Flexible Configurable Stream Processing of Point Data

Jonas Boesch & Renato Pajarola
Visualization and MultiMedia Lab, University of Zürich
Binzmühlestrasse 14, 8050 Zürich, Switzerland

ABSTRACT

To efficiently handle the continuously increasing raw point data-set sizes from high-resolution laser-range scanning devices or baseline stereo and multi-view 3D object reconstruction systems, powerful geometry processing solutions are required. We present a flexible and run-time configurable system for efficient out-of-core geometry processing of point cloud data that significantly extends and greatly improves the stream-based point processing framework introduced in [29]. In this system paper we introduce an optimized and run-time extensible implementation, a number of algorithmic improvements as well as new stream-processing functionality. As a consequence of the novel and improved system architecture, implementation and algorithms, a dramatically increased performance can be demonstrated as shown in our experimental results.

Keywords graphics, point-based, geometry processing, streaming

1 INTRODUCTION

Points as rendering and modeling primitives have become a powerful alternative to polygonal object representation [34, 13, 14]. Note that point samples are the natural raw output data primitives of 3D scanning and reconstruction systems. In fact, 3D point samples are the fundamental geometry-defining entities. Satisfying provably correct sampling criteria as discussed in [26], a set of 3D points fully defines the geometry as well as the topology of a surface including boundaries, components and genus. Here we assume that input point data sets reasonably sample the represented surfaces.

With the continually increasing density and extent of raw point cloud data, effective algorithms and systems are required to cope efficiently with the massive amounts of point samples. Basic data and geometry processing operations must be supported such as smoothing, outlier detection, normal estimation, or data decimation with many more being conceivable. These operations can only be performed efficiently on large data if memory trashing [9] is avoided. Therefore, data must be paged efficiently into main memory and processed coherently with respect to randomly accessing memory locations.

In [29] the concept of stream-processing point data was introduced, which we will discuss and extend in Section3. The basic idea was to sequentialize the unorganized raw input point data and then feed the resulting point stream through a pipeline of local stream operators. While the approach in [29] is conceptually well designed and showed promising results, it nevertheless has a number of limitations that we address in this work. First of all, the configuration of the pipeline (chain) of stream operators had to be defined at compile-time, and in fact, all selectable operators had to be known and implemented as well at that time. Furthermore, there was no concept of an operator-chain overarching data structure to maintain any global information about all points currently residing in main memory. Third, the previous implementation left some room for performance improvements, e.g. pooling of dynamic memory resources. Also, the initial organization of point data for stream-processing has been left to an offline preprocess, which has now been integrated seamlessly into the system. Hence in this system paper we present an improved stream-processing framework that addresses all these issues, and eventually also introduces some new stream-processing functionality. The main technical contributions are:

i) A novel flexible C++-classes framework that defines run-time configurable geometry processing stream operators.

ii) New concept of chain-operators overarching a chain of individually configured stream operators.

iii) Improved implementation of neighborhood search operator and dynamic memory handling.

iv) Seamless integration of the previously separate preprocessing stage.

WSCG 2009 conference proceedings, ISBN
WSCG'2009, , 2009
Plzen, Czech Republic.

2 RELATED WORK

Points as 3D surface modeling and rendering primitives have been introduced as early as in [23] and [15]. A number of efficient hardware supported rendering algorithms such as [36, 35, 5, 4, 30] have been proposed and

subsequently further improved. The fundamental theory and algorithms for point-based modeling and rendering are described in [14], and surveys on point-based rendering (PBR) have been presented in [39, 38] and [22]. Apart from rendering which has been well studied and extended to out-of-core [37, 12, 31] or transparent rendering [45, 46], low-level geometry processing techniques for point data have been discussed in [32, 25, 33, 20, 43]. However, these methods are aimed at processing only moderately sized point sets that fit into main memory.

Sequential organization of point data has been addressed specifically for rendering and network transmission in [7, 31] and [37]. More general low-level geometric operations are applied to a stream of points in [29], which we will discuss in the following section.

Streaming has chiefly been used in processing digital audio and video data which in contrast to 3D geometry is inherently sequentially organized, i.e. in time. The sweep-line concept in geometry processing [8] is conceptually closer than multimedia streaming, since our basic stream-processing follows a similar idea of sweeping a plane over the point cloud data. In the context of 3D geometry, streaming has been introduced for simplification and compression operations on polygonal meshes [17, 44, 18, 42], which generally grow and process mesh regions sequentially in an order that limits its main memory usage. Specifically for rendering, a streaming mesh layout has been proposed in [16]. These streaming approaches on meshes, however, do not support low-level geometry processing operations, and more importantly, do not directly apply to raw point data processing as mesh connectivity is required.

Recently, different streaming frameworks for surface reconstruction from points have been presented. In [3] a Poisson-based multi-resolution streaming framework based on an sparse octree is used in which multiple streams are processed concurrently. In [19] and [2] slice-based streaming algorithms are proposed for Delaunay triangulations. There, the space is partitioned into explicit regions as opposed to the implicit region partitioning used in this paper by using a sweep plane.

Finally, in graphics the concept of streaming images and geometry data has been used in the context of remote rendering where 3D data is to be displayed on a remote display (e.g. [10], [27] or [6]). Again, low-level data processing is not the focus in these approaches but the network transmission of data to a remote device.

3 STREAM-PROCESSING

3.1 Sequential Processing

The basic idea behind stream-processing point data as introduced in [29] is to order and process the data sequentially in such a way that: (1) points can be read from an input-stream into main memory one at a time, (2) the so called *active* points in main memory can efficiently be processed independently[1] from others given only some local spatial information, and (3) points are written to an output-stream as soon as they have been fully processed. Since all data processing is limited to the points in the active working set $\mathscr{A}$, at any time only a very limited fraction of data is kept in main memory, which together with the sequential processing supports efficient out-of-core operation on huge point data sets.

Since raw point data sets rarely come in a spatially ordered sequence, a sorting process is required to linearly order them. Given an ordering measure along one direction in space, such sorting can be achieved efficiently for very large data by external sort techniques [24, 21, 41]. Our solution is presented in Section 3.3.

3.2 Stream Operators

The operations supported in the above described stream-processing framework are defined in [29] as local operators $\Phi(\mathbf{p}_i)$ that perform a computation on a point $\mathbf{p}_i$ and its attributes only taking the point $\mathbf{p}_i$ itself and a limited set of neighbors $\mathbf{p}_j$ into account. The neighborhood $\mathscr{N}_i$ is typically defined as a k-nearest neighbors or points $\mathbf{p}_j$ within a given range r. The attributes A_i associated with a point $\mathbf{p}_i$ can include a wide range of parameters such as color, normal orientation or curvature. From this definition it is clear that a local operator $\Phi(\mathbf{p}_i)$ can be applied to any point if $\mathbf{p}_i$ itself as well as all neighbors $\mathscr{N}_i$ are part of the current working set $\mathscr{A}$. This includes a large group of important geometry processing operators ranging from surface parameter estimation to filtering operations.

Furthermore, the stream-processing framework is designed to chain together a series of stream operators $\Phi_1,\ldots,\Phi_p$ that are applied in succession to a stream of points as illustrated in Figure 1. Each stream operator Φ_k itself acts as a FIFO queue, passing the points from one to the next operator. The so defined concept then postulates that a stream operator $\Phi_k(\mathbf{p}_i)$ can be executed on $\mathbf{p}_i$ as soon as no preceding operator $\Phi_{l<k}$ modifies any neighbor points $\mathbf{p}_j \in \mathscr{N}_i$ anymore, or still depends on $\mathbf{p}_i$ for its completion. Moreover, each stream operator Φ_k only passes a point $\mathbf{p}_i$ to the next operator Φ_{k+1} if the point and its attributes have fully been processed. More details are given in [29].

The fundamental stream operators introduced in [29] include the basic I/O operators for reading (Φ_R) and writing (Φ_W) points from and to the input and output streams respectively, as well as a neighborhood operator (Φ_X) that generates the nearest neighbor sets $\mathscr{N}_i$ for any incoming point $\mathbf{p}_i$. Additional geometry processing operators that have been presented include surface normal estimation (Φ_N), curvature estimation (Φ_C), el-

[1] or more exactly the dependency is strictly limited to a well defined local spatial neighborhood relation

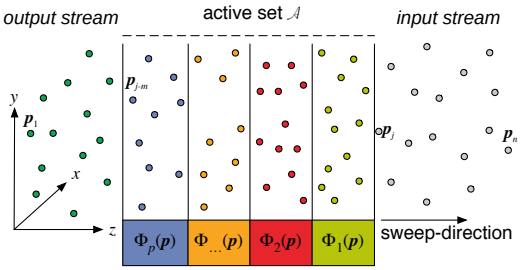

Figure 1: Chain of streamable operators acting on the points passing through the active set.

liptical point splat extent determination (Φ_E) as well as feature preserving surface smoothing (Φ_S).

3.3 Sorting

As for sweep-plane algorithms in computational geometry [8], a stream-processing framework requires the points to be ordered along a spatial direction. In principle, any direction could be used as for example any of the three principal axis of the point data's modeling coordinate system. However, it is typically advantageous to align the data such that the sweep-plane intersection with the object exhibits a smaller outline. Hence for objects with a biased spatial extend, the sweep direction should be aligned accordingly.

Sorting in the direction of the longest axis of a data-aligned tight bounding box can efficiently be achieved in two phases as follows. In the first linear pass over the data points $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^3$, a generic homogeneous covariance $\widehat{\mathbf{M}} = \sum_{i=1}^{n} \hat{\mathbf{p}}_i \cdot \hat{\mathbf{p}}_i^{\mathsf{T}}$ and center of mass of the points $\mathbf{c} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{p}_i$ are accumulated, with $\hat{\mathbf{p}}$ denoting the homogeneous coordinate extension of $\mathbf{p}$. As shown in [28, 30] this allows us to express and post-compute the actual covariance matrix $\mathbf{M} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{p}_i - \mathbf{c}) \cdot (\mathbf{p}_i - \mathbf{c})^{\mathsf{T}}$ elegantly and efficiently in homogeneous space by $\mathbf{M} = \frac{1}{n}\mathbf{T}(-\mathbf{c}) \cdot \widehat{\mathbf{M}} \cdot \mathbf{T}^{\mathsf{T}}(-\mathbf{c})$ with $\mathbf{T}(-\mathbf{c})$ being the translation matrix moving the center of mass $\mathbf{c}$ to the origin. The sorting axis is now given by the eigenvector $\mathbf{v}$ corresponding to the largest eigenvalue of $\mathbf{M}$. In the second phase, the points are transformed and sorted along their projection onto $\mathbf{v}$.

Instead of applying the above transformation and sorting offline as in [29], we have integrated it into the stream-processing framework as an online preprocess. For this purpose we have implemented an efficient out-of-core sorting algorithm based on the radix-sort technique. Besides supporting out-of-core sorting on very large data, a major goal of the sorting preprocess was to provide an incrementally growing sorted stream of data. With radix-sort we can choose to inspect the next bit of one partition first, in a depth-first way, before continuing work on the other partition. Hence sorting can complete and progress from one end of the data range to the other, which is why it was chosen over other out-of-core sorting algorithms. For performance reasons, insertion-sort is used when the partially-sorted partitions fall below a certain size.

Taking advantage of this progressive online sorting approach, the stream-processing pipeline can be fed with the early available sorted data partitions. Thus point processing operations, e.g. such as the costly neighborhood search, can be overlapped with the sorting phase and can start with only minimal latency before the whole data has been preprocessed.

Furthermore, in order to utilize the common availability of multiple cores, the sorting preprocess was not only integrated into the main stream processor, but also adapted to use threadpool-based efficient multi-threading. This improves performance as well as simplifies usage of the stream processor.

4 SYSTEM ARCHITECTURE

4.1 Run-time Configurability

In the original stream-processing approach [29], the operator chain was set up at compile-time. A stream operator defined a set of per-point attribute parameters that it depends on or modifies, some auxiliary data fields used while executing the operator and some attributes that it adds permanently to a point element, which are added to the output stream at the end of the stream operator chain. Every stream operator defines a struct that contains members variables for all required data, and the temporary as wellas the final stream-point structures are defined by multiple inheritance. While the main advantage of this approach is its simplicity, it also lacks in flexibility and consistency. Separate executables for all possible combinatorial configurations of the different stream operators are necessary, which grows exponentially by 2^p with the number p of operators. For an increasing and extensible library of stream operators this is clearly a limiting constraint. Furthermore, there is no automatic mechanism to verify consistency of attribute fields, e.g. such as ensuring that an attribute xy which operator X depends on is provided by another operator Y. A similar problem arises with the order of operators: While including the normal attribute field allows the use of a curvature operator at compile time, it does not make sure that the normal estimation operator is actually applied first in the chain of stream operators.

Therefore, the stream-processing framework was redesigned to allow setting up an arbitrary operator chain at run-time, by using either an external configuration file or by specifying stream operators as command line arguments. Also, since attribute fields of different operators are now specified dynamically at run-time, a registration mechanism can verify that no required attributes are missing, and that the operators are specified in a compatible order.

In the new framework, dependencies between stream operators are defined solely by dependencies on certain

data elements. This has the advantage that every operator can be replaced as long as the replacement operator can generate the same output data fields. Additionally, since there are no direct code-level dependencies between operators, new operators can be integrated by loading them as separate plugins or dynamic shared libraries at run-time.

Run-time Structures In the dynamic run-time configurable version of the stream-processing framework, the size of the structure holding the attribute fields is unknown at compile-time. Therefore, a new *rt_structs* (for run-time structures) concept for the streaming three-dimensional point data structure was designed that enables the use of efficient, type-safe and run-time configurable member variables. Each stream operator stores the accessor objects it needs to extract the required member variables.

Accessors are template classes which are configured with the required data type. This means that the actual type of all dynamic member is fields clearly defined in the header of the operator, and standard C++ type checking can be used when using the run-time structures.

Setup Stage In the first part of the setup stage, each stream-processing operator registers the name and data type of the member fields it requires as input and those it generates as output. This allows simple and efficient consistency checking of the operator chain. Additionally, some configuration settings such as minimum number of neighbors can be negotiated between operators. After that, the size of the point structure is computed and the run-time structure accessors are configured.

Processing Stage Using the templatized run-time structure accessors, each variable can be accessed efficiently. Trying to access a field using a wrong type is detected at compile-time, and the additional cost of accessing a variable as compared to standard C++ classes is only an (inlined) function call and a reinterpret_cast<>().

4.2 Memory Mangement

Pools of objects are used where possible to optimize performance by preventing continuous construction and destruction of objects. Memory pools are used for rt_struct objects, for the *k*d-tree nodes in the *k*d-heap-neighbor operator and for all node types in the new chain operator.

5 OPERATORS

The basic semantic of stream operators has been retained from [29] in the new proposed system architecture. In addition to the standard read and deferred-write I/O and the various geometry operators, we have implemented one new local stream operator $\Phi_O(\mathbf{p}_i)$ for outlier detection and removal.

Moreover, the new stream-processing system has an additional novel chain-operator type Ψ which in its scope overarches the entire chain of individual local stream operators Φ. A stream operator $\Phi_k(\mathbf{p}_i)$ is defined as a local operation on the geometry of point $\mathbf{p}_i$ and its attributes, and is but one element in a chain of stream operators $\Phi_1,\ldots,\Phi_p$ not knowing about the other selected operators. Furthermore, a stream operator only has direct access to the points within its own FIFO queue of points, which is only a subset of all points in the active set $\mathscr{A}$. On the other hand, the chain-operator $\Psi(\mathscr{A})$, which is discussed in more detail in Section 5.4, in its scope spans the entire set of active points $\mathscr{A}$ and has knowledge of all elements in the chain of stream operators $\Phi_1,\ldots,\Phi_p$ as illustrated in Figure 2.

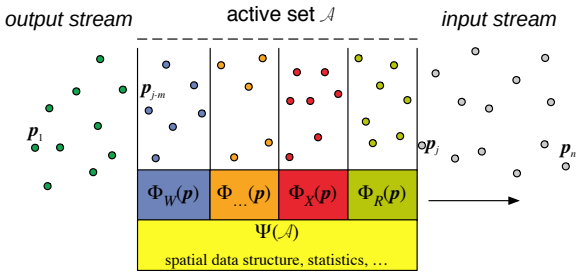

Figure 2: Conceptual diagram of the chain operator Ψ overarching a chain of individual stream operators Φ_k.

5.1 I/O Operators

The read operator Φ_R acts on the input stream of point data. During the setup phase, it reads and parses the data header and maps the point data input file to the input stream. Typically, this is done via memory mapping of the input file and sequential traversal through the input data. During the point processing phase, Φ_R reads the input point data and (optionally) converts it to the proper format. By definition, the read operator Φ_R must be the first in a chain of operators. It uses a pool of rt_struct point objects as mentioned in Section 4.2 to efficiently create the new objects.

In the setup phase, the write operator Φ_W creates and memory maps the output file. During processing, the write operator uses the deferred writing strategy described in [29] to write points out to disk and remove them from main memory as soon as this can be done safely. Φ_W shares a pool of rt_struct point objects with the read operator, as indicated above, to avoid unnecessary memory allocation and deallocation overhead.

5.2 Neighborhood Operator

In the extended stream-processing framework we introduce a new neighborhood operator Φ_X that takes advantage of the spatial data structure provided by the new chain operator Ψ_X, see also below. Upon insertion of a new point $\mathbf{p}_j$ into the active-set $\mathscr{A}$, it will

also be inserted into a spatial data structure maintained by Ψ_X. Moreover, the stream operator $\Phi_X(\mathbf{p}_j)$ then queries $\Psi_X(\mathscr{A})$ immediately for an initial *left-sided* nearest neighbors set $\mathscr{N}_j$ which at that point contains the closest points $\mathbf{p}_i$ with index $i < j$ in the input stream. At the same time, the new point $\mathbf{p}_j$ is tested with existing neighborhood sets $\mathscr{N}_i$ and included if appropriate to update their missing *right-sided* closest neighbors.

The spatial data structure in the chain operator Ψ_X also allows for within-range neighborhood queries. As soon as the next new point $\mathbf{p}_j$ is farther away from an active point $\mathbf{p}_i \in \mathscr{A}$ than the query range r along the streaming dimension, a range query for $\mathbf{p}_i$ can be invoked on $\mathscr{A}$ to find all neighbors within distance r.

While the new nearest neighbor operator is now the default to establish closest points neighborhoods, a *k*d-heap approximate *k*-nearest-neighbor operator as in [29] is still available, but deprecated for performance reasons. See also experimental results in Section 7.

5.3 Geometry Operators

The normal estimation Φ_N, curvature estimation Φ_C, elliptical point splat-extent estimation Φ_E and smoothing Φ_S operators have been ported from [29] to the new dynamic rt_struct member attributes architecture described in Section 4.

A new local outlier detection stream operator $\Phi_O(\mathbf{p}_i)$ has been added which quantifies the likeliness of any nearest neighbor $\mathbf{p}_j \in \mathscr{N}_i$ to be an outlier. The outlier quantifier is defined as

$$O_i(\mathbf{p}_j) = h_i(|\mathbf{p}_i - \mathbf{p}_j|) \cdot |\mathbf{p}_j - \Pi_i(\mathbf{p}_j)|, \qquad (1)$$

with $h_i(x)$ being a smooth weighting function, e.g. a Gaussian, and $\Pi_i(\mathbf{p}_j)$ the projection of $\mathbf{p}_j$ onto the plane with normal $\mathbf{n}_i$ through point $\mathbf{p}_i$. The outlier operator $\Phi_O(\mathbf{p}_i)$ compares the values $O_i(\mathbf{p}_j)$ for all $\mathbf{p}_j \in \mathscr{N}_i$ to a threshold value ε and if greater discards $\mathbf{p}_j$ as a nearest neighbor of $\mathbf{p}_i$.

Therefore, the outlier operator $\Phi_O(\mathbf{p}_i)$ measures the offset of $\mathbf{p}_j$ from the tangent plane at $\mathbf{p}_i$ and quantifies a degree of co-planarity. However, it penalizes points $\mathbf{p}_j$ farther away from $\mathbf{p}_i$ less, so they are allowed to deviate more from the tangent plane. Hence with the support radius of the weighting function $h_i(x)$ being adjusted relative to an estimated local curvature at point $\mathbf{p}_i$, the outlier detection can be made adaptive to the local surface feature size.

5.4 Chain Operators

The main new chain operator $\Psi_X(\mathscr{A})$ sorts all the points in the active set $\mathscr{A}$ into a spatial tree structure. By default, a bucketed PR KD-Tree [40] is used, but the operator can be templatized with other spatial index structures. Any regular stream operator can interact with this tree operator by providing a visitor object [11]. Currently, the tree operator is used by the neighborhood operator Φ_X for efficient *k*-nearest-neighbor searching or range queries. Additionally, the smoothing operator Φ_S uses the new chain operator Ψ_X to update the spatial data structure for point locations modified by the smoothing operation.

The statistics operator $\Psi_S(\mathscr{A})$ collects data about current and maximum data size, extent and memory usage of the active set. Having all statistical functionality in an operator makes data collection optional, and statistics can easily be disabled for performance reasons. This also allows the implementation of different operators to customize statistics collection (complete debug and optimization statistics vs. minimal release-mode statistics).

6 RUN-TIME PROCESS

Setup To demonstrate the run-time procedure of the stream-processing application, a simple processing job will be explained in detail. A raw point data set is to be processed to compute the normal of each point. The resulting pipeline consists of the following stream operators: read, neighborhood, normal and write. The input data set contains only point positions.

During the setup phase, the read operator will be initialized first, reads and parses the input point header and registers a dynamic member field called position. Next, the neighboorhood operator is initialized. This operator requires the chain tree operator and so requests the system to instantiate the chain operator. Then, it will reserve the neighbor-list and neighbor-count dynamic members, and register position as input-dependency. Additionally, it will check for a user-specified neighborhood-size option, or set a default. Next the normal operator is initialized, reserves the normal-vector dynamic member field and adds position, neighbor-list and neighbor-count to the input-dependencies. Finally, the write operator is instantiated. Optionally, a statistics chain operator may be set up. After all operators are set up, the system will run a dependency check that makes sure all requirements for each operator are met. The pipeline specified above is valid and therefore passes that check.

The size of the run-time structure used in this pipeline is computed, the offsets to the dynamic members in all accessor objects of each stream operator are set and an initial pool of memory for points is allocated. The read operator memory-maps the input data file, and the write operator creates and memory-maps a file large enough to contain the final point data, including all the newly created member fields that were marked as persistent output data fields. This phase concludes the setup stage and stream processing can now begin.

Processing The read operator reads the point position from the input data file into factory-allocated new run-time structure instances. The points are inserted into all chain operators, including the tree operator containing a spatial data structure. The neighborhood oper-

ator then receives the points and performs a k-nearest neighbor query by sending a visitor object to the tree operator's data structure. See Section 5.2 for how the k-nearest neighbor search is performed. The normal operator then receives the point from the output buffer of the previous operator and computes an estimated normal based on the neighboring points. Finally, the deferred write operator buffers the processed point until it is not referenced anymore in the previous stream operators. The point-data is written to the output stream and the run-time structure instance is returned to the pool. This process continues until all the input points have been processed.

Notes The new stream-processing system depends on two libraries: boost [1] and an in-house library with some vector-matrix geometry functions. The stream-processing system runs on most UNIX-like operating systems including Mac OS X, GNU/Linux and FreeBSD.

7 EXPERIMENTAL RESULTS

All reported experimental results were achieved on a Apple Xserve with dual Intel Xeon 2.0GHz processors. The models that have been tested are listed in Table 1.

model name	number of sample points
david2mm	4'129'534
lucy	14'022'961
scene	21'749'996
david1mm	28'168'109
st. matthew	102'965'801

Table 1: The point-cloud test models.

In Figure 3 we compare the performance of the new stream-processing system architecture to the original approach introduced in [29]. As we can see, the new architecture is not only much more flexible with its run-time configurability of the stream-operator chain, but it is also significantly more efficient, especially for large point-cloud data sets. The performance differences are mainly due to two factors: the new improved nearest neighborhood operator, and the memory pooling used in the read- and write-operators.

With the new neighborhood operator and other improvements, the average time a point stays active in main memory has also been reduced significantly as shown in Figure 4.

Many of todays point-cloud models are too big to process efficiently all in main memory. One advantage of stream-processing is that only a minimal required subset of the model data has to be in-core. In Figures 6 and 5 the relative sizes of the active set with respect to the total size of the point data set are displayed. The (office) scene model is a worst-case model since it has a well defined direction of longest extent, however, at the same time it exhibits large co-planar point regions exactly perpendicular to the major object extent. Therefore, the basic alignment and sorting leads to

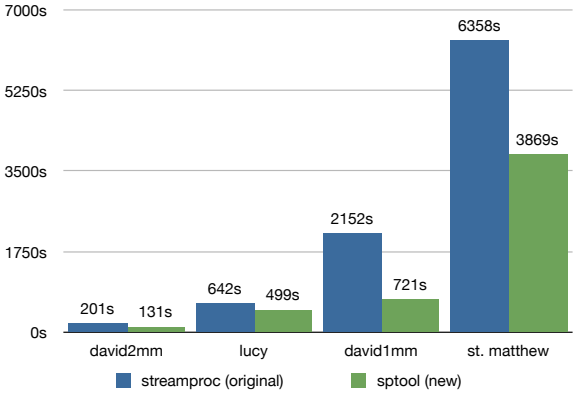

Figure 3: Performance comparison between the original and the new system using a read-neighborhood-normal-curvature-splatsize-write operator chain.

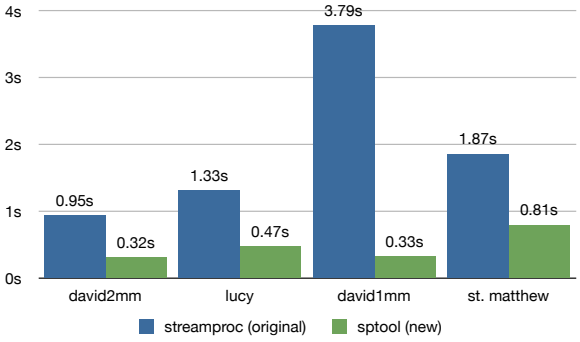

Figure 4: Comparison of average time each point spends in main memory.

a stream-processing where at isolated locations a large number of points get passed by the sweep-plane, leading to few individual spikes in memory consumption. Figure 6 shows the typical in-core effectiveness as well as the above indicated worst-case. Figure 7 displays the spikes in active set size for the worst-case scene model.

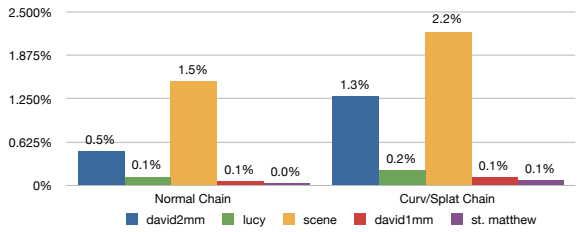

Figure 5: Comparison of average active-set sizes in relation to total point-set-size.

Applying a chain of operators consisting of read (Φ_R), nearest-neighbor search (Φ_X), normal estimation (Φ_N), curvature estimation (Φ_C), splat-extent estimation (Φ_E) and deferred-write (Φ_W) the out-of-core effectiveness of the stream-processing system has remained excellent as in [29]. As shown in Figure 7, the goal of dramatically reducing the number of data elements actively maintained in main memory has well been achieved,

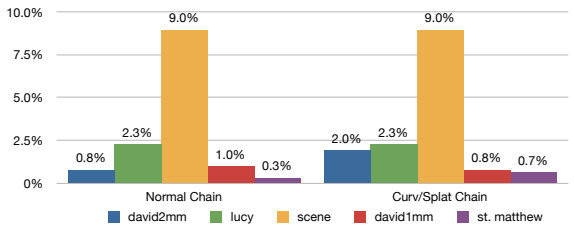

Figure 6: Comparison of maximum active-set sizes in relation to total point-set-size.

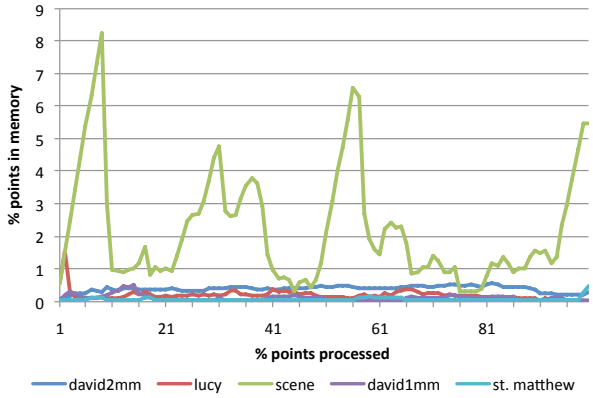

Figure 7: Percentage of active set of points maintained in main memory during stream-processing.

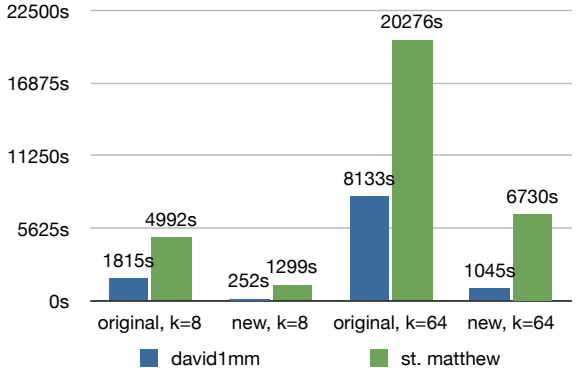

Figure 8: Performance comparison of neighbor detection in large models using 8- and 64-neighborhoods.

as rarely ever more than 1% of data is kept active in main memory. To more efficiently handle worst-case models such as the office scene outlier, the sorting preprocess should in the future take into account specifically expressed dimensions of co-planar data and simply perturb the sweep-plane direction slightly to avoid this worst-case scenario.

8 CONCLUSION

In this paper we have presented a novel stream-processing system architecture, extending and implementing the conceptual framework introduced in [29] more efficiently. The new architecture allows for efficient and flexible run-time configurability of geometry processing operators that can be applied to an ordered stream of point cloud data. This novel definition and implementation of local stream-processing operators allows operators to be dynamically defined and configured at runtime, and not statically at compile-time as previously required. Through our novel stream-operator implementation, the main stream-processing application program can be compiled without specification of which geometric operators and in what order they will eventually be applied to the point data. In fact, at runtime, the available local geometry processing operators can dynamically be selected and configured on-demand. Moreover, the stream-processing application can automatically check for consistency of the selected chain of stream operators.

In addition to a few new and modified stream-operators, we have introduced the new concept of a chain operator. In the context of stream-processing points by passing them from one geometry processing operator to the next in a chain of multiple successive stream operators, a chain operator acts as a global operator overarching the chain of individual stream operators and thus introduces a new stream-processing functionality.

With respect to a seamless integration of the necessary preprocessing task, we have included the stream-sorting operation into the stream-processing framework. A parallel radix-sort based algorithm provides a streaming result of sorted point data which can be operated on by the main stream-operators with minimal latency. Finally, the new system architecture has demonstrated significant performance improvements.

ACKNOWLEDGEMENTS

We would like to thank and acknowledge the Stanford 3D Scanning Repository and Digital Michelangelo projects as well as Cyberware Inc. for providing the 3D geometric test data sets. This work was partially supported by the Swiss National Science Foundation Grant 200021-111746/1.

REFERENCES

[1] Boost c++ libraries. Website, 2007.

[2] Remi Allegre, Raphaelle Chaine, and Samir Akkouche. A streaming algorithm for surface reconstruction. In *Proceedings Eurographics Symposium on Geometry Processing*, pages 79–88, 2007.

[3] Matthew Bolitho, Michael Kazhdan, Randal Burns, and Hugues Hoppe. Multilevel streaming for out-of-core surface reconstruction. In *Proceedings Eurographics Symposium on Geometry Processing*, pages 69–78, 2007.

[4] Mario Botsch and Leif Kobbelt. High-quality point-based rendering on modern GPUs. In *Proceedings Pacific Graphics 2003*, pages 335–343. IEEE, Computer Society Press, 2003.

[5] Mario Botsch, Andreas Wiratanaya, and Leif Kobbelt. Efficient high quality rendering of point sampled geometry. In *Proceedings Eurographics Workshop on Rendering*, pages 53–64, 2002.

[6] Liang Cheng, Anusheel Bhushan, Renato Pajarola, and Magda El Zarki. Real-time 3D graphics streaming using MPEG-4. In *Proceedings IEEE/ACM Workshop on Broadband Wireless Services and Applications*, 2004.

[7] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential point trees. *ACM Transactions on Graphics*, 22(3):657–662, 2003.

[8] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[9] Peter J. Denning. Virtual memory. *ACM Computing Surveys*, 2(3):153–189, 1970.

[10] Klaus Engel, Ove Sommer, and Thomas Ertl. A framework for interactive hardware-accelerated remote 3D-visualization. In *Proceedings EUROGRAPHICS - IEEE TCVG Symposium on Visualization*, pages 167–177, 2000.

[11] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1994.

[12] Enrico Gobbetti and Fabio Marton. Layered point clouds: A simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics*, 28(1):815–826, February 2004.

[13] Markus H. Gross. Getting to the point...? *IEEE Computer Graphics and Applications*, 26(5):96–99, September-October 2006.

[14] Markus H. Gross and Hanspeter Pfister, editors. *Point-Based Graphics*. Morgan Kaufmann Publishers - Elsevier, 2007.

[15] J.P. Grossman and William J. Dally. Point sample rendering. In *Proceedings Eurographics Rendering Workshop 98*, pages 181–192. Eurographics, 1998.

[16] Martin Isenburg and Peter Lindstrom. Streaming meshes. In *Proceedings IEEE Visualization*, pages 231–238, 2005.

[17] Martin Isenburg, Peter Lindstrom, Stephan Gumhold, and Jack Snoeyink. Large mesh simplification using processing sequences. In *Proceedings IEEE Visualization*, pages 465–472. Computer Society Press, 2003.

[18] Martin Isenburg, Peter Lindstrom, and Jack Snoeyink. Streaming compression of triangle meshes. In *Proceedings Eurographics Symposium on Geometry Processing*, pages 111–118, 2005.

[19] Martin Isenburg, Yuanxin Liu, Jonathan Shewchuk, and Jack Snoeyink. Streaming computation of delaunay triangulations. *ACM Transactions on Graphics*, 25(3):1049–1056, July 2006.

[20] Thouis R. Jones, Frédo Durand, and Matthias Zwicker. Normal improvement for point rendering. *IEEE Computer Graphics and Applications*, 24(4):53–56, July-August 2004.

[21] Donald E. Knuth. *The Art of Computer Programming, 3rd Edition*. Addison-Wesley, 1998.

[22] Leif Kobbelt and Mario Botsch. A survey of point-based techniques in computer graphics. *Computers & Graphics*, 28(6):801–814, 2004.

[23] Marc Levoy and Turner Whitted. The use of points as display primitives. Technical Report TR 85-022, Department of Computer Science, University of North Carolina at Chapel Hill, 1985.

[24] John P. Linderman. rsort and fixcut. man pages, 1996. revised June 2000.

[25] G. H. Liu, Y. S. Wong, Y. F. Zhang, and H. T. Loh. Adaptive fairing of digitized point data with discrete curvature. *Computer Aided Design*, 32(4):309–320, 2002.

[26] Gopi Meenakshisundaram. *Theory and Practice of Sampling and Reconstruction for Manifolds with Boundaries*. PhD thesis, Department of Computer Science, University of North Carolina Chapel Hill, 2001.

[27] Yuval Noimark and Daniel Cohen-Or. Streaming scenes to MPEG-4 video-enabled devices. *IEEE Computer Graphics and Applications*, 23(1):58–64, January/February 2003.

[28] Renato Pajarola. Efficient level-of-details for point based rendering. In *Proceedings IASTED Invernational Conference on Computer Graphics and Imaging (CGIM)*, 2003.

[29] Renato Pajarola. Stream-processing points. In *Proceedings IEEE Visualization*, pages 239–246. Computer Society Press, 2005.

[30] Renato Pajarola, Miguel Sainz, and Patrick Guidotti. Confetti: Object-space point blending and splatting. *IEEE Transactions on Visualization and Computer Graphics*, 10(5):598–608, September-October 2004.

[31] Renato Pajarola, Miguel Sainz, and Roberto Lario. XSplat: External memory multiresolution point visualization. In *Proceedings IASTED Invernational Conference on Visualization, Imaging and Image Processing (VIIP)*, pages 628–633, 2005.

[32] Mark Pauly and Markus Gross. Spectral processing of point-sampled geometry. In *Proceedings ACM SIGGRAPH*, pages 379–386. ACM Press, 2001.

[33] Mark Pauly, Richard Keiser, Leif Kobbelt, and Markus Gross. Shape modeling with point-sampled geometry. *ACM Transactions on Graphics*, 22(3):641–650, 2003.

[34] Hanspeter Pfister and Markus Gross. Point-based computer graphics. *IEEE Computer Graphics and Applications*, 24(4):22–23, July-August 2004.

[35] Liu Ren, Hanspeter Pfister, and Matthias Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Proceedings EUROGRAPHICS*, pages 461–470, 2002. also in Computer Graphics Forum 21(3).

[36] Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proceedings ACM SIGGRAPH*, pages 343–352. ACM SIGGRAPH, 2000.

[37] Szymon Rusinkiewicz and Marc Levoy. Streaming QSplat: A viewer for networked visualization of large, dense models. In *Proceedings Symposium on Interactive 3D Graphics*, pages 63–68. ACM SIGGRAPH, 2001.

[38] Miguel Sainz and Renato Pajarola. Point-based rendering techniques. *Computers & Graphics*, 28(6):869–879, 2004.

[39] Miguel Sainz, Renato Pajarola, and Roberto Lario. Points reloaded: Point-based rendering revisited. In *Proceedings Symposium on Point-Based Graphics*, pages 121–128. Eurographics/IEEE VGTC, 2004.

[40] Hanan Samet. *Foundations of Multidimensiona and Metric Data Structures*. Morgan Kaufmann Publishers - Elsevier, 2006.

[41] Jeffrey S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, 2001.

[42] Huy T. Vo, Steven P. Callahan, Peter Lindstrom, Valerio Pascucci, and Claudio T. Silva. Streaming simplification of tetrahedral meshes. *IEEE Transaction on Visualization and Computer Graphics*, 13(1):145–155, January-February 2007.

[43] T. Weyrich, M. Pauly, R. Keiser, S. Heinzle, S. Scandella, and M. Gross. Post-processing of scanned 3D surface data. In *Proceedings Symposium on Point-Based Graphics*, pages 85–94. Eurographics/IEEE VGTC, 2004.

[44] Jianhua Wu and Leif Kobbelt. A stream algorithm for the decimation of massive meshes. In *Proceedings Graphics Interface*, pages 185–192, 2003.

[45] Yanci Zhang and Renato Pajarola. Single-pass point rendering and transparent shading. In *Proceedings Symposium on Point-Based Graphics*, pages 37–48. Eurographics/IEEE VGTC, 2006.

[46] Yanci Zhang and Renato Pajarola. Deferred blending: Image composition for single-pass point rendering. *Computers & Graphics*, 31(2):175—189, 2007.