Eurographics Symposium on Point-Based Graphics (2005)
M. Pauly, M. Zwicker, (Editors)

Progressive Splatting

Jianhua Wu Zhuo Zhang Leif Kobbelt

Computer Graphics Group, RWTH Aachen University, Germany

Abstract

Surface splatting enables high quality and efficient rendering algorithms for dense point-sampled datasets. However, with increasing data complexity, the need for multiresolution models becomes evident. For triangle meshes, progressive or continuous level of detail hierarchies have proven to be very effective when it comes to (locally) adapt the resolution level of the 3D model to the application-dependent quality requirements. In this paper we transfer this concept to splat-based geometry representations. Our progressive splat decimation procedure uses the standard greedy approach but unlike previous work, it uses the full splat geometry in the decimation criteria and error estimates, not just the splat centers. With two improved error metrics, this new greedy framework offers better approximation quality than other progressive *splat decimators. It comes even close to the recently proposed globally optimized* single-resolution *sub-sampling techniques while being faster by a factor of 3.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Curve, surface, solid and object representations

1. Introduction

Point-based geometry receives more attention in recent years and much work has been dedicated to efficient acquisition, modeling, processing and rendering of point primitives [GPZ*04, KB04]. The main advantage of discrete point sets over "standard" polygonal meshes is their simplicity and flexibility, i.e. they do not need to preserve any connectivity information.

In order to visually fill the gap between point samples, point-based models are usually generalized to splat representations that sample and approximate the surface geometry, and almost all available high quality and efficient point-based rendering systems today are adopting splatting techniques either in object space or in image space [RL00, ZPBG01, RPZ02, BK03, ZRB*04, BSK04]. From a geometric point of view, splats (oriented 3D ellipses) are no more than linear surface elements with finite spatial extent overlapping in C^{-1} fashion, opposed to the well established C^0piecewise linear polygonal meshes. Most important, the conceptual flexibility of point-based representations that we can exploit is largely preserved for surface splats, i.e. better optimization techniques can be applied to splat models more easily than polygonal meshes, thus leading to higher quality algorithms (eg. [WK04]).

Thanks to the evolution of modern 3D photography and 3D scanning systems, the data size of point models are ever increasing drastically nowadays [LPC*00]. Despite the fast rendering speed of point models on up-to-date GPUs [DVS03, BK03], their sheer data size has increased so fast that they have brought great challenges for subsequent processing tasks like modeling and rendering. And this would be even more critical for low-end hardware environments like mobile phones or PDAs. In this sense, *progressive* splat representations that can freely adjust the data complexity are of the same importance as the well-studied progressive representation for standard polygonal meshes [GGK02], as progressive or continuous level of detail hierarchies have proven to be very effective when it comes to locally adapt the resolution level of the 3D model to the application-dependent quality requirements.

As most previous work to generate progressive splat-based geometry has utilized the straightforward hierarchical space partitioning method or only focused on infinitesimal isolated points (centers of splats), they usually produce poor outputs at coarser approximation levels which could degrade the visual appearance of level of details rendering in the very beginning. In this paper we transfer the concept of the well-known progressive meshes [Hop96] from triangle

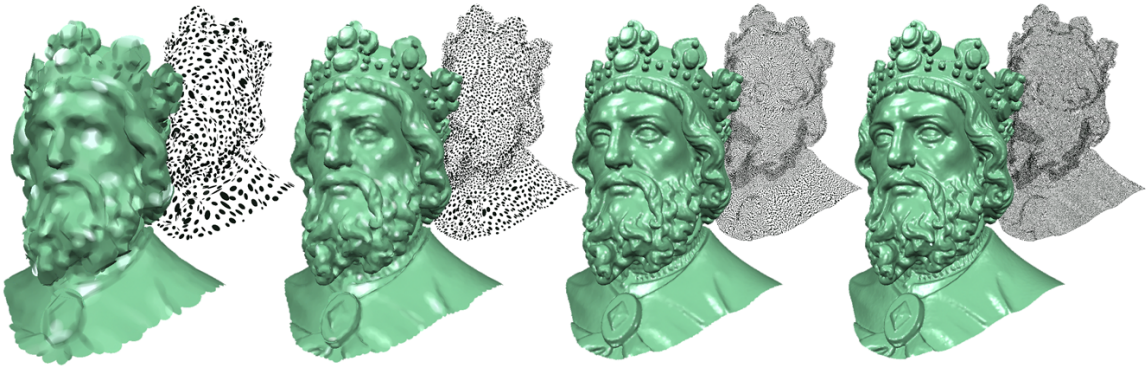

Figure 1: *Progressive splatting of Charlemagne model (*$600K$ *points) from left to right with* $2K, 10K, 70K$ *and* $600K$ *splats.*

meshes to splat-based geometry representations. Our progressive splat decimation procedure uses the standard greedy approach but unlike previous work it uses the full splat geometry (both center and its extent) in the decimation criteria and error estimates and not just the splat centers. As we will demonstrate later, with two improved error metrics generalized from [CSAD04], this new greedy framework offers much better quality than other *progressive* splat decimators. Therefore it is preferable for those progressive applications like rendering and transmission. The quality that we obtain is coming close to the recently proposed *single-resolution* high-quality sub-sampling techniques which are based on global optimization [WK04] while being much faster by an average factor of 3.

1.1. Related Work

Considerable work has been presented during the last years relating to point-based graphics research. A complete survey to many other available point-based techniques is beyond the scope of this paper, while the readers are referred to [GPZ*04, KB04] for the same purpose.

To construct multiresolution point-based models, most previous work has adopted the hierarchical space partitioning strategy aiming at efficient visualization of complex datasets. The input point set is clustered within a hierarchical space partitioning data structure and representative points or splats are created in each node by analyzing the local surface properties. Rusinkiewicz and Levoy [RL00] used bounding sphere hierarchies to perform recursive point rendering. Piecewise constant average points are computed for all spheres in the structure. This idea was followed in [DVS03] and the pre-computed hierarchy was rearranged into a sequential format that can be efficiently processed by nowadays graphics hardware. Many other papers also used octrees as underlying hierarchical structures [BWK02, Paj03, KV03]: while [KV03] only concentrated on C^{-1} piecewise constant points, [BWK02, Paj03] can also generate multlresolution C^{-1} piecewise linear splat geometry. More recently [SPL04] converted the output of [Paj03] to multiple continuously stored harddisk files to render large splat samples in an out-of-core fashion. In general, hierarchial space partitioning techniques are straightforward and efficient. However, due to the lack of algorithmic flexibility, they are often not able to produce progressive models of sufficient quality especially in coarse approximation levels. This results in poor splat representations in the beginning phases of progressive splatting (cf. Fig. 2).

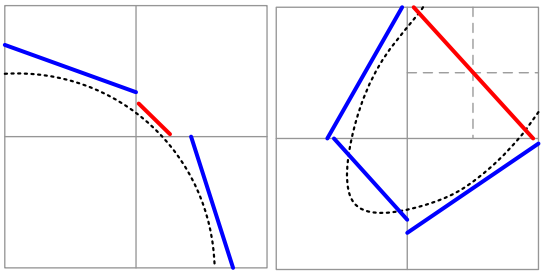

Figure 2: *Hierarchical space partitioning will lead to non-uniform splat sizes (left) or splats connecting two distant parts (right) in the coarse level due to the less flexible global partitioning. Splats are shown as thick lines and displaced for clear illustration, surfaces as dotted curves. Further subdivision (dashed lines, right) could improve these problems but* 1*-to-n partitioning prevents the granularly resolution control of progressive models.*

To achieve better quality, the well established greedy optimization schemes used in mesh simplification has been adapted by various works [Lin01, PGK02, FACOS03] to simplify point-sampled geometry. Their results can naturally lead to a progressive surface format for progressive splatting. Unfortunately, they all focused only on the relationships between splat centers and hence require extra effort to estimate the actual splat spatial extent. On the contrary, taking the full geometry of surface splats into account, Wu and Kobbelt [WK04] presented an optimal splat sub-sampling

algorithm that performs a greedy phase followed by a post global relaxation optimization step. The output quality of this method is superior compared to most other similar schemes, which has inversely compromised its progressive capability due to its non-continuous single-resolution subsampling property.

2. Overview

Targeting at progressive splatting, we introduce an iterative greedy splat decimation framework working directly on C^{-1} piecewise linear surface splats. Intuitively, it functions as the counterpart of the well-known progressive meshes [Hop96] in the C^0 piecewise linear polygonal meshes setting and the iterative point simplification [PGK02] in the C^{-1} piecewise constant points setting.

Specifically, given the input point set, *initial splats* are first created for all point samples. Then all possible *splat merge operators* are arranged in a priority queue according to an *error metric* measuring errors caused by respective operators with top element having minimum error. *Iterative operations* are usually performed repeatedly with applying the top operator and updating possibly affected operators in the queue until the desired number of splats is reached. More details will be discussed in the upcoming sections.

3. Initial Splat Creation

The input of our algorithm is a set of unstructured points $P = \{\mathbf{p}_i\}$ sufficiently sampling and describing a smooth manifold boundary surface $\mathcal{S}$ of a given 3D object. Each output splat T_i is a general 3D ellipse given by its center $\mathbf{c}_i$, its unit normal vector $\mathbf{n}_i$, and two additional *non-unit* vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ defining its major and minor axes.

Similar to [PGK02] and [WK04], in order to analyze local surface properties as well as the associated initial splat T_i of a point sample $\mathbf{p}_i$, the k-nearest neighbors $N_k(\mathbf{p}_i)$ have to be computed beforehand. Then a least square plane L can be found for $\mathbf{p}_i$ and $N_k(\mathbf{p}_i)$ defining the normal $\mathbf{n}_i$ of T_i, with center $\mathbf{c}_i = \mathbf{p}_i$. As we set all initial splats to be circles, initial $\mathbf{u}_i$ and $\mathbf{v}_i$ can be any two orthogonal vectors parallel to L with same length r,

$$r = max \left\| (\mathbf{p}_j - \mathbf{c}_i) - \mathbf{n}_i^T (\mathbf{p}_j - \mathbf{c}_i)\,\mathbf{n}_i \right\|,$$

for all $\mathbf{p}_j \in N_k(\mathbf{p}_i)$.

In the meantime, a virtual graph $\mathcal{N} = (P, E)$, where the edge (i, j) belongs to E iff $\mathbf{p}_j \in N_k(\mathbf{p}_i)$, is also formed to represent the above neighborhood relationship. This graph $\mathcal{N}$ will be the supporting *dynamic* topology during the iterative splat merge operations, i.e., as an extension of the common edge collapse operator [GGK02], a splat merge operator Φ will merge two splats T_l and T_r associated with endpoints $\mathbf{p}_l$ and $\mathbf{p}_r$ of an edge $e \in E$ into one larger splat T_m. Thus the ordering queue will initially contain splat merge operators corresponding to all edges in E. Once an operator Φ is applied, e and other degenerated edges will be removed from the edge set E and $\mathbf{p}_l$ and $\mathbf{p}_r$ will be replaced by a new point $\mathbf{p}_m = \mathbf{c}_m$ in the point set P.

4. Error Metrics and Splat Merge Operators

In order to ensure similar approximation quality as that of the established mesh cases, two different error metrics measuring distance deviation and normal deviation respectively are also generalized and embedded into the new splat decimation framework. As splat merge operators heavily depend on specific error metrics, we also introduce their detail information at same time in this section.

4.1. L^2 Metric

The L^2 error metric is based on Euclidean distance measurement. To be able to compute the deviation error caused by a splat merge operator Φ with respect to the original point set, an additional array of indices $\{f_i\}$ to the original points is kept for each splat T_i and initialized with a single index $\{i\}$ referring to the initial point $\mathbf{p}_i$. When merging two splats, their index arrays will be united and assigned to the new splat. Then for a merge operator Φ, to merge splat T_l and T_r to new splat T_m, the approximation error is defined as:

$$\varepsilon_\Phi = \|e\| \cdot \sum_{f \in \{f_m\}} |dist(\mathbf{p}_f, T_m)|^2,\ \{f_m\} = \{f_l\} + \{f_r\}. \quad (1)$$

Note that the above error metric has been weighted by the length of the edge e measuring the distance between two merged splat centers to penalize merging two distant splats which otherwise would produce over-sized splats.

Given the L^2 error metric (1) and two splats T_l and T_r to be merged, the new splat T_m can be determined by applying Principle Component Analysis (PCA [Jol86]) to the point set $P_m = \{\mathbf{p}_f\}, f \in \{f_m\}$ in 3D directly rather than the projected point set in 2D as [Paj03]. Afterwards, we will have the average point $\bar{\mathbf{p}}$ as well as three real eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$ and the corresponding eigenvectors $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$. Then for T_m, center $\mathbf{c}_m = \bar{\mathbf{p}}$, normal $\mathbf{n}_m = \mathbf{v}_3$, and two axes $\mathbf{u}_m$ and $\mathbf{v}_m$ will have direction $\mathbf{v}_1$ and $\mathbf{v}_2$ respectively with a length ratio $\sqrt{\lambda_1/\lambda_2}$. The final axis lengths are scaled simultaneously so

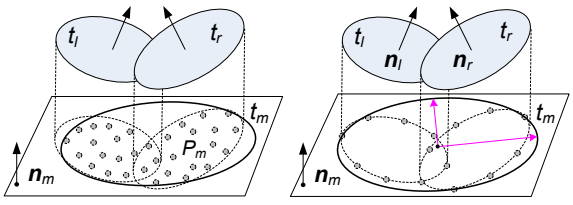

Figure 3: *Splat merge operators according to L^2 error metric (left) and $L^{2,1}$ metric (right).*

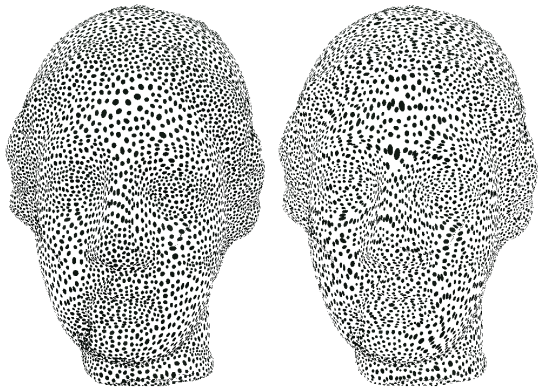

Figure 4: *Igea model (*134*K points) coarsified to same number of* 13*K splats using L^2 error metric (left) and $L^{2,1}$ metric (right) with respective run time 52 sec. and 31 sec. and relative Hausdorff distance error* 0.017% *and* 0.022%.

that the elliptical splat T_m covers all points P_m in 2D when they are projected onto the splat plane (cf. Fig. 3).

Once a splat merge operator Φ is applied, the graph $\mathcal{N}$ will be updated as we have mentioned in Section 3. Similar to standard mesh simplification, all neighboring merge operators to T_l and T_r in the ordering queue have to be updated with new merge errors or removed from the queue if they are degenerated and not valid any more.

4.2. $L^{2,1}$ Metric

$L^{2,1}$ error metric measures the deviation of normal directions and is extended from the original metric first presented in [CSAD04]. In this case, the error computation is simpler and we do not need to keep the index array either. Given the splat merge operator Φ, the respective area $|T_l|$ and $|T_r|$ of two splats T_l and T_r to be merged, similar to (1), the edge-length weighted error is calculated as:

$$\varepsilon_\Phi = \|e\| \cdot |T_l| \cdot |T_r| \cdot \|\mathbf{n}_l - \mathbf{n}_r\|^2 . \qquad (2)$$

According to the $L^{2,1}$ metric, the geometry of new splat T_m is defined as center

$$\mathbf{c}_m = \frac{|T_l| \cdot \mathbf{c}_l + |T_r| \cdot \mathbf{c}_r}{|T_l| + |T_r|};$$

and normal

$$\mathbf{n}_m = \frac{|T_l| \cdot \mathbf{n}_l + |T_r| \cdot \mathbf{n}_r}{|T_l| + |T_r|}.$$

The extent of splat T_m is computed in the same way as for the L^2 metric with the only difference that rather than projecting the point set P_m (which we do not keep), we uniformly sample n points (usually 8 is enough) on both boundaries of splat T_l and T_r and project all of them to the splat plane of T_m to find the main axis directions and proper scaling (cf. Fig. 3).

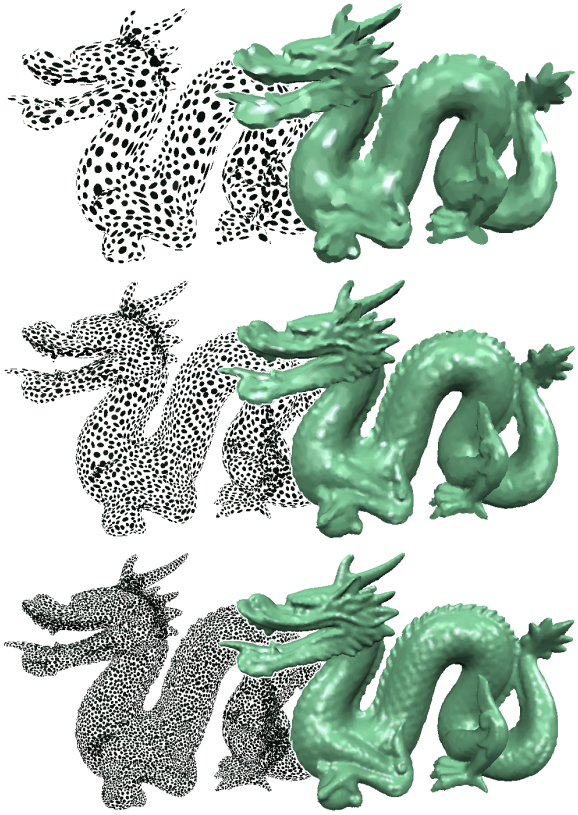

Figure 5: *Progressive splatting of Dragon model (*438*K points) from top to bottom with* 2*K*, 6*K and* 20*K splats.*

We have compared the effects of the different L^2 and $L^{2,1}$ error metrics in Fig. 4. It is not surprising that we have similar observations as [CSAD04]: $L^{2,1}$ metric can better capture the anisotropy of the surface geometry, while L^2 metric will generate a more uniform splat distribution. Due to its simple computational efforts, algorithms using $L^{2,1}$ metric naturally run much faster than those using L^2 metric. Because the error is measured as one-sided maximum Hausdorff distance from the original point set to coarser splat approximations (the percentage of the major bounding box diagonal length), it is also clear to see that this distance-based error measurement favors distance-based L^2 metrics over normal-based $L^{2,1}$ metrics.

5. Progressive Splats Format

The sequence of splat merge operators $\{\Phi_i\}$ can be recorded during the splat decimation procedure. Together with the remaining coarse base splat set $\mathbf{T}_B$, the splat merge operators $\{\Phi_i\}$ and their straightforward inverses, splat split operators, form a progressive splats format in the similar way as progressive meshes [Hop96] with the major difference that progressive splats do not maintain topology information.

Looking into details, the progressive splats format is composed with a base splat set $\mathbf{T}_B$ and a set of continuous detail operators $\{\Psi_i\}$. Each single detail operator Ψ_i is derived from corresponding recorded splat merge operator Φ_i and will contain three splat indices l, r, m and the geometry of three splats T_l, T_r, T_m. The data storage is amount to 48 Bytes per operator under single floating point precision. For refinement, the splat of index m will be split and replaced with two smaller splats T_l and T_r; For coarsification, two splats of respective indices l and r will be merged and replaced with a larger splat T_m. Note that by utilizing this progressive splats format we can perform both up-streaming and down-streaming without any extra data storage, and thus can produce splat models at arbitrary resolution. This also will support efficient applications like progressive rendering and transmission of surface splats.

6. Results

We have tested our progressive splatting algorithm with various point-based datasets on a standard Linux PC with 2.8 GHz CPU and 2 GB main memory. If not otherwise specified, k-nearest neighbors is always set to $k = 8$ and the $L^{2,1}$ error metric is applied to produce the reported results. Shaded results to show the overall visual quality can already be found in Fig. 1 and Fig. 5, where splats are also shrinked to demonstrate their shapes and distribution.

6.1. Comparison to Splat Decimation Methods

We conduct comparative analysis of our *progressive* splatting algorithm (denoted as PSP) with the other two typical *progressive* splat decimators, the LOD point rendering (LOD) [Paj03] and iterative point simplification (IPS) [PGK02], and the *single-resolution* optimal splat subsampling scheme (OSS) [WK04] respectively. For LOD, splats in the same Octree levels will be collected and for IPS, an extra step is necessary to convert its point-based output to the splat representation using a strategy similar to the one discussed in Section 3.

Quality The result quality of different methods is estimated in the following aspects:

- **Error measurement** captures the statistical distances between decimated splat approximations and the original point model. Fig. 7 compares the three progressive splat decimators while in Fig. 8 errors caused by our progressive PSP algorithm and discrete OSS scheme are reported.
- **Visual quality** depends on the rendering effects as well as the splat shapes and distribution (see Fig. 8 and Fig. 10).
- **Area ratio** between the area sum of splat approximations and the mesh surface area of the original point model (see Tab. 1). With same number of splats, the smaller the ratio, the smaller the area of splats to be rasterized in the fragment shader of GPU, and the faster the rendering speed.

n_{splat}	PSP	LOD	IPS	OSS
415	2.18	2.34	4.14	1.63
2591	2.52	2.71	4.12	2.15
11588	3.23	3.22	4.23	3.13

Table 1: *Area ratio for different splat decimation methods, normalized to the initial surface area of the triangle mesh. This factor measures how much overdraw occurs in the rasterization of the splats.*

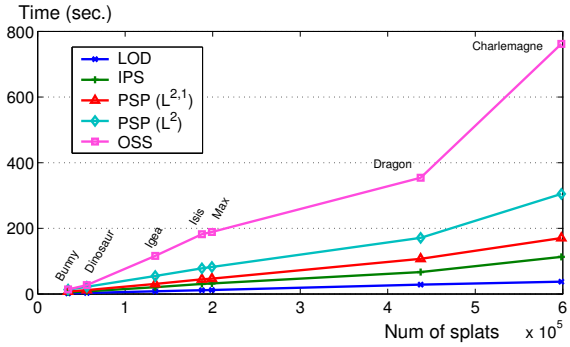

Figure 6: *Computation times for different splat decimation methods where for PSP, OSS and IPS, times are measured for a simplification to* 1% *of the input model size and LOD is its whole structure creation time.*

Considering of all above three criteria in combination, it is not difficult to tell that, among the three *progressive* splat decimators, our PSP algorithm always performs better than both LOD and IPS. Especially on coarser scales, we find that because LOD merely adopted octree space partitioning scheme and IPS only considered points, i.e. splat centers rather than whole splats (one reason for its large area ratio too), they could not produce as promising results as ours. In addition, although the *single-resolution* OSS usually produces best quality due to its global optimization, OSS can not create progressive splat representations and even in some aspects (e.g. error measurement) our PSP method comes quite close to the best OSS solution already.

Timing Computation times of different splat decimation methods are shown and compared in Fig. 6 as functions of input model size. No wonder that LOD runs fastest as it has a quite simple algorithmic structure. Although both using the same greedy framework, our PSP algorithm is slower than IPS which has adopted efficient quadric error metric (QEM) [GH97], as it has to compose and solve least-square systems in each splat merge step. And it is not a surprise that the best-quality OSS needs most running time because of its complex global optimization techniques. Nonetheless, since high computational costs have been traded with improved output quality, and since all splat decimation schemes are pre-processing procedures, the amount of running time we have reported is always endurable.

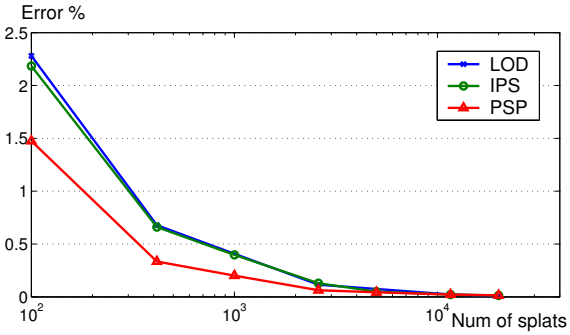

Figure 7: *Error comparisons on bunny model (35K points) for different* progressive *splat decimators.*

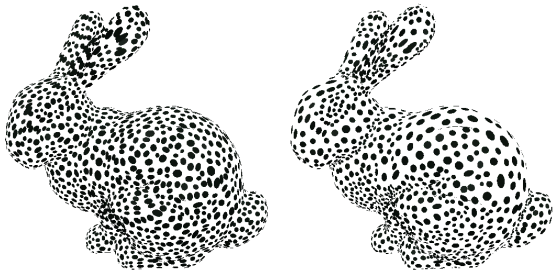

Figure 8: *Bunny model (cf. Fig. 10) decimated to similar number of splats by progressive PSP (left,* 2591*) and single-resolution OSS (right,* 2577*) algorithms. Although PSP and OSS have quite close errors (*0.103% *to* 0.092%*), being able to concentrate more splats on regions of high curvatures, OSS gives better splat shapes and distribution than PSP.*

Discussion In general, LOD and IPS run very fast while producing results of inferior quality. On the contrary, OSS can create output with best quality but is significantly slower. Our progressive PSP algorithm is intended to fill this gap: it runs only a little slower than LOD and IPS while much faster than OSS, and it offers result quality very close to the best OSS. In fact, what we have provided here is also a *practical guide* to choose different splat decimation strategy based on the specific quality-time tradeoff.

6.2. Comparison to Mesh Simplification

We also compare in Fig. 9 our PSP splat decimation algorithm to QEM mesh simplification method [GH97] that can produce progressive meshes [Hop96]. The two methods can produce quite similar results due to the same greedy piecewise linear approximation nature, but it is also worth to mention the lower approximation error of our results as splats are more flexible and can overlap with each other while triangles are rigidly connected in a C^0 fashion.

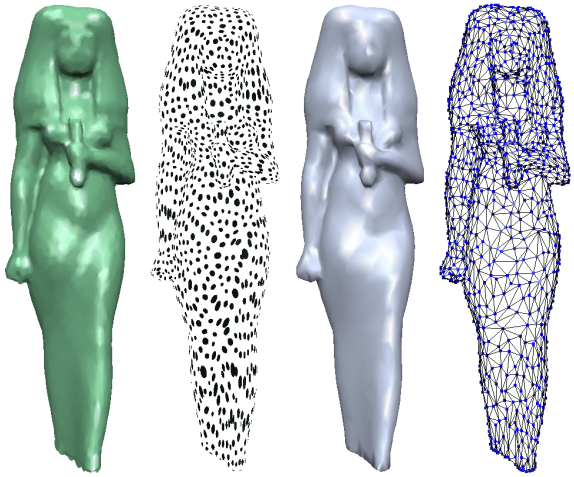

Figure 9: *Isis mesh (*188*K points) decimated by our PSP algorithm (left two) and QEM simplification (right two) to same* 2*K splats and vertices with respective run time 31 sec. and 28 sec. and relative error* 0.51% *and* 0.63%*.*

7. Conclusions

Multiresolution representations are always necessary to deal with large datasets. Previous work on progressive splat decimation usually utilizes a hierarchical space partitioning strategy to create corresponding multiresolution formats, or adopts a greedy framework with the strategy of focusing only on the relationships between splat centers. As we have shown in the paper, although running very fast, these two strategies can not produce plausible results at the coarse approximation level. This may be problematic in some cases, e.g. during progressive rendering from low resolution to higher one, much lower rendering quality will appear in the very beginning (cf. Fig. 10). In this paper we have presented a greedy progressive splat decimation algorithm which utilized the full splat geometry in the decimation criteria and error estimates not just the splat centers. It offers much better quality than other *progressive* splat decimators and also comes quite close to the recently proposed globally optimized *single-resolution* sub-sampling techniques while runs much faster. In summary, our splat decimation algorithm can be a new candidate to generate progressive splat models with a good time-quality tradeoff.

One of the most obvious future steps would be the out-of-core implementations of the proposed method like streaming processing for polygonal meshes [WK03] as they are essential to deal with those massive point models. In addition, because our current progressive splat format still needs quite a lot storage space, the simplification of progressive structures is always important for applications like progressive data transmission. Progressive geometry compression schemes [GD02, AG05] could also be employed in our splat

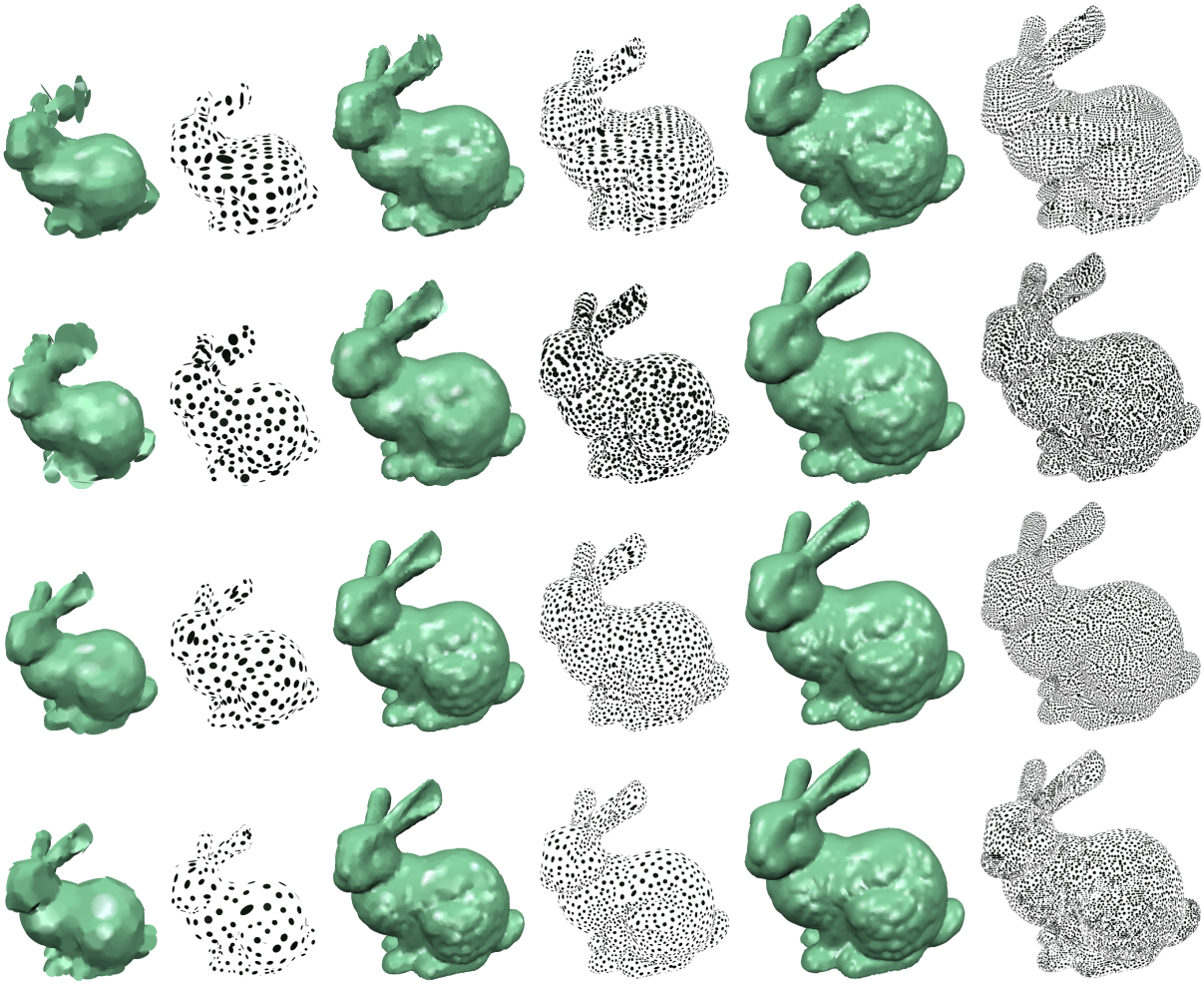

Figure 10: *Visual comparisons of LOD (top row), IPS (top middle), our algorithm PSP (bottom middle) and OSS (bottom) where the bunny model (35K points) is approximated with same number of* 415 *(left),* 2591 *(middle) and* 11588 *(right) splats for LOD, IPS and PSP and* 419, 2577 *and* 11564 *splats for OSS respectively.*

setting to further reduce the data size. Finally selective LOD splat rendering systems need to be developed based on this progressive splatting techniques.

Acknowledgements

We would like to thank the anonymous reviewers for their insightful comments and Martin Habbecke for proofreading the paper. Datasets used in the paper are courtesy of the Stanford Graphics Group and the Cyberware website.

References

[AG05] ALLIEZ P., GOTSMAN C.: Recent advances in compression of 3d meshes. In *Advances in Multiresolution for Geometric Modelling* (2005), Springer-Verlag, pp. 3–26. 6

[BK03] BOTSCH M., KOBBELT L.: High-quality point-based rendering on modern GPUs. In *Proceedings of Pacific Graphics 2003* (2003), pp. 335–343. 1

[BSK04] BOTSCH M., SPERNAT M., KOBBELT L.: Phong splatting. In *Eurographics Symposium on Point-Based Graphics* (2004), pp. 25–32. 1

[BWK02] BOTSCH M., WIRATANAYA A., KOBBELT L.: Efficient high quality rendering of point sampled geometry. In *Eurographics Workshop on Rendering* (2002). 2

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN

M.: Variational shape approximation. *ACM Transactions on Graphics. Special issue for SIGGRAPH conference 23*, 3 (2004), 905–914. 2, 4

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential point trees. *ACM Transactions on Graphics 22*, 3 (SIGGRAPH 2003), 657–662. 1, 2

[FACOS03] FLEISHMAN S., ALEXA M., COHEN-OR D., SILVA C. T.: Progressive point set surfaces. *ACM Transactions on Graphics 22*, 4 (2003), 997–1011. 2

[GD02] GANDOIN P.-M., DEVILLERS O.: Progressive lossless compression of arbitrary simplicial complexes. *ACM Transactions on Graphics 21*, 3 (SIGGRAPH 2002), 372–379. 6

[GGK02] GOTSMAN C., GUMHOLD S., KOBBELT L.: Simplification and compression of 3d-meshes. In *Tutorials on Multiresolution in Geometric Modeling* (2002), Springer. 1, 3

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings ACM SIGGRAPH 1997* (1997), pp. 209–216. 5, 6

[GPZ*04] GROSS M., PFISTER H., ZWICKER M., ALEXA M., PAULY M., STAMMINGER M.: Point-based computer graphics. In *SIGGRAPH 2004 Course Notes 6* (2004). 1, 2

[Hop96] HOPPE H.: Progressive meshes. In *Proceedings of ACM SIGGRAPH 1996* (1996), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 99–108. 1, 3, 4, 6

[Jol86] JOLLIFFE I.: *Principle Component Analysis*. Spring-Verlag, 1986. 3

[KB04] KOBBELT L., BOTSCH M.: A survey of point-based techniques in computer graphics. *Computers & Graphics 28*, 6 (2004), 801–814. 1, 2

[KV03] KALAIAH A., VARSHNEY A.: Statistical point geometry. In *Eurographics Symposium on Geometry Processing* (2003), pp. 107–115. 2

[Lin01] LINSEN L.: Point cloud representation. In *Technical report No. 2001-3* (2001), Faculty for Computer Science, Universitaet Karlsruhe. 2

[LPC*00] LEVOY M., PULLI K., CURLESS B., RUSINKIEWICZ S., KOLLER D., PEREIRA L., GINZTON M., ANDERSON S., DAVIS J., GINSBERG J., SHADE J., FULK D.: The digital michelangelo project. In *Proceedings SIGGRAPH 2000* (2000), pp. 131–144. 1

[Paj03] PAJAROLA R.: Efficient level-of-details for point based rendering. In *Proceedings IASTED Computer Graphics and Imaging* (2003). 2, 3, 5

[PGK02] PAULY M., GROSS M., KOBBELT L.: Efficient simplification of point-sampled surfaces. In *Proceedings IEEE Visualization 2002* (2002), pp. 163–170. 2, 3, 5

[RL00] RUSINKIEWICZ S., LEVOY M.: Qsplat: A multiresolution point rendering system fo large meshes. In *Proceedings SIGGRAPH 2000* (2000), pp. 343–352. 1, 2

[RPZ02] REN L., PFISTER H., ZWICKER M.: Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Eurographics 2002 Conference Proceedings* (2002), pp. 461–470. 1

[SPL04] SAINZ M., PAJAROLA R., LARIO R.: Extreme splatting: External memory multiresolution point visualization. In *Technical Report UCI-ICS-04-14* (2004), University of California Irvine. 2

[WK03] WU J., KOBBELT L.: A stream algorithm for the decimation of massive meshes. In *Graphics Interface 2003 Proceedings* (2003), pp. 185–192. 6

[WK04] WU J., KOBBELT L.: Optimized sub-sampling of point sets for surface splatting. *Computer Graphics Forum 23*, 3 (2004), 643–652. (Proc. Eurographics 2004). 1, 2, 3, 5

[ZPBG01] ZWICKER M., PFISTER H., BAAR J. V., GROSS M.: Surface splatting. In *Proceedings SIGGRAPH 2001* (2001), pp. 371–378. 1

[ZRB*04] ZWICKER M., RÄSÄNEN J., BOTSCH M., DACHSBACHER C., PAULY M.: Perspective accurate splatting. In *Graphics Interface 2004* (2004), pp. 247–254. 1