Mesh Addition Based on the Depth Image (MABDI)

Lucas Chavez[1] and Ron Lumia[1], *Fellow, IEEE*

***Abstract*— Many robotic applications, especially those whose goal is to aid or assist through human-robot interaction, utilize a rich map of the world for reasoning tasks such as collision detection, path planning, or object recognition. Such map, and the method used to produce it, must take into consideration real-world constraints. Most mesh-based mapping algorithms resemble a "black box" and do not provide a mechanism to close the loop and make decisions about the incoming information. MABDI leverages the global mesh by finding the difference between what we expect to see and what we are actually seeing, and using this to classify the incoming measurements as novel or not. This allows the surface reconstruction method to be run only on data that has not yet been represented in the global mesh. The result is an algorithm that becomes computationally inexpensive once the environment is known, but can also react to new objects.**

I. Introduction

Many robotic applications, especially those that involve human-robot interaction, often require a rich representation of the environment in order to perform such behavior as path planning and obstacle avoidance. In general, a rich representation, or map, is useful for providing situational awareness to an autonomous agent. A map is also important for applications such as teleoperation [1].

The methodology to build this representation is a continuously evolving subject in the field of robotics. The origins of the research into this problem date back roughly 25 years [2]. Since then the methods and the representations themselves have continued to evolve at an impressive rate. The main catalyst behind this growth is the advancement of sensing technologies over the same time period. In general, sensors have continued to generate measurements at higher rates, higher resolution, and lower cost over the years. This has provided an amazing opportunity to build richer and more useful representations of the environment.

In robotics, map building in an unknown environment is referred to as the Simultaneous Localization and Mapping (SLAM) problem [3]. This label describes the fact that a methodology which solves the SLAM problem must simultaneously locate the robot in the environment as well as map the environment. The focus of this work is the mapping aspect of the SLAM problem. Fig. 1 gives a visualization of the goal.

There are different types of data structures that can define a map. All types have both intrinsic characteristics that impact the algorithms that generate them and constraints that must be considered for real-world applications. In addition, we are concerned with rich representation types, in contrast to sparse representation types [4], because rich types have the most use in applications such as human-robot interaction.

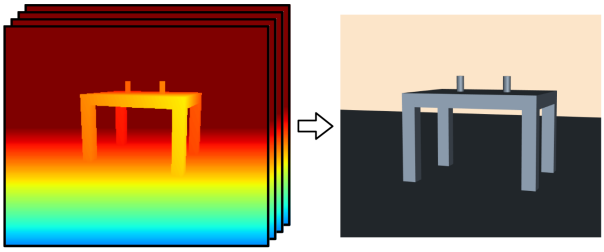

Fig. 1. Goal is to create a map from depth images

TABLE I

Comparison of constraints for different map types

	Supported	Computationally Inexpensive	Low Memory Requirement
Point Clouds	x	x	-
Surfels	-	x	x
Implicit Functions	x	-	-
Mesh	x	x	x

When considering which type of map is best for real-world applications, we must consider the constraints imposed by each type:

- Supported - Is there software, tools, research, algorithms, etc., for this type of map?
- Computationally Inexpensive - Can the algorithms run quickly on low cost computers (rather than specialized hardware)?
- Low Memory Requirement - Can the algorithms run on hardware with a standard amount of RAM?

Table I compares the constraints of common map types. We can see, in general a mesh type map satisfies real-world constraints. It has been used extensively by the gaming and graphics communities, and so benefits from an incredible amount of continued research and advances in hardware such as Graphics Processing Units (GPUs).

Currently, one of the issues with mesh mapping techniques is they are generally "black box" methods. Meaning the data comes in from the sensor, those measurements are turned into a mesh, and then that mesh is appended to a global mesh. Fig. 2 visualizes this common pipeline in black. The goal

[1]Mechanical Engineering Department and Electrical & Computer Engineering Department, University of New Mexico, Albuquerque, NM, USA golucasplus@gmail.com, {lucasc,lumia}@unm.edu

of this work is to design an algorithm to close the loop (as visualized in red) and allow the system to make decisions about the incoming data based on what it already knows.

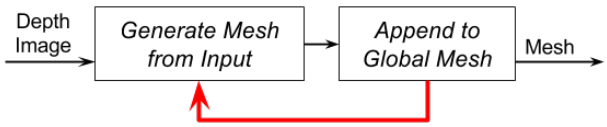

Fig. 2. Common "black box" pipeline in black. The contribution of MABDI in red.

II. Related Works

Works related to MABDI are generally based on RGB-D sensors. This type of sensor has become very popular since the release of the Kinect from Microsoft, which was the first mass produced RGB-D sensor of its kind. RGB-D sensors are inexpensive and produce noisy 640x480 depth images at 30fps. The RGB-D sensor has excited the robotics community because this has been the first time that depth data has been so readily accessible from such an inexpensive sensor. Therefore, methodologies that use RGB-D data must be able to quickly deal with very high rates of information.

One highly-related work was published by Whelan et al. in 2012 [5] and more recently in 2013 [6]. The system they developed was named Kintinuous and was able to produce a high quality mesh representation of the environment. Their hybrid system utilized the KinectFusion method [7] of Newcombe et al. to create a volumetric representation of the portion of the environment in front of the sensor. As the sensor moves, portions of the environment that leave the volume in front of the sensor are ray cast and turned into a mesh. They obtain very impressive results but also mention a limitation of their system for future work. The limitation is that the mesh can not be updated once created, which is an issue when revisiting parts of the environment. One of the most impressive current works which has an adaptable mesh came from Cashier et al. in 2012 [8]. In this work, they were able to generate and update a mesh with new measurements from a ToF sensor. They used the difference between the existing model and the actual measurements to decide whether to adapt the mesh or add new elements. The mesh topology was not adaptive to the environment and their experiments only showed results of mapping a single flat wall with no robot movement. The system needs to be tested for object addition and removal.

Research and development of new mapping algorithms trend towards leveraging the information in the global map to make decisions about the incoming data. One can see parallels with how we as humans see the world. MABDI proposes do this in a computationally feasible way by simply using differencing and thresholding imaging methods.

III. Approach

The algorithmic structure of MABDI can be seen in Fig. 3. The diagram is very similar to Fig. 2 with the exception of the Classification component, shown in blue. This Classification component is MABDI's contribution to the state-of-art in mesh based mapping algorithms, and is what gives MABDI the ability to make decisions about the incoming data.

The Classification component consists of two elements:

1) *Generate Expected Depth Image* E - Here we take the global mesh M, render it using computer graphics, and use the depth buffer of the render window to create a depth image E of what we expect to see from our sensor. This method requires the current pose P of the actual sensor (simulated for our experiments).
2) *Classify Depth Image* D - Here we classify the actual depth image D (simulated for our experiments) by first taking the absolute difference between E and D and thresholding. If the differences are small, those points are thrown away and if the differences are large, those points are kept as D_n. The idea behind this is, if the difference is large, the measurements are coming from a part of the environment that has not been seen before, i.e. novel. The implication of this assumption is that this version of MABDI cannot handle object removal. It is worth noting that MABDI can be extended to handle object removal by using the sign of the difference between E and D instead of the absolute value.

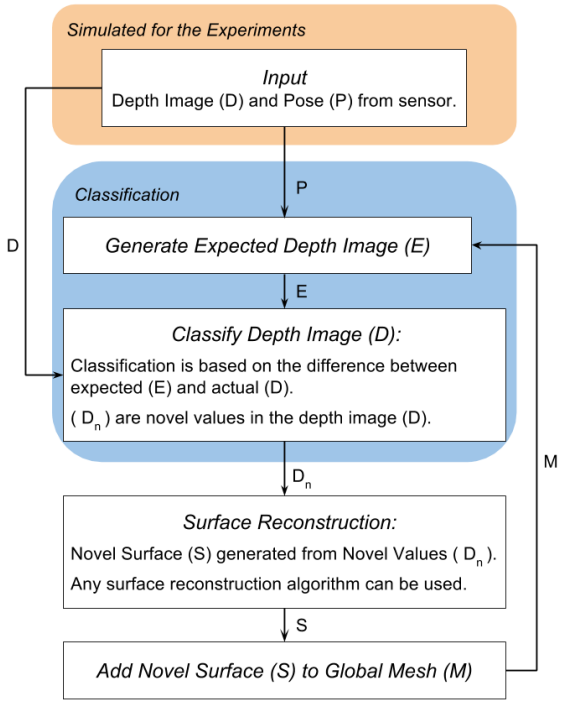

Fig. 3. MABDI system diagram

From a software perspective, the major difficulty of implementing the MABDI algorithm was found to be creating both the simulated depth image D and the expected depth image E. In addition, managing the complexity of the data pipeline needed to run the algorithm and the simulation of

the sensor proved to be difficult. Thankfully, Kitware, who is a leading edge developer of open-source software, created the Visualization Toolkit (VTK) [9], [10]. At the time of this writing the VTK Github repository has over 60,000 commits and is contributed to by supporters such as Sandia National Labs [11].

MABDI implemented in Python and uses VTK. The code is available upon request to golucasplus@gmail.com. At the time of this writing, it consists of over 1,400 lines. The code that implements the MABDI algorithm itself is around 750 lines.

VTK is aptly designed for the implementation of MABDI for many reasons. Perhaps the most important is the concept of a vtkAlgorithm (often called a Filter). This allows a programmer to create a custom and modular processing pipeline by defining classes that inherit vtkAlgorithm and then defining the connections between these classes. For example, you could have a pipeline that reads an image from a source (component 1), performs edge detection (component 2), and then renders the image (component 3). Using this concept, the individual elements of MABDI can be succinctly defined in individual classes. With that in mind, we can see in Fig. 4 the layout used in my implementation of MABDI. Note, vtkImageData and vtkPolyData are VTK types used to represent an image and mesh respectively:

- *Source* - Classes with the prefix Source define the environment that is used for the simulation and provide a mesh in the form of a vtkPolyData.
- *FilterDepthImage* - Render the incoming vtkPolyData in a window and output the depth buffer from the window as a vtkImageData. The output additionally has pose information of the sensor.
- *FilterClassifier* - Implements the true innovation of MABDI, takes the difference between the two incoming depth images (vtkImageData) and outputs a new depth image where the data that is not novel is marked to be thrown away.
- *FilterDepthImageToSurface* - Performs surface reconstruction on the novel points. In this simple implementation the topology of the mesh is defined in the image coordinates and can be thought of as a checkerboard pattern with two triangles in every square. The data is then projected to real-world coordinates. The topology and the real-world coordinates are combined to define a surface and output as a vtkPolyData.
- *FilterWorldMesh* - Here we simply append the incoming novel surface to a growing global mesh that is also output as a vtkPolyData.

IV. Experimental Setup

It was decided to develop and test MABDI in a completely simulated environment so that all results could be repeatable and also to facilitate the ability to debug during the development process. This ability was truly invaluable as some components of the algorithm proved to be complex from an implementation perspective. In addition, we can now compare the resultant global mesh to ground truth.

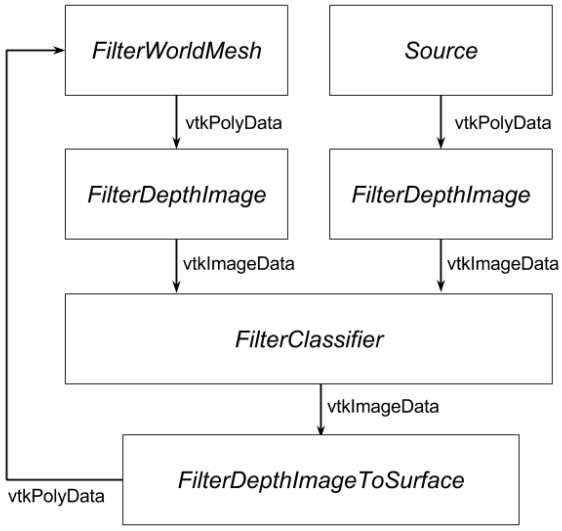

Fig. 4. MABDI software diagram

A. Simulation Parameters

The simulation was designed to be highly configurable and is implemented by a class named MabdiSimulate. The class is initialized with parameters that control all aspects of the simulation. Parameters of a particular importance are discussed in more detail here:

- Environment - This parameter specifies the environment used to generate the simulated depth images. *Table* is an environment consisting of a table and two cups placed on the table. The table is 1 meter tall. *Bunnies* is an environment consisting of three bunnies who are around 1.5 meters tall. These bunnies are created using the Stanford Bunny [12], a well known data set in computer graphics.
- Noise - If true, adds noise to the depth image of the simulated sensor.
- Dynamic - If true, adds an object during the simulation. In the case of this analysis, a third bunny is added half-way through the simulation.
- Iterations - The number of iterations the simulation will have. This parameter affects the distance the sensor travels from frame to frame.

For this paper we will be exploring three experimental runs to demonstrate the ability of the MABDI implementation to generate valid results. Additionally, the experimental runs will be able to show the capabilities of the MABDI algorithm such as handling object addition in the environment.

All experimental runs define a helical path for the sensor to follow during the simulation. The path circles the objects in the environment twice. A helical path was chosen because it returns to a part of the environment that has been already mapped and is thus “known” to the global mesh. Also, because

TABLE II

DESCRIPTION OF THE EXPERIMENTAL RUNS.

	Environment	Noise	Dynamic	Iterations
Run 1	Table	False	False	30
Run 2	Bunnies	True	False	50
Run 3	Bunnies	True	True	50

the path is a helix and not just a circle, the sensor views the environment from a slightly different position on each pass.

B. Analysis of Simulated Noise

In order to realistically simulate the sensor in a real-world environment we add noise to the depth image D. See Fig. 3. The magnitude of the noise added is based on the accuracy of real-world sensors. As new RGB-D sensors have been developed, such as the Asus Xtion and the Kinect for Xbox One, the accuracy of the sensors has continued to improve [13]. For this work, we take a conservative approach and utilize the well known error modeling work of Khoshelham [14] that is based on the original Kinect.

The depth image used by the MABDI algorithm E and the depth image that comes from simulating the environment D both use a pinhole camera model. This method has been validated in the localization work of Fallon [15]. The intrinsic camera parameters of the pinhole hole model were chosen to emulate the Kinect sensor [16]. The pinhole model defines a transformation matrix used to transform between viewpoint coordinates and real-world coordinates. The z component of the viewpoint coordinates constitutes the depth image and are defined to vary between 0 and 1. Due to how the transformation works, differences in the depth image do not linearly correspond to changes in real-world coordinates as can be seen in Fig. 5.

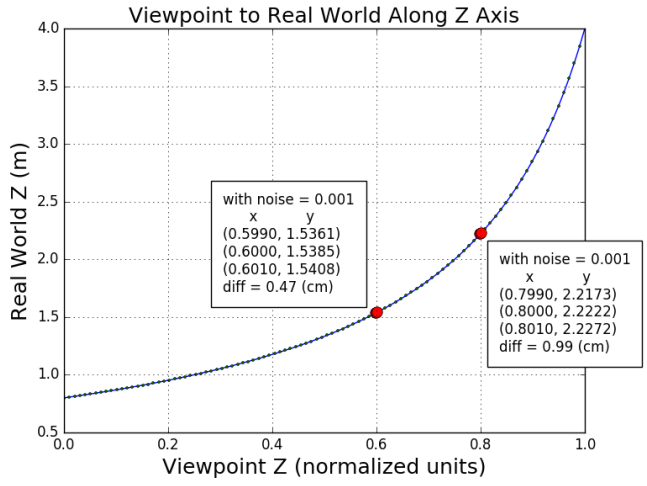

Fig. 5. Viewpoint coordinates to real world coordinates analysis.

The noise added to D is defined by the equation below. The standard deviation σ=0.001 was chosen so that the resultant errors in the real-world coordinates would correlate to the error model in [14]. The text boxes in Fig. 5 show the resultant real-world error for two values of D; they match the error model of [14].

$$D_{noisy}(i,j) = D(i,j) + D(i,j) * \mathcal{N}(\mu{=}0, \sigma{=}0.001)$$

V. RESULTS

An overview of the experimental runs is given in Fig. 7. The figure shows a set of six views of information for each run. These views are created at every iteration and generate a movie of the run. Fig. 7 shows a snapshot of these views at different points of the simulation for each run. The views are described below:

- Top Left - View of the global mesh M from a third perspective. The wire frame corresponds to the viewing frustum of the sensor. The light blue helical line is the path of the sensor. Gray is the simulated environment. The multi-colored mesh is the global mesh M. The mesh is multi-colored in order to show the passage of time. For example, in Run1, The mesh is colored yellow, light green, and dark green for iterations 1, 2, and 3 respectively.
- Top Middle - Same as Top Left except it shows the novel surface S instead of the global mesh M.
- Top Right - Plot showing the number of elements in the global mesh M up to this iteration.
- Bottom Left and Bottom Middle - Actual D and expected E depth image respectively.
- Bottom Right - The classified depth image. Novel point D_n are shown in black. Points to be thrown away are shown in white.

We can notice important aspects of MABDI using Fig. 7. Notes on each of the runs:

- Run 1 - Top Left shows how the M gets composed over time. It is important to note that the mesh is not overlapping itself. This can be understood by noticing S from Top Middle is the same as the section of M this is colored dark green.
- Run 2 - This run shows clearly how the classification process is able to distinguish novel points. This can be seen by noticing the valley in-between the ear and the eye of the bunny closest to the sensor. The sensor was not able to see these points from the prior iteration due to occlusion. From the sensor's current perspective, the points can now be seen. Notice how the valley is missing in the *expected* depth image E, classified as novel in Bottom Right, and thus reconstructed into S. This is a clear example of the concept behind MABDI.
- Run 3 - This run shows how MABDI reacts to object addition. At this iteration the middle bunny is suddenly added. We can follow the data: D shows the new bunny, E shows what we expect to see (the middle bunny is not there because it is not in M), Bottom Right shows all points corresponding to the new bunny as marked as novel, and finally the novel points are reconstructed

as S and appended to M. Top Right also shows a large jump in the number of elements in M due to the new bunny.

Fig 6 shows the resultant global mesh from all of the runs along with a plot of the number of elements in the mesh over iterations. These plots show the main contribution of MABDI because they level-off as the environment becomes more known as opposed to traditional "black box" reconstruction methods where the number of elements increases linearly over time.

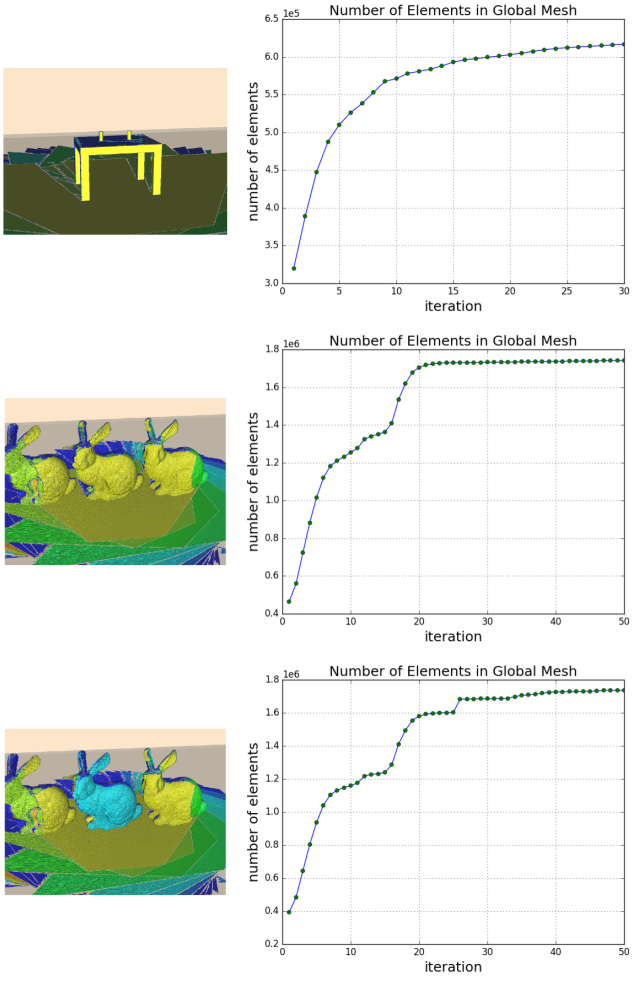

Fig. 6. Global mesh results. Top: Run1. Middle: Run2. Bottom: Run3

VI. Conclusion

The goal of MABDI is to determine data from the sensor that has not yet been represented in the map and use this data to add to the map. MABDI does this by leveraging the difference between what we are actually seeing and what we expect to see. MABDI can work in conjunction with any "black box" mesh-based surface reconstruction algorithm, and can be thought of as a general means to provide introspection to those types of reconstruction methods.

The MABDI implementation was able to successfully perform in a realistic simulation environment. The results show how novel sensor data was successfully classified and used to add to the global mesh. Also, the MABDI algorithm runs at around 2Hz on a consumer grade laptop with an Intel i7 processor. This performance means that it is capable of real-world applications.

Currently MABDI is only designed to handle object addition, but the idea can be extended to handle both object addition and removal as discussed in Section III. This would give the system the capability to handle highly dynamic environments such as a door opening and closing.

VII. Acknowledgment

This work was support in part by Sandia National Laboratories under Purchase Order: 1179196 and NSF grant OISE #1131305.

References

[1] M. W. Kadous, R. K.-M. Sheh, and C. Sammut, "Effective user interface design for rescue robotics," in *Proceeding of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction - HRI '06*. New York, New York, USA: ACM Press, mar 2006, p. 250.

[2] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Computer*, vol. 21, no. 4, pp. 163–169, 1987.

[3] S. Thrun, "Robotic mapping: A survey," *Exploring artificial intelligence in the new millennium*, no. February, 2002.

[4] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, jun 2001.

[5] T. Whelan, M. Kaess, and M. Fallon, "Kintinuous: Spatially Extended KinectFusion," 2012.

[6] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. B. McDonald, "Robust Tracking for Real-Time Dense RGB-D Mapping with Kintinuous," in *ICRA 2013*, no. MIT-CSAIL-TR-2012-031. Computer Science and Artificial Intelligence Laboratory, MIT, sep 2012.

[7] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. IEEE, oct 2011, pp. 127–136.

[8] L.-K. Cahier, T. Ogata, and H. Okuno, "Incremental probabilistic geometry estimation for robot scene understanding," in *ICRA 2012*. Ieee, may 2012, pp. 3625–3630.

[9] W. J. Schroeder, B. Lorensen, and K. Martin, *The visualization toolkit*. Kitware, 2004.

[10] Kitware. (2016) Vtk the visualization toolkit. [Online]. Available: http://www.vtk.org/overview/

[11] S. N. Labs. (2016) Snl computational systems and software environment. [Online]. Available: http://www.sandia.gov/asc/computational_systems/

[12] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *SIGGRAPH '94*. New York, New York, USA: ACM Press, 1994, pp. 311–318.

[13] E. Lachat, H. Macher, M. Mittet, T. Landes, and P. Grussenmeyer, "First experiences with kinect v2 sensor for close range 3d modelling," *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 40, no. 5, p. 93, 2015.

[14] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of Kinect depth data for indoor mapping applications." *Sensors (Basel, Switzerland)*, vol. 12, no. 2, pp. 1437–54, 2012.

[15] M. F. Fallon, H. Johannsson, and J. J. Leonard, "Efficient Scene Simulation for Robust Monte Carlo Localization using an RGB-D Camera," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, may 2012, pp. 1663–1670.

[16] Microsoft. (2016) Microsoft robotics kinect sensor. [Online]. Available: https://msdn.microsoft.com/en-us/library/hh438998.aspx

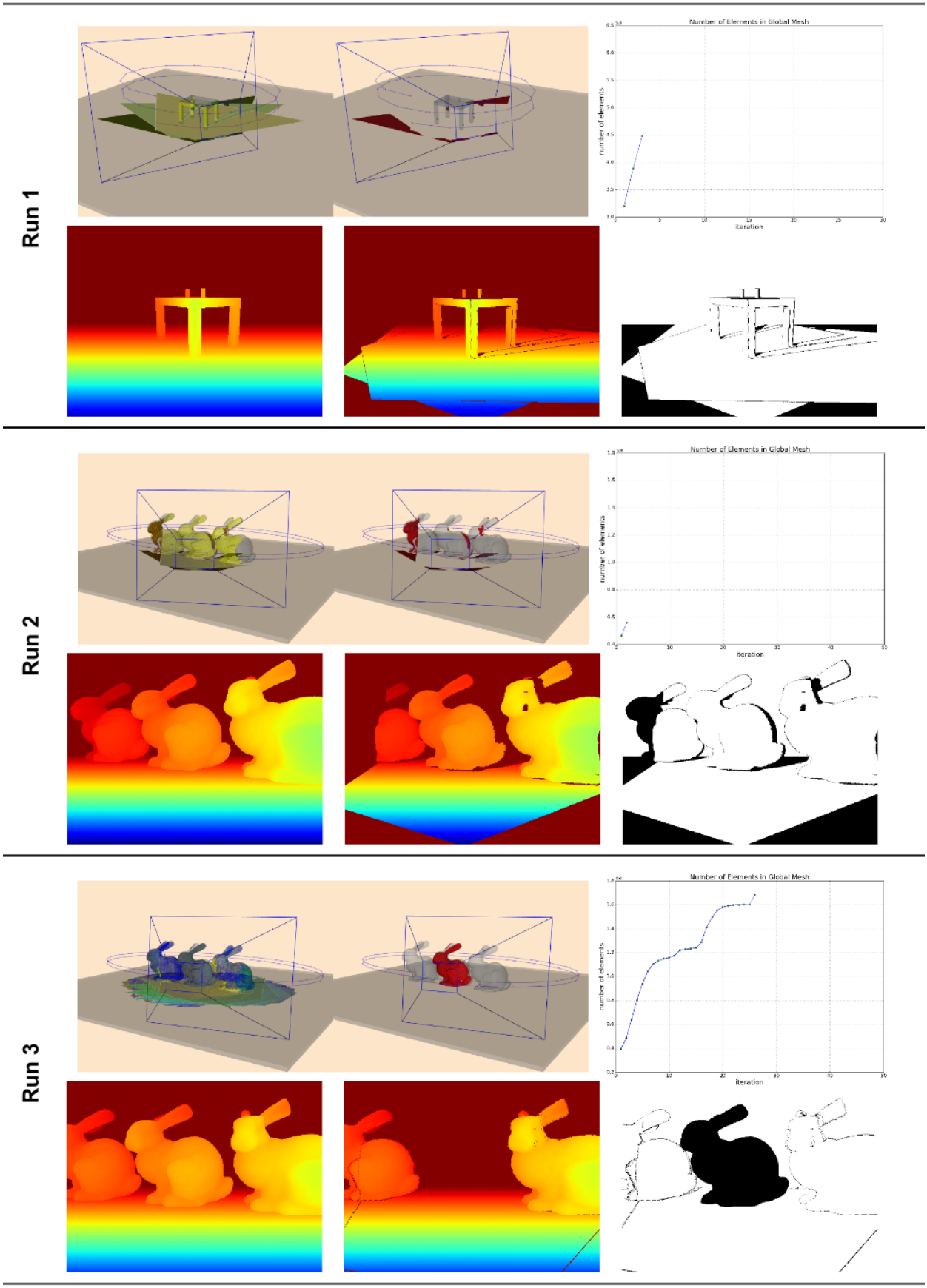

Fig. 7. Results of all experimental runs. See Results Section for discussion.