Distributed Triangle Mesh Processing

Daniela Cabiddu
CNR-IMATI
Via De Marini 6
16149, Genova, Italy
daniela.cabiddu@ge.imati.cnr.it

Marco Attene
CNR-IMATI
Via De Marini 6
16149, Genova, Italy
marco.attene@ge.imati.cnr.it

ABSTRACT

We propose a web-based system to remotely and distributedly process triangle meshes. Users can implement complex geometric procedures by composing simpler processing tools that, in their turn, can be provided by researchers who publish them as appropriate Web services. We defined an efficient geometric data transfer protocol in order to resolve the potential mesh delivery bottleneck caused by the transfer of large models to the various servers on typical long-distance connections with limited bandwidth. We have experimented our system on several large models and on diverse processing scenarios, and we have concluded that our transfer protocol significantly reduces the overall time needed to produce the result.

Keywords

distributed processing, compression

1 INTRODUCTION

Geometry processing is now a mature research area where new algorithms and processes are continuously produced on top of state-of-the-art, already complex previous works.

Researchers in this field often need reimplementing algorithms from paper descriptions but, due to the aforementioned level of complexity, this easily becomes a costly and error prone operation. Nonetheless, researchers need to invent new algorithms and to fairly compare them against previous works, and such a need called for approaches to share shape models and algorithms. Collaborative environments often need to run experiments in distributed labs, where hardware and software requirements must be satisfied by each local machine in order to rebuild source codes and install stand-alone applications.

We prove that the integration of web service technologies and workflow-based frameworks provides an effective solution to this problem. Our system makes it possible to use a standard Web browser to remotely run complex geometric algorithms by distributing the various sequential steps on different servers that expose state-of-the-art algorithms in form of Web services. Thanks to the system, experiments can be run from any location, with no need of any local installation or reimplementation of previous works. Since transferring large models to the various servers would represent a bottleneck in the process, we defined an efficient geometric data transfer protocol.

2 RELATED WORK

Running experiments is a fundamental activity in geometry processing research. A typical experiment in this area consists in considering an input data set, performing a sequence of operations on it, and analyzing the results. Sometimes a fixed sequence of operations is used to process a variety of data sets, whereas some other times the operation list is slightly changed while keeping the input constant.

In computer graphics and geometry processing, polygon meshes are the dominant representations for 3D objects, and diverse mesh processing software tools exist. Among them, MeshLab [CCR08] and OpenFlipper [MK12] allow to interactively edit a mesh, save the sequential list of executed operations and locally re-execute the workflow from their user interfaces. Pipelines can be shared in order to be rerun on different machines where the stand-alone applications need to be installed.

To get rid of any specific software, hardware, and operating system, Campen and colleagues published an online service called WebBSP [Cam10] which is able to remotely run a few specific geometric operations. The system is accessible from a standard web browser and the user is required to upload an input mesh; then, a single geometric algorithm must be selected from a set of available operations. The algorithm is actually run on the server and a link to download its output is sent to the user by email. The available operations are not customizable by users, only one of them can be run at each call, and the service is accessible only from the WebBSP graphical interface.

Geometric Web services were previously considered by Pitikakis [Pit10] with the objective of defining semantic requirements to guarantee their interoperability. Though in Pitikakis's work Web services are stacked into hardcoded sequences, users are not allowed to dynamically construct workflows, and geometric issues such as the evaluation of mesh qualities (necessary to support conditional tasks and loops) and the transmission of large models are not dealt with.

3 THE GEOMETRIC WORKFLOW SYSTEM

Our system have been designed with the objective of supporting computer graphics and geometry processing research activities. Specifically, it allows the user to build workflows to process and analyse 3D triangular meshes and efficiently share the experiments through the Internet. The "bricks" that constitute a workflow are geometry processing algorithms that read an input mesh and produce an output mesh. Any such algorithm can be used as long as it is available online and accessible as a web service, independently of the physical location of the host server. Our framework currently provides some "in-house" geometry processing algorithms, but the architecture is open and fully extensible by simply publishing a new algorithm as a web service and by communicating its URL to the system.

The framework architecture (see Figure 1) is organized in three layers, according to Hollingsworth WFM specifications [Hol95] and inspired to the already existing WFMs in life science areas [TS07]. The first layer includes a graphical user interface that allows building new workflows from scratch, uploading existing pipelines and invoking available ones. A workflow can either encode its input mesh (e.g. to allow replicating an experiment) or accept a different input (e.g. to perform a new experiment based on the same procedure). The second layer contains the workflow engine responsible of runtime execution, while the third layer includes the web services that wrap geometry processing tools.

3.1 The Workflow Engine

The workflow engine is the core of the system and orchestrates the invocation of the various web services involved. From the user interface it receives the specification of a geometry processing workflow and possibly the address of an input mesh to be downloaded from the Internet. When all the data is available, the workflow engine sequentially invokes the various algorithms and returns the URL of the eventual result to the user

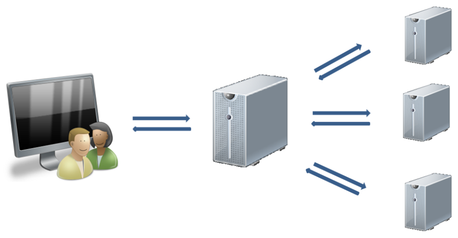


Figure 1: The three-layered system architecture. A graphical user interface allows to upload and run workflows. The workflow engine is responsible of workflow execution and manages the ordered list of tasks that need to be run. Geometric tasks are made available by distributed servers as web services.

interface. Note that in the long term the same operation can be provided by more than one server. The engine is responsible to select the most appropriate server providing each of them, by considering connection efficiency and load balancing. In order to enable the definition of non-trivial workflows, the engine is also able to manage the execution of conditional tasks and loops, and delegates the evaluation of the condition itself to specific web services.

3.2 The Web Services

To the best of our knowledge, no existing repository provides web service support to geometry processing computations. Thus, we have implemented some of them (see Table 1), and each can be considered as a black box able to run a simple operation on a 3D triangular mesh using possible input parameters, store the output on the server where it is located and make the output available by returning its address.

Note that a single server (i.e. a provider) can expose a plurality of web services implementing a variety of algorithms. Also, in order to support execution of loops and conditional tasks, appropriate *boolean* web services must be provided. For example, if it is necessary to invoke two different algorithms depending on whether the mesh is manifold or not, a "isManifold" web service is invoked that reads the mesh and returns a boolean value to be used when checking the condition. Examples of supported checks are shown in Table 2.

Web Service	**Parameters**
Remove Smallest Components	NONE
Remove Degenerate Triangles	NONE
Add Noise	% rel. bb diagonal
Laplacian Smoothing	# iterations
Hole Filling	NONE

Table 1: Web services that perform geometry processing tasks.

Web Service	Quality type
Check # vertices	**Integer**
Check # edges	**Integer**
Check # triangles	**Integer**
Check manifoldness	**Boolean**
Check orientation	**Boolean**
Check orientability	**Boolean**
Check minimum triangle angle	**Double**
Check maximum triangle angle	**Double**
Check average normal instability	**Double**

Table 2: Web services able to check mesh qualities.

4 MESH TRANSFER PROTOCOL

To support the idea of including web services provided by third parties, and to allow input models to be stored on remote servers, we require that web services are designed to receive the URL of the input mesh and to download it locally; also, after execution of the algorithm, the output must be made accessible through a standard URL to be returned to the calling service. Not surprisingly, we have observed that the transfer of large-size meshes from a server to another according to the aforementioned protocol constitutes a bottleneck in the workflow execution, in particular when slow connections are involved. Mesh compression techniques can be used to reduce the input size, but they do not solve the intrinsic problem. In order to improve the transfer speed and thus efficiently support the processing of large meshes, we designed a mesh transfer protocol inspired on the prediction/correction metaphor used in data compression. The general idea of prediction/correction works as follows. A sender S needs to transmit a data set to a receiver R, but instead of transmitting the whole data set at once, S sends a piece of data only, let it be d_0. Then, R tries to *predict* what the next piece of data d_1 will be based on the previously received information d_0. At the same time S does exactly the same prediction and, instead of sending the next piece of data, sends the difference between the prediction and the actual data to be sent, that is $c_1 = d_1 - d_0$. Thus, R can calculate d_1 by *correcting* the prediction using c_1. The benefits of all this machinery become evident when the predictions are accurate enough: in this case the corrections to be sent are small if compared with the original data and thus can be encoded with fewer bits. A typical example in geometry processing is the so-called “parallelogram rule” [TG98] used for mesh compression.

We have observed that there are numerous mesh processing algorithms that simply transform an input mesh into an output by computing and applying local or global modifications. Furthermore, in many cases modifications can be only local (e.g. sharp feature restoration), may involve the geometry only while keeping the connectivity unaltered (e.g. most mesh deformation algorithms), or may modify both geometry and connectivity while minimally changing the overall shape (e.g. remeshing). In all these cases it is possible to predict the result by assuming that it will be identical to the input, and it is reasonable to expect that the corrections to be transmitted can be more compactly encoded than the explicit result of the process.

4.1 Concurrent Mesh Transfer

The aforementioned observation can be exploited in our setting. Figure 2 shows an example of execution of a simple workflow composed by three tasks. The engine reads the whole workflow and, for each of the three tasks requested, looks for a server exposing an appropriate web service (i.e. a web service which implements the task). Then, the engine sends the address of the input mesh to all the servers that have been identified so that they can download it (Figure 2a). Right after having sent the address, the engine triggers the first web service (Figure 2b) to locally run the algorithm. Such an algorithm produces both the output mesh and the list of changes applied on the input to obtain the result (e.g. vertices/edges/triangles that have been added, removed or modified). Both the output mesh and the list of changes (i.e. the correction) are compressed and made available through two URLs which are communicated to the workflow engine. In its turn, the engine forwards this information to the next two web services to be triggered, so that both of them can download the compressed correction from the first server, and can reproduce the output of the first step by decoding and applying the correction to the mesh that was previously downloaded. At this point the engine triggers the second web service (Figure 2c) that follows the same protocol by running the algorithm and publishing the URLs of the output and the correction. Finally, the third web service corrects its prediction, runs its task and returns the URL of the final result (Figure 2d).

In a more general setting, the protocol works as follows. Through the user interface, the user selects/sends a workflow and possibly the URL of an input mesh to the workflow engine. The engine analyses the workflow, locates the most appropriate servers hosting the involved web services, and sends in parallel to each of them the address of the input mesh. Each server is triggered to download the input model and save it locally. At the first step of the experiment, the workflow engine triggers the suitable web service that runs the algorithm, produces the result, and locally stores the output mesh and the correction file (both compressed). Their URLs are returned to the workflow engine that forwards them to all the subsequent servers involved in the workflow. Each server downloads the correction and applies it to the mesh it already has in memory in order to update the local copy of the model. Then, the workflow engine

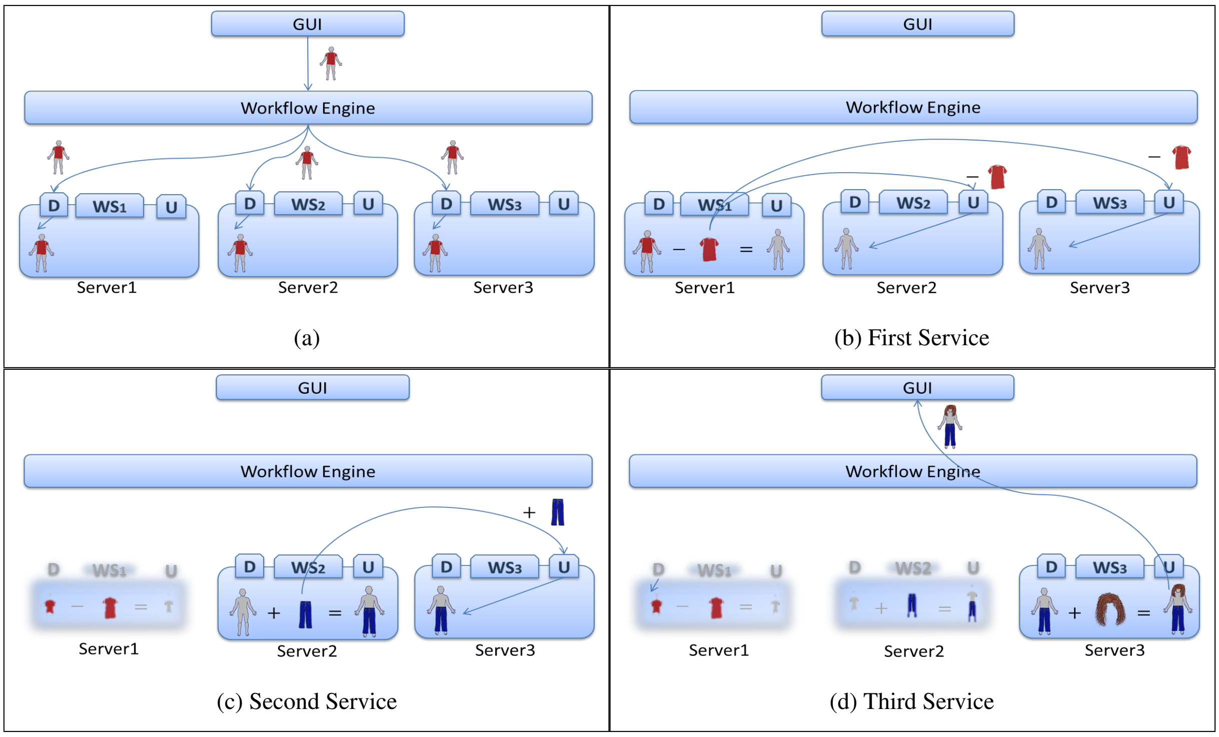


Figure 2: Mesh Transfer Protocol Example. Three servers are involved into the workflow execution. Each of them exposes a web service to support a geometry processing algorithm and two modules able to download (D) meshes and update (U) the previously downloaded mesh by applying the corrections. (a) The engine shares in parallel the address of the input mesh with all the involved servers that proceed with the download. (b) The first service runs the task, produces the corrections and returns the corresponding address to the engine that shares it in parallel to the following involved servers. Both download the file and correct the prediction. (c) The second service is invoked, runs the task and makes the correction available, so that the third involved server can download it and update its local copy of the mesh. (d) The engine triggers the third service that runs the algorithm and makes available the modified output mesh so that it can be directly downloaded by the user.

triggers the next service for which an up-to-date copy of the mesh is readily available on its local server. At the end of the workflow execution, the engine receives the address of the output produced by the last invoked web service and returns it to the user interface, so that the user can proceed with the download.

The Workflow Engine maintains a list of *active tasks* whose corresponding Web services need to receive the corrections and update their local copy of the mesh. Initially all the tasks involved in the workflow are *active*. Upon termination, a “regular” task (i.e. neither an “IF” nor a “WHILE”) is removed from the list meaning that it is no longer involved in the workflow. The same happens when an “IF” task is encountered with true condition. Conversely, after an “IF” or a “WHILE” whose condition is false, all the tasks constituting the body are removed from the list. After a “WHILE” with true condition, an additional copy of all the tasks in its body is added to the list.

In this scenario, the entire input mesh is broadcasted only once at the beginning of the process, whereas the final result is transmitted only once at the end. Inbetween, only the corrections are broadcasted to the subsequent servers. Thus, when the corrections are actually smaller than the partial results, this procedure produces significant benefits. In any case, each web service produces both the correction and the actual result so, should the former be larger than the latter, the subsequent web services can directly download the output instead of the corrections. Thus, our mesh transfer protocol improves the overall performances when the aforementioned conditions hold, while no degradation is introduced otherwise.

4.2 Representation of the correction

A triangle mesh can be defined by an abstract simplicial complex that specifies its connectivity endowed with a set of vertex positions that uniquely identify its geometric realization [Att13]. An algorithm that modifies an input mesh can act on its geometry only (e.g. by changing the position of the vertices), on its connectivity only (e.g. by triangulating boundary loops), or on both. Typically, such an algorithm includes an analysis part that performs the calculations to derive what to add, modify or remove, and an editing part that applies such changes to the mesh. Depending on the algorithm these two parts may be not necessarily sequential, but the editing operations can always be tracked and are sufficient to reconstruct the result. In the worst case where the mesh is completely rebuilt from scratch, this list is a sequence of “add vertex” and “add triangle” operations preceded by a “clear all” (see Table 3).

Simplex	Operation	Encoding
Triangles	Add	T A v_1 v_2 v_3
	Remove	T *id* R
	Split (center)	T *id* SPC
	Split (generic point)	T *id* SP x y z
Vertices	Add	V A x y z
	Move	V *id* M x y z
	Move All	V MA x_1 y_1 z_1 x_2 y_2 z_2 ...
Edges	Add	E A v_1 v_2
	Swap	E *id* SW
	Collapse	E *id* C
	Split (midpoint)	E *id* SP
	Split (generic point)	E *id* SP x y z
All	Clear All	CLEAR

Table 3: Supported editing operations. *Italic* labels in the encodings indicate either simplex identifiers (i.e. indexes) or vertex coordinates.

In our setting, each web service runs a geometry processing algorithm, keeps track of the editing operations, and saves them along with the final result. To do this, the algorithm itself must be enriched with proper code to stream such operations into the correction file. In our current implementation we support atomic operations to encode the insertion, removal and modification of single simplexes of any order (i.e. vertices, edges and triangles). Each operation is identified by a unique opcode, while each simplex is uniquely identified by an integer ID. Thus, to represent an “edge swap” we need an opcode representing the swap operation and an integer identifying the edge to be swapped. Besides such atomic operations, we include some derived functionalities that group atomic changes for the most diffused editing operations. In many cases this allows to further save storage space (and thus transmission time). For example, let us suppose that we need to subdivide a triangle into three subtriangles by inserting a new vertex: in this case we would need to encode a “remove triangle” (1 opcode + 1 ID), a “create vertex” (1 opcode + 3 coordinates), and three “create triangle” (3 opcode + 9 IDs for the vertices) operations. Conversely, if we include a single “split triangle” operation in our set, we can simply use its opcode endowed with the identifier of the triangle to be split and the coordinates of the splitting vertex. Additional operations are defined both at the level of the connected components (e.g. removal, translation, rotation, ...) which, just as the simplexes, are identified using IDs, and at the level of the whole mesh.

When an algorithm terminates, the produced sequence of operations is further compressed through arithmetic coding to minimize redundancy [Sai02]. The application of the correction by the subsequent web services requires less computational efforts and time than the rerun of the algorithm because of the fact that its analysis part and the operation precondition checks are not needed anymore.

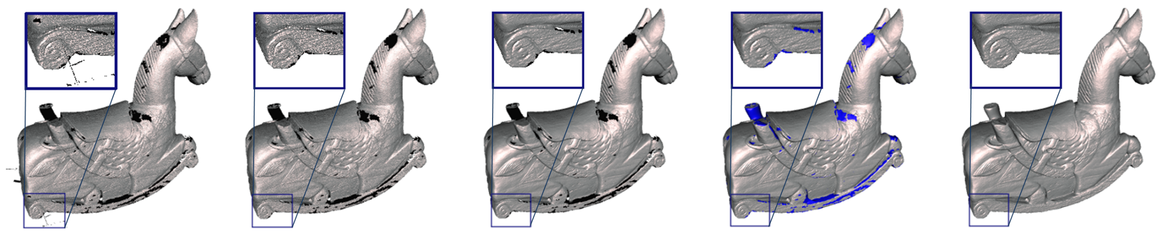
Notice that sometimes a careful analysis of the algorithm at hand allows to avoid streaming all the operations. For example, let us consider an algorithm that performs N iterations of Laplacian smoothing on a mesh with V vertices. At each iteration all the vertices are moved to the center of mass of their neighbors, thus by a naive approach we would stream $N * V$ vertex shifts. A more clever implementation, however, can simply stream the eventual global shift once for each vertex, thus reducing the size by a factor of N.

5 RESULTS AND DISCUSSION

For the sake of experimentation, the proposed Workflow Management System has been deployed on a standard server running Windows 7, whereas web services implementing atomic tasks and check mesh qualities have been deployed on different machines to constitute a distributed environment. However, since all the servers involved in our experiments were in the same lab with a gigabit network connection, we needed to simulate a long-distance network by artificially limiting the transfer bandwidth to 5 Mbps.

Then, to test such a system we defined multiple processing workflows involving the available web services. The dataset has been constructed by selecting some of the most complex meshes currently stored within the Digital Shape Workbench (see Table 4).

As an example, one of our test workflows is depicted in Figure 3. The input model (Figure 3a) has 160 spurious disconnected components that are removed by the first web service (Figure 3b). Then one iteration of laplacian smoothing is applied (Figure 3c) by the second web service to enhance the surface, while its 404 holes are patched by the third web service implementing Liepa’s hole filling algorithm [Lie03] (Figure 3d). Finally, degenerate triangles are removed by the fourth web service (Figure 3e). This test gives a first idea of the benefits provided by our transfer protocol: for example, consider that all the simplexes removed in the first step ($\approx 3K$ vertices, $\approx 7.5K$ edges and $\approx 4.5K$ triangles) could be encoded within a 11 KB correction



(a) (b) (c) (d) (e)

Figure 3: Local mesh repairing [ACK13]: a typical example of geometry processing workflow. (a) The raw model. (b) Smallest components removed. (c) Laplacian smooth applied. (d) Holes filled. (e) Degenerate triangles removed.

Mesh	Isidore	Nicolo	Neptune	Ramesses	Raptor	Dancers
Vertices	1.071.671	945.924	1.321.838	775.712	1.000.080	703.207
Triangles	2.128.494	1.886.968	2.643.684	1.537.462	2.000.000	1.399.805
Components	161	103	1	308	51	1
Boundaries	404	157	0	824	0	105

Table 4: Dataset extracted from the Digital Shape Workbench.

file, whereas the compressed output mesh file size was 20.5 MB.

The same workflow was run on all the other meshes in our dataset to better evaluate the performance gain achievable thanks to our concurrent mesh transfer protocol. Table 5 reports the size of the output mesh and the size of the correction file after each operation (both after compression) whereas Table 6 shows the total time spent by the workflow along with a more detailed timing for each single phase. In both the tables tasks are indicated by acronyms as follows: Removal of Smallest Components (RSC), Laplacian Smoothing (LS), Hole Filling (HF), and Removal of Degenerate Triangles (RDT).

As expected, the corrections related to tasks that locally modify the model (eg. RSC, HF, RDT) are significantly smaller than the whole output mesh by several orders of magnitude; corrections regarding more "global" tasks (eg. LS) are also smaller than the output mesh, although in this latter case the correction file is just two/three times smaller than the whole output. Nevertheless, these results confirm that the proposed concurrent mesh transfer protocol provides significant benefits when the single steps produce mainly little or local mesh modifications.

For each mesh in our dataset, Table 6 reports the time spent by each algorithm to process the mesh (columns RSC, LS, HF, RDT), the time needed to transfer the correction file to the subsequent web service (columns $T_1 \dots T_3$), and the time spent to update the mesh by applying the correction (columns $U_1 \dots U_3$). For the sake of comparison, below each pair (T_i, U_i) we also included the time spent by transferring the whole compressed result instead of the correction file, and the overall relative gain achieved by our protocol is reported in the last column. It is worth noticing that, in all our test cases, the sum of the transfer and update times is smaller than the time needed to transfer the whole mesh, with a significant difference when the latter was produced by applying little local modifications on the input. Clearly, the additional instructions introduced in the geometry processing algorithms to stream out the corrections should be considered for a fair comparison, but we have verified that such an overhead is definitely negligible with respect to the overall processing time of each algorithm, and therefore has not been reported in Table 6.

As an additional example, Table 7 shows results concerning the execution of a workflow involving a "while" loop where Laplacian smoothing is applied as long as the average normal instability exceeds a threshold value. Times are related to a single workflow step (eg. a web service execution or a data transfer), while numbers between parenthesis indicate how many times the corresponding step is run during loop execution. Similar tests have been done involving the execution of conditional tasks. It is worth noting that, in our test cases, the mesh transfer protocol reduces the execution time if the mesh satisfies the required mesh quality and therefore the subsequent web service is invoked. No advantage and no degradation are introduced when the operation precondition does not hold due to the fact that no output file is transferred from one server to the others.

Mesh	RSC	LS	HF	RDT
Isidore	20.573	23.333	23.717	25.497
	11	9.433	154	2
Nicolo	19.498	21.447	20.601	20.171
	3	9.296	48	2
Neptune	39.881	40.131	39.891	39.937
	1	15.237	1	1
Ramesses	17.484	19.544	19.934	19.802
	3	8.754	149	3
Raptor	14.465	15.621	15.552	15.441
	688	10.195	1	1
Dancers	16.457	18.037	18.325	18.116
	1	7.220	80	1

Table 5: Output sizes (in KB). For each mesh and for each task, the first line shows the size of the compressed output mesh, while the second line reports the size of the compressed correction. Acronyms indicate Removal of Smallest Components (RSC), Laplacian Smoothing (LS), Hole Filling (HF), and Removal of Degenerate Triangles (RDT).

To summarize, our tests show that the concurrent mesh transfer protocol considerably reduces the amount of data transferred among the servers, and thus the total elaboration time.

As previously mentioned, if the output mesh produced by a web service is smaller than the correction file, then such an output is forwarded to the subsequent servers that simply replace their local copy of the mesh. In an ideal system, the time needed to apply the correction should be taken into account as well before choosing whether to forward the whole mesh or the correction file. Unfortunately the update time depends on too many factors (i.e. architecture of the host server, current workload, ...) to be accurately guessed, but we argue that this is not a real issue because this case appears to be unlikely to happen in practice.

Regarding the tasks that globally modify the mesh, such as the Laplacian Smoothing, we suspect that a clever analysis combined with coordinate quantization can provide a much more compact representation of the correction file.

Finally, we recognize that some algorithms that completely rebuild the mesh (e.g. from an intermediate representation such as in [Ju04]) can hardly be reproduced in a compact way through our current set of local editing operations. In these cases our mesh transfer protocol does not provide any advantage and the whole output mesh should be transmitted.

6 CONCLUSION

We proposed a workflow-based framework to support collaborative research in geometry processing. It allows scientists to remotely run geometric algorithms provided by other researchers as Web services and to combine them in order to create geometric workflows to be executed on any appropriate input mesh.

By distributing the workload, our system can count on considerable computational resources and can be easily extended if necessary, while the potential mesh delivery bottleneck has been resolved by our concurrent data transfer protocol.

To further simplify the work of potential contributors, we are currently investigating innovative techniques to automatically compute the list of editing operations by simply comparing the input and the output of each Web service [DP13] introducing no degradation in the system performance. Such a comparison module would automatically derive the correction, without the need for the contributor to turn the algorithm into an appropriate version able to stream the operations.

7 ACKNOWLEDGMENTS

This work has been partly supported by the PO CRO Fondo Sociale Europeo Regione Liguria 2007-2013 Asse IV “Capitale Umano” Ob. Specifico I/6 through the project “Tecniche di visualizzazione avanzata di immagini e dati 3D in ambito biomedicale”, and by the European Commission under grant agreement 262044 “VISIONAIR”. The authors are grateful to all the colleagues at IMATI and Softeco Sismat Srl for the helpful discussions.

8 REFERENCES

[ACK13] M. Attene, M. Campen, and L. Kobbelt. Polygon mesh repairing: An application perspective. *ACM Computing Surveys (CSUR)*, 45(2):15:1–15:33, March 2013.

[Att13] M. Attene. Surface mesh qualities. In *GRAPP/IVAPP*, pages 79–85, 2013.

[Cam10] M. Campen. Webbsp 0.3 beta. http://www.graphics.rwth-aachen.de/webbsp, 2010.

[CCR08] P. Cignoni, M. Corsini, and G. Ranzuglia. Meshlab: an open-source 3d mesh processing system. *ERCIM News*, (73):45–46, April 2008.

[DP13] J. D. Denning and F. Pellacini. Meshgit: Diffing and merging meshes for polygonal modeling. *ACM Trans. Graph.*, 32(4):35:1–35:10, July 2013.

[Hol95] D. Hollingsworth. Workflow management coalition - the workflow reference model. Technical report, Workflow Management Coalition, January 1995.

[Ju04] T. Ju. Robust repair of polygonal models. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 23(3):888–895, 2004.

Mesh	**IB**	**RSC**	**T_1**	**U_1**	**LS**	**T_2**	**U_2**	**HF**	**T_3**	**U_3**	**RDT**	**Total**	**Benefits**
Isidore	33,0	7,7	0,0	5,8	12,4	15,1	7,1	8,4	0,2	6,0	13,8	109,5	67%
	33,0	7,7	32,9		12,4	37,3		8,4	37,9		13,8	183,4	
Nicolo	31,2	6,5	0,0	4,8	10,5	14,9	6,1	7,5	0,1	4,9	11,5	98,0	69%
	31,2	6,5	31,2		10,5	34,3		7,5	33,0		11,5	165,7	
Neptune	63,8	13,0	0,0	0,0	18,6	24,4	11,0	12,6	0,0	0,0	14,4	157,8	99%
	63,8	13,0	63,8		18,6	64,2		12,6	63,8		14,4	314,2	
Ramesses	28,0	6,7	0,0	4,3	9,6	14,0	5,4	7,0	0,2	4,5	10,3	90,0	70%
	28,0	6,7	28,0		9,6	31,3		7,0	31,9		10,3	152,8	
Raptor	26,7	7,7	1,1	5,6	9,6	16,3	5,8	6,0	0,0	0,0	9,3	88,1	50%
	26,7	7,7	23,1		9,6	25,0		6,0	24,9		9,3	132,3	
Dancers	26,3	4,9	0,0	0,0	7,3	11,6	4,3	5,2	0,1	3,6	7,0	70,3	92%
	26,3	4,9	26,3		7,3	28,9		5,2	29,3		7,0	135,2	

Table 6: Elaboration times (in seconds). Acronyms indicate Input Broadcasting (IB), Removal of Smallest Components (RSC), Laplacian Smoothing (LS), Hole Filling (HF), and Removal of Degenerate Triangles (RDT). Cells labelled by $\mathbf{T}_i$ indicate the time needed to transfer the correction file. Cells labelled by $\mathbf{U}_i$ indicate the time needed to update the mesh by applying the correction.

Mesh	**IB**	**C_AVG_NI**	**LS**	**T_N**	**U_N**	**Total**	**Benefits**
Isidore	33,0	6,3 (4)	12,7 (3)	20,1 (3)	7,1 (3)	178,1	26%
	33,0	6,3 (4)	12,7 (3)	42,8 (3)		224,8	
Nicolo	31,2	5,4 (4)	10,7 (3)	17,8 (3)	6,3 (3)	157,4	22%
	31,2	5,4 (4)	10,7 (3)	35,7 (3)		192,1	
Neptune	63,8	11,2 (5)	19,2 (4)	24,8 (4)	11,7 (4)	342,7	32%
	63,8	11,2 (5)	19,2 (4)	64,3 (4)		453,6	
Ramesses	28,0	5,4 (5)	9,8 (4)	14,7 (4)	5,7 (4)	175,6	28%
	28,0	5,4 (5)	9,8 (4)	32,7 (4)		224,8	
Raptor	26,7	6,3 (5)	11,5 (4)	19,3 (4)	7,1 (4)	209,5	5%
	26,7	6,3 (5)	11,5 (4)	29,1 (4)		220,7	
Dancers	26,3	4,5 (4)	8,5 (3)	13,4 (3)	5,3 (3)	125,8	28%
	26,3	4,5 (4)	8,5 (3)	30,5 (3)		161,3	

Table 7: Elaboration times (in seconds). The test has been run on our dataset after the artificially addition of noise. Acronyms indicate Input Broadcasting (IB), Laplacian Smoothing (LS) and Check Average Normal Instability (C_AVG_NI). Cells labelled by $\mathbf{T}_N$ indicate the time needed to transfer the correction file. Cells labelled by $\mathbf{U}_N$ indicate the time needed to update the mesh by applying the correction. Each elaboration time is related to a single workflow step. The number between parenthesis indicates how many times the workflow step is run during loop execution. Note that the precondition check is run once more, the last time it returns false and breaks the loop.

[Lie03] P. Liepa. Filling holes in meshes. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '03, pages 200–205, Aachen, Germany, 2003.

[MK12] J. Möbius and L. Kobbelt. Openflipper: An open source geometry processing and rendering framework. In *Curves and Surfaces*, volume 6920 of *Lecture Notes in Computer Science*, pages 488–500. Springer Berlin / Heidelberg, 2012.

[Pit10] M. Pitikakis. *A Semantic Based Approach For Knowledge Management, Discovery and Service Composition Applied To 3D Scientif Objects*. PhD thesis, University of Thessaly, School of Engineering, Department of Computer and Communication Engineering, 2010.

[Sai02] A. Said. Introduction to arithmetic coding - theory and practice. In *Lossless Compression Handbook*, pages 101–152. Academic Press, 2002.

[TG98] C. Touma and C. Gotsman. Triangle mesh compression. In *Graphics Interface*, pages 26–34. Canadian Human-Computer Communications Society, 1998.

[TS07] A. Tiwari and A. K. T. Sekhar. Workflow based framework for life science informatics. *Computational Biology and Chemistry*, (5-6):305–319, 2007.