Memory-Efficient Sliding Window Progressive Meshes

Pavlo Turchyn
University of Jyväskylä, Finland
pturchy@cc.jyu.fi

Abstract

Progressive mesh is a data structure that encodes a continuous spectrum of mesh approximations. Sliding window progressive meshes (SWPM) minimize data transfers between CPU and GPU by storing mesh data in static on-GPU memory buffers [For01]. The main disadvantages of the original SWPM algorithm are poor vertex cache usage efficiency, and big resulting datasets. Connectivity-based algorithm [KT04] achieves a good vertex cache coherence but it does not address the problem of high memory utilization. In this paper, we describe estimates for the size of memory buffers, and describe methods to reduce the index datasets. We achieve 20% reduction due to the use hierarchical data structures (clustered patches); further reduction (50% or more) is possible if one can optimize connectivity of input meshes, or is willing to decrease the number of mesh approximations stored in the progressive mesh.

Keywords: level of detail, GPU, progressive meshes, independent sets, face clusters.

1 INTRODUCTION

Progressive mesh data structure encodes a continuous spectrum of mesh approximations [LRC*02, pp.70–72]. Efficient hardware rendering of the progressive meshes is discussed in a number of papers. Many proposed methods do not depend on mesh simplification procedure; they address only creation and maintenance of GPU-optimal sequences of triangles, such as triangle strips, while permitting the use of an arbitrary simplification algorithm (see e.g. [LRC*02, pp.168–169], [Ste01, BG01, SP03]). These methods process the data on CPU, and then transfer the results to GPU via generally slow procedure. Since the use of progressive meshes implies frequent LOD changes, the data transfers will be a bottleneck in most cases.

The problem of CPU-GPU data transfers is addressed in some view-dependent schemes, which partition a mesh into the set of triangle patches, e.g. [YSG05, CGG*05]. Each patch is updated once per several frames, thus the cost of updating the entire model is distributed over time. However, these schemes impose significant overheads when rendering smaller meshes (e.g. <100K triangles) with multiple instances, which are found in many applications, such as videogames.

Sliding window progressive meshes (SWPM) allow both efficient rendering of the instantiated meshes, and fast geomorphing that exploits frame-to-frame coherence. SWPM simplification algorithm partitions a mesh into the set of patches, which are sequentially replaced with simpler triangulations in a *view-independent* fashion. Unlike the view-dependent schemes, SWPM use very small patches (about six triangles each), so the typical difference between two subsequent mesh approximations is two triangles. The patches are stored in static on-GPU memory buffers that tend to be relatively large; this problem is augmented by the fact that simplification with higher quality requires even larger buffers.

This paper is organized as follows. Sec. 2 describes preliminaries about SWPM algorithm. Sec. 3 contains contribution of this paper; we derive estimates for the size of memory buffers, and propose methods to reduce memory consumption.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright UNION Agency – Science Press

2 SLIDING WINDOW ALGORITHM

Simplification operators. In this paper, we consider *half-edge collapse* and *vertex removal* operators. Let $M = (V,T)$ be a 2-manifold triangle mesh, where $V = \{v_i | v_i \in \mathbb{R}^3\}$ is a set of vertices, and $T = \{\triangle_\ell | \triangle_\ell = (v_i, v_j, v_k), v_i, v_j, v_k \in V\}$ is a set of triangles. Vertex removal operator $\mathscr{R}(v_i)$ deletes from the mesh the vertex v_i and all incident triangles, and then re-triangulates the resulting hole with a triangulation algorithm. Half-edge collapse operator $\mathscr{H}(v_i, v_j)$ replaces the vertex v_i with the vertex v_j in all triangles of the mesh M and then removes the resulting degenerate edge and corresponding degenerate triangles from the mesh. One can interpret $\mathscr{H}$ as a vertex removal with the specific triangulation algorithm.

Mesh simplification algorithm builds a sequence of mesh approximations $\mathbf{M} = \{M_0, M_1, \ldots, M_n\}$ from the initial mesh M_0. An approximation $M_{k+1} \in \mathbf{M}$ is the result of applying the operator $\mathscr{H}(v_i, v_j)$ (or $\mathscr{R}(v_i)$) with some $v_i, v_j \in V$ to $M_k \in \mathbf{M}$ (the vertices v_i and v_j are chosen according to the value of an *error metric*).

Sliding window algorithm. Sliding window scheme was described in [For01] as an efficient method for rendering of instantiated geometry. Consider a mesh $M = (V,T)$. Let $S(v_i)$ be the set of the triangles incident

to the vertex $v_i \in V$, i.e. $S(v_i) = \{\triangle_\ell | \triangle_\ell \ni v_i, \triangle_\ell \in T\}$ ($S(v_i)$ is also called *star* of the vertex v_i). We say that a subset $P(v_i)$ of the set T is a *patch*, if $S(v_i) \subseteq P(v_i) \subseteq T$; here v_i is the *inner vertex*.

Let $\mathbf{P} = \{P_1, P_2, \ldots, P_n\}$ be a list of patches, where $P_1 \cup P_2 \cup \cdots \cup P_n = T$, and any triangle belong to only one patch. The inner vertices of such patches belong to an independent set. We define a list of *simplified patches* $\mathbf{Q} = \{Q_1, Q_2, \ldots, Q_n\}$. Each *simplified patch* $Q_i \in \mathbf{Q}$ is obtained by applying a simplification operator to the corresponding patch $P_i \in \mathbf{P}$, i.e. $\mathscr{R}(v_j) : P_i(v_j) \longmapsto Q_i$, or $\mathscr{H}(v_j, v_k) : P_i(v_j) \longmapsto Q_i$, where $v_k \in P_i(v_j)$. We stress that a simplification operator removes the inner vertex only. Thus, we can replace any P_i with Q_i while maintaining conformity (i.e. no cracks or T-joints) of the mesh.

Let $||$ denote the lists concatenation operator. $A||B$ builds the list that contains the elements of A followed by the elements of B, i.e. $\{a_1, \ldots, a_n\}||\{b_1, \ldots, b_m\} = \{a_1, \ldots, a_n, b_1, \ldots, b_m\}$. We construct the *index buffer* (which is a memory buffer that holds triangles connectivity information) from the list of patches $\mathbf{P}$ and the corresponding list of simplified patches $\mathbf{Q}$ as follows

$$I(\mathbf{P}, \mathbf{Q}) = P_1||P_2||\cdots||P_n||Q_1||Q_2||\cdots||Q_n. \quad (1)$$

The important outcome of obtaining such I is that we can render a LOD simply by choosing the appropriate window in I, e.g. the original mesh M_0 is rendered using the window $P_1||P_2||\cdots||P_n$, M_1 is rendered using the window $P_2||P_3||\cdots||Q_1$, M_2 is rendered using the window $P_3||P_4||\cdots||Q_1||Q_2$, etc. No memory update is required to change mesh LOD; the index buffer may be shared among all mesh instances.

Geomorphing. The sliding window scheme enables efficient geomorphing to smooth LOD transitions. Consider $M_0 \to M_1$ transition; only the patch P_1 changes and actually needs morphing. In the case of an arbitrary $M_i \to M_j$ transition ($i < j$), the patches $P_{i+1}, P_{i+2}, \ldots, P_j$, which require morphing, form a continuous subsequence that may be rendered separately from the patches, which do not need morphing. One can expect that the majority of patches may be processed without morphing because in practice i is ofen close to j due to the frame-to-frame coherence. Since the morphing via linear interpolation effectively doubles the amount of data transferred per vertex, the sliding window scheme reduces by a factor of two the memory bandwidth consumed by geomorphing.

Vertex caching. One have to construct the list of patches $\mathbf{P}$ in such a way that any two subsequent patches $P_i \in \mathbf{P}$ and $P_{i+1} \in \mathbf{P}$ share via a vertex cache as many vertices as possible [KT04]. Thus, the sequence of simplified patches $\mathbf{Q}$ is also cache-optimized. Therefore, vertex cache hit rate remains high for any mesh LOD. Alternatively, one can optimize patches order for efficient Z-culling [NBS06].

3 SWPM INDEX DATASET

Consider a mesh that contains 60K triangles and 30K vertices. Below we show that SWPM index dataset takes about 2Mb of memory; on the other hand, only 0.7Mb buffer is required for storing two 3-vectors (e.g. position and normal) per vertex. Thus, reduction of the index dataset, which is described in this section, may be the main tool to reduce the size of geometric data for memory-limited applications.

Cascade simplification. Suppose, we have obtained the list of patches $\mathbf{P}_0$ (and the corresponding list of simplified patches $\mathbf{Q}_0$) on the mesh M_0. We construct SWPM index buffer $B_0 := I(\mathbf{P}_0, \mathbf{Q}_0)$. Let M_n be the simplest mesh approximation that can be rendered using the buffer B_0. It is possible to derive the limits (7) for the number of triangles M_n contains. However, many application may require simpler approximations. Thus, the simplest approximation M_n must be simplified further; we have to construct the buffer $B_1 := I(\mathbf{P}_1, \mathbf{Q}_1)$ where M_n serves as an initial mesh. Generally, we construct the buffers $B_0, B_1, \ldots, B_z$, where the buffer B_z contains a sufficiently coarse mesh.

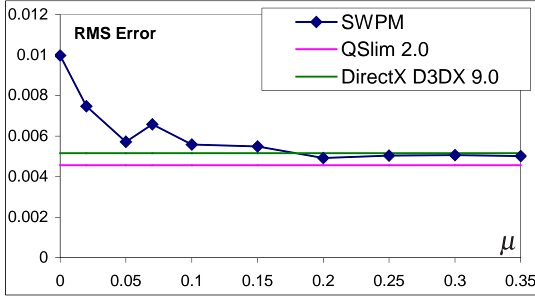
We denote $|B_i|$ the number of triangles, which are stored in the SWPM buffer B_i. Let β be the total number of triangles in all SWPM buffers

$$\beta := \sum_{i=0}^{z} |B_i|. \quad (2)$$

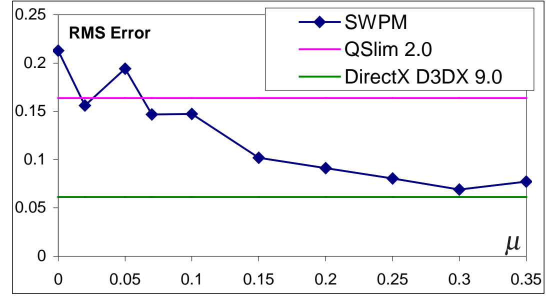
Under certain assumptions it is possible to derive a simple estimate (10) for the minimum value of β. The mesh-dependent parameter $\overline{d}$ is a mean degree of the vertices of the *maximum independent set*. Using this estimator, one can expect that SWPM index dataset is at least 3–5 times larger than the index buffer of the original mesh (since typically $\overline{d} \in [4; 6]$).

The factor of error control. A drawback of the simplification algorithms that remove independent vertices is their limited ability to adapt mesh triangulation according to the original surface geometry. Consider the mesh shown in Fig. 5a. Conservatively, we start simplification by reducing complexity of the flat upper part of the mesh while leaving the curved bottom part untouched as shown in Fig. 5c. However, we delete vertices uniformly from the top and the bottom when removing the maximum independent set.

One possible solution is as follows. First, we sort all vertices of the mesh $M = (V, T)$ according to the error induced by the removal of each vertex. Then we prohibit removal of k vertices with the largest error; any such vertex may not become an inner vertex. Let $\mu = k/|V|$ be a user-defined fraction of the vertices that may not be removed. Typical relations between μ and the simplification error are shown in Fig. 1. In practice, setting $\mu \in [0.05; 0.2]$ is often sufficient to achieve plausible simplification results. Similar argumentation concerning large meshes can be found in [FS01].

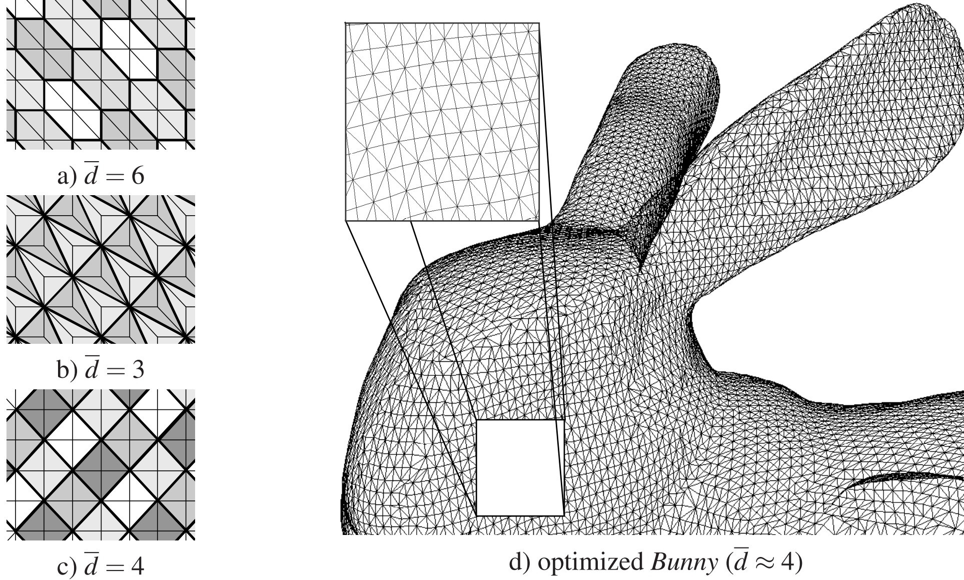


a) *t72* mesh is simplified from 2450 to 640 triangles



b) *cessna* mesh is simplified from 7446 to 580 triangles

Figure 1: Relationship between the simplification error and the fraction of non-removable vertices μ



a) $\overline{d} = 6$

b) $\overline{d} = 3$

c) $\overline{d} = 4$

d) optimized *Bunny* ($\overline{d} \approx 4$)

Figure 2: a) the patches on regular triangulation b) the result of applying Optimize procedure to the mesh shown in Fig. 2a c) the result of Optimize when $\overline{d}$ is limited to $\overline{d} \geq 4$ d) the latter procedure is applied to Bunny mesh (the model is courtesy of Stanford Computer Graphics Lab)

When $\mu > 0$, the following estimator tends to be relatively accurate

$$\beta = |T_0|(\overline{d} - 1)(\mu + 1). \tag{3}$$

Thus, one can expect that SWPM index dataset is 4–7 times larger than the index buffer of the original mesh (since typically $\overline{d} \in [4;6]$ and $\mu \in [0.05;0.2]$).

3.1 Mesh connectivity optimization

It follows from (10) and (3) that the size of index dataset is proportional to the mean degree of the inner vertices. At the same time, we notice that a greedy search for the maximum independent set, which is based on the minimum-degree heuristics, tends to include the vertices of lower degree to the independent set. Therefore, one way of minimizing $\overline{d}$ is creating low-degree vertices in the mesh artificially, e.g. at the preprocessing step that precedes the simplification process. One may perform such preprocessing by applying a sequence of *edge swap* operators. *Edge swap* operator $\mathscr{E}(v_k, v_\ell, v_m, v_n)$ replaces two triangles $\triangle_i = (v_k, v_\ell, v_m)$ and $\triangle_j = (v_\ell, v_k, v_n)$, which share the edge (v_k, v_ℓ), with the triangles $\triangle'_i = (v_k, v_n, v_m)$ and $\triangle'_j = (v_\ell, v_m, v_n)$. As a result, degrees of the vertices v_k and v_ℓ decrease.

One simple strategy is applying the operator $\mathscr{E}$ iteratively until the degree of any vertex cannot be reduced any further.

function *Optimize*(M)
  $X \leftarrow \{v_1\}$, $Y \leftarrow \{v_n\}$, where v_n is connected to v_1
  for each $v_i \in X$
    find the edge (v_i, v_m) such that $v_m \in Y$
    for each edge (v_i, v_j) starting with (v_i, v_m) in *ccw* order
      if $v_j \notin Y$ and $v_k, v_\ell \notin X$ **then**
        $M \leftarrow \mathscr{E}(v_i, v_j, v_k, v_\ell)M$
        $X \leftarrow X \cup \{v_j\}$
        $Y \leftarrow Y \cup \{v_k, v_\ell\}$
  return M

For example, *Optimize* produces the triangulation shown in Fig. 2b from the triangulation shown in Fig. 2a. It decreases the mean degree $\overline{d}$ down to three, which is the lowest value. However, it is easy to see that such optimization also dramatically worsens mesh

quality in planar mesh regions since the majority of resulting triangles would have obtuse angles. Therefore, one have to implement a check for planarity; if a curvature in a vertex is below some threshold, then degree of this vertex should not be decreased below four. Such modification of the algorithm is straightforward; we omit its details. The modified optimization produces the mesh with *diamond* pattern (see Fig. 2c) from the regular triangulation shown in Fig. 2a. Optimized *Bunny* model is shown in Fig. 2d.

3.2 Clustered patches

In this subsection we describe an improvement to the algorithm of SWPM dataset creation. In order to reduce the size of index dataset, we reuse the simplified patches to construct a new set of the patches via clustering.

We demonstrate the idea of clustering on a simple example. We define the list of seven patches $\mathbf{P} = \{P_1, P_2, \dots, P_7\}$ and the corresponding list of simplified patches $\mathbf{Q} = \{Q_1, Q_2, \dots, Q_7\}$. The simplified patches are shown in Fig. 3b. The simplified patches contain 28 triangles; the numbering of triangles is shown in Fig. 3a. By X denote the fragment of the index buffer $X := Q_1||Q_2||\cdots||Q_7 = \{\triangle_1, \triangle_2, \dots, \triangle_{28}\}$. The fragment X is shown in Fig. 4a; it is located at the end of index buffer $I(\mathbf{P}, \mathbf{Q})$.

Using a clustering algorithm, we construct a new list of patches $\mathbf{P}^h = \{P^h_1, P^h_2, \dots, P^h_n\}$, where each patch $P^h_i \in \mathbf{P}^h$ is a concatenation of one or more consecutive simplified patches of the list $\mathbf{Q}$. In our example, we define two patches $P^h_1 = Q_1||Q_2||Q_3$ and $P^h_2 = Q_4||Q_5||Q_6||Q_7$ (see Fig. 3c), thus $\mathbf{P}^h = \{P^h_1, P^h_2\}$. Then, we build the corresponding list of simplified patches $\mathbf{Q}^h = \{Q^h_1, Q^h_2\}$, and the index buffer $I(\mathbf{P}^h, \mathbf{Q}^h)$ in the same way as we do it for the basic sliding window algorithm. The index buffer starts with the fragment $Y := P^h_1||P^h_2 = Q_1||Q_2||\cdots||Q_7 = \{\triangle_1, \triangle_2, \dots, \triangle_{28}\}$ (see Fig. 4b), which coincides with the fragment X located at the end of index buffer $I(\mathbf{P}, \mathbf{Q})$ (see Fig 4a). The memory buffer $D := I(\mathbf{P}, \mathbf{Q})||I(\mathbf{P}^h, \mathbf{Q}^h) = P_1||P_2||\cdots||P_7||X||Y||Q^h_1||Q^h_2$ contains a large amount of duplicated data because $X \equiv Y$. However, instead of D we can use the buffer $D^h := P_1||P_2||\cdots||P_7||X||Q^h_1||Q^h_2$ that contains both $I(\mathbf{P}, \mathbf{Q})$ and $I(\mathbf{P}^h, \mathbf{Q}^h)$. The buffer D^h is smaller than D because the repeated subsequence is omitted.

The goal of clustering is building the lists of patches in such a way, that two or more index buffers are partially coincident, thus we can construct their compact memory representation, which is smaller than simple union of the buffers. As shown in Appendix, the size of SWPM index dataset decreases as the number of patches increases. In order to increase the number of patches, we introduce a notion of *linked patches*.

In our example, define patches $P^\ell_1 = Q_1||Q_2||\{\triangle_9 \in Q_3\}$, $P^\ell_2 = (Q_3 \setminus \{\triangle_9\})||Q_4||\{\triangle_{17} \in Q_5\}$, and $P^\ell_3 = (Q_5 \setminus \{\triangle_{17}\})||Q_6||Q_7$ (see Fig. 3d). We call P^ℓ_i and P^ℓ_{i+1} *linked patches* when they both include triangles of the same simplified patch. Linked patches P^ℓ_1 and P^ℓ_2 share the simplified patch Q_3; P^ℓ_2 and P^ℓ_3 share Q_5. We build the list of patches $\mathbf{P}^\ell = \{P^\ell_1, P^\ell_2, P^\ell_3\}$ and its corresponding list of simplified patches $\mathbf{Q}^\ell = \{Q^\ell_1, Q^\ell_2, Q^\ell_3\}$. Again, a fragment $P^\ell_1||P^\ell_2||P^\ell_3$ of the index buffer $I(\mathbf{P}^\ell, \mathbf{Q}^\ell)$ (see Fig. 4c) coincide with the fragment $X \subset I(\mathbf{P}, \mathbf{Q})$, thus we can construct more compact memory representation of these index buffers.

Linked patches have to be subsequent in the list $\mathbf{P}^\ell$. As a counterexample, consider such list $\mathbf{P}^\ell = \{P^\ell_2, P^\ell_1, P^\ell_3\}$, where linked patches P^ℓ_2 and P^ℓ_3 are not subsequent. Any continuous triangles subsequence of $I(\mathbf{P}^\ell, \mathbf{Q}^\ell)$ cannot coincide with Q_5; thus, $I(\mathbf{P}^\ell, \mathbf{Q}^\ell)$ and $I(\mathbf{P}, \mathbf{Q})$ cannot partially coincide, and the advantage of the use of clustering is lost.

Agglomerative clustering algorithm. The input of clustering algorithm is the set of simplified patches $\mathbf{Q}_0$ and the simplest mesh $M_n = (V_n, T_n)$ produced as a result of simplification of the mesh M_0. The only clustering criterion is maximization of the number of clusters. We can formulate this problem as a maximum independent set problem on a graph G^h. Each vertex v_i of the independent set is the inner vertex of a patch $P^h(v_i)$, which is formed by the simplified patches that include the vertex v_i. By $G(T)$ denote the connectivity graph of the triangulation T. The graph G^h is initialized with $G(T_x)$; then for each simplified patch $Q_j \in \mathbf{Q}_0$ we find the additional edges, which connect all previously unconnected vertices of Q_j (i.e. the edges required to make $G(Q_j)$ complete), and insert them to G^h.

In order to find the maximum independent set on G^h, one may use the greedy search described in [SvK97]). First, we mark high-error vertices (the number of such vertices is controlled by the user-defined parameter μ), then we delete all marked vertices by decrementing the degrees of their neighbors. While unmarked vertices remain, we choose a vertex of lowest degree for the independent set and delete it and its neighbors by marking them and decrementing the degrees of their neighbors.

Additionally, we implement a single-step lookahead to search for linked patches. For example, suppose the patch $P^\ell_1 = Q_1||Q_2||Q_3$ is found in the first iteration of the greedy search. Then we attempt to find a patch P^ℓ_2, which shares Q_1, Q_2, or Q_3 with P^ℓ_1. This is a local search since the linked patches are in the neighborhood of P^ℓ_1. If several candidates are found, we choose the one of the lowest inner vertex degree. If the linked patch P^ℓ_2 is found, we attempt to find a patch P^ℓ_3, which is linked to P^ℓ_2, etc. When no more linked patches can be constructed, we proceed with the basic greedy search for a next independent vertex.

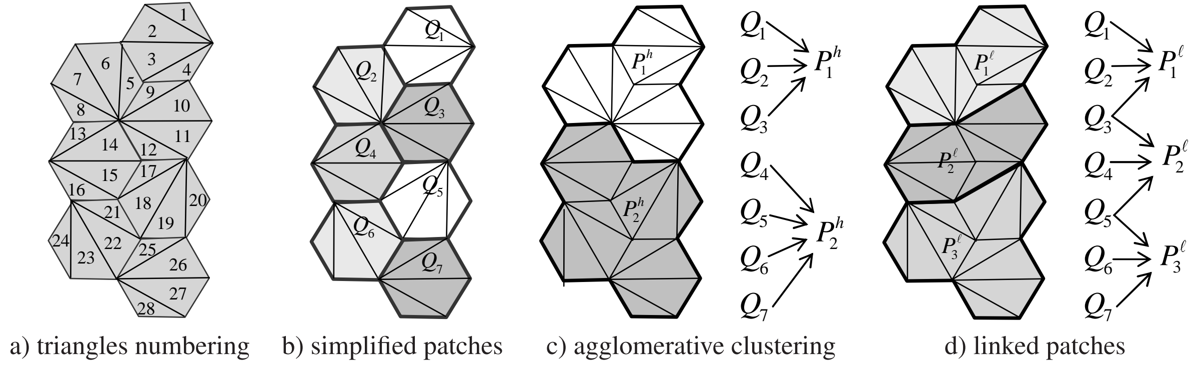


Figure 3: The use of clustering to construct new patches from the set of simplified patches

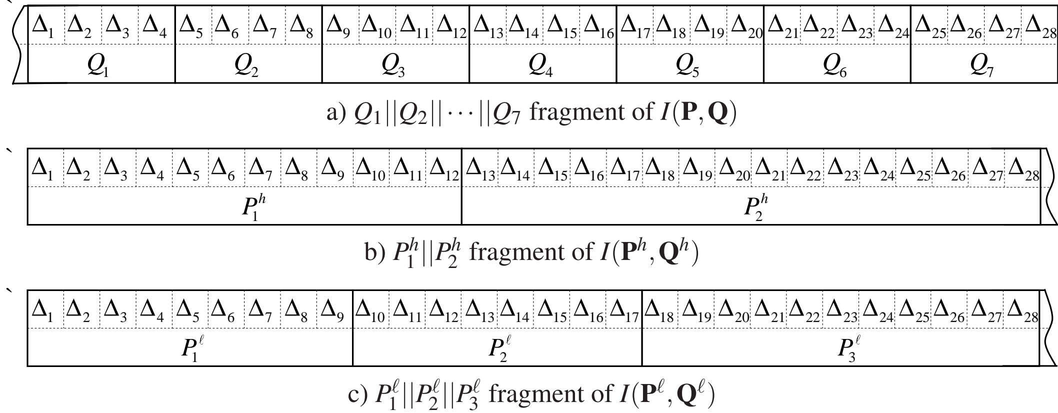


Figure 4: Coinciding fragments of the index buffers from the example shown in Fig. 3

3.3 Patches with multiple inner vertices

Consider the patch $P_1^h = Q_1||Q_2||Q_3$ and its corresponding simplified patch Q_1^h. As we simplify the mesh, the following sequence of replacements occurs: $P_1 \to Q_1$, $P_2 \to Q_2$, $P_3 \to Q_3$, $P_1^h \to Q_1^h$. To reduce the size of index dataset, we may choose not to store Q_1, Q_2, and Q_3; we directly replace $P_1||P_2||P_3 \to Q_1^h$ during rendering. Thus, we can reduce the size of index data at the expense of storing fewer approximations. In order to describe such process we have to extend our previous definition of a patch.

We define a *patch with multiple inner vertices* $P^*(v_i, v_j, \ldots, v_k)$ as a list of triangles, such that $S(v_i) \cup S(v_j) \cup \cdots \cup S(v_k) \subseteq P^*$. We construct the corresponding simplified patch Q^* by sequentially applying simplification operator $\mathscr{R}$ (or $\mathscr{H}$), which removes the inner vertices, e.g. $Q^* = \mathscr{R}(v_i)\mathscr{R}(v_j)\cdots\mathscr{R}(v_k)P^*(v_i, v_j, \ldots, v_k)$. Given the list of patches $\mathbf{P}^* = \{P_1^*, P_2^*, \ldots, P_n^*\}$ and its corresponding list of simplified patches $\mathbf{Q}^* = \{Q_1^*, Q_2^*, \ldots, Q_n^*\}$, we build the index buffer $I(\mathbf{P}^*, \mathbf{Q}^*)$ and render the mesh approximations as described in Sec. 2.

A patch with several inner vertices is simplified more aggressively than a patch with single inner vertex. Thus, the patches with multiple inner vertices are particularly useful for the overtesselated models that contain a lot of redundant coplanar triangles. Suppose the entire flat upper part of the mesh shown in Fig. 5a is included into a single patch P^*. Fig. 5c shows that we avoid simplifying the curved bottom part by simplifying P^* only, and reduce the size of index data. A planarity measure (e.g. *dual quadratic metrics* [GWH01]) may be used to select the patches that are suitable for having more than one inner vertex.

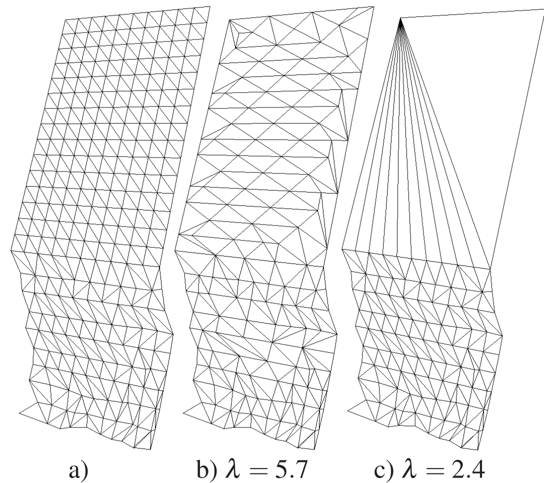


Figure 5: a) original mesh, b) simplification using single inner vertex per patch ($\mu = 0.25$), c) the entire flat upper part is simplified as a patch with several inner vertices

One method to build the patches with multiple inner vertices is the clustering algorithm; we simply discard

the patches P^h as described above. This can be done in run-time when there is not enough videomemory available in the system. Alternatively, a mesh partitioning algorithm may be used to directly obtain the patches with several inner vertices.

Generally, the patches with multiple inner vertices provide to the user a way to trade the number of mesh approximations in SWPM for the smaller index dataset.

4 NUMERICAL RESULTS

Rendering performance. One popular method for rendering of instantiated geometry is the *discrete LOD* method where several discrete mesh approximations are created using a simplification software, such as QSlim. Then each approximation is optimized for the efficient hardware processing (we use *NVTriStrip* package for this purpose). In run-time, the system selects the most appropriate mesh approximation to display for an object. We compare discrete LOD and SWPM in an application that renders 7000 diffusely lit instances of *t72* mesh on the system with Athlon64 2.2Ghz and GeForce 7800GTX. The application is not shader-bounded since we use relatively simple vertex shader (diffuse lighting only). Two 3-vectors (position and normal) are defined per vertex. In this test, the camera moves with varying velocity over the scene that contains regularly spaced mesh instances. The results are shown in Fig. 6. LOD selection function is the same for all methods. When geomorphing is not in use, performance of the discrete LOD (a) and SWPM (b) is the same because their rendering paths coincide. However, geomorphing causes a notable slowdown of the discrete LOD rendering (c) since it doubles the bandwidth required for the vertex data (note that its impact may be less serious for shader-bounded applications since geomorphing is arithmetically inexpensive). Geomorphing in the sliding window scheme (d) is not expensive due to the use of frame-to-frame coherence (as discussed in Sec. 2). This advantage disappears when the frames are highly incoherent. For example, a fast camera movement lasts for 8 seconds starting from 9th second of the timedemo; both methods (c) and (d) demonstrate similar performance during this period. The curve (e) shows performance of CPU-based progressive meshes implementation (as found e.g. in DirectX D3DX 9.0) that streams the list of indices from CPU to GPU each frame; the application generates approx. 500Mb of indices per second, which makes such scheme impractical.

Index dataset reduction. Table 1 demonstrates the results of dataset reduction techniques. The parameter λ in (9) relates the size of resulting SWPM index dataset to the number of triangles in the original mesh. In Table 1, the coefficient λ corresponds to the progressive mesh obtained using the basic sliding window algorithm; dataset reduction techniques decrease this value according to the percentages (a)–(d). The values (d) are obtained by discarding some simplified patches (as discussed in Sec. 3.3) from the progressive meshes obtained for the case (c); this normally produces the patches with approximately three inner vertices.

Following [BG01], we use the *average number of cache misses per triangle* (ACMR) as a measure of vertex cache usage efficiency. The clustering is an additional factor (in addition to vertex cache coherence), which is taken into account when the order of patches is determined. Thus, the clustering may cause an increase in ACMR in some cases (see Fig. 7).

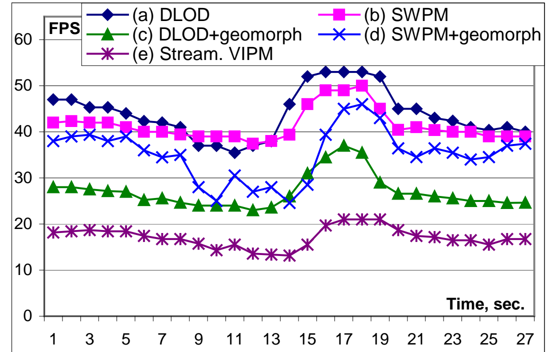


Figure 6: Rendering speed of several LOD algorithms in frames per second over time

Basic SWPM			Index dataset reduction, %			
Mesh	$\lvert T_0 \rvert$	λ	(a)	(b)	(c)	(d)
Regular	49920	5.4	39%	18%	41%	58%
TIN	77632	5.2	28%	16%	31%	51%
Hugo	16374	5.1	23%	18%	29%	51%
Bunny	69451	5.4	19%	15%	27%	49%
Cessna	7446	5.3	9%	19%	27%	53%
Cow	5804	5.5	11%	18%	26%	49%
t72	2450	3.8	12%	19%	29%	50%
Elf	1274	4.6	6%	20%	31%	57%

Table 1: Index dataset reduction via connectivity optimization (a), clustering (b), the combination of optimization and clustering (c), or the combination of optimization and the patches with three inner vertices per patch (d)

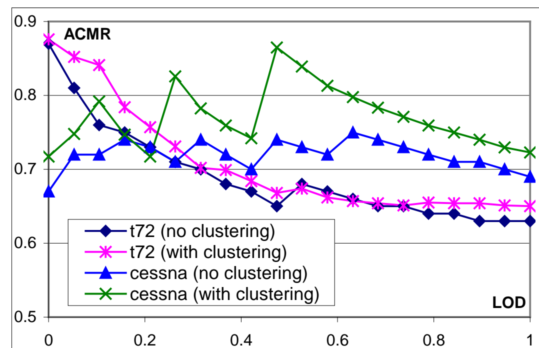


Figure 7: ACMR (vertex cache size is 16 entries) for all resolutions of t72 and cessna SWPM created with or without clustering

Simplification quality. We use *quadratic error metrics* [LRC*02, pp.133–134] and half-edge collapse operator to create SWPM. Fig. 8 shows *t72* mesh simplified with help of various algorithms. Generally, although SWPM is an *ad-hoc* algorithm for efficient rendering, it typically produces mesh approximations comparable with the results of conservative mesh simplification algorithms.

CONCLUSIONS

Proposed methods for datasets reduction extend applicability of SWPM algorithm for memory-limited applications. The use of SWPM may be an interesting extension to batch-processing view-dependent LOD schemes, such as [YSG05, CGG*05].

References

[BG01] BOGOMJAKOV A., GOTSMAN C.: Universal rendering sequences for transparent vertex caching of progressive meshes. In *Proceedings of Graphics Interface 2001* (2001), Watson B., Buchanan J. W., (Eds.), pp. 81–90.

[CGG*05] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: Batched multi triangulation. In *Proceedings IEEE Visualization* (2005), IEEE Computer Society Press, pp. 157–164.

[For01] FORSYTH T.: Comparison of vipm methods. In *Game Programming Gems 2* (2001), DeLoura M., (Ed.), Charles River Media, pp. 363–376.

[FS01] FRANC M., SKALA V.: Parallel triangular mesh decimation without sorting. In *SCCG'01: Proceedings of the 17th Spring conference on Computer graphics* (Washington, DC, USA, 2001), IEEE Computer Society, p. 22.

[GWH01] GARLAND M., WILLMOTT A., HECKBERT P. S.: Hierarchical face clustering on polygonal surfaces. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (New York, NY, USA, 2001), ACM Press, pp. 49–58.

[KT04] KOROTOV S., TURCHYN P.: Topology-driven progressive mesh construction for hardware-accelerated rendering. *Computer Graphics and Geometry (Internet) 6*, 3 (2004), 100–119.

[LRC*02] LUEBKE D., REDDY M., COHEN J., VARSHNEY A., WATSON B., HUEBNER R.: *Level of Detail for 3D Graphics*. Computer Graphics and Geometric Modeling. Morgan Kaufmann, 2002.

[NBS06] NEHAB D., BARCZAK J., SANDER P. V.: Triangle order optimization for graphics hardware computation culling. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2006), ACM Press, pp. 207–211.

[SP03] SHAFAE M., PAJAROLA R.: Dstrips: Dynamic triangle strips for real-time mesh simplification and rendering. In *Proceedings Pacific Graphics Conference* (2003).

[Ste01] STEWART A. J.: Tunneling for triangle strips in continuous level-of-detail meshes. In *Proceedings of Graphics Interface* (2001), Watson B., Buchanan J. W., (Eds.), pp. 91–100.

[SvK97] SNOEYINK J., VAN KREVELD M. J.: Linear-time reconstruction of delaunay triangulations with applications. In *ESA '97: Proceedings of the 5th Annual European Symposium on Algorithms* (London, UK, 1997), Springer-Verlag, pp. 459–471.

[YSG05] YOON S.-E., SALOMON B., GAYLE R.: Quick-vdr: Out-of-core view-dependent rendering of gigantic models. *IEEE Transactions on Visualization and Computer Graphics 11*, 4 (2005), 369–382. Member-Dinesh Manocha.

A THE BOUNDS FOR SWPM INDEX DATASETS

Consider the buffer $B_0 := I(\mathbf{P}_0, \mathbf{Q}_0)$, where the approximations $M_0, M_1, \dots, M_n$ are stored. First part of the buffer, which contains the non-simplified patches from the list $\mathbf{P}$, forms the set of triangles T_0 of the initial mesh $M_0 = (V_0, T_0)$, i.e. $T_0 = P_1 \cup P_2 \cup \dots \cup P_n$. Remaining part of the buffer, which contains the simplified patches from the list $\mathbf{Q}$, forms the triangles of the approximation $M_n = (V_n, T_n)$, where $T_n = Q_1 \cup Q_2 \cup \dots \cup Q_n$. Then, the number of triangles in the buffer B_0 is a sum of the number of triangles in T_0 and the number of triangles in T_n, i.e. $|B_0| = |T_0| + |T_n|$. In what follows we assume that the meshes are 2-manifolds without holes. Thus, the operator $\mathscr{H}$ (or $\mathscr{R}$) removes exactly two triangles from a patch, so

$$|T_n| = \sum_{i=1}^{n} |Q_i| = \sum_{i=1}^{n} (|P_i| - 2) = |T_0| - 2n, \tag{4}$$

where n is the number of patches. Hence, one have to maximize n in order to minimize $|B_0|$. It follows from the definition of a patch that maximization of n requires solving the *maximum independent set* problem on a mesh connectivity graph (the set of inner vertices is a *maximum independent set*). This is a classical *NP-hard* problem. There exist polynomial-time algorithms for finding its approximate solution with a good accuracy. One greedy algorithm for planar graphs is described in [SvK97]. The algorithm's lower bound for the size of independent set is $n > |V_0|/6$. It follows from the Euler's relation that

$$n > \frac{|V_0|}{6} > \frac{|T_0|}{12} \tag{5}$$

On the other hand, we note that each patch contains at least three triangles, so

$$n \leqslant \frac{|T_0|}{3}. \tag{6}$$

Combining (4)–(6), we obtain the bounds

$$\frac{1}{3}|T_0| \leqslant |T_n| < \frac{5}{6}|T_0|. \tag{7}$$

Using M_n as an initial mesh, we construct another SWPM buffer. Denote this buffer B_1. The simplest triangulation within the buffer B_1 may not be sufficient for the application. Thus, one has to continue constructing SWPM buffers $B_2, B_3, \dots, B_z$ until the required mesh complexity is achieved. Our main interest is estimating the parameter β in (2), which determines the amount of memory required for storing these buffers.

The approximation M_n serves as an initial mesh when we construct the buffer B_1. Let $M_m = (V_m, T_m)$ be the simplest

approximation of the buffer B_1. Following the same reasoning, which is used to obtain (7), we have

$$\frac{1}{3}|T_n| \leqslant |T_m| < \frac{5}{6}|T_n|.$$

In order to obtain the upper bound for $|T_m|$, for $|T_n|$ we take the upper bound provided by (7). Then

$$|T_m| < \frac{5}{6}|T_n| < \left(\frac{5}{6}\right)^2 |T_0|,$$

so

$$|B_1| = |T_n| + |T_m| < \frac{5}{6}|T_0| + \left(\frac{5}{6}\right)^2 |T_0|.$$

Generally,

$$|B_i| < \left(\frac{5}{6}\right)^i |T_0| + \left(\frac{5}{6}\right)^{i+1} |T_0|.$$

By summing up the inequalities for every $|B_i|$, we obtain upper bound for β

$$\beta < |T_0| + 2\left(\frac{5}{6}\right)|T_0| + \cdots + 2\left(\frac{5}{6}\right)^{z-1}|T_0| + \left(\frac{5}{6}\right)^z |T_0|.$$

We can interpret the right hand side as a converging geometric series. This is justified since in practice z is relatively big. Thus, we conclude that

$$\beta < 11|T_0|.$$

It is possible derive the lower bound for β using a similar reasoning. Finally, we arrive to the following inequalities

$$2|T_0| \leqslant \beta < 11|T_0|. \quad (8)$$

It is natural to express β as a size of initial mesh triangulation multiplied by a coefficient

$$\beta = |T_0|\lambda, \quad (9)$$

where $\lambda \in [2; 11)$ according to (8). We derive the bounds for the coefficient λ under relatively general assumptions about the input mesh. However, memory-limited applications require more precise estimation of λ for specific meshes. Thus, we have to derive an estimate for λ that exploits properties of a particular mesh.

A *degree* of a mesh vertex is the number of triangles incident to this vertex. Let $\overline{d}$ be a mean degree of the mesh vertices of *maximum independent set*. The value of $\overline{d}$ depends mainly on the structure of triangulation. We assume that

1. the choice of simplification operators and their parameters preserves the structure of initial triangulation, so $\overline{d}$ is approximately the same for any approximation M_i;
2. the size of a patch is approximately equal to the degree of its inner vertex, thus $\overline{d}$ is a mean patch size.

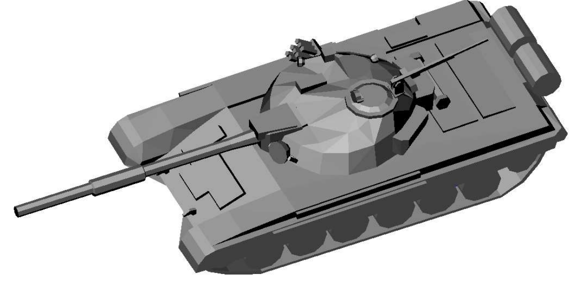
Under these assumptions, the number of patches is a ratio of total number of triangles to $\overline{d}$. Applying this relation to (4), it is easy to show that

$$|T_n| = |T_0|(1 - 2/\overline{d}),$$
$$|B_0| = |T_0| + |T_n| = |T_0|(2 - 2/\overline{d}),$$
$$|B_i| = |T_0|(2 - 2/\overline{d})(1 - 2/\overline{d})^i.$$

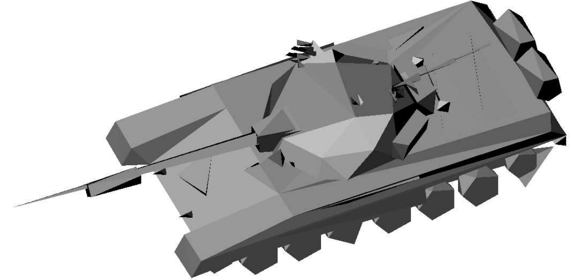
Again, β is expressed as a converging geometric series

$$\beta = |T_0|(\overline{d} - 1). \quad (10)$$

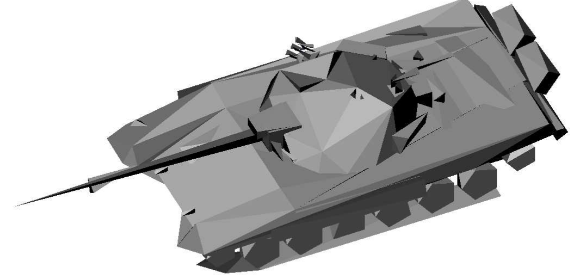
Thus, the size of resulting dataset is proportional to $\overline{d}$, which is a mesh-dependent parameter.



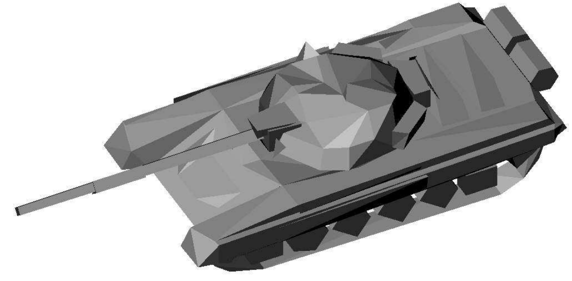
a) original mesh



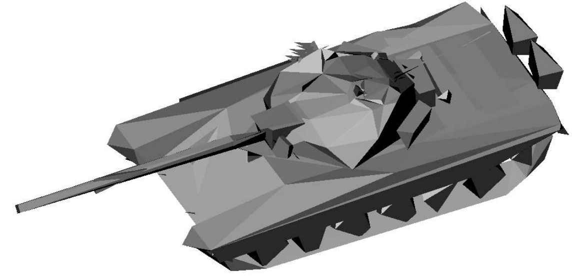
b) DirectX D3DX 9.0



c) VizUp 1.7



d) QSlim 2.0



e) SWPM ($\mu = 0.2$)

Figure 8: t72 mesh is simplified from 2450 to 640 triangles using different simplification algorithms. As discussed in Sec. 3, the parameter μ determines tradeoff between the size of SWPM index dataset and the simplification quality; relations between μ and RMS simplification error are shown in Fig. 1.