An Efficient Volumetric Method for Building Closed Triangular Meshes from 3-D Image and Point Data

Gerhard Roth and Eko Wibowoo

Institute for Information Technology, Building M 50, Montreal Road
National Research Council of Canada, Ottawa, Canada K1A 0R6
E-mail: roth@iit.nrc.ca Web: http://www.iitsg.nrc.ca/˜roth

Abstract
We present a volumetric method that can efficiently create triangular meshes from 3-D geometric data. This data can be presented in the form of images, profiles or unordered points. The mesh model can be created at different resolutions and can also be closed to make a true volumetric model.

Keywords: Mesh Creation, Triangular Meshes, Model Building, Range Data, Virtual Reality.

1 Introduction

This paper presents an algorithm to build a geometric model from 3D data of the surface of an object obtained by a range sensor [1]. Such sensors are also called geometric sensors because they are able to directly capture the geometry of an object. Typically this is done by using an optical source, such as a laser, to obtain the distance to the object's surface [2]. Another possibility is the use of X-ray tomography [3] to obtain the cross sections of the object cut by the X-rays. There are an ever increasing number of geometric sensors being developed by different companies and research organizations [4]. As is shown in Figure 1, some geometric sensors can acquire entire images, others can only acquire a single 3D profile or slice of an object, while still others can only acquire a single point at a time. If the 3D data is produced in point form there is no known neighbour relationship between data points. Such an unordered set of data points is called cloud data, and is common in industrial practice [5].

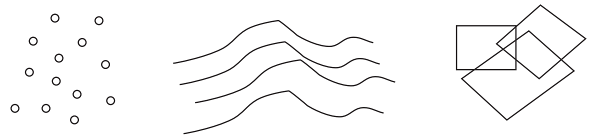

Figure 1: **Different types of 3D data: points, profiles and images.**

To scan an entire object the sensor must view the object's surface from a sufficient number of different views. This 3D data from each view must then be registered so that it is all in a single co-ordinate frame. This registration process can be done mechanically, by moving the geometric sensor using an accurate positioning device. It can also be done from the data itself, by minimizing the error between overlapping 3D data regions. Both methods are common and have been shown to work successfully [6, 7]. In this paper we will assume that all the 3D data has already been properly registered.

In order to make practical use of the registered 3D data it is necessary to construct a geometric model from this data. The first question is what type of geometric model to create. One of the most common model formats is a triangular mesh, which consists of a large number of triangular faces. If the 3D data is presented as a set of images it is trivial to create such a mesh by simply triangulating each image. However, since there is often considerable overlap between the 3D images from different views, a mesh created in this fashion will have many redundant faces. It is desirable to create a non-redundant mesh, in which there are no overlapping faces.

This paper, which is a continuation of previous work [8], describes a voxel-based algorithm which has the following characteristics.

- It uses a simple voxel data structure which is very efficient in both space and time.
- It is able to process 3D data in image, profile and point cloud format.
- It has a number of different ways of handling noisy and spurious 3D data points.
- It can fill holes in the triangulation to close the mesh and create a true volumetric model.
- It can report the accuracy of the triangular mesh relative to the original 3D data.
- It can handle 3D data which has an associated intensity or colour value.

2 Previous work

The methods in the literature that are able to create non-redundant meshes from multi-view 3D data can be divided into two groups, the volumetric approaches, and the surface approaches.

The volumetric approaches [9, 10, 8, 11, 12, 13, 14] store the 3D data points into a volumetric data structure, typically a voxel grid or an octree. The triangular mesh is then created using an Iso-Surface extraction algorithm, usually marching cubes [15], which operates on this volumetric data structure. The surface approaches [16, 17, 18, 19, 20] create an initial set of triangular regions from the original 3D images. These regions are then stitched together to make the final mesh.

Surface approaches are limited to processing 3D data in image format. They do not handle cloud data, and it is not clear how any surface approach could ever achieve this goal. In fact, many voxel approaches also do not handle cloud data (the only exception is [9]), but the underlying volumetric data structure clearly makes this possible. Another serious problem with the surface approaches is the requirement that all the original data points be in memory while the algorithm is in operation. By contrast, in the volumetric methods, once a point has been processed by the Iso-Surface algorithm it can be discarded. Therefore for a volumetric approach the required space is proportional to the number of occupied volume elements, while for a surface approach it is proportional to the number of data points.

If the goal is to create the most accurate mesh possible relative to the original data then a surface approach is superior to a volumetric approach. This is because surface methods triangulate the data at the original resolution. By contrast the volumetric methods can not set the size of their volumetric data structure to the same resolution as the 3D data because then each volumetric element would contain too few data points [11]. However, in practice we find that more 3D data is collected than is necessary. Therefore the number of data points is typically at least two or three times greater than the number of triangles that we want to have in the final mesh. In this case a volumetric approach will succeed because each volumetric element will contain a sufficient number of data points.

For both the volumetric and surface methods an issue that has rarely been dealt with in the past is how to close the final mesh. The mesh is often passed to a rapid prototyping system to create a physical duplicate [21], or is decimated further by a compression algorithm [22] for faster display. Both types of processing require that the input mesh be closed and topologically correct. Therefore it is essential to be able to produce a mesh which has these characteristics. Achieving this goal is equally difficult for both the volumetric and surface approaches.

A recent volumetric method [12] is similar to our proposed algorithm. However, there are significant differences. We can handle both cloud and image data, while this method handles only image data. Our way of closing the triangular mesh is less general, but is simpler and more efficient. Also, because of our efficient voxel data structure our method is about one order of magnitude faster.

3 Data Structures

The basic data structure we use is a voxel grid of fixed dimensions in x, y and z. This voxel grid will contain the original data points, along with the final mesh triangles. It is essential to be able to access the voxels efficiently, and to be able to traverse and check the topology of the final mesh.

3.1 Accessing Voxels

One way that a voxel needs to be accessed is randomly, by using its x, y and z index in the grid. This is accomplished by using a lookup table which maps the voxel indices to a voxel pointer [23]. If there is a pointer for each possible voxel, then this table would be very large. However, since geometric sensors scan only the surface of an object, only the pointers to the occupied surface voxels need to be stored. If the average x, y or z dimension of the hash table is n, then the total number of possible voxels in the 3D grid is n^3. However, in general the number of surface voxels is $O(n^2)$, which is much smaller. For this reason we use a hash table to store only the pointers to the occupied voxels. The size of this hash table should be close to the number of occupied voxels, which must be estimated before the actual 3D data is processed.

We have computed the percentage of occupied voxels for a large number of different objects and voxel grid sizes. In general, we find that between 1% and 6% of the total number of possible voxels are occupied. The percentage decreases as the total number of voxels in the grid increases. We dynamically adjust the size of the hash table as a function of the total number of possible voxels in the grid. Table 1 shows our hash table size as a percentage of the total number of voxels, for different sized voxel grids. The sizes in this table represent a conservative upper bound on our experiments. Even if the number of occupied voxels is greater than this percentage, the hash table will still work, but it will be slower.

Total Number of Possible Voxels	*Expected Percentage of Occupied Voxels*
$< 1,000,000$	6
$> 1,000,000$ and $< 2,000,000$	4
$> 3,000,000$	1

Table 1: **The allocated size of the hash table as a percentage of the total number of possible voxels.**

The hash table is excellent for random access of any voxel element. It will simply return a null pointer when an attempt is made to access a voxel which is not occupied. However, we often need to traverse all the occupied voxels in a single pass. This is the case, for example, when we wish to do some post processing such as closing holes in the triangular mesh. To find all the occupied voxels using the hash table would mean referencing every possible voxel to check if it is occupied, which is not practical.

For this reason we use a linked list structure to connect all the occupied voxels together. In this way we can traverse the occupied voxels very quickly. These two data structures allow both efficient random access (hash table) and efficient traversal of all occupied voxels (linked list).

3.2 Individual Voxel Structure

The marching cubes Iso-Surface algorithm creates the triangular mesh elements that define the surface. Regardless of how many triangles exist in each voxel, every triangle vertex must be on one of the twelve possible voxel edges. Triangles in a voxel often share an edge with triangles in the neighbouring voxels. For this reason each voxel is linked to up to twelve shared triangle vertices through a set of pointers. A triangle vertex also points back to each of the up to four possible voxels of which it is a member.

This data structure has two advantages. First, it enables the marching cubes algorithm to be implemented efficiently, since every triangle vertex that is shared between voxels needs only be computed once. Second, it is possible to determine if a triangle in a voxel shares an edge with a neighbouring voxel by checking the vertex-to-voxel pointers of each vertex in that edge. This operation is important in the hole closing algorithm that is described in Section 4.7.

4 Algorithm Overview

With this voxel grid as the underlying data structure the following sequence of operations are executed to create the triangular mesh.

```
Set the voxel size automatically or manually.
Add each data point to the appropriate voxel.
Eliminate spurious data points.
Compute the local normal for each data point.
Smooth the normals with a relaxation algorithm.
Run marching cubes to get the triangulation.
Close any small holes that exist.
Compute the intensity or colour of each triangle.
Remove small isolated triangle regions.
Find the mesh accuracy relative to the 3D data.
```

4.1 Set the Voxel Size

The first requirement is to set the dimensions of the voxel grid. This affects the number of triangles in the mesh, as well as the accuracy of the mesh relative to the 3D data. Here we have two possibilities. The first is to set the voxel size manually. The second is to do this automatically using a simple heuristic: take one hundred random 3D points and find the closest neighbouring point to each of these points. Then set the voxel size to three times the average of these one hundred minimum inter-point distances. This assures us that the each voxel grid element is likely to contain at least one data point [11].

4.2 Add Each Data Point to the Appropriate Voxel

Once the size of the voxel grid has been set it is necessary to add each 3D data point to the appropriate voxel. If the voxel does not already exist then it is first created and attached to the list of occupied voxels. Besides the actual 3D data any other relevant information, such as the local surface normal or the colour also must be saved.

4.3 Eliminate Spurious Points

There are often spurious points in 3D data due mostly to the problem of edge curl [24]. This occurs with optical sensors when the optical source (usually a laser beam) is half on and half off the edge of an object. In this case an invalid 3D point is often the result. Such a point does not actually exist on the surface of any object: it is truly spurious. This phenomenon becomes more noticeable as the distance to the scanned objects increases, or the amount of reflected light from the laser beam decreases.

The key to dealing with such points is to note that if the sensor were moved to a different viewpoint, then similar artifacts would appear, but in different locations. Therefore these spurious points are often pierced by rays joining valid surface points to the sensor viewpoint, as is shown in Figure 2. This fact has been noted previously and used to deal with the problem of hole closing [12] and to increase the measurement confidence of 3D data [20]. We use this principle to remove spurious data points.

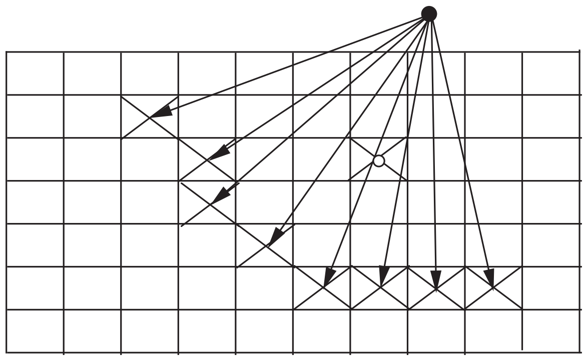

Figure 2: **Voxels which are pierced often (unfilled circle) by rays from the viewpoint are likely to be spurious.**

For each data point we walk along the voxel grid from this point towards the the sensor viewpoint. All the occupied voxels along this traversal are voted against. Those voxels that have many votes against them are considered to be spurious, and are removed.

We only perform the voxel traversal for a small distance from each voxel. This avoids the requirement to traverse the entire viewing volume, but still removes spurious points that are close to valid points. Spurious points that are far from any valid surface points tend to produce isolated and small triangular mesh regions in the final triangulation. For this reason such points can easily be removed by a 3D connectivity algorithm, as described in Section 4.9. While this approach is very effective it only works when the sensor viewpoint is available, which is not the case for cloud data. How to remove spurious points in cloud data is still an active area of research.

4.4 Compute The Normals

The marching cubes algorithm that is described in the next section requires that each data point have an associated normal. For image and profile data the normal can be computed by simply finding the closest neighbouring row and column points. These two neighbours along with the original data point are then used to compute the local normal estimate. When doing this computation it is important not to use neighbouring points that cross a depth discontinuity, or jump edge [1]. For image or profile data the orientation of this computed normal is toward the sensor viewpoint.

Computing the normal when the input data consists of 3D points in unordered form (cloud data) is more complex. In this case we find the closest two neighbouring points in the voxel grid, and then use these points along with the original point to compute the normal. The difficulty is in setting the normal orientation. Since by definition a cloud data point has no associated viewpoint, there is no obvious way to decide on which way the normal should be oriented. If the normal points in the wrong direction then the marching cubes algorithm will produce errors. This is because it requires a signed distance, and a flipped normal will produce the wrong sign.

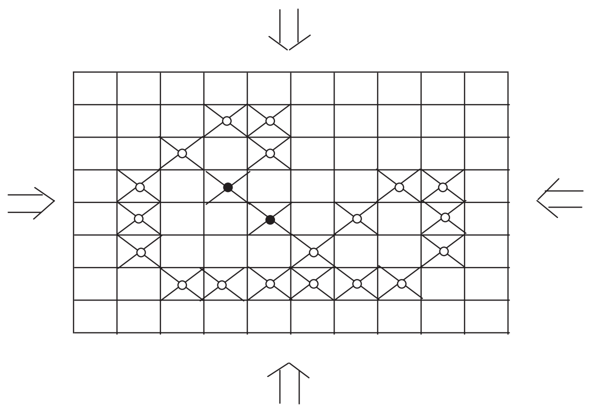

Figure 3: **Setting the normal signs in a voxel slice: Unfilled circles are set from the orthogonal view directions, while filled circles are set by normal propagation.**

Consider the voxel grid and the six axis directions $(+/-x, +/-y, +/-z)$. If we look from infinity down each axis into the voxel grid then those voxels that are visible must have their normals point towards the viewing direction. We fix the normal direction for these visible points. Then we propagate the normal direction to their neighbouring voxels. The results of this propagation process are shown for a single slice of the voxel grid in Figure 3. This heuristic only works when the voxel grid defines a closed object, one without any missing data. This is a reasonable assumption since in many cases no attempt is made to create a model until all the surface data is available.

4.5 Smooth the Normals Using a Relaxation Algorithm

Depending on the quality of the sensor data it may be necessary to smooth the normals that are associated with each data point. First we find the voxel whose normal most agrees with its occupied neighbours. This voxel is then used as the seed of a recursive smoothing algorithm. This algorithm sets each voxel normal to the average of the normals of the voxel neighbours. When noise in the original 3D data produces inaccurate normal estimates then this method tends to produce a smoother triangulation.

4.6 Run the Marching Cubes Algorithm

Marching cubes is an Iso-Surface algorithm which extracts the zero set of a signed distance function [15]. In this application the signed distance function must be created from the 3D data points and their normals. For each voxel vertex this signed distance, which we call the field value, is computed by taking the weighted average of the signed distances of every point in the eight neighbouring voxels. Once the field value at each voxel vertex is known then a linear interpolation process finds the intersection of the underlying surface with each edge of the voxel. Each of these intersection points is a vertex of the final triangulation. The triangles that approximate this surface in the voxel are found using a lookup table.

One advantage of the marching cubes algorithm is the ability to weight each of these signed distances. In our application it is sensible to weight the individual points according to their estimated accuracy [25, 12, 11]. The most significant source of inaccuracy occurs when the surface normal is close to being orthogonal to the line of sight of the sensor. When this happens, the reflected laser spot tends to spread and distort which causes significant errors in the computed depth. For this reason the Cosine of the angle between the local surface normal and the viewing direction is used as a weighting value [12].

4.7 Closing holes

There are often small gaps or holes between the triangles in each voxel. If the data is unevenly sampled, or the size of the voxel grid is too high, then there will be some voxels that do not have any data points. Also, there are sometimes areas of the object's surface that have not been scanned by the sensor.

These gaps need to be closed in order to create a model which has no holes. Producing a closed model is important for two reasons. The first is that only such a model can be sent to a rapid prototyping machine to make a physical duplicate [21]. The second is that a model that has even a small number of holes is difficult to process further by such operations as mesh compression [22].

In the first step of the closing algorithm we find all triangle edges that are not connected to other triangles. Such edges are called free edges and indicate a hole in the triangulation. A hole loop is a closed sequence of free edges which run in the proper direction. This direction is found using the right hand rule: if the thumb of the right hand is placed along the normal of the triangle face then the edges of the triangle must point in the direction of the index finger. The direction of the free edge traversal in a hole loop must be opposite to this direction.

The first requirement is to find the hole loops. In our voxel data structure each triangle vertex points back to all the voxels that have triangles which contain that vertex. This pointer structure is used to traverse the free edges in each hole loop using a recursive search. The recursion is efficient because usually about half the hole loops contain three edges (one triangle in size), and only 10% have more than five edges.

Once a hole loop has been found, it must be triangulated. We fit a plane through the 3D vertices of the hole loop edges. Then we project these vertices onto this plane, and check for self intersections in the hole loop. If there are none, then we triangulate the projected hole loop in this plane [27]. This same sequence of triangle vertices is then used to triangulate the 3D hole loop.

This approach will not work when a hole loop contains a triangle island; that is a number of valid triangles that are inside this loop. However, we have found that such cases are quite rare because the hole loops are usually small. It will also not work if the 3D hole loop is not coplanar. This is rarely the case for small holes, but is sometimes true for large holes. As opposed to some authors [12], we believe that large holes should be closed by obtaining more 3D data, and do not expect our hole closing algorithm to handle this case.

4.8 Colour and Intensity mapping

For each 3D data point there is sometimes an associated intensity or colour value provided by the 3D sensor [4]. In this case we assign to each triangle vertex the colour or intensity value of the closest data point. If the voxel grid resolution is not too coarse the result is a very realistic mapping of colours on top of the triangulation.

4.9 Remove Isolated Mesh Regions

Generally any noise points that have not been eliminated by the method described in Section 4.3 are not close to any valid surface points. Therefore, when the remaining noise points are triangulated they tend to produce small and isolated triangular mesh regions. We apply a 3D connectivity algorithm to find the size of each set of connected triangles in the final mesh. Then those connected triangle regions that contain fewer than a small number of triangles are removed, since they are likely to be noise. This deals effectively with any remaining noise points.

4.10 Compute the accuracy of the final mesh

When we have created the final triangular mesh we would like to know the accuracy of this mesh relative to the original 3D data points. For each point in a voxel we find the distance to the closest triangle that is part of the same voxel. We then find the maximum of this closest distance for all the data points. This number tells us the maximum deviation of the triangular mesh from the original 3D data. It can be no greater than the length of the longest side of the voxel grid.

The goal in mesh creation is to achieve a specified mesh accuracy relative to the original data. Usually this required accuracy is in the range of 1/10 mm to 2 mm. Note that when we speak of accuracy we are talking about the faithfulness of the final triangulation relative to the 3D data. That is not the same as the accuracy of the original 3D data relative to the true object geometry.

It is possible to increase the mesh accuracy by simply reducing the voxel size. As we have noted previously, the voxel grid size must be at two to three times greater than the sampling density of the 3D data [11]. This is a limitation of all voxel approaches to mesh creation. However, since 3D data is usually oversampled, a mesh of the desired accuracy can usually be obtained.

5 Experiments

We have taken 3D data in both cloud and image format from various sources and created a number of mesh models. The experiments were run on a Silicon Graphics Challenge processor, with 512Mbytes of physical memory. The results are summarized in Table 2. The source of the data for each model is acknowledged by referencing the paper that used this data, or by referencing the organization that provided the data. Some of models such as the duck were created from both cloud and image data.

In general the results validate our claim that our method is an order of magnitude faster than others in the literature. However, not all of these models were closed properly. The soldier and teapot have large regions of the objects surface where there is no 3D data [18]. As we have stated previously, our closing algorithm can not handle such large holes. We believe that in such cases more 3D data should be obtained. Figure 4 shows the rendered mesh created from 3D data in both image and cloud format while Figure 5 shows the rendered colour mesh for a number of different colour objects.

6 Conclusions and Future Work

Our voxel approach to creating meshes has shown itself to be efficient, simple and general. Any mesh creation algorithm must read the input data, and store the final triangulation. Assume there are n 3D data points, and the final triangulation has k triangles. Then the time requirement of any mesh creation algorithm is at least $O(n)$, and the space requirement is at least $O(k)$. In our case the required space is proportional to the number of occupied voxels, and the execution time is proportional to the number of 3D data points. Therefore our algorithm is optimal, at least in the asymptotic sense.

Since there may be many points in a voxel, the number of 3D data points is often much larger than

Model Name	*Data Source*	*Data Type*	*Number Points*	*Voxel Dimensions*	*Create and Close Times (secs.)*	*Number Triangles*
Duck	NRCC	Image	68K	96, 73, 51	8, 2	35K
Duck	NRCC	Point	68K	96, 73, 51	11, 2	35K
Boat	NRCC	Image	314K	99, 177, 99	43, 10	150K
Elephant	NRCC	Image	312K	149, 65, 125	24, 5	98K
Teapot	NRCC [18]	Image	67K	79, 131, 80	8, 5	56K
Soldier	NRCC [18]	Image	91K	116, 60, 49	8, 3	47K
Bunny	Cyber. [16]	Image	354K	169, 167, 131	43, 10	195K
Dragon	Cyber. [12]	Image	1.7M	347,420,393	235, 37	642K
Club	U. Wash. [9]	Point	17K	45, 68, 11	2, 1	6K
Star	Daimler	Point	106K	108, 75, 127	13, 2	36K
IMS Part	Daimler [26]	Point	1.0M	173, 257, 125	137, 21	354K

Table 2: **The results of running the algorithm on various 3D data sets.**

the number of voxels. Therefore storing only the occupied voxels enables our approach to handle very large 3D data sets. In this paper we have only shown examples of object models. We are currently applying this same algorithm to environment modelling. Here the goal is to model large environments, such as factories. In this case the amount of 3D data increases by at least an order of magnitude over object modelling. Because we can handle such large data sets our approach is well suited to the task of environment modelling.

The web site listed in the title page has some of the mesh models created by this algorithm, along with other related research. In the future, we plan to make the executable version of this program available on the same web site.

References

[1] P. J. Besl and R. C. Jain, "Three dimensional object recognition," *ACM Computing Surveys*, vol. 17, pp. 75–145, Mar. 1985.

[2] M. Rioux, "Laser rangefinders based on synchronized scanning," *Applied Optics*, vol. 23, pp. 3837–3844, 1985.

[3] J. C. Russ, *The image processing handbook.* CRC Press, 1995.

[4] P. Besl, "Active, optical range imaging sensors," *Machine Vision and Applications*, vol. 1, no. 1, pp. 127–152, 1988.

[5] R. Fisher, A. Fitzgibbon, A. Gionis, M. Wright, and D. Egger, "A hand-held optical surface scanner for environment modeling and virtual reality," Tech. Rep. DAI No.778, University of Edinburgh, Dec. 1995.

[6] H. Gagnon, M. Soucy, R. Bergevin, and D. Laurendeau, "Registration of multiple range views for automatic 3-d model building," in *Proceedings of IEEE Computer Vision and Pattern Recognition Conference*, (Seattle, Washington), pp. 581–586, June 1994.

[7] Y. Chen and G. Medioni, "Object modelling by registraion of multiple range images," *Image and Vision Computing*, vol. 10, pp. 145–155, Apr. 1992.

[8] G. Roth and E. Wibowo, "A fast algorithm for making mesh models from multi-view range data," in *Proceedings of the DND/CSA Robotics and Knowledge Based Systems Workshop*, (St. Hubert, Quebec), pp. 349–356, Oct. 1995.

[9] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized data points," in *Computer Graphics 26: Siggraph'92 Conference Proceedings*, vol. 26, pp. 71–78, July 1992.

[10] C. Bajaj, F. Bernardini, and G. Xu, "Automatic reconstruction of surfaces and scalar fields from 3d scans," in *Computer Graphics: Siggraph '95 Proceedings*, pp. 109–118, 1995.

[11] A. Hilton, A. Toddart, J. Illingworth, and T. Windeatt, "Reliable surface reconstruction from multiple range images," in *Fourth International European Conference on Computer Vision*, vol. 1, pp. 117–126, Apr. 1996.

[12] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Computer Graphics: Siggraph '96 Proceedings*, pp. 221–227, 1996.

[13] M.-E. Algorri and F. Scnmitt, "Surface reconstruction from unstructured data," *Computer graphics forum*, vol. 15, no. 1, pp. 47–60, 1996.

[14] M. Wheeler, Y. Sato, and K. Ikeuchi, "Consensus surfaces for modelling 3-d objects from multiple range images," Tech. Rep. CMU-CS-TR-96-168,

Carnigie Mellon Univ., School of Computer Science, Pittsburg, PA, 1996.

[15] W. E. Lorenen and H. E. Cline, "Marching cubes: a high resolution 3d surface reconstruction algorithm," in *Computer Graphics: Siggraph'87 Conference Proceedings*, vol. 21, pp. 163–169, July 1987.

[16] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *Computer Graphics (Siggraph '94)*, vol. 26, pp. 311–318, 1994.

[17] M. Rutishauser, M. Stricker, and M. Trobina, "Merging range images of arbitrarily shaped objects," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 573–580, 1994.

[18] M. Soucy and D. Laurendeau, "A general approach to the integration of a set of range views," *IEEE Transactions On Pattern Analysis and Machine Intelligence*, vol. 17, pp. 344–358, Apr. 1995.

[19] M. Soucy and D. Laurendeau, "A dynamic integration algorithm to model surfaces from multiple range views," *Machine Vision and Applications*, vol. 8, no. 1, pp. 53–62, 1995.

[20] R. Pito, "Mesh integration based on co-measurements," in *International Conference on Image Processing*, pp. 397–400, 1996.

[21] R. Aubin, "A world wide assessment of rapid prototyping technologies," in *Proceedings of the Intelligent Manufacturing Systems International Conference on Rapid Prototyping*, (Stuttgart, Germany), pp. 45–48, 1994.

[22] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh optimization," in *Computer Graphics: Siggraph '93 Proceedings*, pp. 19–25, 1993.

[23] G. Wyvill, C. McPheeters, and B. Wyvill, "Data structure for soft objects," *Visual Computer*, no. 2, pp. 227–234, 1986.

[24] B. Curless and M. Levoy, "Better optical triangulation through spacetime analysis," in *Fifth International Conference on Computer Vision*, (Cambridge, Massachusetts), pp. 987–994, 1995.

[25] P. Hebert, D. Laurendeau, and D. Poussart, "Scene reconstruction and description: geometric primitive extraction from multiple view scattered data," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, (New York), pp. 286–293, 1993.

[26] P. Boulanger, G. Roth, and G. Godin, "Applications of 3-d active vision to rapid product development," in *Proceedings of the Intelligent Manufacturing Systems International Conference on Rapid Prototyping*, (Stuttgart, Germany), Feb. 1994.

[27] R. Seidel, "A simple and fast incrmental randomized algorithm for triangulating polygons," *Computational geometry: theory and applications*, vol. 1, pp. 51–64, 1991.

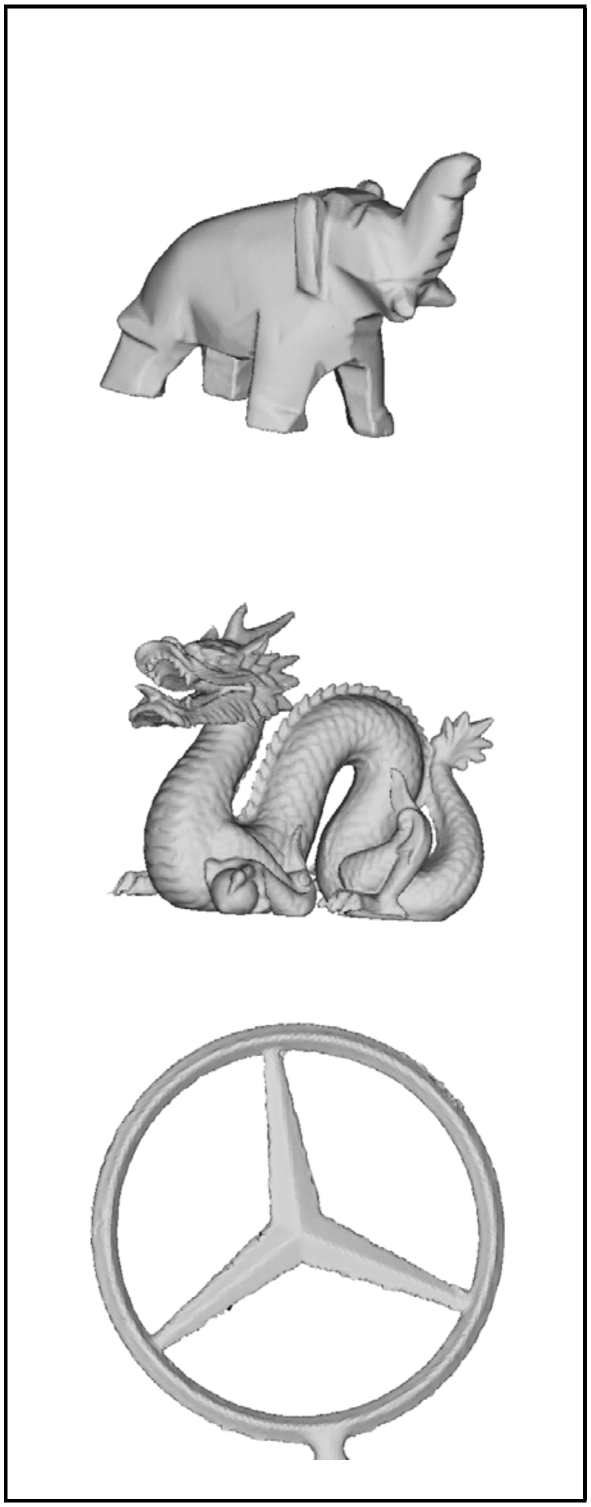

Figure 4: **Two models created from image data (the elephant and dragon), the other (Mercedes-Benz star) created from cloud data.**

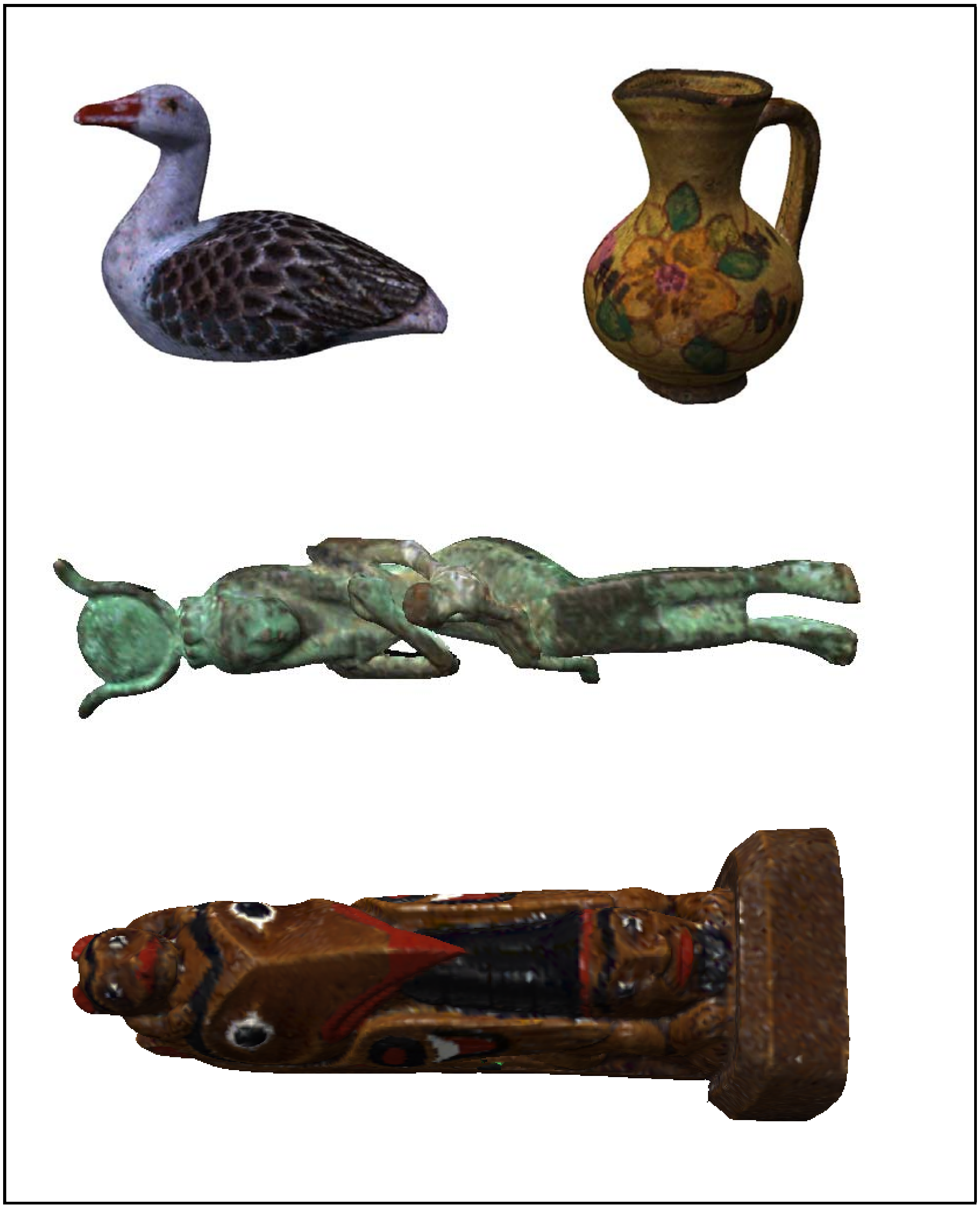

Figure 5: **Four colour models (duck, vase, mummy, and totem) produced by vertex colour mapping and Gouraud shading.**