Salient Clustering for View-dependent Multiresolution Rendering

Rodrigo Barni, João Comba
Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre, Brazil

Amitabh Varshney
Department of Computer Science and UMIACS
University of Maryland
College Park, United States of America

Figure 1. Saliency-guided view-dependent rendering: saliency (a) is used to perform clustering (b). Viewing position and saliency are used to estimate cluster importance (c), which is passed to a fragment program that calculates the right resolution for each cluster inside the GPU. Final rendering (d).

***Abstract*—Perceptual information is quickly gaining importance in mesh representation, analysis and rendering. User studies, eye tracking and other techniques are able to provide ever more useful insights for many user-centric systems, which form the bulk of computer graphics applications. In this work we build upon the concept of Mesh Saliency — an automatic measure of visual importance for triangle meshes based on models of low-level human visual attention — applying it to the problem of mesh segmentation and view-dependent rendering. We introduce a technique for segmentation that partitions an object into a set of face clusters, each encompassing a group of locally interesting features; Mesh Saliency is incorporated in a propagative mesh clustering framework, guiding cluster seed selection and triangle propagation costs and leading to a convergence of face clusters around perceptually important features. We compare our technique with different fully automatic segmentation algorithms, showing that it provides similar or better segmentation without the need for user input. We illustrate application of our clustering results through a saliency-guided view-dependent rendering system, achieving significant framerate increases with little loss of visual detail.**

***Keywords*-saliency; segmentation; multiresolution;**

I. Introduction

Perceptual considerations are becoming increasingly important in mesh processing, analysis and display techniques [1]. User-centric systems form the bulk of computer graphics applications, and displaying results that are visually appealing while maintaining interactivity for datasets that become larger and more detailed is proving to be a difficult task.

Many techniques have been developed to tackle such a problem; among them, mesh simplification and segmentation proved to be particularly useful in reducing the overall size of datasets to be processed or displayed [2]. Limitations exist, though: greatly simplified meshes may lose features of interest, and segmentation may group faces into clusters that do not represent interesting regions of an object. If regions of high interest in a particular dataset could be reliably identified, these processes could be greatly improved.

Mesh saliency [3] provides exactly such information. Using multi-scale curvature analysis based on a center-surround operator, it reliably separates what most observers consider to be interesting regions in an entirely automated process. In this work we build upon the concept of mesh saliency, integrating it into a mesh segmentation framework; results are illustrated through mesh simplification and view-dependent rendering applications. The main contributions of this work are illustrated in Figure 1 and can be summarized as follows:

- **Salient clustering**: Saliency is used to determine cluster starting points and propagation, joining triangles into meaningful groups, each containing a set of locally interesting features. Segmentation results are as good as or better than comparable techniques, and are particularly suitable for perceptually-guided processes, such as view-dependent rendering.
- **Multi-resolution salient simplification**: Mesh simplification is achieved through saliency-guided QSlim [4], weighted by a cluster-localized saliency map to preserve interesting cluster-level features. A multiresolution representation of the object is created, where

1530-1834/09 $25.00 © 2009 IEEE
DOI 10.1109/SIBGRAPI.2009.34

regions of high cluster-level importance are preserved.

- **Saliency-guided view-dependent rendering**: Each cluster is analyzed independently based on factors such as saliency and expected screen-space area. Resolution is preserved for clusters with high estimated saliency and visibility. A GPU vertex contraction algorithm computes updated vertex index lists in real time, thus improving framerates while maintaining visual fidelity.

II. Related Work

Sullivan *et al.*'s report on perceptually adaptive graphics [1] discusses how perceptual information is used to evaluate and improve many visual applications, showing that solutions where human perception is taken into account provide greatly improved visual results. Howlett *et al.* have conducted eye-tracking experiments to identify and predict feature saliency for three-dimensional objects [5], and show that higher visual fidelity can be attained on low level-of-detail optimizations when using saliency as guiding element.

The work of [6] describes a method for partial matching of triangular meshes, through identification of unique salient features over a set of low level surface descriptors; it does not assign perceptual importance to different regions of an object. The concept of mesh saliency was first introduced on [3], as a measure of regional importance based on multi-scale curvature estimations. This technique is based on computational models of human visual perception and is able to reliably identify regions that most observers would find of high visual importance. In [7] a user-guided simplification method is introduced, allowing preservation of key perceptual features on very low level-of-detail optimizations through user selection of interesting areas. No automatic measure of perceptual importance is proposed.

Though methodologies vary — from adaptations of vector quantization algorithms [8] to Dijkstra-like propagation tuned for best rectangular fit [9] — most clustering algorithms are guided either by simple geometric information, such as distance and normal deviation [10] or require user input to define interesting areas [9]. Exceptions exist, though: [11] introduces a hierarchical segmentation technique based on fuzzy clustering and spectral cuts. It results, generally, in cuts at regions of deep concavities. A technique for mesh decomposition based on expanding spheres around each vertex is presented in [12]; while shape analysis is performed at multiple scales, this is not translated into a measure of perceptual importance. The segmentation technique introduced in [13] uses visually-salient spectral cuts, but saliency is defined only for entire mesh regions and not on the vertex level. Intuitively meaningful results can be obtained through this method, but no relative perceptual importance is assigned to regions, compromising its use in perceptually-guided multiresolution applications. Katz *et al.* [14] introduces a method for segmentation through feature point and core extraction. Feature points are defined as those resting on the tips of prominent components of a model, and not through perceptual importance estimation.

III. Salient Clustering

Mesh segmentation consists of partitioning a triangular mesh into a set of face clusters, each comprised of a subset of the original triangles. Faces assigned to each cluster share common properties, such as spatial proximity and orientation. Salient clustering, as introduced here, is the process of segmenting a triangle mesh into a group of clusters, where each cluster encompasses a local set of interesting features as defined by mesh saliency. This section describes a segmentation approach that reliably detects and groups triangles belonging to interesting features by integrating vertex-level mesh saliency information into a propagative clustering framework.

A. Mesh Saliency

Inspired by the concept of saliency for 2D images presented in [15], mesh saliency was introduced in [3] as a vertex-level measure of regional importance. This method effectively separates important regions from the surrounding context, providing a reliable automatic measure of perceptual importance for triangle meshes. Mesh saliency applies the center-surround operation presented in [15] to triangle meshes, substituting local mesh curvature for image color. Vertex-level mesh saliency $\mathscr{S}_i(v)$ for a given vertex v is given by:

$$\mathscr{S}_i(v) = |G(\mathscr{C}(v),\sigma_i) - G(\mathscr{C}(v),2\sigma_i)| \qquad (1)$$

where $G(\mathscr{C}(v),\sigma)$ corresponds to a Gaussian-weighted average of the mean curvature $\mathscr{C}(v)$, and σ_i corresponds to the standard deviation of the Gaussian filter at scale i (Figure 2). For all results presented in this work, we have used five scales $\sigma_i \in \{2\varepsilon, 3\varepsilon, 4\varepsilon, 5\varepsilon, 6\varepsilon\}$, where ε is defined as 0.3%

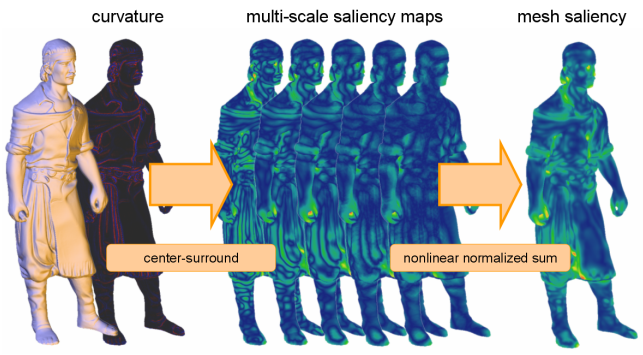


Figure 2. Saliency computation uses a center-surround mechanism to combine curvature information obtained at different scales.

of the length of the diagonal of the object's bounding box. Note that σ_i only determines the size of the neighborhood being considered — for a given saliency computation, all measurements are extracted from the same mesh.

B. Cluster Determination

Salient clustering is a variation of the chartification algorithm presented in [10], modified to consider mesh saliency information during seed initialization and cluster computation. Our method is composed of a seed initialization phase, as well as iterative cluster growth and re-seeding phases which are executed until convergence is reached.

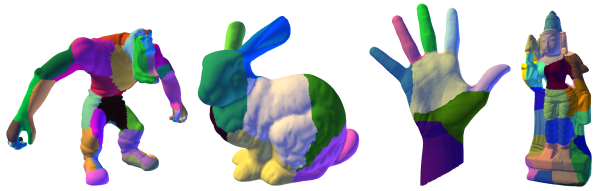


Figure 3. Segmentations for the Grunt, Bunny, Hand and Goddess datasets.

Seed initialization is done by sampling regions of highest importance in the mesh. A vertex pool is created, containing the $\lambda_c * C$ highest-saliency vertices, where λ_c is a user-defined multiplier and C is the desired cluster count; C seeds are then randomly chosen from this pool. Each cluster is initialized with saliency, normal and position values. Faces that share a seed vertex are placed on the cluster propagation queue. For all our experiments λ_c was set to 5, which was found to provide a large enough pool of perceptually important vertices for adequate initial seed distribution. Definition of cluster count is not addressed in this work, but it should be a number closely matching the number of important salient features in the mesh.

Cluster growth works through Dijkstra-like searches, performed simultaneously for all clusters. The dual graph of the mesh, where pairs of faces are connected by edges, is used as propagation medium. Each search starts on the highest-saliency face already in the queue; subsequent faces are evaluated through a cost function that considers normal deviation, distance and saliency variation. This promotes fairly planar clusters over areas of similar saliency, thus tending to encompass single prominent salient features or sets of similar and closely-packed features. More accurately, the edge cost between a face f on a cluster c and a candidate face f' adjacent to it is defined by:

$$\begin{aligned} cost(f,f') = &(\lambda_n - (\mu(\overrightarrow{n_c}) \cdot \overrightarrow{n_{f'}})) * \\ &(|\mu(\mathscr{S}_c) - \mathscr{S}_{f'}| * |\mathscr{S}_{c_0} - \mathscr{S}_{f'}| * |\mathscr{S}_f - \mathscr{S}_{f'}|) * \\ &(|\overline{x_{f'}} - \overline{x_f}|) \end{aligned} \tag{2}$$

where $\mu(\overrightarrow{n_c})$ is the mean cluster normal, $\overrightarrow{n_{f'}}$ is the normal vector for f', $\overline{x_f}$, $\overline{x_{f'}}$, $\mathscr{S}_f$ and $\mathscr{S}_{f'}$ are, respectively, the centroids and saliency values for f and f', $\mu(\mathscr{S}_c)$ is the cluster mean saliency and $\mathscr{S}_{c_0}$ is the cluster starting saliency value. Relative weight for normal contribution is adjusted by λ_n. Note that the scale of each component is not important — since each component multiplies the others, only relative magnitude needs to be considered. Scale is assumed to be constant for any given component in any given mesh. Growth proceeds until all mesh faces are assigned to a cluster. For all our experiments λ_n was set to 1.

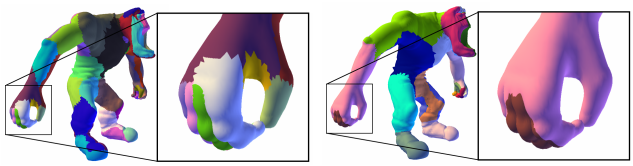
Cluster re-seeding is performed next. Each cluster has its seed and initial data updated; the new seed should be the most interior high-saliency vertex possible. New seeds are also selected through Dijkstra search, now starting from cluster borders and advancing inwards. Edge cost is now defined by a combination of distance and saliency variation:

$$cost(f,f') = (\overline{x_{f'}} - \overline{x_f}) * (\mathscr{S}_{f'} / max(\mathscr{S}_c)) \tag{3}$$

where $max(\mathscr{S}_c)$ is the cluster maximum saliency value detected during the growth phase. The new seed will be the highest-saliency vertex belonging to the last face reached during propagation.

Convergence happens through successive growth and re-seeding phases: both are repeated in order until the new seeds are identical to those encountered on the previous iteration. As in [10], we check for the existence of possible cycles in the convergence process; in such a case, any point on the cycle is acceptable.

Figure 4 compares examples of triangle meshes partitioned using salient clustering and non-salient propagative clustering. Notice how in Figure 4.a each finger is effectively isolated in a single cluster, while features are much less distinct in 4.b.



(a) (b)

Figure 4. Comparison of clustering results: Salient clustering (a) and non-salient propagative clustering (b).

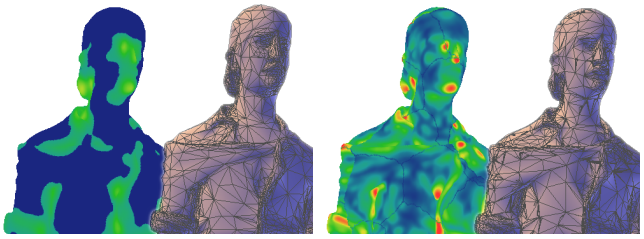
C. Multi-resolution Salient Simplification

We extend the salient simplification method presented in [3] to generate multi-resolution representations of a clustered triangular mesh. This can be seen as an extension of the quadrics-based simplification method (QSlim) where mesh saliency is incorporated as a quadric weighting factor.

Our extension uses a cluster-localized saliency map to guide simplification contractions, better preserving detail that is significant inside a specific cluster. This map is computed by normalizing saliency values inside each cluster, on a linear scale that ranges from zero (at the cluster base saliency) to one (at the cluster peak saliency), and then adding the normalized saliency values to the cluster mean saliency:

$$\mathscr{S}_{ci} = ((\mathscr{S}_i - min(\mathscr{S}_c))/(max(\mathscr{S}_c) - min(\mathscr{S}_c))) + \mu(\mathscr{S}_c) \quad (4)$$

This localized map is smoothed and amplified as in [3], and used to weight the quadrics computed by QSlim. Optimization is performed over the entire mesh, so connectivity at cluster boundaries can be maintained. Figure 5 compares smoothed and amplified saliency to cluster-normalized saliency.



(a) (b)

Figure 5. Saliency Comparison: smoothed and amplified saliency and simplified mesh (a) and cluster-localized saliency and simplified mesh (b).

Globally important details are better preserved when the original saliency is used, while regions with high local saliency are preserved when we use cluster-localized saliency values. Since our goal is to adaptively preserve detail in a view-dependent manner during render-time, the second option is preferred: using global saliency can lead to vast regions of the model suffering from heavy optimization while detail is carefully preserved on relatively small areas. This would jeopardize our ability to select an adequate level-of-detail in render-time; i.e. when regions that originally had high saliency values are selected for extensive simplification.

IV. View-dependent Rendering

Multi-resolution rendering involves selecting the proper level-of-detail for each rendered object based on application requirements, which typically emphasize shorter rendering times while maintaining the best possible visual result. We propose a saliency-guided system, taking advantage of results generated by saliency-guided mesh segmentation to effectively estimate visual importance in real time, feeding a GPU-based vertex contraction system. Figure 6 illustrates the interconnection between system components.

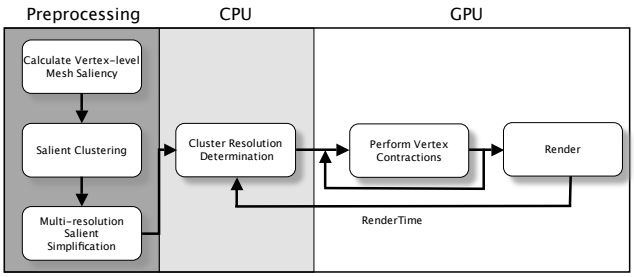


Figure 6. System Overview: In a preprocessing stage a salient-based clustering algorithm generates mesh clusters that are simplified based on saliency information. CLOD selection through vertex contractions is performed on the GPU using view-dependent information, generating an array of vertex indices that define the multi-resolution mesh to be rendered.

Our technique works by estimating cluster screen-space visibility and perceptual importance by comparing its position and orientation relative to the camera against a set of cluster attributes. This data is then used to determine the cluster rendering resolution, which serves as input to a GPU-based level-of-detail selector that builds the proper vertex index arrays to be used for rendering. While resolutions are selected independently for each cluster and vertex indices reference localized cluster vertex arrays, a global resolution-matching step is executed to guarantee hole-free cluster boundaries, removing the need for additional zippering and minimizing face overlap. Salient clustering is an essential component of our solution because it fulfills the following requirements:

- **Identify sets of triangles that comprise distinct salient features of the original object:** Different features can have different levels of visual importance to an observer. Having similar and spatially-close features within a single cluster allows us to quickly select the level of detail at run-time without the need for extensive computation.
- **Reduce processing and bandwidth requirements for level-of-detail selection during render-time:** Processing a vertex list for multi-resolution rendering is a time-consuming task; hence the use of GPU processing. Dividing an object into clusters can reduce the number of bits required to store a vertex index in GPU memory, thus reducing the required bandwidth.

We evaluated different clustering techniques, and while most of them were able to generate results that satisfied the second requirement, all failed on properly detecting and grouping perceptually interesting features without user input, excepting saliency-guided mesh segmentation.

A. Cluster Resolution Determination

Once a global resolution target (R_{global}) for an object to be rendered is chosen, we must determine how each cluster will be optimized to generate the best possible visual results. Two factors are taken into account when level-of-detail for a cluster is being calculated: estimated screen-space visibility and estimated visual importance.

Estimated screen-space visibility (A_{est}) is basically a factor of the cluster distance from the camera, total area, mean normal and peak normal deviation. We assume, conservatively, that when the vector pointing from the camera to the cluster centroid is perfectly opposite to the cluster mean normal, the entire cluster is visible. As the angle between these vectors decreases, visible area decreases proportionally; when it is smaller than the cluster peak normal deviation visibility can be expected to be very low. More specifically, we define the visualization angle factor as:

$$\lambda_{angle} = \angle(\overrightarrow{v_{cam}}, \mu(\overrightarrow{n_c})) - peak(\overrightarrow{n_c}) \quad (5)$$

where $\overrightarrow{v_{cam}}$ is the vector from the camera to the cluster centroid, $\mu(\overrightarrow{n_c})$ is the cluster mean normal, and $peak(\overrightarrow{n_c})$ is the cluster peak normal deviation (maximum angle between the normal of a face belonging to the cluster and the cluster's mean normal). While this is not a precise measure of visibility — to have such a measure visibility for every face in the cluster would have to be evaluated — it is a suitable approximation for our purposes. Additionally, distance from the camera must be taken into account: the farther away from the camera an object is, the smaller its associated area will be. Distance factor is defined as:

$$\lambda_{dist} = |\overline{x_c} - p_{cam}| - r_c \quad (6)$$

where $\overline{x_c}$ is the cluster centroid, p_{cam} is the camera position, and r_c is the radius of the cluster bounding sphere. Multiplying the visualization angle by the inverse of the distance factor and by the cluster area gives us the combined screen-space area estimate:

$$A_{est} = \lambda_{angle} * 1/\lambda_{dist} * A_{cluster} \quad (7)$$

Note that the case where the distance factor (λ_{dist}) is zero must be considered. Treatment is application-specific; for our tests, when a value of zero was found, the last non-zero value previously encountered was applied.

Visual importance estimation (I_{est}) takes into account a cluster's total, mean and peak saliency values. Highest contribution is obtained when high values exist for the three saliency metrics, giving us the visual importance estimate:

$$I_{est} = (\mathscr{S}_c * peak(\mathscr{S}_c) * \mu(\mathscr{S}_c))/\mathscr{S}_{mesh} \quad (8)$$

where $\mathscr{S}_{mesh}$ denotes accumulated saliency for the entire mesh. Examples of importance estimation from different points of view are shown in Figure 7.

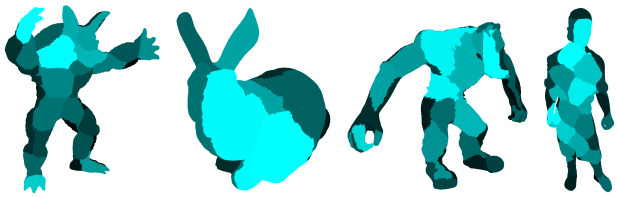


Figure 7. Visual Importance Estimation: Lighter regions have greater importance values.

Contraction distribution sorts clusters by $A_{est} * I_{est}$. Those with lower area and importance values are rendered with lower resolution, while higher values have more detail preserved. For practical purposes, we define a cluster's resolution by the number of vertex pair contractions to be performed over its multi-resolution representation; contractions are then distributed across the cluster list in order. The lowest importance cluster is assigned:

$$R_{local} = \lambda_{perc} * R_{global} \quad (9)$$

contractions, where λ_{perc} is a user-defined percentage value. R_{local} is clamped to the maximum number of contractions allowed on the cluster. This number is then subtracted from the global contraction target and the next cluster is evaluated, until the global contraction target has been reached. If R_{global} is not reached after the last cluster on the list is evaluated, the process continues from the beginning of the list. For all our experiments λ_{perc} was set to 0.02.

Neighbor matching is executed after all clusters have received a contraction target: a vertex pair contraction can only be performed if all contractions leading to it have already been performed. To guarantee this, we keep the global order of the last required pair contraction associated with each vertex contraction. For each cluster, the largest parent-contraction order is determined, and neighboring clusters are evaluated; if the neighbor contraction stop point is smaller than this largest contraction order, the neighbor stop point is replaced by this value. This ensures gap-free boundaries, and minimizes face overlap problems. Some overlap may still occur, especially over regions where significant difference in resolution targets for a local group of clusters is encountered, but such artifacts are rare.

B. Level of Detail Selection

Vertex pair contractions over each cluster vertex list are performed until a resolution is determined. This is considerably time consuming: index lists must be processed several times until all contractions are performed. Updated lists must be uploaded to the GPU for rendering, thus increasing CPU-GPU bandwidth usage. To speed up this process, we introduce a GPU-based level-of-detail selector, which performs the entire process on the GPU, and reduces

both execution and index list update times. Our GPU-based level-of-detail selector is composed of a single fragment shader, two framebuffer objects and a set of data textures.

Five textures hold the information required for the contraction process. The first two textures encode, for each vertex of a given cluster, its initial and subsequent contractions:

- Base Vertex Index (*BVI*) texture (2D, RGBA32F): initial contraction: vertex index (v_i), contracted vertex index ($VC(v_i)$), 1D coordinates for the CVI texture (i_{cvi}) and for both the SCI and CSP textures (i_{sci}).
- Contracted Vertex Index (*CVI*) texture (2D, RGBA32F): subsequent contractions, same format as the BVI texture.

Since a vertex is processed separately for each cluster, we ensure that a vertex that appears on multiple clusters undergoes the same number of contractions, thus avoiding cracks in the final mesh. The remaining three textures encode information used in this process:

- Shared Cluster Indices (*SCI*) texture (2D, RGBA32F): IDs for all clusters associated with a given vertex.
- Vertex Contraction Order (*VCO*) texture (2D, R32F): global pair contraction order of a given vertex.
- Cluster Stop Point (*CSP*) texture (1D, R32F): cluster stop points selected during resolution estimation.

Framebuffer objects (*FBOs*) are used for both processing (*FBOP*) and data format conversion (*FBOC*). The first (*FBOP*) is used during the iterative vertex pair contraction process and matches the information stored on the BVI and CVI textures, having two surfaces (*FBOPr* and *FBOPw*) for ping-pong rendering. The second (*FBOC*) is used for data type conversion and copy. Both framebuffer objects share the size defined for *BVI*. The vertex contraction shader takes *CVI*, *SCI*, *VCO* and *CSP* as input textures, as well as the texture associated with *FBOPr* and the widths and heights for the *CVI*, *SCI* and *VCO* textures. It is executed over the domain defined by *FBOP*, where each pixel maps directly to one vertex index in the clustered index buffer. Level-of-detail selection works as follows: The value *maxContractionLevel*

Algorithm 1 VERTEX CONTRACTION ALGORITHM

```
1: Read surface on FBOPr with initial BVI values
2: for i = 1 to maxContractionLevel do
3:     Execute shader over FBOPr, writing on FBOPw
4:     Swap FBOPr and FBOPw
5: end for
6: Write contents of FBOPr to FBOC
7: Copy FBOC to vertex index buffer for rendering
```

represents the highest level of any pair contraction target that must be reached among all clusters, and can be calculated with no additional complexity during the cluster resolution determination step. The contraction shader is detailed on Figure 8.

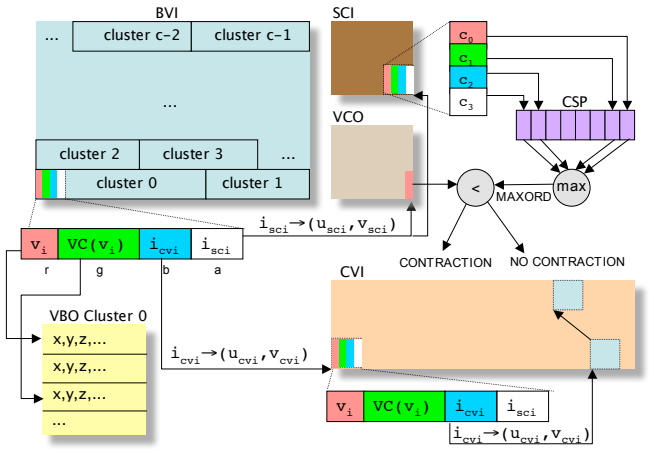


Figure 8. Vertex contraction shader: first it checks if a vertex has a contraction target (channel *G*: vertices with no target receive value -1). If no target exists, the value is maintained; otherwise, coordinates stored on channels *B* and *A* are decoded. The first is used to load contraction data from *CVI*, while the second is used to access data from both *SVI* and *VCO*. Each channel on *SCI* is used for additional texture indirection, now reading from CSP; the four values read from *CSP* are compared, and the highest order value *MAXORD* is selected. If the global vertex order, read from *VCO*, is smaller than *MAXORD* the vertex must be contracted, and the output value is updated with one read from *CVI*.

After the vertex contraction shader has executed *maxContractionLevel* times, *FBOPr* contains the updated index array. We copy these values, through a single rendering pass, to *FBOC* - which shares the same internal format as the index buffer - from whence values are copied to the index buffer to be used for rendering. Rounding problems when *unsigned integer* values are manipulated by fragment shaders were observed in some graphics cards. *Unsigned short* did not present such errors, with the added benefit of lower transfer times, so this format was adopted instead. However, it limits clusters to a maximum of 2^{16} vertices, a restriction which can be easily enforced during pre-processing.

C. Rendering

Rendering is relatively straightforward. A single vertex index buffer, with cluster-localized vertex indices, is bound at all times. For each cluster, its vertex data buffer (where vertex positions, normals, color, etc. are stored) is bound, and the portion of the index buffer relative to the cluster is rendered. Other cluster-level optimization algorithms, such as partial ordering [16], can be used in conjunction with our method without the need for adaptations.

V. Results

We have tested our segmentation algorithm with several datasets, covering a broad range of geometric and topological complexity. All experiments were conducted on an Athlon 64 3500+, with 2GB of RAM and an NVIDIA GeForce 8800 GTX 768 graphics card. Table I gives more specific information on each dataset used. Execution times for segmentation are given in Table II; while k-means clustering takes considerably less time than our technique, its results are relatively poor. Times for propagative segmentation are similar to those of our technique, but it fails to detect salient features, generally targeting flat regions.

Table I
Datasets

Dataset	Vertices	Faces	Edges	Clusters
Grunt	26143	52282	78423	45
Bunny	35947	69451	104288	10
Hand	136663	273060	409724	10
Ganesh	206618	413236	619827	30
Goddess	137406	274822	412231	30
Armadillo	172974	345944	518916	45
Laçador	653891	1307794	1961691	45

Significantly better frame-rates - when compared both to full resolution representations and optimization through CPU-based pair contraction - were observed for all models when using our method, for meshes rendered at 5% of the original resolution. Computation time for each mesh update was amortized over 10 frames for both the GPU and CPU implementations. Table III summarize these results.

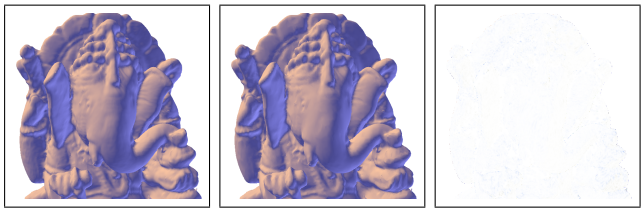
Table II
Saliency computation and clustering times, in seconds.

Dataset	Saliency Comp.	Salient Seg.	Prop. Seg.	K-Mean Seg.
Grunt	8.516	13.859	21.203	0.14
Bunny	5.187	30.359	64.844	0.172
Hand	78.219	247.39	218.687	0.625
Ganesh	80.516	545.328	925.672	2.531
Goddess	56.531	578.61	833.843	0.906
Armadillo	91.047	417.265	623.688	1.609
Laçador	1747.53	6945.98	7141.34	8.063

Table III
FPS for full mesh, GPU vertex contration and reference CPU vertex contraction implementation.

Dataset	No Opt.	CPU (5%)	GPU (5%)	(35%)	(60%)
Grunt	1087	685	2220	1759	1419
Bunny	817	475	2147	1507	1126
Hand	241	116	823	518	357
Ganesh	160	64	482	283	222
Goddess	216	127	839	423	328
Armadillo	184	93	783	341	262
Laçador	50	21	234	105	75

Advantage in relation to rendering the full-resolution mesh decreases when less optimization is performed, but even at as much as 60% of the original resolution improvements in framerate are significative when amortization is performed. Still, this loss could be minimized through greater amortization, setting a cutoff point (optimizing meshes only when significant simplification is required) or other render-time resolution selection techniques. Rendering multiple instances of the same optimized mesh, such as characters or repeating scenery objects in massively populated environments, will also decrease relative costs. As an example, rendering nine instances of the Armadillo dataset at 60% of its original resolution more than doubles the framerate: from 20 FPS when rendering the full resolution mesh to 42 FPS when our method is enabled, with minimal visual loss. Results practically indistinguishable from the original mesh can be obtained at up to 35% simplification (Figures 9 and 10) and with minimal difference at 5% (Figure 11).



(a) Original (b) 35% (c) Difference

Figure 9. View-dependent rendering results for the Ganesh dataset, at 35% of original resolution, with framerate increasing from 64 to 283 FPS. Original mesh (a), simplified mesh (b) and inverted difference image, showing nearly no difference (c).

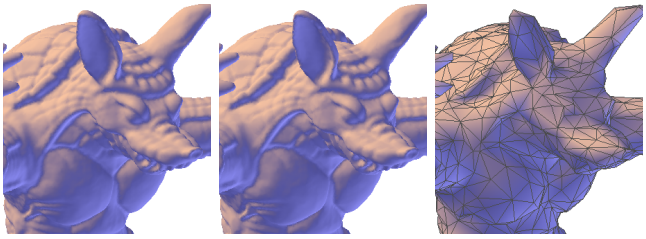


Figure 10. Armadillo results. Original Mesh, Saliency-guided Simplification to 35% of original resolution and to 1% or original resolution (progressive mesh backstop point). Most relevant details were preserved on the mesh simplified to 35% when compared to the full resolution.

VI. Conclusions and Future Work

A method to reliably segment a triangle mesh into a set of clusters containing visually interesting features was described in this work. Mesh saliency serves as a computational model of perceptual importance to iteratively determine face clusters through propagation from a starting point selected from a pool of high-saliency vertices. No user input is required, and salient regions are correctly captured. Segmentation of equal or better quality than comparable methods is achieved, in similar computational

times. Additional validation was performed through a view-dependent rendering system, where salient clusters were assigned different mesh resolutions at render-time based on their visual importance, calculated from viewpoint information and combined cluster saliency data; high simplification rates were achieved with little visual loss, resulting in greatly improved framerates.

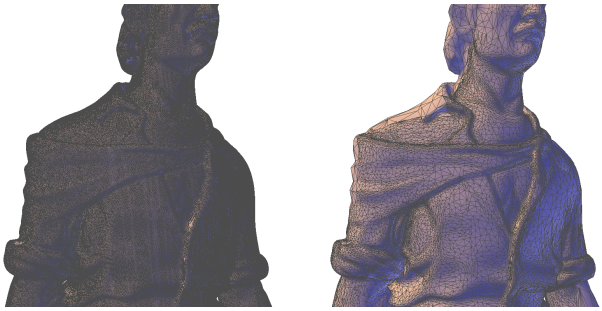


Figure 11. Laçador Results. Detail from Original Mesh and Saliency-guided Simplification to 5% of original resolution.

Further refinements are possible. Saliency computation can benefit from better metrics for neighborhood determination, such as geodesic distance; salient feature detection — recognition of individual interesting features by local saliency peak analysis — may improve both the results of the segmentation process and of the cluster importance estimation step. Validation of our visual importance metric through user studies can provide a more solid basis for employing mesh saliency as an automatic metric for perceptual importance.

REFERENCES

[1] C. O'Sullivan, S. Howlett, Y. Morvan, R. McDonnell, and K. O'Conor, "Perceptually Adaptive Graphics," in *STAR-Proceedings of Eurographics 2004*, ser. State of the Art Reports, C. Schlick and W. Purgathofer, Eds., no. STAR-6. INRIA and the Eurographics Association, 2004, pp. 141–164. [Online]. Available: http://eg04.inrialpes.fr/Programme/STAR/STAR6.html

[2] D. Luebke, B. Watson, J. D. Cohen, M. Reddy, and A. Varshney, *Level of Detail for 3D Graphics*. New York, NY, USA: Elsevier Science Inc., 2002.

[3] C. H. Lee, A. Varshney, and D. W. Jacobs, "Mesh saliency," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 659–666, 2005.

[4] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 209–216.

[5] S. Howlett, J. Hamill, and C. O'Sullivan, "Predicting and evaluating saliency for simplified polygonal models," *ACM Trans. Appl. Percept.*, vol. 2, no. 3, pp. 286–308, 2005.

[6] R. Gal and D. Cohen-Or, "Salient geometric features for partial shape matching and similarity," *ACM Trans. Graph.*, vol. 25, no. 1, pp. 130–150, 2006.

[7] Y. Kho and M. Garland, "User-guided simplification," in *SI3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*. New York, NY, USA: ACM Press, 2003, pp. 123–126.

[8] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, vol. 28, pp. 84–95, 1980.

[9] N. A. Carr, J. Hoberock, K. Crane, and J. C. Hart, "Rectangular multi-chart geometry images," in *Proceedings of the 4th Eurographics Symposium on Geometry Processing*, 2006.

[10] P. V. Sander, Z. J. Wood, S. J. Gortler, J. Snyder, and H. Hoppe, "Multi-chart geometry images," in *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 146–155.

[11] S. Katz and A. Tal, "Hierarchical mesh decomposition using fuzzy clustering and cuts," in *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*. New York, NY, USA: ACM Press, 2003, pp. 954–961.

[12] M. Mortara, G. Patané, M. Spagnuolo, B. Falcidieno, and J. Rossignac, "Blowing bubbles for multi-scale analysis and decomposition of triangle meshes," *Algorithmica*, vol. 38, no. 1, pp. 227–248, 2003.

[13] H. Zhang and R. Liu, "Mesh segmentation via recursive and visually salient spectral cuts," in *Vision, Modeling, and Visualization 2005*, G. G. et al., Ed. Berlin: Akademische Verlagsgesellschaft Aka GmbH, November 2005, pp. 429–436. [Online]. Available: http://www.vmv2005.uni-erlangen.de

[14] S. Katz, G. Leifman, and A. Tal, "Mesh segmentation using feature point and core extraction," *The Visual Computer*, vol. 21, no. 8-10, pp. 649–658, 2005.

[15] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 11, pp. 1254–1259, 1998.

[16] D. Nehab, J. Barczak, and P. V. Sander, "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: ACM Press, 2006, pp. 207–211.