Domitilla Catacomb Walkthrough – Dealing with more than 1 Billion Points

Claus SCHEIBLAUER
Institute of Computer Graphics and Algorithms, Vienna University of Technology

In this paper we would like to present our application for editing and visualizing huge point clouds. We also show how we used our application to build a model of the Domitilla Catacomb in Rome, which is the largest catacomb in Rome. It is an Early-Christian necropolis spanning 15 km of subterranean galleries. The catacomb contains tombs and community burials, where some of the tombs are painted. The catacomb was documented during 8 scan campaigns where range laser measurements were taken, and also color photos were taken. We used the data from these scan campaigns to build a model that consists of more than 1.2 billion points. After the model was built it is possible with our application to make interactive walkthroughs through the model. We describe the current state of our development of our system, because it is still work-in-progress. At the moment we are already able to build, merge, edit, and visualize multiple point clouds with a size of up to 2 billion point samples per point cloud.

point-based rendering, out-of-core processing, range scanning, virtual reconstruction

Introduction

Range laser scanning evolved as a means for documenting buildings or archeological excavation sites. An example is the Domitilla Catacomb in Rome (Fig. 1). The quality of the documenting scans depends on the density of points that are taken during the scanning. So the point clouds resulting from these laser scans can consist of hundreds of millions of points for a high quality model of the scanned object. After scanning, the

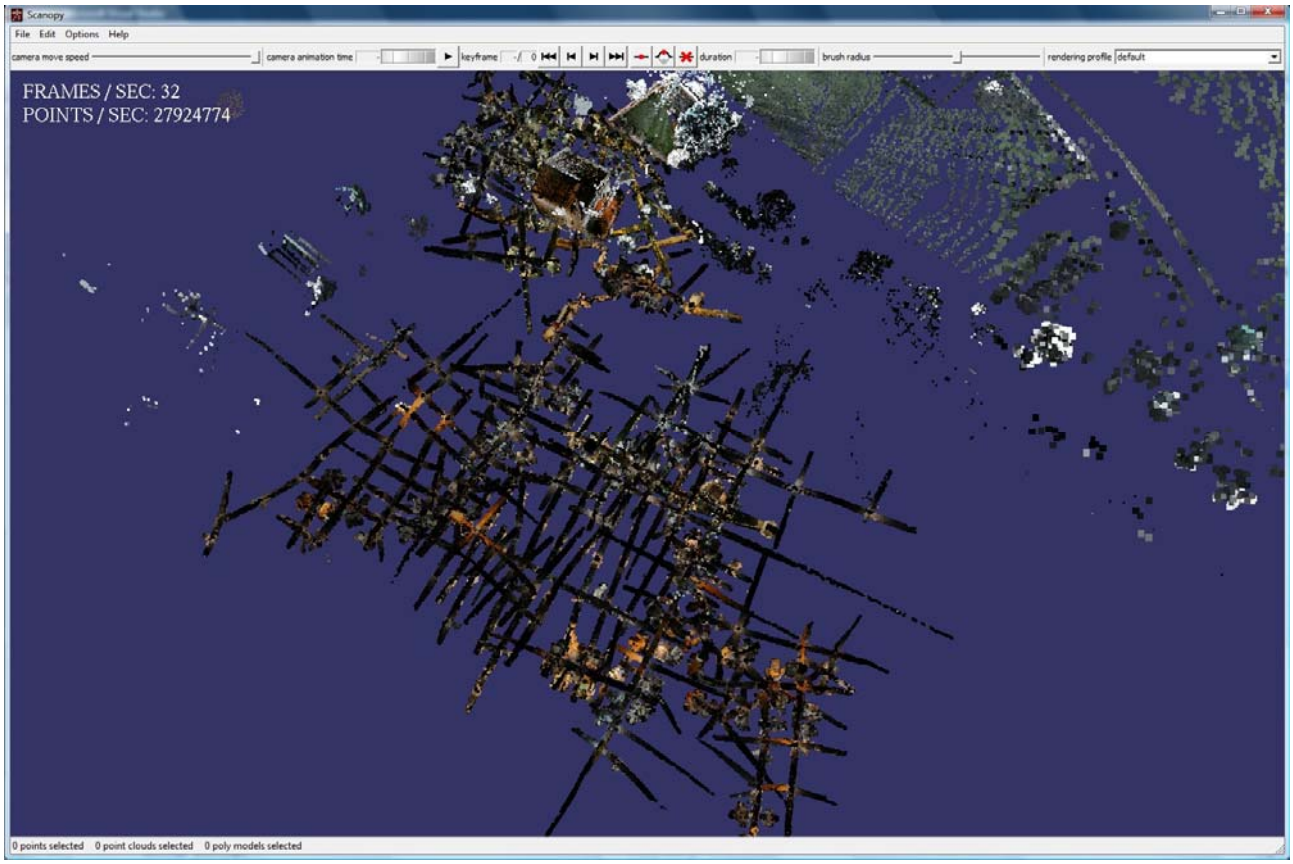

Figure 1 Overview of the complete Domitilla model.

data has to be processed to get a clean model that can be used for high quality renderings. To get a clean model from this vast amount of data needs several person months. Instead we try to avoid this work and visualize the data that is coming from the laser scanner directly. The quality of the models that are produced without cleaning the data is not the same as the quality of the models produced after cleaning the data, but it is possible to get an overview of the scanned objects directly after taking a scan. This can be useful for archeologists, e.g., during a scanning campaign to identify areas where are still too few samples taken.
The models in our viewer are combined point clouds from several scan positions. The models can become so large, that even with today's computer hardware it is not possible to display all points of such a model at the same time for an interactive walkthrough. Because of this we included a level-of-detail (LOD) algorithm and an out-of-core algorithm. The LOD algorithm displays the model in a level-of-detail that is sufficient for the current user viewpoint. The out-of-core algorithm manages the swapping of points in and out of main memory.
Our system supports nearly arbitrarily large point clouds for viewing but also for editing. It is possible to add new point clouds to an existing model, and we also allow for deleting points from the model. Furthermore we developed a heuristic to estimate point sizes, which enables the viewer to display objects with closed surfaces without a special preprocessing step. Note that the surfaces only appear closed, because the point size is chosen such, that the holes in the model become closed for most areas of the model. For the point size heuristic it suffices to know the positions of the points.

Data Structure

The models are stored in a data structure called NestedOctree (Wimmer and Scheiblauer, 2006), which organizes points spatially. This is done in a way, so that levels of detail (LODs) can be used without additionally created points. This means that we only use points from the original point clouds and we do not change their positions. The data structure is based on an octree, which is a recursively defined regular grid, where 8 cells are organized as a cube, and the length of the cube is 2 cells along each axis. In a NestedOctree each node holds points, no matter if it is an inner node or a leaf node. A LOD can now be created by adding up all points from the root node to the nodes that build the visibility front. The visibility front is the set of all nodes in the NestedOctree that are furthest down in the octree hierarchy and are still visible from the current viewpoint.
The points are stored in files on the hard disk, and each file represents one node in the NestedOctree. During rendering all nodes which are currently visible are read in from disk. Reading point files is done in a separate thread, so that rendering is not interrupted during loading the point files.
The user interacts with the point clouds by a graphical user interface (GUI), where he can load point clouds, buildup models from separate point clouds, and move through point clouds interactively. The user interface and data structure are part of our application, which is called Scanopy. This is also the name of the project that is sponsoring our work.

Build Up

One model is compound of several smaller point clouds, which are typically the results of different laser scans. The buildup process merges the points from several smaller point clouds into one large point cloud. This is necessary for later to know the density of points at any area of the model, because the density can then be used to efficiently display the LODs and to calculate the point size heuristics. The buildup of the

model is accomplished with an out-of-core algorithm, since not all points of a large model fit into main memory. The nodes of the NestedOctree hold references to the points that are stored at the nodes. During the buildup process the nodes are always kept in memory, only the points are swapped in and out of memory. The swapping is managed by a Least-Recently-Used (LRU) cache.
The points are inserted one-by-one from the root node, and if a point cannot be inserted into a node, the point is rejected from this node and it is tried to insert the point into a node further down the hierarchy. This is a recursive procedure, until the point finds a node where it fits into.

Point Size Heuristics

The model is most easily rendered with constant point size. This has the drawback that points are rendered either too large or too small, because the points are not equally distributed throughout the model, and holes become visible at the surface of the model. If the point size would be appropriate to fill in the holes in one part of the model, the same point size would not be appropriate (e.g., too large points or too small points) in another part of the model. Therefore we developed a point size heuristic, which dynamically determines the point size at each rendered node of the model, depending on the number of points that are loaded in the visible NestedOctree nodes. The point size is therefore the same for all points of one node of the NestedOctree.

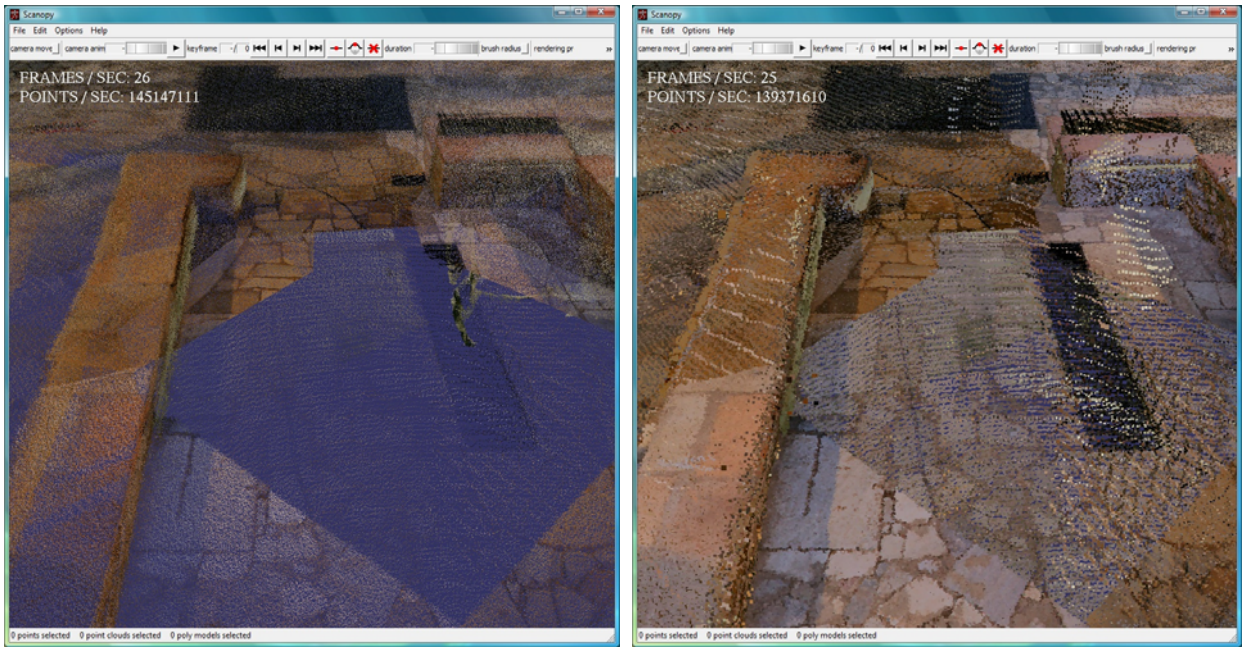

Figure 2 The left image shows the floor of a scanned building rendered with constant point size of 1. The image on the right side shows the same area of the floor rendered with dynamic point size.

Using the point size heuristic it is possible to visualize the model with a closed surface, but without estimating normals or point sizes in a pre-processing step (Fig. 2). One drawback of the point size heuristic is that errors in the colorization become more visible, as can be seen in the right image of (Fig. 2). This is due to different lighting conditions during scanning from different positions. The surface is not in correspondence to the real surface of the model, it is just an estimation of the model's surface that is derived from the density of the points. The points are rendered as quadratic screen aligned splats, so aliasing artifacts remain visible, especially at the edges of objects.

Selection

An application of the point size heuristic is the selection of points. The user can move a selection-sphere along the visible surface of the model (Fig. 3). Since the model has a closed surface the movement of the selection-sphere is smooth along the surface of the model. When the user presses the left mouse button, points inside the selection-sphere become selected, when the user presses the right mouse button, selected points can be deselected again.

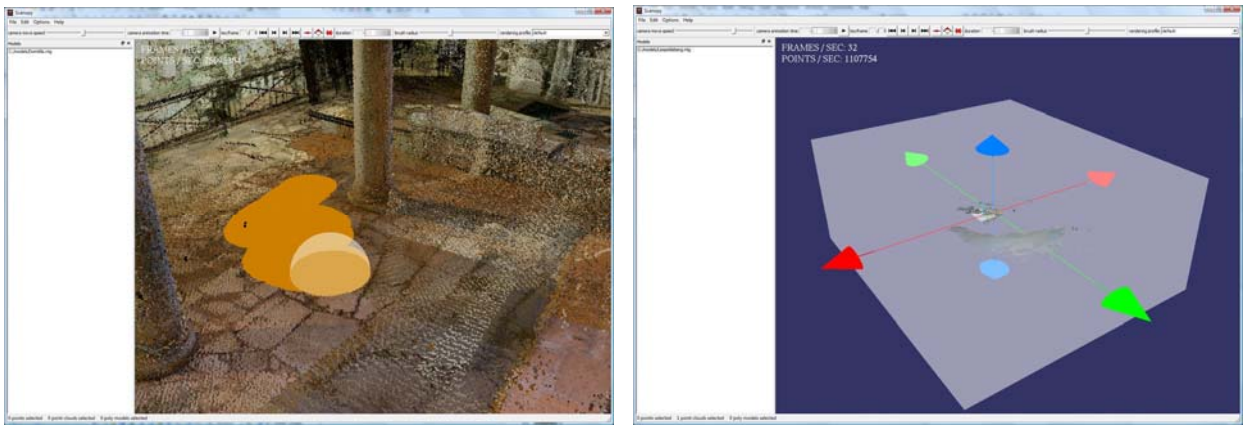

Figure 3 The left image shows the selection sphere and some selected points within a point cloud. The right image shows a point cloud surrounded by the clipping box. The user can drag the arrowheads to change the size of the clipping box.

Editor

Our application also incorporates basic editing operations on point clouds. Points can be selected by the selection-sphere and can then be deleted from the model. If there are new points available, e.g., from an additional scan, they can be merged into an already existing model. It is also possible to delete outliers in a fast and intuitive way, by defining a clipping box around the model (Fig. 3). Only points within the clipping box remain in the model. The user can interactively move the faces of the clipping box and can see immediately the result of the changed clipping box size.

Results

We got the data for the Domitilla model from the FWF START project “Domitilla-Katakombe in Rom”. The data consists of some 1200 single scans, divided into 8 scan campaigns. The scans were taken with a Riegl LMS-Z420I Range Laser Scanner, and color pictures were shot during the scanning process. Colorization of the point clouds was done later manually, which means that every scan had to be colorized with RiScanPro, the software that is accompanied with the laser scanner.

Figure 4 The left image shows the cathedral from the outside with some cubiculae accessible from the cathedral. The middle image shows a cubiculum from the inside. The right image shows the cathedral from the inside.

After colorization we can build a model of the Domitilla catacomb. In the Scanopy application we load the already colorized point clouds from the original scans. The point clouds also have to be already registered within the project coordinate system, because we cannot do this in our program. Then we can remove outliers with the clipping box. This can be done for each colorized point cloud separately, and the point cloud can then be saved without the deleted outliers. After defining the clipping box for each point cloud, which is an optional step, we can then select the point clouds that should be merged into a large model.
The buildup process is taking quite some time, so it is running in a separate thread and does not prevent interaction with the program. After the buildup is complete, the model can be loaded into the viewer and can then also be edited like any other point cloud.
The complete Domitilla model consists of more than 1.2 billion point samples, and takes some 19.5 GB of disk space. Each points stores its position and color as attributes. Some details of the model are shown in (Fig. 4). The buildup process took some 2h 45min on a computer with a RAID 0+1 disk subsystem. Please note that the memory requirements of the model are the same as the memory requirements of the contributing point clouds, i.e., no memory overhead is introduced because no additional points are needed for our LOD rendering.
Deleting small areas in the complete model, e.g. an unintentionally recorded scanner stand, lasts only some seconds. To delete an area of 60 million points takes about 4 minutes.

Conclusion

We have shown that in our system it is possible to execute the whole processing pipeline from reading in the point clouds, doing some editing, building a model, and then doing an interactive walkthrough. Currently most of the time in development is spent on expanding the data structure to allow for arbitrary attributes per point and to make the management of the attributes easily accessible for the user.

Thanks

We thank the FFG project “Scanopy” for sponsoring our work. We thank the FWF START project “Domitilla-Katakombe in Rom” and the ÖAW for access to the Domitilla scans.

References

Wimmer, M and Scheiblauer, C 2006: *Proceedings Symposium on Point-Based Graphics 2006*, pages 129-136.