dimly lit museum. The glass casing around the vase made it impossible to use the flash. Nevertheless, the resulting model is well reconstructed. The same can be said about the statue in (b), which was captured under daylight conditions. In (c) we can see some further models produced by the system.

6 Conclusion

We have presented an automatic system for the creation of textured meshes out of images taken with a digital camera. Both the initial and final part of a traditional 3D reconstruction pipeline have been enhanced, with functions for camera parameter retrieval and mesh generation respectively. Result are delivered both as an archive with the full resolution model, and as a link that opens the low resolution version in a WebGL model viewer. We also provide the original un-cleaned depth maps and the recovered camera parameters.

Our system is currently on-line on `http://www.arc3d.be` and open to the public, completely free for non-commercial purposes.

Acknowledgments

The research leading to these results has received funding from the European Communitys Seventh Framework Program (FP7/2007-2013) under grant agreement n. 231809 (IP project 3D-COFORM).

We further wish to express our gratitude to Dr. Sven Havemann of TU Graz and Dr. Sorin Hermon of the Cyprus Institute for providing us with images sets.

References

1. http://www.khronos.org/webgl (2011)
2. Sons, K., Klein, F., Rubinstein, D., Byelozyorov, S., Slusallek, P.: XML3D: interactive 3D graphics for the web. In: Web3D. (2010)
3. Vergauwen, M., Van Gool, L.: Web-based 3d reconstruction service. MVA **17** (2006) 411–426
4. http://www.photosynth.net (2011)
5. http://www.3dtubeme.com/ (2011)
6. Snavely, N., Seitz, S.M., Szeliski, R.: Photo tourism: Exploring photo collections in 3D. In: SIGGRAPH. (2006)
7. Furukawa, Y., Ponce, J.: Accurate, Dense, and Robust Multi-View Stereopsis. In: CVPR. (2007)
8. Moons, T., Gool, L.J.V., Vergauwen, M.: 3d reconstruction from multiple images: Part 1 - principles. Foundations and Trends in Computer Graphics and Vision **4** (2009) 287–404
9. Hartley, R.I., Zisserman, A.: Multiple View Geometry in Computer Vision. Second edn. Cambridge University Press (2004)
10. Triggs, B.: Autocalibration and the absolute quadric. In: Int. Conf. Computer Vision and Pattern Recognition, Puerto Rico (1997)
11. Pollefeys, M., Verbiest, F., Gool, L.V.: Surviving dominant planes in uncalibrated structure and motion recovery. In: ECCV. (2002)
12. Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: Speeded-up robust features. Computer Vision and Image Understanding **110** (2008) 346–359
13. Nistér, D.: An efficient solution to the five-point relative pose problem. TPAMI **26** (2004) 756–777
14. Haralick, B.M., Lee, C.N., Ottenberg, K., Nlle, M.: Review and analysis of solutions of the three point perspective pose estimation problem. International Journal of Computer Vision **13** (1994) 331–356
15. Pollefeys, M., Van Gool, L., Vergauwen, M., Verbiest, F., Cornelis, K., Tops, J., Koch, R.: Visual Modeling with a Hand-Held Camera. International Journal of Computer Vision **59** (2004) 207–232
16. E.P.J, T.: Broadcast engineer's reference book / edited by E.P.J. Tozer. Thirteenth edn. Boston, Massachusetts: Elsevier; Focal Press (2004)
17. Goesele, M., Snavely, N., Curless, B., Hoppe, H., Seitz, S.M.: Multi-View Stereo for Community Photo Collections. In: ICCV. (2007)
18. Furukawa, Y., Curless, B., Seitz, S.M., Szeliski, R.: Towards internet-scale multi-view stereo. In: CVPR. (2010)
19. Kazhdan, M., Bolitho, M., Hoppe, H.: Poisson surface reconstruction. In: SGP. (2006)
20. Hoppe, H.: Progressive Meshes. In: SIGGRAPH. (1996)