RMBL3D: Building Smooth Virtual Reality Maps Using 3D Objects

Bradley J. Wimpey
wimpey@cs.uga.edu

Walter D. Potter
potter@uga.edu

Computer Science Department
The University of Georgia
Athens, Georgia

ABSTRACT
In this paper, we describe a 3D modeling program called RMBL3D, Realistic Maps Built like Legos, which manipulates architectural repetition in building structures to produce realistic 3D models of environments. This paper details the features and aspects of the RMBL3D program and tests the viability of using 3D objects as building blocks to build a virtual map. The program takes in a description of the map in the form of character symbols representing objects in the environment and reproduces the 3D version of the map, one that can be explored and navigated by a user.

Categories and Subject Descriptors
I.3.7 [**Three-Dimensional Graphics and Realism**]: Virtual Reality; I.6.3 [**Simulation and Modeling**]: Applications.

General Terms
Design, Experimentation.

Keywords
Mapping, 3D Modeling, Robotics.

1. INTRODUCTION
The use of 3D maps and models has become quite important. 3D models give the user information on object locations that 2D maps provide while also adding depth and realism. 3D maps of foreign cities can provide a tourist with a virtual tour of the land and building layouts before the person ever sets foot in the city. 3D maps can also be used in military training scenarios to prepare soldiers before their lives are put into any danger. The more accurate the textured images and location information and relationships of the digitized objects, the more realistic the experience is to the viewer.

Man-made architectural structures include repeated segments of objects. Inside of buildings you find column after column of bricks or cinder blocks, and long stretches of smooth drywall occasionally interrupted by doors or hallway archways. Where objects are located in relation to each other in the environment is sometimes the greatest detail distinguishing objects from one another. Furthermore, if one had realistic looking 3D models representing different objects found in real life, such as a column segment of red bricks, a column of white cinder blocks, or a door to name a few, and if one was able to place these objects into 3D space, then one could conceivably build accurate 3D representations of the environment.

3D mapping has been used to map environments dangerous for humans; [14] used a sensor equipped cart and a mobile robot for 3D mapping of abandoned mines. 3D modeling has also been used for heritage site recording and archiving techniques. [3] applied their methods to modeling churches in Italy. They described combining image based modeling for large structural elements with laser range scans for detailed modeling. Their image based modeling, however, was interactive. [11] introduced CityEngine, a program that used a set of production rules to generate an urban model of buildings, roads, and highways. [9] described their shape grammar for modeling different building types, CGA Shape, which was combined into CityEngine. They created production rules used to model Pompeii based on ground plans and architecture styles. The result is a stunning "What it might have looked like" visualization technique. In [10], they procedurally recreated stone buildings of the ancient city of Xkipché in Mexico. CityEngine with CGA Shape produced amazing results, but this program is geared more for stochastic model generation for, say, games or movies, as opposed to mapping. [4] described copying and pasting repeated architectural components in image based modeling. The user first models one component that is repeated in the structure to be modeled, for example a window of a building, and then identifies other areas in the overall model that the component can be reused. The technique is interactive, however, and it requires a user to select points common between the modeled component and the areas the component is to be reused. A popular choice for modeling is the use of laser range finders, but different techniques exist for representing the model data. [7] used a mobile robot to map part of the campus of USC, but the range data result was presented as point clouds, leaving it to the viewer to interpret objects and details. [13] produced a mesh of range data in an octree representation. [6] modeled built structures by identifying large planar shapes in the range data results, but no object separation or texturing was applied. [1] produced a 3D model of an office environment by using a mobile robot equipped with a panoramic camera and a laser range finder. The resulting model,

however, lacked quality and detail in the images textured on the model. [12] attempted to create CAD models from range and image data. They use some human interaction to combine the 3D data with the 2D image data. While the models are CAD representations, they do not use repeated architectural objects in construction of the model. [5] created urban environment models by incorporating height data acquired from airborne laser range finders, but they also used information from existing ground plans. Our approach can be used to render models without being locked into ground plan data.

Our motivation was to manipulate architectural repetition in order to render realistic maps in three dimensions. For accessibility to test our idea, we set out to use objects found within the Boyd Graduate Studies Research Center on the campus of the University of Georgia. Using images and size information of objects from the interior of Boyd, we designed 3D models of objects to be placed into the 3D world. The RMBL3D program takes in a map of object symbols in the form of a text file and produces a 3D rendering accordingly, using the defined 3D objects and predetermined spacing found within Boyd. Thus, depending on the objects specified in the symbol map, the 3D objects are placed into the 3D world like Lego pieces, and the result is a smooth, navigable 3D map of the environment. Since RMBL3D is equipped with 3D model objects representative of the objects one might find in Boyd, it is not only able to render 3D maps of Boyd hallways and foyers, but it is also able to produce maps of non-existent areas like kilometer long hallways or square kilometer foyers, just with the objects common to Boyd. The goal of RMBL3D is to be the rendering program of an autonomous mobile robot, in which the robot explores the environment and builds an object symbol map which is then fed into RMBL3D to view the resulting 3D map. Using RMBL3D, a user would then be able to traverse the virtual environment. The entire system of the exploring mobile robot, object mapping and 3D rendering is ongoing research, but we present here the RMBL3D program in order to adequately explain its features and details.

2. ASSESSING BOYD FOR DIGITIZATION

In order to realize the goal of realistic 3D models and maps, we had to assess the environment to make sure we encode realistic wall heights, door widths, elevator widths, and so forth. If we were also going to use objects as building blocks, we had to decide how much space those building blocks would take up in the 3D environment. The Boyd Graduate Studies Research Center on the campus of the University of Georgia has an interior one might find in other office buildings, and since it has convenient access to researchers, we took it as our location for experimentation.

We had to decide on a segmentation size that would not be too big, which might lead to the creation of unrealistic 3D models, but also not be too small, eliminating the benefits gained from recognizing and mapping repeated structures. We found that delimiting the environment into one meter segments, with each meter segment being a wall object, door object, or another object from the environment, we could map out a floor of Boyd. An elevator was a special case object; it needed to be two meters wide in order to be of appropriate size to represent an elevator and to be of realistic size when navigating the 3D map. In segmenting the environment into meters, not everything will match perfectly. Using our modeling approach, we used a rule such that object locations and sizes will be rounded to the nearest meter. For example, if a door along a hallway is separated from another door by only two feet of a brick wall object, the brick wall segment in the 3D map will be rounded to a whole meter. A column of Boyd is about 66 centimeters square, and we model it using a square meter column object. The widths of hallways were modeled as 2 meters across. Using these mapping rules, the models rendered are not exact, but for our application, the approximate models are robust and the rounding error does not cause any major problems. Furthermore, Boyd walls were of different material, often bricks or cinder blocks, and since doors do not go from floor to ceiling, brick or cinder overhangs would also need to be modeled over doors. Hallway turns also needed to be represented in the model in the form of L or T junctions. In order to model a foyer, we needed segments of empty space. Furthermore, we needed floor tiles, ceiling tiles, and all of this took place on a floor of Boyd with a height (floor to ceiling) of 3.048 meters.

3. REPRESENTING THE SYMBOL MAP

The symbol map is what is produced onboard the autonomous mobile robot. Since the origin of the symbol map is of no concern to the RMBL3D program, as long as the symbols make sense to the program, a 3D model can be rendered, regardless of how the symbol map was created. This modularity allows us to develop and test the extent of the RMBL3D capabilities without it specifically relying on the mobile robot to, say, traverse a kilometer long map. Since this was a mobile robot project, however, we had to keep in mind the way in which the robot would detect the objects and how it should be represented in a symbol map. Therefore, the symbol map layout is formatted to support an incremental, robocentric mapping behavior.

3.1 Object Symbols

According to our robocentric mapping behavior, for each represented meter segment, the objects the robot detects on its left and its right are recorded as pairs of symbols in the symbol map. The robot would then move and detect again. The robot would update the map incrementally and keep the map as a string of characters. Therefore, the format of the symbol map is $O_{R1}O_{L1}O_{R2}O_{L2}\ldots$ where for each object symbol O_{Di} : O is one of the symbols {D,d,b,w,E,e,c,n}; D is the robocentric direction the object is located (left or right); i is the number of traveled meters. For example, a symbol map string that starts “DbbD” represents a door with brick overhang on the right and a brick wall segment on the left in meter number one, and the second meter of the map is of a brick wall on the right and a door with brick overhang on the left. For a list of the object symbols implemented, see Table 1.

3.2 Control Symbols

To represent a single 90 degree turn in the symbol map, we used the five symbol format $DO_{E1}O_{E2}O_{B1}O_{B2}$ where D is the robocentric direction of the 90 degree turn (‘l’ or ‘r’ for ‘left’ or ‘right’, respectively), O_{Ei} is an object symbol representing one of the two end cap objects, and O_{Bi} is an object symbol representing one of the two beginning cap objects. We use the terms “end cap” and “beginning cap” to represent the space to be filled in by objects when the robot makes a turn or when two hallways join. Since our modeled hallways are two meters across, if a robot is

Table 1. Implemented Object Symbols

D	door with brick overhang
d	door with cinder overhang
b	brick wall
w	cinder block wall
E	elevator with brick overhang
e	elevator with cinder overhang
c	column
n	empty space

mapping from the center of the hallway and needs to turn to start mapping an intersecting hallway, the "caps" are the extra objects to be mapped that join one hallway with the next. Alternatively, the caps are also the objects to be mapped when a robot makes a turn in a foyer. Figure 1 highlights this area of interest. An example of the five symbols to represent a 90 degree turn in the symbol map would be "rdwwd", which would mean a 90 degree turn to the robot's right, with the end cap objects being 'door with cinder overhang' and 'cinder block wall' and the beginning cap

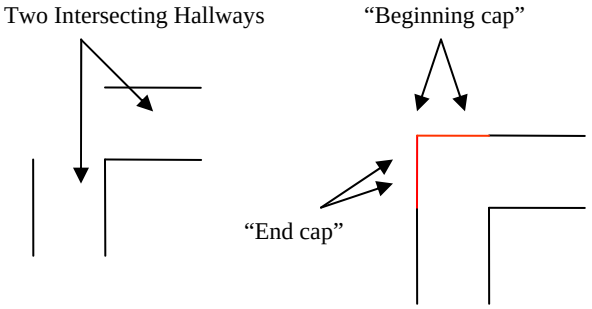


Figure 1: Hallway Caps

objects being 'cinder block wall' and 'door with cinder overhang'. This is a typical example of an L intersection. Should the robot encounter a turn, and it is mapping a foyer or a T intersection, in which there are two directions it could take, it would need to insert empty space into the map. A five symbol example of a T intersection in which the robot chooses to continue mapping to the left but leave the right area open for future mapping might be "lnnww", where the end caps are left as empty space and the beginning caps are two cinder block wall segments. Visual examples of intersections and turns that needed to be represented are shown in Figure 2, and how they were modeled in RMBL3D can be seen in Figure 3. The '+' symbol was a control symbol implemented such that we could map multiple levels of a building into one 3D model map. The '+' symbol would move the mapping plane 4 meters up.

Whereas using laser range finders on robots will allow the robot to map areas from a greater distance away, we found that object detection via image processing combined with this grid-like mapping, the robot would be able to map areas in a snake-like pattern, two meters in width at a time. This allows for more methodical and therefore more reliable object detection and mapping.

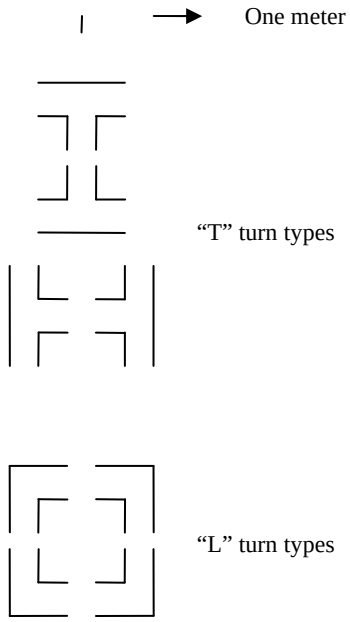


Figure 2: Turn Types

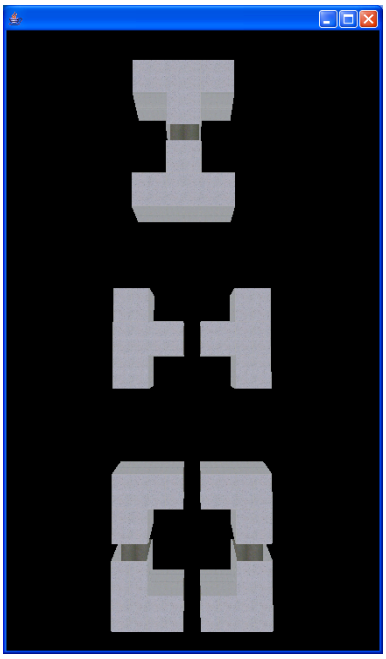


Figure 3: RMBL3D Representations of the Turn Types (bird's eye view)

4. IMPLEMENTATION

The RMBL3D program was written in Java and used Java 3D [8] and the JWSU Virtual Reality toolkit [2]. The Java 3D package is freely available from Sun Microsystems. Java 3D has been developed for years and includes a robust API and many tutorials. The JWSU toolkit is a powerful and free utility that allows for mobility in the 3D world and also includes support for virtual

reality devices such as head mounted displays, powerwalls, and PinchGloves or CyberGloves.

4.1 The Java 3D World and Object Models

Since the spatial environment in Java 3D is represented in meters, modeling the objects and specifying coordinates based on our previous assumptions was straightforward. Furthermore, the Java 3D engine handles scale, such that if one creates an object one meter in height and another object two meters in height, Java 3D handles the size ratios accordingly in the 3D world. We use the Java 3D geometry primitive classes for boxes and spheres in conjunction with Java 3D texture mapping classes to texture the objects with detailed images. A RMBL3D view of the object models can be seen in Figure 4. Using Java 3D, it was also easy to create objects that run in different directions by simply switching around two of the object's dimensions. For example, suppose a viewer was facing a three dimensional wall object that was one meter wide, three meters tall, and ten centimeters in depth. One might also describe this object, from the viewer's perspective, as running east to west in front of the viewer. To create an object running north to south, using Java 3D it was simply a matter of switching the object's width and depth properties, producing an identical object flowing in a perpendicular direction.

There were some slight oddities in describing object sizes in Java 3D. In creating a box object model, it would seem the intuitive way to describe it would be to provide the object's volume dimension values, that is, its width, height, and depth. Using the box geometry class of Java 3D, however, the object is centered at the origin and stretched to the given X, Y, and Z dimension

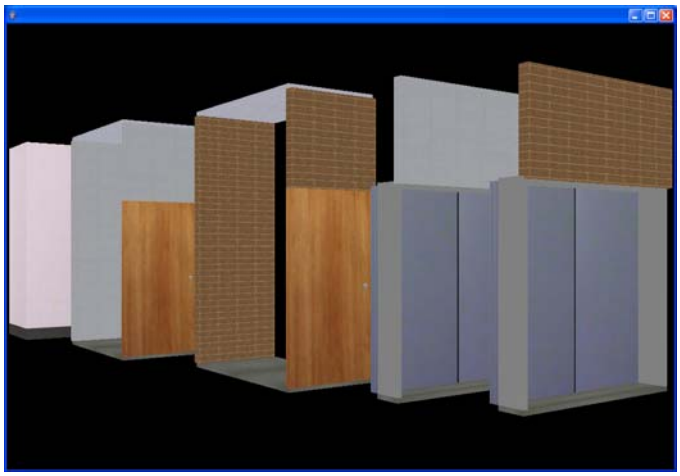


Figure 4: RMBL3D Object Models

parameter values. Thus, if one does not keep this in mind, one might unintentionally end up with an object double the intended size. Also since objects were created and centered at the origin, one had to take care to translate objects of different sizes to slightly different (X,Y,Z) locations in order for them to align on a 3D plane. We moved such mundane translational bookkeeping details into the object creation functions, leaving higher level functions to simply state the whole meter coordinate location at which the object was detected and if it was detected on the left or the right.

After the symbol map is read into the RMBL3D program, the map is parsed into either pairs of objects or into the set of five symbols representing a turn. During the parsing, the program simulates the movements the robot took in mapping the environment by keeping an updated directional variable and its total area traversed, allowing it to correctly place objects into the 3D world with updated (X,Y,Z) values. The program starts building the 3D virtual map at the origin (0,0,0), and it assumes the robot started out facing north. The symbol map is robocentric, meaning left and rights are always in relation to the robot and the direction it is traveling. The program therefore had to account for which direction the robot was traveling when it recorded the symbols. The program updates the direction as turns are encountered in the symbol map. Also, since we assumed objects were in meter segments, and since the Java 3D universe is in meters, as a new pair of symbols is parsed by RMBL3D and needs to be inserted into the 3D map, RMBL3D simulates taking a meter step in the environment depending on the current directional variable. Updating the (X,Y,Z) location variables as the program creates the map is simply a matter of incrementing and decrementing

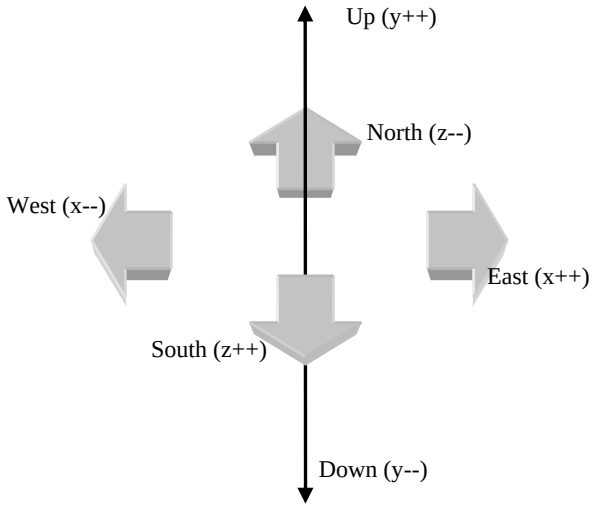


Figure 5: Taking a one meter step in the Java 3D world

according to Figure 5. Also, updating the directional variable follows the graph in Figure 6, with the nodes being the up-to-date direction and the edge labels R or L being the direction symbol encountered in the symbol map and the new direction which results from such a turn. When RMBL3D encounters the '+' symbol in the symbol map, which means start mapping on a new floor, the program jumps the Y variable up 4 meters so that now it simulates the robot on a new plane in the 3D world. Using this, we can map multiple levels of the same building in the same 3D map.

Once the program finishes reading through and placing all of the objects, turns, and level changes into the 3D world, it displays the results as a smooth flowing 3D map that one can navigate through and explore. Many of the traversal behaviors are enabled by using the JWSU toolkit and the keyboard arrow keys and mouse, but Java 3D also allows for implementation of custom keyboard

navigation features. To test this and to allow for some precise camera viewing locations, RMBL3D implements different camera views by translating the user's viewing location to the origin, to +100 meters in the Y direction above the origin (which would move the user to a flyover location), or +100 meters in the Z direction from the origin (which would move the user's view 100 meters backwards from the origin).

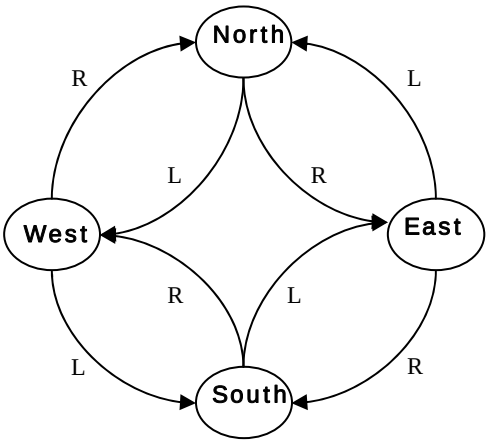


Figure 6: RMBL3D Direction Change graph

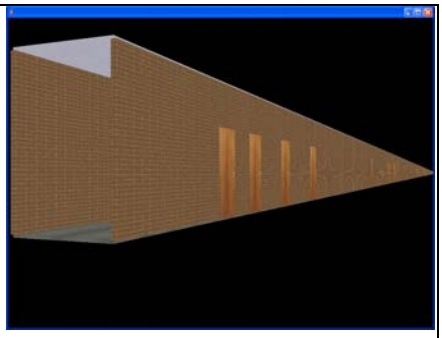
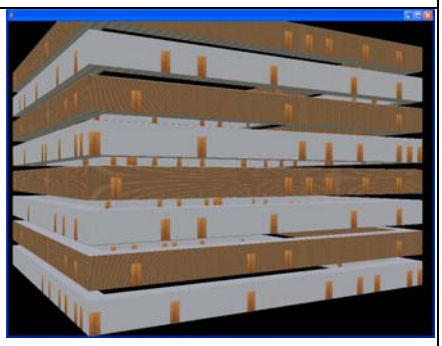
5. RESULTS

Identifying and using repetition and creating a virtual 3D map from a string symbol map, we achieved excellent results (see Table 2).

6. CONCLUSIONS AND FUTURE WORK

Virtual reality offers a rich, realistic experience for exploring and navigating 3D maps. 3D maps have become useful in site archiving, in training, and in virtual tourism. RMBL3D is a program developed for the 3D visualization of an object map. One can test any symbol map with the RMBL3D program to see the 3D map it creates, within the limits of the defined symbols and objects. Future work includes adding the robot into the system to explore the environment and autonomously acquire the object symbol map.

Table 2. Results Photos

 Boyd Foyer	 RMBL3D Boyd Foyer	 1 Kilometer Long Hallway in RMBL3D
 Boyd Hallway	 RMBL3D Boyd Hallway	 Multiple floors in RMBL3D

7. REFERENCES

[1] Biber, P., Andreasson, H., Duckett, T., Schilling, A. 3D modeling of indoor environments by a mobile robot with a laser scanner and panoramic camera. In Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004. (Sendai, Japan, Sept. 28- Oct. 2, 2004). IROS 2004, Vol. 4, pp. 3430-3435.

[2] Deligiannidis, L., Weheba, G., Krishnan, K., Jorgensen, M. JWSU: A Java3D framework for virtual reality. In Proc. of the International Conference on Imaging Science, Systems, and Technology. CISST '03, pp. 312-319, June 2003.

[3] El-Hakim, S., Beraldin, J., Picard, M., Vettore, A. Effective 3D modeling of heritage sites. Proc. Fourth International Conference on 3-D Digital Imaging and Modeling, 2003. (Banff, Canada, Oct. 6-10, 2003) 3DIM 2003, pp. 302-209.

[4] El-Hakim, S., Whiting, E., Gonzo, L. 3D modeling with reusable and integrated building blocks. 7th Conference on Optical 3-D Measurement Techniques, Oct. 3-5, 2005, Vienna, Austria.

[5] Haala, N., Brenner, C., Statter, C. An integrated system for urban model generation. In Proc. ISPRS Congress Comm. II, Cambridge, pp. 96-103.

[6] Hahnel, D., Burgard, W., Thrun, S. Learning compact 3D models of indoor and outdoor environments with a mobile robot. Robotics and Autonomous Systems, Vol. 44, pp. 15-27, July 2003.

[7] Howard, A., Wolf, D., Sukhatme, G. Towards 3D mapping in large urban environments. In Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004. (Sendai, Japan, Sept. 28- Oct. 2, 2004) IROS 2004, Vol. 1, pp. 419-424.

[8] Java 3D homepage, https://java3d.dev.java.net/

[9] Müller, P., Wonka, P., Haegler, S., Ulmer, A., Van Gool, L. Procedural modeling of buildings. ACM Transactions on Graphics (TOG). Vol. 25, Issue 3, pp. 614-623, July 2006. DOI= http://doi.acm.org/10.1145/1141911.1141931.

[10] Müller, P., Vereenooghe, T., Wonka, P., Paap, I., Van Gool, L. Procedural 3D reconstruction of Puuc buildings in Xkipché. 7th International Symposium on Virtual Reality, Archaeology and Cultural Heritage. VAST '06, pp. 139-146.

[11] Parish, Y. and Müller, P. Procedural modeling of cities. In Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques. 2001, pp. 301-308, DOI=http://doi.acm.org/10.1145/383259.383292.

[12] Stamos, I., Allen, P. 3-D model construction using range and image data. 2000 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (Hilton Head Island, SC, USA, June 13-15, 2000). CVPR '00, Vol. 1, pp 531-536.

[13] Surmann, H., Nuchter, A., Hertzberg, J. An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. Robotics and Autonomous Systems, Vol. 45, pp. 181-198, December 2003.

[14] Thrun, S. et al. A system for volumetric robotic mapping of abandoned mines. In Proc. of the IEEE International Conference on Robotics and Automation. ICRA '03, Vol. 3, pp. 4270 – 4275, May, 2003.