# Work in Progress: GPU-assisted Surface Reconstruction and Motion Analysis from Range Scanner Data

Daniel Wesslén*
University of Gävle

Stefan Seipel†
University of Gävle

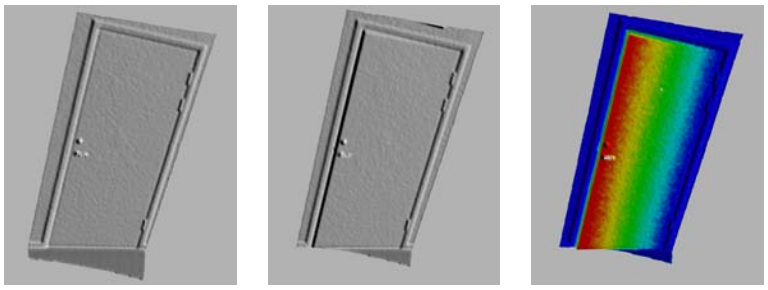

Figure 1: Reconstructed surfaces from two scans of the same door and exaggerated, color coded difference.

## Abstract

We present a method for rapid GPU-assisted surface reconstruction from range scanner data producing meshes suitable for visualization and analysis of very slow-moving objects from multiple scans of the same area.

**CR Categories:** I.3.6 [Computer graphics]: Methodology and techniques—Graphics data structures and data types, Interaction techniques

**Keywords:** surface reconstruction, meshing, point clouds, laser scanner

## 1 Introduction

Usually, range scanners or other point measuring equipment is used to sample an object from which a surface will be generated, reduced or otherwise processed. The final model is expected to be used on many occasions and therefore the time spent on constructing it is considered expendable. [Amenta et al. 2002; Bajaj et al. 1995; Curless and Levoy 1996]

We are facing a different situation—scans are to be acquired at regular intervals and the difference between them used to detect motion. Analysis will be both manual and automatic. A display will be provided for users to look at the data, which will have to be exaggerated for differences to become detectable. Automatic analysis will run continuously, alerting a user when potentially interesting data is detected.

*e-mail: dwn@hig.se
†e-mail: ssl@hig.se

In this scenario meshes become irrelevant when new data is available, so time spent on creating a perfect mesh is wasted—especially if nobody is looking at the display, in which case it will be used only once for automated analysis.

It is also desirable to quickly be able to bring up a mesh representing changes between any two previous scans, providing further incentive to find a solution that executes as fast as possible.

### 1.1 Previous Research

Automated surface reconstruction from point cloud data sets and depth images has seen much research, though usually focused on finding meshes that optimally fit the input point set in some way or that use as few triangles as possible. We have found no existing method suited for our conditions.

## 2 Surface Reconstruction

Surfaces are generated as regular grids, much like the input data. Optionally, meshes may be generated from multiple point sets at once, in which case corresponding vertices in the two meshes will correspond to the same direction from the scanner. This is the most common mode of operation since it facilitates motion analysis.

Reconstruction is fairly strightforward—input data is cleaned up, matched to the target raster, cleaned up again, and normals are then generated—the interesting part is that this process is almost entirely performed on the graphics card.

### 2.1 Cleanup

Input point sets may contains both valid points, invalid points where no measurement was recorded, points where the light was reflected from multiple surfaces, and spurious incorrect values which do not correspond to any surface. When light is reflected from more than one surface the reported distance will lie between the correct alternatives, disconnected from both surfaces.

The first task is thus to clean the data by removing invalid points. Apart from actually loading the data, this is the only part of the algorithm that is currently performed by the CPU.

A simple sweep of the data is performed, replacing the values of all points which are invalid or disconnected from the surface. Values are considered disconnected if their distance from neighbouring values in all directions is greater than a threshold. Coordinates are also restored to scanner-centered spherical coordinates to ease processing in later stages.[1]

### 2.2 Matching and Meshing

For motion analysis to work, vertices in one mesh must match corresponding vertices in the other. This would be trivial if all data sets were sampled from exactly the same directions, but different scans may use differing areas of interest and resolution, meaning that the advantage of a regular grid input is lost at this point.

We calculate the spherical coordinate bounds of the two meshes and use it to calculate the raster size of the final mesh. Two floating point frame buffers which will be used for further processing are created at this size. Linearization of the data and is then performed by splatting points at one of the floating point buffers, starting with single pixel points and followed by successively larger points. Depth testing is used to avoid overwrites of pixels that have already been set at smaller sizes.

At this stage the offscreen framebuffer contains only valid points located at mostly correct positions. Some pixels will contain duplicate values from when larger points were drawn. To remove duplicates and place all points in a regular grid, the frame buffer is copied to a vertex buffer and used to draw a triangle mesh. Interpolation over the triangles will give all covered points a linearly weighted depth corresponding to their position in the mesh. Duplicate points will result in degenerate triangles and simply disappear.

Some cleanup is also performed—the points created here could be used as a mesh directly, but drawing triangles blindly will result in artifacts where surfaces are disconnected as triangles connecting the edge verices will still be drawn. Such triangles are eliminated by alpha testing—the test is set to $\alpha > 0$ and vertices located near discontinuities in the mesh have their $\alpha$ set to 0. Interpolation will cause triangles near discontinuities to have $\alpha \in (0,1)$, which will still keep them visible. Only triangles where all vertices have $\alpha = 0$ will be invisible.

Some errors are always introduced in the scanning so the penultimate step is a smoothing of the mesh, performed by simple averaging of adjacent depth values.

Finally, all points are converted to cartesian coordinates and normals are calculated.

## 3 Display

In order to display differences, vertex buffers from two meshes are bound simultaneously and a vertex shader is used to interpolate or extrapolate a coordinate from the provided two. Additionally, a fragment shader samples a color ramp texture to provide color based on the distance between the two meshes.

[1] Points are stored in cartesian coordinates in all formats supported by the capture software used.

Figure 1 displays two different scans of the same door. The scans use slightly differing areas of interest and the door is ajar in scan number two. Image 3 displays color-coded and slightly exaggerated difference in the area covered by both scans. This is the intended main display mode in the final application.[2]

## 4 Performance

Graphics card locality is a major concern for performance. With its drastically higher throughput we wish to offload as much work as possible to the GPU, but reading this data back to the host is a bottleneck. The current solution can be executed entirely on the graphics card, given a card and drivers that support render-to-vertex-array. Unfortunately, such drivers are not yet available for the Radeon 9800 used in development so we have to copy data from card to host and back again at multiple points during execution.

Only one stage in the vertex processing is currently performed on the CPU—the cartesian to polar coordinate conversion. A bounding box needs to be calculated from the polar coordinates, which is far easier to do on the CPU, leading to a choice between performing the conversion directly on the CPU or on the GPU followed by a readback and loop for bounds calculation. The ideal solution would perhaps be to calculate bounds on the GPU so that only a very small readback is required.

The source data for the door displayed in Figure 1 contains 149 211 points. The application generates a mesh from the point set in 0.22 seconds in its current state—crippled by unnecessary readbacks.

## 5 Discussion and Future Work

The weakest point is currently that polygons are generated between disjoint surfaces. While these are never seen, fragments need to be generated and processed for them. A solution would be to generate new index sets that omit these triangles. Ideally, this would be performed entirely by the GPU so that data must never be read back to the host. Implementing this will likely be the next improvement made.

Overall, the system works well for the intended use—it is optimized for rapid mesh generation from two point sets and subsequent comparative display of these.

## References


AMENTA, N., CHOI, S., DEY, T. K., AND LEEKHA, N. 2002. A simple algorithm for homeomorphic surface reconstruction. *International Journal of Computational Geometry and Applications 12*, 1-2, 125–141.

BAJAJ, C. L., BERNARDINI, F., AND XU, G. 1995. Automatic reconstruction of surfaces and scalar fields from 3D scans. *Computer Graphics 29*, Annual Conference Series, 109–118.

CURLESS, B., AND LEVOY, M. 1996. A volumetric method for building complex models from range images. *Computer Graphics 30*, Annual Conference Series, 303–312.


[2] The door is used for testing and is not related to the final application.