SANDIA REPORT

SAND98–0584 • UC–2060
Unlimited Release
Printed March 1998

Real Time 3D and Heterogeneous Data Fusion

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

RECEIVED
MAR 2 0 1998
OSTI

MASTER

Charles Q. Little, Daniel E. Small

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

19980420 057

SF2900Q(8-81)

Real Time 3D

and Heterogeneous Data Fusion

Charles Q. Little

Intelligent Systems Sensors and Controls

Daniel E. Small

Advanced Engineering and Manufacturing Software Development

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185 - 1003

Abstract

This project visualizes characterization data in a 3D setting, in real time. Real time in this sense means collecting the data and presenting it before it delays the user, and processing faster than the acquisition systems so no bottlenecks occur. The goals have been to build a volumetric viewer to display 3D data, demonstrate projecting other data, such as images, onto the 3D data, and display both the 3D and projected images as fast as the data became available. We have examined several ways to display 3D surface data. The most effective was generating polygonal surface meshes. We have created surface maps from a continuous stream of 3D range data, fused image data onto the geometry, and displayed the data with a standard 3D rendering package.

In parallel with this, we have developed a method to project real-time images onto the surface created. A key component is mapping the data on the correct surfaces, which requires a-priori positional information along with accurate calibration of the camera and lens system.

Acknowledgment

We wish to thank the efforts of several others. Colin Selleck is the person responsible for the LAMA structured lighting system, which was the primary 3D sensor used. His contributions made the continuous scanning possible. Chris Wilson provided good technical work and feedback in the implementation of algorithms tried, and was responsible for the vision systems used. Dan Horschel was the program manager and was instrumental in providing support for us to accomplish this work.

Contents

Figures

Real Time 3D and Heterogeneous Data Fusion

Introduction

Three-dimensional world models are desirable for a large number of applications for visualization and simulation. For many applications, it is sufficient to use well characterized pre-built models in an established environment. However, to effectively interact with real world environments, the model must match the real world, both in its geometry and the richness and quality of the information. The traditional approach to create models by hand is costly and very slow. In some cases, the time needed to generate the models often exceeds the useful life of the model, making it ineffective to exploit the model for visualization or task planning. This problem is compounded when the application includes working within the environment, where things move around or the scene changes. In this case, a new model or updates to register the change into the model are required.

• Background

The Intelligent Systems and Robotics Center at Sandia National Labs has a strong interest in 3D models for simulation and automated robotics. Automated systems need knowledge of their environment to interact with it; to avoid obstacles or pick things up. A useful way to store this knowledge is as a 3D model of the real world environment.

The research presented here builds on our 3D modelling work over the past few years. Recently, we have begun to replace the mostly manual CAD modelling method by direct modelling using 3D range sensors. These sensors gather surface data in the form of x-y-z points, which are currently being meshed together to form surface maps. This has reduced model generation time by an order of magnitude; from days to hours. This still does not approach where we would like it to be, which is creating and updating models as fast as the application interacts with the environment.

• Statement of the Problem

The problem we are addressing is the rapid generation of an accurate, feature rich 3D model of a dynamic real environment, such as digging up dirt in a buried waste site, slurping sludge from a tank, cutting pipes in a facility, or modeling a moving target [1]. Models of static environments will benefit by the ability to add complex scene information by fusing images onto the 3D model. A reduced time to model will be necessary to keep up with a changing environment.

A concurrent problem is that a model generated solely with range-mapped data is not sufficient to represent a real environment. Although the data has positional accuracy, the scene looks like a room ready for painting with tarps draped over everything. Other information is usually necessary to know where specific tasks must take place. Segmentation algorithms can be used to separate the scene into spatial objects. However, other sensors are often deployed to gather this discriminating data. It can be very desirable to map this data directly onto the 3D model. For example, a camera image

mapped onto a surface greatly increases user perception, both in object recognition and depth queues. Texture mapping is the standard method to do this on polygonal surface geometry, but these methods often fall apart in complex scenes. Some data is inherently volumetric in nature, such as radiation fields. It may be desirable to view this data volumetrically, but this can be a problem in a polygonal model because it is difficult to decide where to attach the data.

System Description

A description of the 3D modelling and data fusion hardware and software used is given to aid the discussion of the processes. The data collection system consists of an SGI Indy computer (low-end system) with a pair of color video cameras mounted on pan & tilt units, which we refer to as the visual targeting system. The 3D range sensor uses the same computer and incorporates an additional camera and a laser, also mounted on the same pan & tilt units. We refer to the range system as the LAser-MApper or LAMA. The camera and laser target the scene from different angles. The laser line is automatically located in the camera view and 3D range is calculated using triangulation. An infrared camera was also acquired for the project, and mounted on a third pan & tilt unit. Each device was calibrated, meaning quantitative measurements were taken to allow the position, orientation, and field of view, in space, of each unit to be known. Position and orientation parameters for each device are stored in a homogeneous transform matrix. Each camera also has internal calibration parameters that define the focal length, image plane size, and location of the optical axis in the image plane.

A high-end graphics computer, an SGI Onyx with Reality II graphics, is used to take advantage of the hardware for 3D geometric display and texture mapping.

Approach

• Current Method

Our current method of generating 3D models relies on a polygonal based surface geometry approach. Range data is translated from a series of scattered dots to a polygonal mesh that describes the surface. This approach usually requires filtering and other data reduction techniques to come up with a reasonable data size. One disadvantage is the time needed to go from range data to polygons (mentioned earlier as hours over a complex environment). A second problem is in model updating. Our current surfaces tend to be monolithic; the entire scan is presented as one surface. There is no easy method to update a portion of the model. It is difficult to delete parts of the old surfaces that are no longer there. Merging data from multiple views is also difficult, due to registration errors that degrade the outcome of surfaces that appear in two views. This becomes a bigger problem when it is necessary to capture multiple views to fill occluded spots.

• Volumetric Mapping

Work of others in area of 3D range mapping using a volumetric methods [2,3] lead us to develop a volumetric approach to handle this problem. These methods use a volume space to capture and store range data. This has the advantage of direct data storage,

meaning that all range data is stored directly in the volume. This is in effect a quantized version of the workspace, where each voxel is a digitized volume is space. The problem of updating changes in the model is bypassed by the very nature of data storage. Old data disappears when new data is entered. Merging data from multiple views is not a problem because the merge occurs during data storage, before model creation. An additional bonus comes with merging. Fusing data from multiple views, and using a statistical method to combine data within a voxel, greatly decreases the false readings from noise and other inherent sources such as reflective surfaces, increasing accuracy at little cost.

The basis of this method is in the mapping of range data to the volume. The voxels are nominally assigned one of three values:

- *Unknown* – All voxels begin as unknown.

- *Surface* – Where a range point falls into a voxel, it is declared a surface.

- *Empty* – This is the value given for empty (air) space.

Since we know the location of the sensor, we cast a ray from the sensor to each range point. Voxels along the path are set to *empty* until the range value is hit, at which point the voxel is set to *surface*. Figure 1 shows a 2D diagram of a ray cast and its effect on voxels in the path. This algorithm is modified when adding statistical averaging by assigning a multiplying factor. Multiple hits from different ray casts will increase the probabilities of the assignment.

Figure 2 shows a 3D example of volume rendering, with voxels assigned to *unknown*, *empty*, and *surface*.
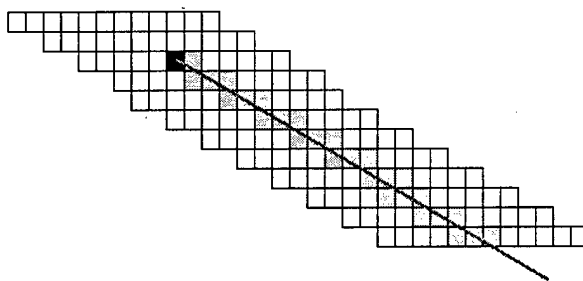

Figure 1. 2D data mapping; showing unknown (clear), empty (gray), and surface (black)
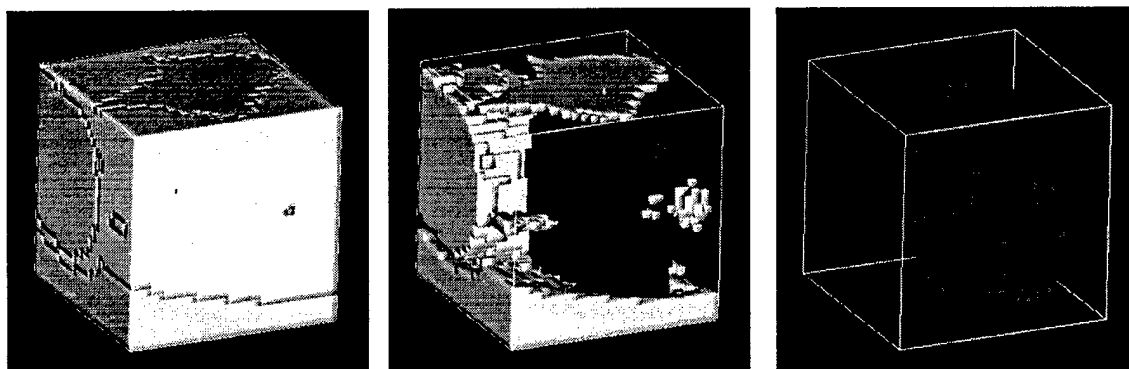

Figure 2. Volume rendering a) *unknown* + *empty*, b) *unknown* + *surface*, c) *surface*

9

The initial goal of the project was to directly view the data from the volume using a standard volume rendering method. Early results however were disappointing in terms of speed of rendering. Rather than spend effort in writing a new visualization method, efforts where spent on an alternative approach using surface rendering, which is described next.

- **Continuous Surface Mapping**

The primary goal was to display model data as fast as we can receive it. An alternate method to accomplish this was to modify the current method of gathering range data and creating a triangle mesh to represent the surface. Two steps were taken.

The first change was to redesign the interface to the LAMA sensor to allow it to send a continuous stream of data. In this mode, the sensor data is not buffered until a scan is complete as before. Rather, the data is sent on a continuous basis. The modification required two parts. First, the communications to the sensor where modified to use a CORBA communication protocol, that allows for a continuous data stream connection. The second included a scan mode that itself is continuous. The result is very useful where the same scene is to be continually scanned from a fixed point, or if the sensor is placed on a moving platform that can roam a large area.

A second step was to implement a turntable platform. This now enables us to move a small target through a complete rotation instead of moving the sensor. This allows us to gather multiple views from different locations and continuous scanning without the need of other hardware to move the sensor. The latter is particularly useful in developing algorithms for combining multiple views that rely on well registered data, that is, knowing precisely where one data set is relative to the other. This is usually accomplished using fairly expensive motion hardware to move the sensor or time consuming calibration after moving the sensor manually. Figure 3 is data from a continuous scan, showing the raw positional points and the resulting surface. Figure 4 consists of two fixed views taken at different views. This is an example of multiple views from different positions.
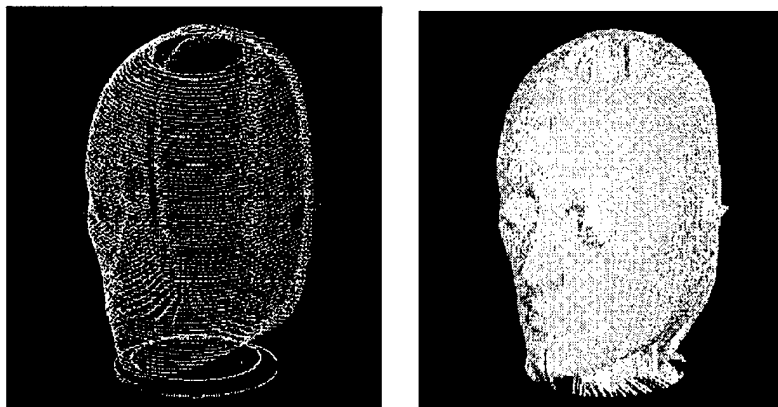

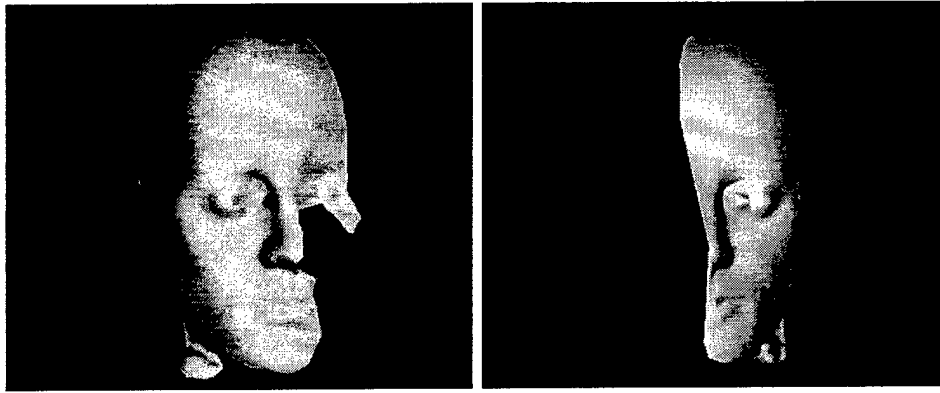Figure 3. a) 3D range data, b) continuous surface of a head model

Figure 4. Multiple registered views of head model

Heterogeneous Data Fusion

3D surface information is very often only the beginning of a useful world model. Virtual reality, for example, places much more emphasis on texture mapping than geometry because of its power in user perception. Our goal is to maintain high resolution geometry, primarily because of its use in robot planning, but fusing other data is also desirable. However, fusing heterogeneous data, such as radiation or intensity images, onto the surface model has not been feasible in the past because of the complexity of the problem.

• Discussion of Static Texture Mapping

Texture mapping is the mapping of a 2D image to the surface of a 3D model [4,5,6]. Static texture mapping is very useful for data fusion, especially in areas that don't change, such as background scenes. Traditional texture mapping onto polygons works well for simple surfaces -- one image for one rectangular polygon. However, it runs into major trouble when one image needs to drape over many polygons, particularly when they are oriented in a haphazard manner, precisely what we see with the surface maps generated from real world environments. The technique which we explored required an offline process of running each polygon vertex (the corners) through a matrix multiplication with the camera parameters to translate the point into the 2D (u, v) image plane. By performing this calculation for each vertex we can associate every vertex in the model with a point in the image, which gives us the data necessary to perform texture mapping. Although this is computationally intensive, the graphics hardware handles much of this processing.

Problems occur when the image and surface area do not quite correspond. This shows up when the projection of the image falls off the surface model and where the surface model extends beyond the image. These are usual events with our mapping system, because the cameras and range sensors are in different places. Texture mapping of areas that fall outside the image is ill defined. A common approach is to stretch the edge pixels over the uncovered areas. This effect results in registration errors. Figure 5 is an image that we desire to texture map. It shows the top half of a door. Figure 6 shows a simple box surface model of the wall that contains the door. It then shows the result of standard texture mapping. Note the stretching or smearing visible from the edge of the image.
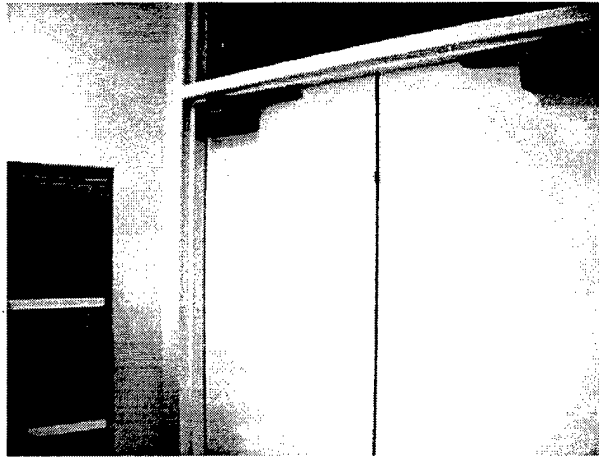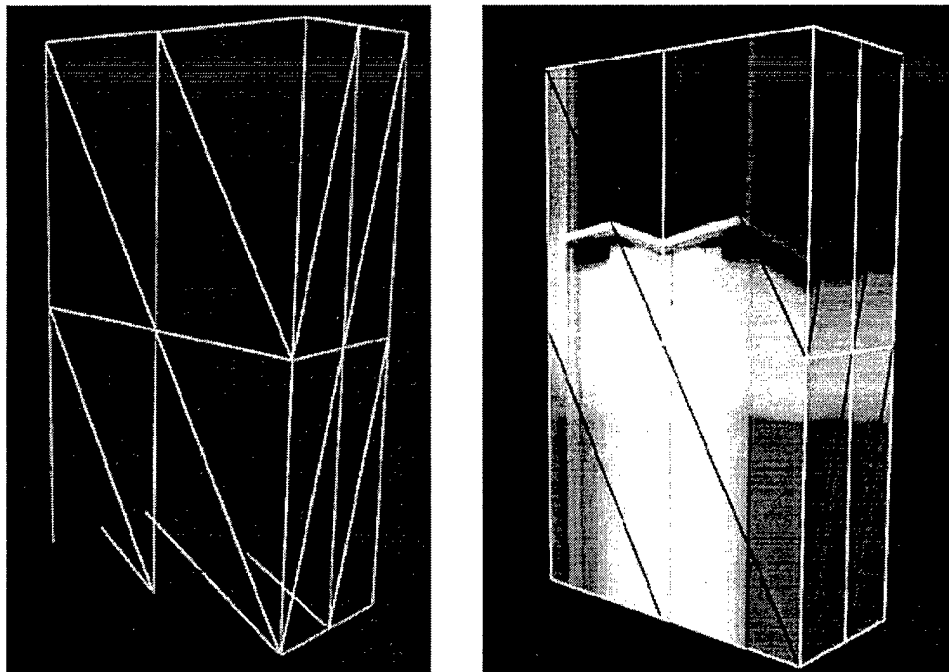
11

Figure 5. Image to be texture mapped.

Figure 6. Surface model a) without and b) with texture mapping. Note edge distortion.

- **Solving the Overlap Problem**

The static mapping technique requires a polygon and a corresponding image. For an accurate model, each polygon must be associated with the proper image at the proper location. Every camera image gathers data over a volume of space. This volume is a frustum, and it is defined by the camera position in space and the lens parameters. It can be thought of as a prismatic cone, which begins at the camera lens and expands through the camera field-of-view (the width and height of the camera image) into space. Polygons inside the frustum can be mapped. Those on the outside cannot. Those that fall in between (overlap) are subject to mapping error.

This problem of image overlap is solved by running the entire data set (geometry and image data) through an algorithm, which regroups the surface into sets defined by the

12

image's frustum. The surface is defined as a collection of polygons, usually stored as a list of triangles. This algorithm is designed to filter a triangle list into 2 new lists; an *in* list and *out* list. The discriminator used to place the triangle on the *in* or *out* list is the camera frustum. Specifically, the vertices of each triangle are compared against the planar equations that make up the frustum.

Pseudocode for this algorithm follows:

1. If a triangle is inside the frustum it gets put on the *in* list.

2. If a triangle is outside the frustum it gets put on the *out* list.

3. If the frustum splits a triangle, then the triangle is split into smaller triangles. Those new triangles are then re-filtered according to the previous criteria by placing them either on the original triangle list, or on the *out* list. The entire process is repeated until the original triangle list is completely processed.

Figure 7 illustrates this algorithm. The frustum appears as a prism in space, beginning at the camera center and extending along the field-of-view of the image. Where this prism intersects the surface model, the model is re-triangulated, actually creating two model patches; one inside the frustum space and the outside. The inner patch results in a matching correspondence of the surface polygons with the image, thus solving the overlap problem.

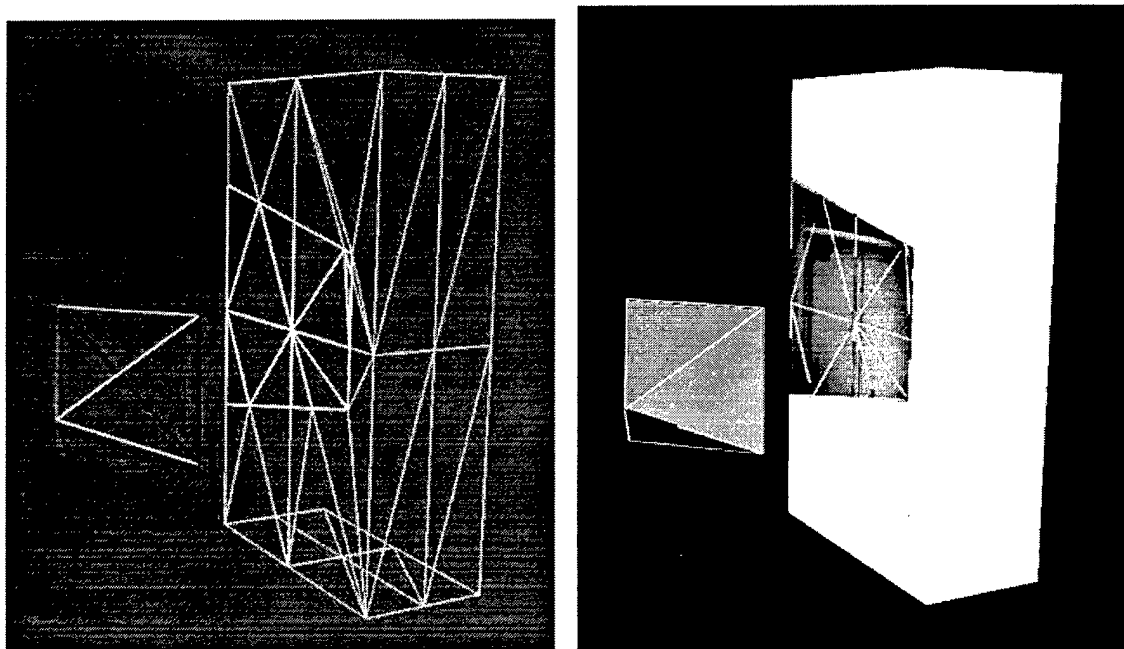Figure 7. Re-triangulated surface model a) without and b) with texture mapping.

• Data Fusion by Projected Texture Mapping

The projective texturing method directly projects 2D image data onto a 3D surface, as if you were using a slide projector. It requires knowledge of the location and orientation of the camera to present the data for projection, but this is available due to the manner in

13

which we deploy the imaging sensors (we calibrate sensors at their deployment locations). It also requires an extra rendering pass through the scene graph, but it is possible to feed the image and position data into the system in real-time, thus updating the projection far faster than updating the static textures. This was implemented using IRIS Performer on an SGI workstation [7,8].

We implement this using shared memory on the rendering machine to store the image-data and the camera pose information. (Shared memory allows for continuous updates to shared memory via an external process which frees the rendering process from performing the data exchange over the network from the sensor system). A virtual camera is created in the scene, which has the same position, orientation, and lens characteristics as the camera. The shared memory is read and the virtual projector data is updated several times per second. The image is then projected onto the geometry. Figure 8 shows a block diagram of the process.
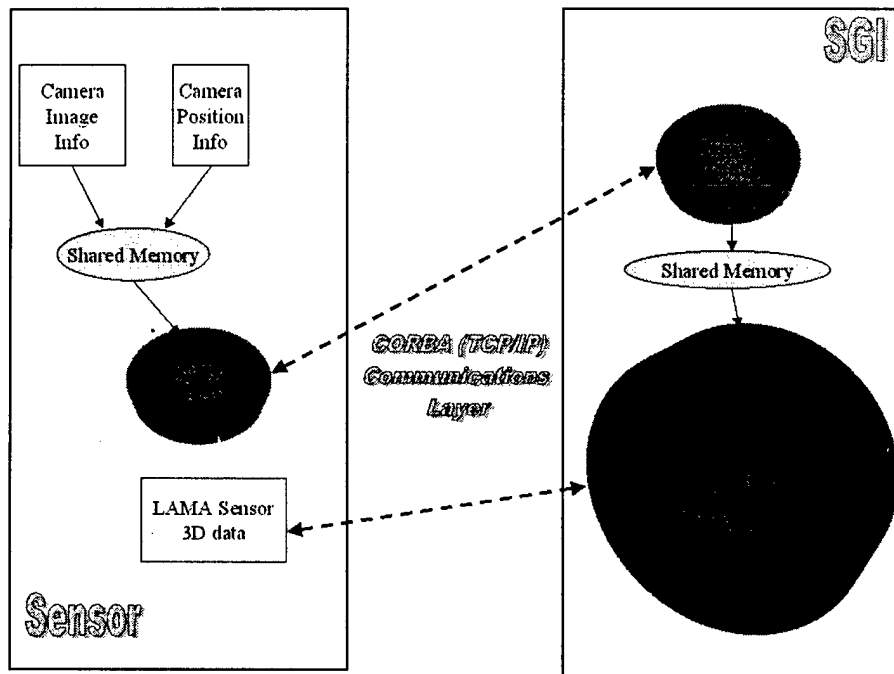

Figure 8. Block Diagram showing real-time image data path.

The "virtual camera" can update the texture on the geometry it "sees" dynamically. This is useful for parts of the model that will change appearance rapidly. In addition, by synchronizing the position of the real and virtual camera, the projection stays current and registered to the scene even as the camera moves. Another key advantage to using this method is that the projection can be rendered onto a surface that has been statically texture-mapped, thus allowing a new method for fusing multiple data sources in the 3D environment.

Figure 9 is an infrared image. The hot spots (dark or red) are a heating pad inside the box and a cup of hot coffee. Figure 10 shows projected texture mapping of this image onto a surface model. Because the data is projected onto the scene in real-time, the user can observe the objects heating up and cooling down. Figure 11 shows the fusion of real-time

projection of infrared data onto the 3D surface, which has been previously texture-mapped using the static texture-mapping technique.
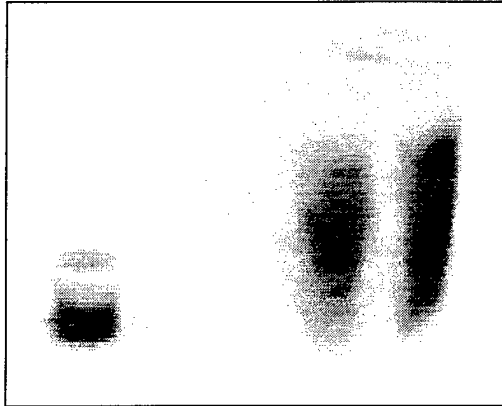

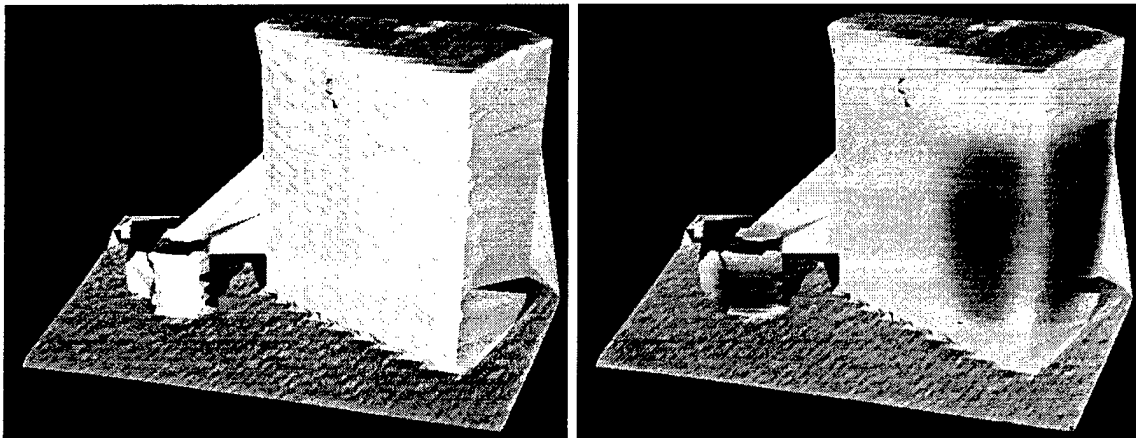Figure 9. Infrared image of coffee cup and box with a heating pad.

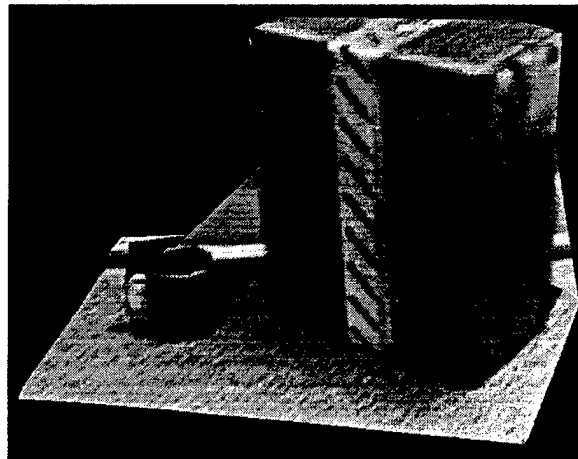Figure 10. A surface map without and with a projected texture visible on the mesh.

Figure 11. Infrared projection added to a statically texture mapped surface.

15

Additional Examples

Here are two more examples from world models that we have made. The first is from our lab (see Figure 12). The initial surface map (the walls and table), where taken with the LAMA sensor in a conventional scan. This took about 45 minutes. These surfaces are statically texture mapped, using 15 views over 25 surface maps. The time needed for static texturing is about 5 seconds per surface map. This figure shows the progression from range data to surface data to texture-mapped surface data.
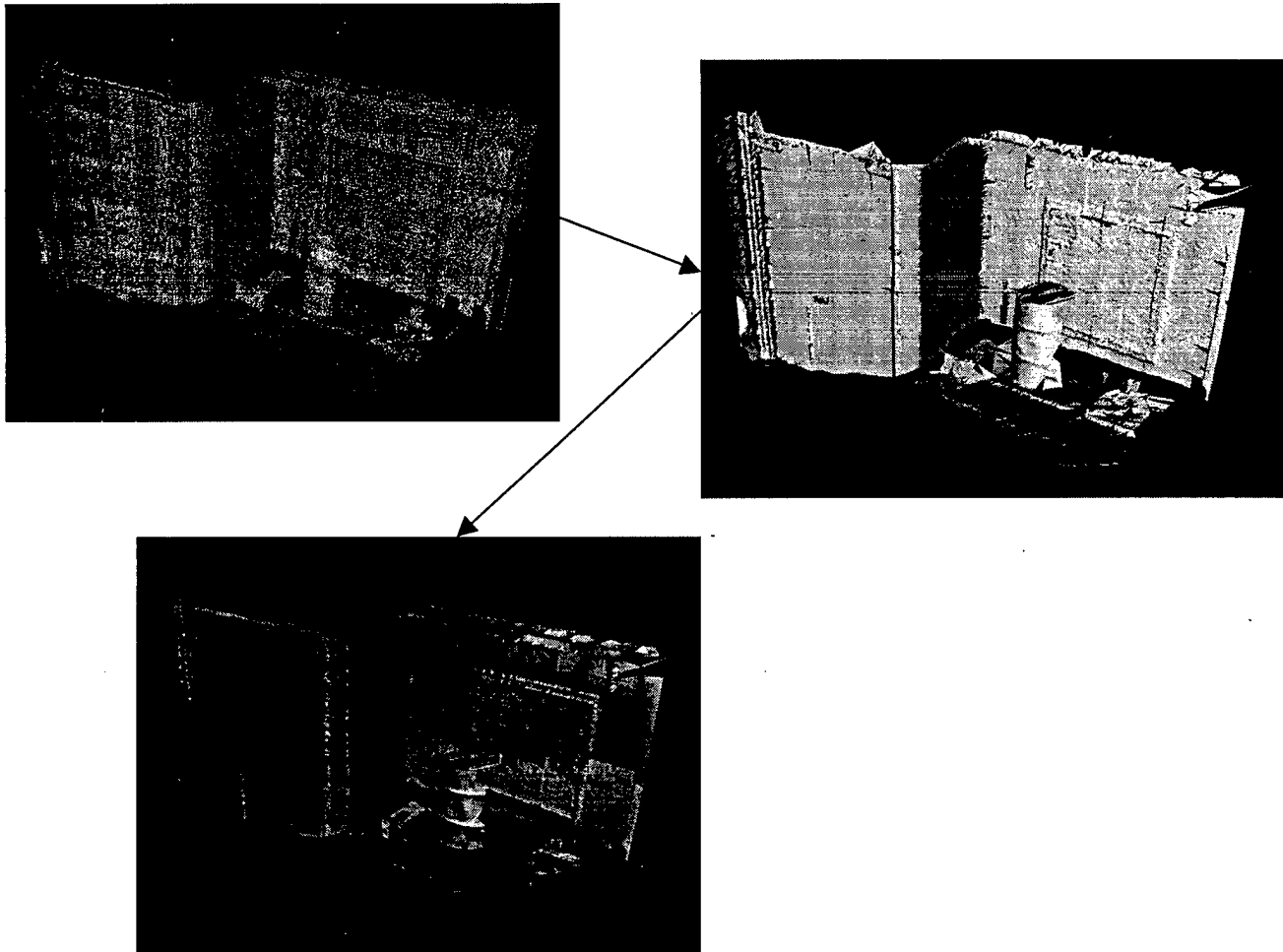

Figure 12. 3D world model of lab: a) raw range data, b) surface map, and c) texture mapped surface.

The second example is of a pipe header (see Figure 13). The area is about 15 feet by 15 feet. Color differences in the texture map are the result of using an auto-iris on the camera, which adjusts the iris by average light entering the lens, which changes as the camera is pointed in different directions. There are roughly 90,000 polygons in the final model, with 20 scans. There are 16 images in the texture mapping. Again, the bulk of the time is in the raw range data collection.
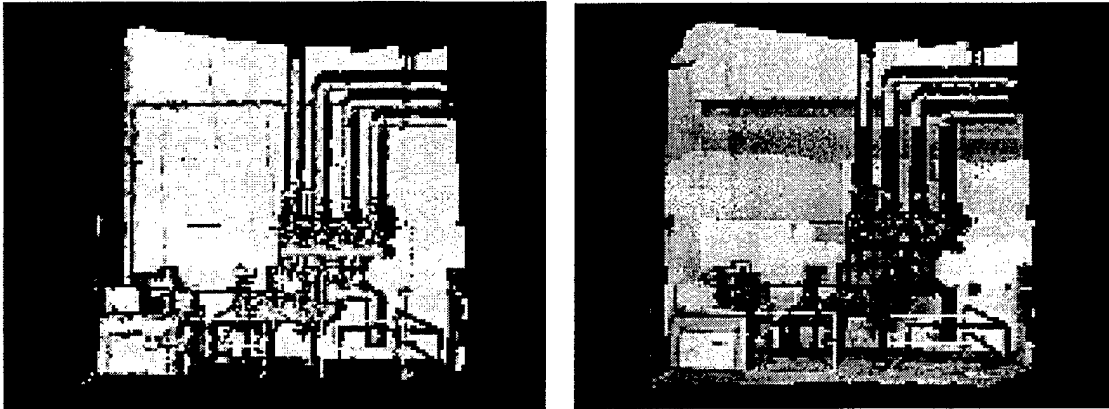
Figure 13. 3D map of pipe nest: a) surface map, b) texture mapped surface.

Future Directions

There are two main thrusts to move this program forward. These are to port the 3D range sensor to a PC, and to deploy it on a mobile platform to enable rapid data collection from multiple views. We would also like to use the Sandia-developed Scannerless Range Imager and the Synthetic Aperture Radar systems for real-time data sources. Revisiting the volume rendering method would be next, to exploit its benefits.

Conclusion

The goal of this project has been to visualize characterization data in a 3D setting, in near real time. Real time in this sense refers to the time needed between collecting the data and presenting it to a user. Along these lines, we set three main milestones. First was to build a volumetric viewer to display 3D data. Second was to demonstrate projecting other data, such as images, onto the 3D data. The third was to display both the 3D and projected images as fast as the data became available. We have met all three milestones.

We have examined several ways to display 3D surface data. The most effective to date has been generating polygonal surface meshes. We are creating surface maps from this data and passing this to a commercial 3D geometric display package.

In parallel with this, we have developed a method to project real-time images onto the surface created. A key component is mapping the data to fall on the correct surfaces, which requires a-priori positional information, along with calibration of the camera and lens system.

To achieve a continuous display, we have modified a 3D range sensor to provide a continuous stream of range data. The processing for surface rendering currently is faster than the cycle time for the range sensor (roughly 5 to 10 Hertz). The projected texture is added independently.

We have experimented with standard video images because of the ease in getting this equipment. Lately we have added a thermal intensity camera for a different heterogeneous data type. Other sensors that produce a 2D image, such as a gamma camera, would be trivial to add. Sensor data from point sources, such as chemical

sniffers, can be added directly as a geometric entity or mapped on the surface through a simple pseudo-color intensity scheme.

References

[1] R. Barry, C. Little, B. Burks, "Requirements and Design Concept for a Facility Mapping System", American Nuclear Society Sixth Topical Meeting on Robotics and Remote Systems, February 5-10, 1995

[2] B. Curless, M. Levoy "A Volumetric Method for Building Complex Models from Range Images", Proceedings of SIGGRAPH 96, ACM Press

[3] T.S. Yoo, U. Nuemann, H. Fuchs, S.M. Pizer, T. Cullip, J. Rhoades, R. Whitaker, "Direct visualization of volume data," IEEE Computer Graphics and Applications, volume 12 number 4, July 1992, pg. 63

[4] OpenGL Programming Guide, chapter 9—Texture Mapping, SGI Developer Reference Library, Addison—Wesley Publishing Company

[5] J. Foley, A. Van Dam, Fundamentals of Computer Graphics, 1982 by Addison— Wesley Publishing Company, pg. 589

[6] J. Blinn, Jim Blinn's Corner: A Trip Down the Graphics Pipeline, 1996 by Morgan Kaufmann Publishers, Inc., pg. 171

[7] Programming on Silicon Graphics Systems: An Overview, chapter 6—IRIS Performer, SGI Developer Reference Library

[8] OpenGL Programming Guide, chapter 3—Viewing, SGI Developer Reference Library, Addison—Wesley Publishing Company

Distribution

1	MS 0151	G. Yonas, 9000
1	0188	C. E. Meyers, 4523
1	1002	P. J. Eicker, 9600
3	0755	D. S. Horschel, 6233
5	1003	C. Q. Little, 9611
5	1010	D. E. Small, 9622
1	1003	R. D. Robinett, 9611
1	1003	C. B. Selleck, 9611
1	1010	M. E. Olson, 9622
1	1010	A. Ames, 9622
1	1005	B. Davis, 9601
1	1004	R. W. Harrigan, 9623
1	1006	P. Garcia, 9671
1	1007	A. T. Jones, 9672
1	1008	J. Fahrenholtz, 9621
6	1008	S. Blauwkamp, 9621
1	1125	A. K. Miller, 9652
1	9018	Central Technical Files, 8940-2
2	0899	Technical Library, 4916
2	0619	Review & Approval Desk, 12690 For DOE/OSTI