Consolidated Visualization of Enormous 3D Scan Point Clouds with Scanopy

Claus SCHEIBLAUER[1] / Michael PREGESBAUER[2]

[1] Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria / [2] Government of Lower Austria, Austria

Abstract: Terrestrial laser scanners are used at many excavations for documentation and inventory taking. For the documentation of the excavation Amphitheatre 1 in Bad Deutsch-Altenburg, Austria, the terrestrial laser scanner Riegl LMS Z420i was used. Overall, the entire excavation was covered by about 200 positions. With each scan position the amount of data recorded increases and this leads to performance problems in the visualization of the data. Due to the enormous number of points a consolidated representation of the entire point cloud is not possible with conventional software. The software Scanopy was developed for presenting and editing huge amounts of point data. Furthermore, it is possible to load polygonal models and display them together with point clouds in one scene. This allows an exact documentation of large archaeological excavation sites. Another application is the visualization of polygonal models of different excavation levels. The simplest visualization of point clouds on screen is a representation of the points as square rectangles. This, however, creates geometric inaccuracies, and colored point clouds are presented with sudden color changes. When the points are displayed by using semi-transparent circles the 3D points can be blended and lead to a more homogenous visual representation of the scanned objects. Thus the visual impression of the point cloud will be improved considerably. The developed software is demonstrated on the excavation mentioned above.

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

Introduction

The roman amphitheater in Bad Deutsch Altenburg/Austria is one of the most attracting monuments of the archaeological park Carnuntum. Therefore the provincial government of Lower Austria carried out several 3D laserscanning data acquisitions to document this ancient site. Within a period between 2007 and 2010 about 200 scan positions have been placed to acquire the whole historical building structure as well as to document the ongoing excavation process. To acquire the data a Riegl LMS 420i Laserscanner with a Nikon D 70 Digital Camera mounted on the Scanner has been used. The whole data has been georeferenced in the national coordinate system by using tie points for a coarse registration and identical patches for the multi-station adjustment. An overall accuracy of 1 - 2 cm for each scan position has been accomplished.

For visualizing the point model of the above mentioned excavation we are using the point cloud viewer/editor Scanopy (SCHEIBLAUER and WIMMER 2011). With Scanopy it is possible to view huge point clouds that do not fit completely into the main memory of the computer, and also to edit them. Furthermore, polygonal models with detailed high-quality textures can be rendered in a combined viewer (MAYER, SCHEIBLAUER and MAYER 2011), allowing to show the whole area of an excavation site by a point-based model, and

interesting details, e.g., walls with paintings, by textured polygon-based models. In the following we will describe a rendering method for point-based models, which allows showing details of the scanned objects even when using splats larger than 1 pixel for the points.

Point Cloud Rendering

The visualization of huge point clouds coming from laser scans is a challenging problem due to several issues. First the amount of data produced by the laser scanner has to be handled. The latest generation of laser scanners can generate a billion (10^9) points with one single scan (RIEGL 2012). Only storing the positions of the scanned points will need 12GB of memory (assuming 3 “float” data type values per point for the position, which results in 3*4 bytes per point position), making direct visualization of this data infeasible for current generation computer hardware. Second, if colored points are used, the colors of the points have to be harmonized. The colors are derived from photographs, which were taken at the scan positions during the scan process. The lighting conditions often change when the scanner is moved to a new scan position, be it due to artificial lighting or due to changing position of the sun when scanning outdoors, or other reasons. When points from two different scan positions, which sample the same area, are colored by different photographs, then neighboring points in the combined point cloud will exhibit different colorizations for the scanned area. These color differences can become quite noticeable for a user. And third, the density of the points in a point cloud that is combined from several scan positions will vary locally. This creates unwanted holes in a visualization of the point cloud. To alleviate the problem a varying point size for rendering can be used, so that areas sampled at different densities often appear with closed surfaces regardless.

While we already proposed methods for two of the before mentioned issues, i.e., how to handle the huge amount of data and how to account for the local varying point density during rendering (SCHEIBLAUER and WIMMER 2011), the issue of the noisy colors in point clouds remains. In the following we will first describe how point cloud rendering can be done with simple OpenGL splats, and then we will describe rendering with Gaussian splats, a method which alleviates the color noise issue, and which also allows showing details of the scanned objects when using splats larger than 1 pixel for a point. Note that we use unprocessed point clouds, i.e., no pre-processing is done on the point clouds except for colorization. Therefore the point clouds we use for visualization will exhibit noise in position of the point samples as well. Furthermore we make no assumptions about the uniformity of the sampling, i.e., the point clouds can have widely varying local sampling densities. Executing no pre-processing enables us to use the original point data coming from the laser scanner.

Rendering with Box Splats

We are using an out-of-core approach for rendering a huge point cloud (SCHEIBLAUER and WIMMER 2011). The points are sorted into an octree-based data structure, and for the current viewpoint the most important points are chosen for rendering. The importance is a function that can be defined by the user, for example points projected to the center of the screen shall be preferred. Points which are currently not available for rendering are streamed in from the disk to the memory. The octree-based data structure also

allows for level-of-detail (LOD) rendering, meaning that only few points will be rendered when the user's viewpoint is far away from the point model, and more points will be rendered when it is close to the model.

We also incorporate a point size heuristics to account for differences in the local density of the point cloud (SCHEIBLAUER and WIMMER 2011). The point size heuristics adapts the rendered point size (i.e., the splat size of the projected points on screen) for the points of each octree node, i.e., all points of one octree node have the same point size, but the point size is different for each octree node. The point size heuristics is dependent on the number of points in an octree node and on the depth of an octree node in the hierarchy. The denser the point cloud and the deeper down the hierarchy, the smaller the points will be rendered. This also means that points which have neighbors further away will be rendered larger, possibly closing holes in the point model. Since it is a heuristics it does not work for all areas of a point model.

Fig 1 The image shows a detail of the point model of the Amphitheater 1 in Bad Deutsch-Altenburg. The points are rendered with one color per splat. The single stones of the wall are not clearly visible due to color noise and overlapping splats.

After choosing the points and calculating the point sizes with the point size heuristics (these two steps are done in every rendered frame), the points are then projected to screen and rendered as simple screen-aligned OpenGL splats. These splats are drawn as square rectangles. When using color for the points, this rendering method will lead to color noise (a term which refers to the abrupt changes in color for neighboring splats). The reason for the color noise is that the splats which are used for rendering might be overlapping in screen space. Rendering with box splats and one color per splat results in images like Fig. 1. As can be seen, the whole point model exhibits a lot of color noise. The point size heuristics fills most holes in the point model, only at the staircase in the center of the image holes in the model become visible. Improving the quality of the rendered point cloud would mean to increase the sampling of the geometry.

Since the point samples are given we cannot increase the sampling. Therefore we implemented a rendering method based on signal reconstruction theory, which is described in the next chapter.

Rendering with Gaußian Splats

Instead of rendering each splat only with its color, the rendering technique termed Gaußian splats blends the contribution of neighboring splats overlapping in screen-space. This way a reconstruction of the underlying signal, which consists of the colored point samples, can be performed at the pixels in screen space.

Texture Mapping and Resampling Framework

Gaußian splats are based on the work by ZWICKER, PFISTER, VAN BAAR and GROSS (2001), who introduced the elliptical weighted average (EWA) texture mapping and resampling framework for point-based rendering. With this framework it is possible to apply textures to irregularly sampled point-based models in highest quality. We adapt the framework for application on noisy point clouds with widely varying sampling densities.

The EWA framework is based on the idea that point-based models can be interpreted as a discretized function of the surface geometry. For colored point clouds also the texture of the surface geometry is therefore discretized. The reconstruction of the texture for every part of the surface geometry makes it necessary to first reconstruct the texture signal (i.e., the intensity in each color channel: red, green, blue) as a continuous signal, then warp the continuous signal from object space to screen space, band limit the warped signal, and finally sample the band limited signal for display on screen (ZWICKER, PFISTER, VAN BAAR and GROSS 2001). The third step, band limiting the warped signal, is necessary to avoid aliasing artifacts in screen space when projected splats become smaller than a pixel.

The original point-based EWA framework was slightly simplified to make it amenable for use on today's graphics processing units (GPUs). The complete evaluation of the EWA framework is compute intensive, and so some approximations were derived which allow using the EWA framework also for fast rendering of point clouds (BOTSCH, HORNUNG, ZWICKER and KOBBELT 2005). For unprocessed point clouds, as the ones we are using, we are making further simplifications. As we do not have a normal per point, shading and orientation of the splats are neglected. Also the blending depth, as described later, is set to a constant user-defined value throughout the point cloud for all points. In uniformly sampled point clouds the blending depth is usually set to the average distance of points or to the radius of a single splat, but in noisy point clouds both measures are unreliable estimates, therefore the blending depth can be set by the user.

Splat Rasterization

In our simplified version of the EWA framework the splats of the points are drawn as screen aligned square rectangles, which are then shaped according to radial basis functions centered at the projected point positions in screen space, i.e., they appear as circular disks. The size of the square rectangles is calculated by the point size heuristics. Since we use unprocessed point clouds, we do not calculate an exact splat radius (so that neighboring splats are just slightly overlapping while still creating a closed surface) for each point, but use the point size heuristics instead.

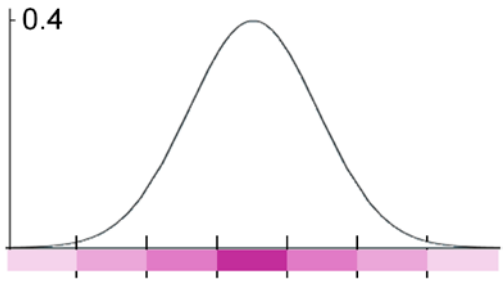

Fig 2 A Gaußian weighting function. The color of a point projected onto the center pixel is weighted very high at the center pixel but weighted lower at the neighboring pixels, according to the Gaußian distribution.

Implementation for use on Graphics Hardware

Rendering a point cloud with the Gaußian splats is done in three steps, which are executed in every rendered frame. First all points that are available in the view frustum are projected to screen space and the so created point splats are used to fill the depth buffer of the viewport. The depth buffer is used as input to the next step as depth image. Note that in the first step no color information is used. In the second step, all points are projected to screen space again, but this time the previously calculated depth image is used to discard points that are invisible. The pixels of the splats produced by these hidden points would be discarded anyway, but the reason why this is important in the second step is, that the information only from really visible points is blended. The blending of the splats is done according to a Gaußian weighting function (Fig 2). There pixels which are close to the projected point position get the highest weight for the point's color. When blending two splats, the weights for the colors from the two splats are used to determine the contributions of the colors for the resulting pixel. In the third step the accumulated color information for each pixel is normalized, since usually the contributions of weights do not sum up to one for each pixel. This would lead to artifacts in the blended colors.

Results

An example for a point cloud rendered with Gaußian splats can be seen in Fig. 3. The details of the stone wall are much more visible compared to the simple rendering in Fig. 1, and the color noise is greatly

Fig 3 The image shows a detail of the point model of the Amphitheater 1 in Bad Deutsch-Altenburg. It is the same area as in Fig. 1. The points are rendered with Gaußian splats. The single stones of the wall are clearly visible due to reduced color noise and blending overlapping splats.

reduced. Some noise is still visible, which has two sources. One source is outliers, which could be removed by manually cleaning the point cloud. The second source is points which are colored by photographs with very diverging lighting compared to their neighboring points, so that blending the splats cannot compensate for the differences in the color.

Conclusion

We have presented a rendering method for huge unprocessed point clouds, which allows showing details of the scanned objects even when using splats that are larger than 1 pixel per point. This is achieved by a blending algorithm, which is based on the point-based EWA framework. We showed the rendering algorithm on a point cloud of the Roman amphitheater in Bad Deutsch Altenburg/Austria, which consists of 105 million points. We hope this rendering method will increase the usability of visualizations of raw unprocessed point scans for everyday users, like archaeologists.

Acknowledgements

This work was funded by the Austrian Research Promotion Agency (FFG) through the FIT-IT project “Terapoints”. We thank the government of Lower Austria for access to the scans of the Roman amphitheater in Bad Deutsch Altenburg/Austria.

References

BOTSCH, M., HORNUNG, A., ZWICKER, M. and KOBBELT, L. (2005) High-Quality Surface Splatting on Today’s GPUs, *Proceedings of Symposium on Point Based Graphics*, 17-24, Stony Brook, NY, USA.

MAYER, I., SCHEIBLAUER, C. and MAYER, J.A. (2011) Virtual Texturing in the Documentation of Cultural Heritage, *Proceedings of the XXIIIrd International CIPA Symposium*, September, Prague.

RIEGL Laser Measurement Systems (2012) http://riegl.com/

SCHEIBLAUER, C. and WIMMER, M. (2011) Out-of-Core Selection and Editing of Huge Point Clouds, *Computers & Graphics*, 35(2), 342-351, April.

ZWICKER, M., PFISTER, H., VAN BAAR, J. and GROSS, M. (2001) Surface Splatting, *Proceedings of SIGGRAPH*, August, Los Angeles.