Parallelization of skeleton extraction from 3D models and point clouds

Nikolas Hamran *
Supervised by: Martin Madaras

Institute of Computer Engineering and Applied Informatics
Faculty of Informatics and Information Technologies STU
Bratislava / Slovakia

Abstract

We present a GPU accelerated algorithm for skeleton extraction from 3D meshes and point clouds. Our method performs skeletonization in a pipeline, where each stage ensures an optimal transformation of the input data to achieve a satisfactory result. The input vertices are converted to the medial surface of the mesh with the shrinking spheres method for improved accuracy of the final skeleton. Vertices are then sampled from the medial surface and contracted to a thin shape with the L1 method. These steps are done in parallel. The final stages include skeleton reconstruction and post processing on the CPU. We introduced modifications to the original CPU based algorithms for them to be suitable for parallel execution. These modifications include speedups for algorithm convergence and data dependency removal during computation. An advantage of our method is that it does not require connectivity information between vertices, as opposed to Laplacian based mesh contraction methods. Experimental results show near real time performance of our algorithm. The implementation uses the CUDA API for efficient utilization of the GPU resources.

Keywords: Curve skeleton, GPU, CUDA

1 Introduction

The curve skeleton is a geometric graph structure which encodes the shape and topology of a 3D mesh. Curve skeletons have many important use cases in computer graphics and other fields of research. They are vital for 3D mesh animation, 3D mesh repair and compression, virtual navigation, visualization improvement and many more. Finding a precise and fast approach to curve skeleton extraction remains an open problem despite many attempts in the field [14].

With the rise of massively parallel processing architectures and the CUDA API, many algorithms were successfully implemented in the GPGPU environment and achieved significant speedup. 3D mesh skeletonization is a fairly expensive operation and the main pitfall of many recent approaches is the lack of real-time performance. From many publications in this domain, only a few focus on accelerated curve skeleton extraction [3, 7]. This is probably due to the GPGPU programming model being significantly less flexible than the sequential approach. This model imposes many limitations on memory access, data dependencies and program flow execution during computation, therefore our aim is to design a skeletonization algorithm based on existing methods which abides all limits of the parallel approach.

In this paper we present a skeletonization pipeline, shown in Figure 1, adapted to the GPGPU batch processing model. Our method is based on the principle of L1 median contraction as proposed by Huang et al. [6]. The L1 median calculation from a set of vertices is a data independent task well suited for the GPU. In addition to the L1 median, the paper introduces a regularization constraint for maintaining the correct layout of skeleton vertices during energy minimization. The regularization constraint requires moderate amounts of data exchange between threads during computation which had to be removed in order to conform to the GPGPU scheme. To compensate for the introduced changes in the L1 contraction phase and to increase the precision of the result we first perform a medial surface extraction with the shrinking spheres method as proposed by Ma et al. [11]. This method, albeit expensive on the CPU, is well suited for the parallel computation model as all processed elements are data independent. The medial surface serves as an approximation of the curve skeleton for the L1 phase. After the L1 contraction, a skeleton reconstruction phase is performed with a recursive method as proposed by Lee et al. [8]. The skeleton is reconstructed with a principal component analysis (PCA) measure along the regression line that passes trough the local area of skeleton vertices.

The proposed algorithm requires the mesh vertex positions and normal vectors as input and produces a list of skeleton vertex positions with an adjacency list to indicate connections between skeleton vertices. We demonstrate our method on example meshes and provide a table of extraction times for individual phases.

*Master study programme in field: Intelligent Software Systems, nikolas.hamran96@gmail.com

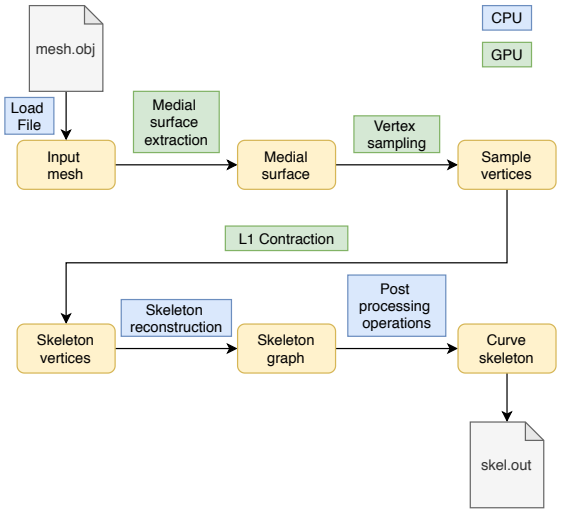


Figure 1: Skeletonization pipeline, phases colored green are performed on the GPU, phases colored blue are performed on the CPU

2 Background

GPU-Based approaches As stated in Section 1, not many skeletonization algorithms were proposed for the GPGPU platform. A recent method by Baldacci et al. [3] relies on the shape diameter function as defined by Shapira et al. [13]. This method uses ray-tracing for determining the local breadth of a mesh for each sampled vertex. Ray tracing is performed in a cone shape centered at each mesh vertex along the inner normal vectors of said vertices. This is done in parallel, for no dependencies exist between rays. The algorithm looks for intersections of rays with the "other side" of the mesh. Local breadth is calculated as the weighted average of ray lengths based on the angle between incoming rays and intersected triangular faces of the mesh. A skeleton vertex is obtained as a point halfway along the line defined by the inner normal and calculated mesh breadth.

A different parallel approach proposed by Jalba et al. [7] applies a pipeline of transformations with gradual convergence to the final skeleton. First the medial surface is extracted and then a curve skeleton is found using SSG - shortest straightest geodesics. During medial surface extraction, the flux of vertices is recorded. The curve skeleton is extracted based on the border lines between areas of flux with different directions. We adopted this pipeline approach, for it is a natural fit for the GPU computing model.

Medial surface extraction The medial surface is a two dimensional manifold that arises from the set of centers of maximally inscribed spheres in a 3D mesh [4], and is the 3D equivalent of the medial axis. As opposed to the curve skeleton, the medial axis represents the true compact representation of a 3D mesh, from which the mesh can be fully reconstructed. Obtaining the medial surface in a parallel environment is a relatively straightforward task. A solution exploiting topological thinning was proposed by Palagyi et al. [12]. This method uses mesh voxelization and iterative thinning. The surface skeleton is obtained by the removal of voxels in layers until a stop condition is met. A significant disadvantage of the method is the production of holes in the resulting medial surface. Holes and cavities in the result surface skeleton are created when the stop condition is not met in time.

An analytic method based on iterative contraction was proposed by Ma et al.[11] which we adopt and explain in Section 3.

Mesh contraction Many state of the art methods utilize a energy minimization principle which results in the contraction of a 3D mesh to a thin shape resembling the curve skeleton [2, 6, 9]. The curve skeleton reconstruction is performed during or after the contraction phase. These methods prove to be resilient against surface noise and produce a robust skeleton with well preserved homotopy.

The method proposed by Au et al. [2] utilizes iterative contraction based on the mesh Laplacian. Vertices are contracted by balancing contraction and attraction constraints to achieve a well distributed thin shape. After contraction, a connective surgery step is performed in which the contracted mesh edges are decimated to a 1D shape - the curve skeleton. The contraction process relies on solving a over-determined system of linear equations with a least squares method. While there are library facilities which implement the solution of such systems on the GPU, the main disadvantage of said method is the numerical instability and the need for connectivity information between mesh vertices.

The method proposed by Huang et al. [6] exploits the L1 geometric median and is the foundation of our approach. We address the details in Section 3. A modified method proposed by Li et al. [9] uses the local optimal projection (LOP) operator to achieve a higher quality skeleton. This method relies on the medial surface of the mesh. First the medial surface is extracted and then marker points are spread throughout the medial surface with the LOP operator. The LOP ensures an uniform distribution of marker points on the medial surface by mutual repulsion. The curve skeleton is then extracted by contracting the marker points with the L1 median method. The uniformity of the marker points ensures a higher quality skeleton.

3 Overview

The input of our pipeline is a set of mesh vertices and their respective normal vectors $M = \{p_i, n_{p_i}\}, i \in I$. The output of the pipeline is a skeleton represented as a graph $X = \{V, E\}$, where V is the set of skeleton vertices and E is a set of skeleton edges.

Medial surface The first step in our pipeline performs the medial surface extraction $M_{med} = \{c_i\}, i \in I$. We adopt the technique of shrinking spheres proposed by Ma et al. [11]. The algorithm searches for the best approximation of the centers c of a maximally inscribed medial balls $\mathscr{B} = (c, \rho)$ with radius ρ for each point $p \in M$. If we assume that $\mathscr{B}$ intersects M at p and that $\mathscr{B}$ is tangent to p then c lies on the line defined by n_p. A sphere is uniquely defined by two points, therefore our goal is to find a second point $\widetilde{p} \in M, p \neq \widetilde{p}$ which is equidistant from c with respect to p. c is found iteratively. We present a high level overview of the algorithm.

For each $p \in M$ in parallel do $i \in \{1,2,...,k\}$ iterations of center point approximations. In each iteration

1. Select a random point $\widetilde{p}^0 \in M, p \neq \widetilde{p}^0$,
2. Calculate ρ^0 of the medial ball $\mathscr{B}^0$ defined by p, n_p and $\widetilde{p}^0$,
3. Calculate $c_p^0 = p - n_p * \rho^0$,
4. If $abs(|c_p^i p| - |c_p^i \widetilde{p}|) < \varepsilon$[1], then exit,
5. Find a new $\widetilde{p}^i \in M, \widetilde{p}^i \neq p$ as the closest neighbor to the current approximation of the best center point c_p^i,
6. Calculate ρ^{i+1} of the medial ball $\mathscr{B}^{i+1}$ defined by p, n_p and $\widetilde{p}^i$,
7. Calculate $c_p^{i+1} = p - n_p * \rho^{i+1}$,
8. Go to 4.

The new radius is calculated as $\rho^{i+1} = \frac{d(p,\widetilde{p}^i)}{2cos\theta_p^i}$, where d is the euclidean distance, θ is the smaller angle between n_p and the vector $\widetilde{p}^i p$ [11]. Next we address the introduced modifications to the algorithm.

We perform a brute force approach to finding $\widetilde{p}^i$ - the closest neighbor of c in a "carousel" memory access pattern. Each thread i performs I checks of distances between c_i and $c_{(i+a) \bmod I}$ in a cycle with $a \in \{1,2,...,I\}$ as the iteration variable. This avoids memory access divergence, for in any given time, all lanes in the memory bus are in use. The brute force approach is acceptable on GPU implementations as suggested by Li et al. [10].

As opposed to the approach by Ma et al. [11], we perform the iterative contraction exclusively on the GPU. A thread which finds its c stops and waits for the other threads in the group to finish. The original method utilized medial center convergence checking between iterations on the CPU. We deemed this approach to have significant overhead due to buffer copying between the CPU/GPU memory and repetitive kernel calls with each iteration. Our approach might introduce some warp divergence after some medial centers have been found, but we consider this a smaller penalty compared to the overhead of repetitive data copying.

[1] ε accounts for floating point errors

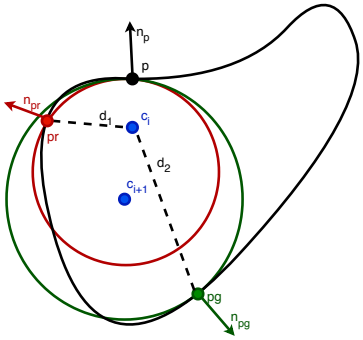
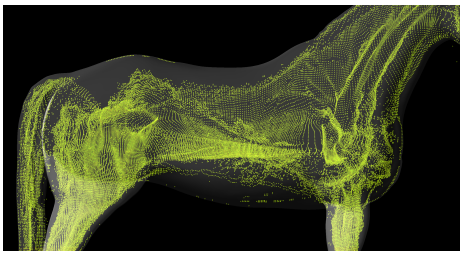
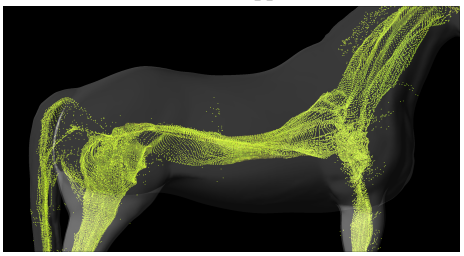


Figure 2: The shrinking spheres algorithm is augmented with the ν coefficient, this ensures a better input for the next phase.



(a) Without applied ν



(b) With applied $\nu = -0.8$

Figure 3: The ν coefficient causes the contracted medial surface to be more compact.

We contribute to the original approach by introducing a coefficient $\nu \in \langle -1, 0)$, which produces a better input for the L1 stage. During the nearest neighbor search, the dot product between vertex normal vectors $n_{dot} = n_p \cdot n_{\widetilde{p}^i}$ is calculated. Each vertex p whose n_{dot} is above the threshold given by ν is disqualified from this iteration of nearest neighbor search. Therefore the new $\widetilde{p}^i$ is guaranteed to lie on the "opposite side" of the input mesh, putting the center point c approximately halfway in between p and $\widetilde{p}^i$. The result is a 2D manifold slightly more contracted than the real medial surface. This property is useful, as it brings the input for the next phase of the algorithm closer to the curve skeleton. The principle is demonstrated in Figure 2, where despite p_r being closer to c_i ($d_1 < d_2$), p_g is chosen instead and the new center c_{i+1} is recalculated accordingly. The improved result after medial surface extraction can be seen in Figure 3. Figure 3b is more contracted as opposed to Figure 3a.

Vertex sampling The next step in the pipeline involves vertex sampling from M_{med} with a uniform stochastic method. A random number $r \in \langle 0,1 \rangle$ is generated for each vertex in M_{med} (in parallel). If r is smaller than a given sample vertex selection probability $R \in \langle 0,1 \rangle$, e.g. $r < R$, then the vertex is marked as selected. The selected vertices are then compressed to a new set of input vertices $M_{sampl} = \{x_j\}, j \in J$ with parallel reduction primitives (prefix sum and map). We chose a 5% sampling rate ($R = 0.05$). The parallel random number generator utilizes a seed value to make the sampling deterministic.

L1 Contraction The sampled vertices M_{sampl} are now contracted to a thin shape with reference to M_{med} based on minimization of:

$$X = \underset{x}{argmin} \sum_{j=1}^{J} \sum_{i=1}^{I} (||x_j - c_i|| \theta(||x_j - c_i||)) + R(X) \quad (1)$$

as proposed by Huang et al. [6] for each sampled vertex. The minimization is done with:

$$x_j^{k+1} = \frac{\sum_{i \in I} c_i \alpha_{ij}^k}{\sum_{i \in I} \alpha_{ij}^k} + \mu \sigma_j^k \frac{\sum_{j' \in J \setminus \{j\}} (x_j^k - x_{j'}^k) \beta_{jj'}^k}{\sum_{j' \in J \setminus \{j\}} \beta_{jj'}^k} \quad (2)$$

$$\alpha_{ij}^k = \frac{\theta(||x_j^k - c_i||)}{||x_j^k - c_i||}; \beta_{jj'}^k = \frac{\theta(||x_j^k - x_{j'}^k||)}{||x_j^k - x_{j'}^k||^2}, j' \in J \setminus \{j\}$$

The vertices in M_{med} are fixed in place. The first term of Equation 1 $||x_j - c_i|| \theta(||x_j - c_i||)$ is responsible for calculating and minimizing the weighted sum of euclidean distances to the closest neighbors in M_{med} for each sampling vertex in M_{sampl}. The distances are weighted with a fast decaying smooth function $\theta = e^{-(x/(h/2))^2}$ where h is the neighborhood support radius. h is initially set to $h_0 = 2d_{bb}/\sqrt[3]{I}$ with d_{bb} being the length of the bounding box cross diagonal of M_{med}. h is gradually increased between iterations based on $h = h + \frac{h_0}{2}$. The second term, the regularization constraint $R(X)$, involves the calculation of σ. σ is a weighted PCA conformity measure to assure an even vertex distribution along local skeleton branches during contraction. $R(X)$ causes the contracted vertices to repel each other along the dominant PCA directions of forming branches. This direction is given as the dominant eigenvector of a weighted covariance matrix from the local neighborhood of sample vertices in M_{sampl}. The repulsion strength is regulated with μ and is set by default to $\mu = 0.35$ [6].

Calculating the first term is trivial to parallelize. Each thread performs the weighted distance calculation for one vertex in M_{sampl}. A problem arises with the calculation of $R(X)$. The calculation of σ requires the current positions of vertices in M_{sampl}. This introduces a data dependency problem. We solved this issue by partitioning the algorithm to iterations and performing global synchronization between them. To avoid data copying, we employed a double-buffering approach with buffer pointer swapping so only the overhead of kernel calls is in effect.

A second, more pressing, issue we encountered was a gradual skeleton reconstruction step between iterations. This step involves marking the contracted vertices with flags to halt their further contraction and recursive skeleton branch construction. Initially a smoothed σ measure is calculated for each vertex by averaging the σ of k-nearest neighbors. Vertices with the smoothed σ above a threshold are considered as candidates for skeleton branch vertices. Branches are then recursively built from said candidates along dominant PCA directions, discarding stray branches and non-candidates. Branch vertices become fixed in place and are no further contracted [6]. A troubling matter with this approach is the recursion. Stacked traversal and recursive function calls are not GPU friendly and therefore we had to discard the intermediate skeletonization and the stop condition. Instead we opted for a fixed number of L1 contraction iterations and separated the skeleton reconstruction to a new phase. This introduced the problem of unbounded neighbor radius growth which causes overcontraction of certain input parts and the loss of fine details of a skeleton. We solved this issue partially by a small, fixed number of contraction iterations. We estimated 5 iterations to be sufficient for most inputs.

Skeleton reconstruction We propose a custom skeleton reconstruction and down sampling method from the contracted vertices in M_{sampl} based on approaches by Huang et al. [6] and Lee et al. [8]. We perform the skeleton reconstruction on the CPU.

The reconstruction begins with assigning a flag to each vertex in M_{sampl}, which determines the status of a vertex. The possible flags are:

- **Branch vertex:** a vertex that is part of the final skeleton.
- **Non-branch vertex:** a candidate for becoming a branch vertex.
- **Invalid vertex:** a vertex that is not part of any branch and is ignored.

At the start, all vertices are labeled as non-branch vertices. The algorithm begins by examining a non-branch vertex (colored green in Figure 4). This vertex serves as a seed point for a new skeleton branch. A recursive depth-first traversal is performed. The skeleton is reconstructed along the regression curve, which is determined by the dominant eigenvector of the localized covariance matrix as proposed by Lee et al. [8]. This eigenvector determines the forward and backward direction of traversal as seen in Figure 4. The algorithm searches for the farthest point (colored red and blue in Figure 4) from the currently processed vertex in each hemisphere of the sphere defined by the local

search radius h_r. Only non-branch or branch points are considered as valid neighbors. We propose h_r to be calculated as $h_r = d_{bb}/\sqrt{I}$ similarly as in the L1 phase. After the search is finished and at least one farthest neighbor is found, all vertices in the local area are marked as invalid, the processed vertex is marked as branch vertex and the algorithm is recursively applied to the farthest found neighbors (if said neighbor is a non-branch vertex). If no neighbors are found, the search terminates and the algorithm starts a new branch from an unprocessed non-branch vertex. This is done until only branch vertices or invalid vertices remain.

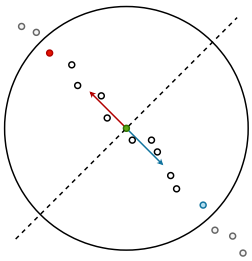


Figure 4: Hemispheres defined by the dominant eigenvector

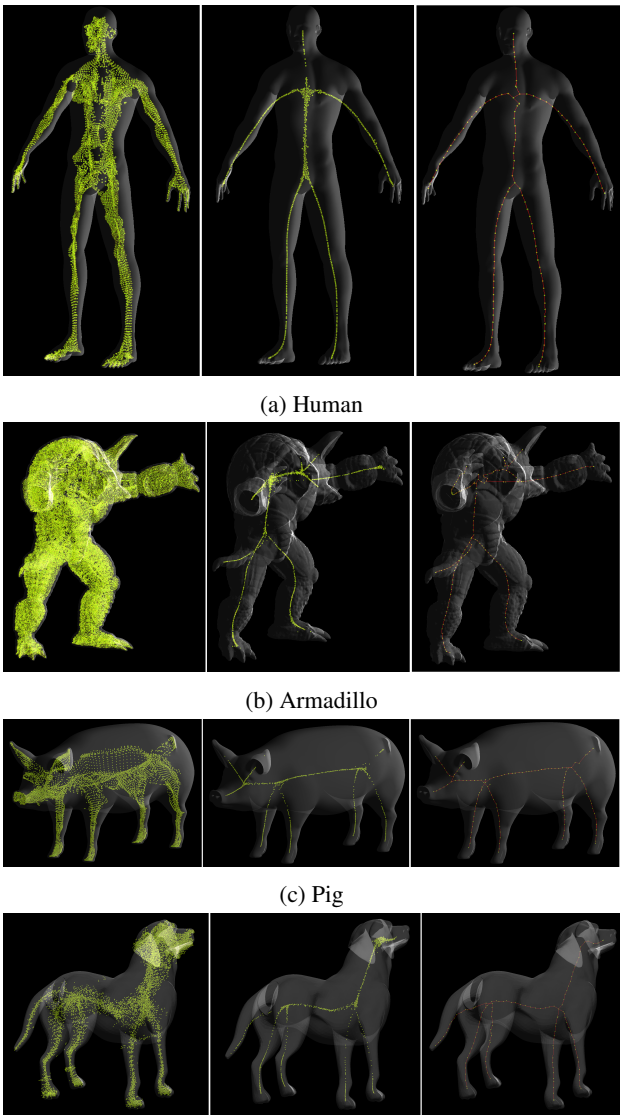
The recursive reconstruction produces a set of disconnected skeleton branches which need to be connected. We treat these components as nodes in a complete graph where edges are weighted by the shortest possible distance between all component pairs. The edge weights incorporate the closets pair of vertices each belonging to their respective component. Next, the components are connected with Kruskal's algorithm for finding the minimum spanning tree thus creating the curve skeleton.

Post processing The final stage of the pipeline involves post processing of the extracted curve skeleton for further improvements. Currently no operations are done in the post processing stage. A skeleton re-centering step is planned to be added soon.

4 Results

We tested our implementation on meshes with different input sizes and measured the execution times which we summarized in Table 1. The column "Total" shows the sum of execution times of individual phases of our pipeline. We provide an overview of our skeletonization process in Figure 5 where the output from individual stages in the skeletonization pipeline is displayed. The hardware used during tests is shown in Table 2. Our tests were performed with $\nu = -0.8$. The unbound contraction radius makes it difficult to produce an exact curve skeleton with our method, therefore we prioritize the speed of extraction.

Extraction times Our algorithm provides a significant improvement of skeleton extraction time which is in the range of milliseconds up to a few seconds. We tested the approach proposed by Huang et al. [6] by performing skeletonization on the hardware listed in Table 2 with an executable from the authors of said article. This executable is accessible from the page [1]. The results are shown in the column "L1 reference implementation" of Table 1. The original CPU based approach performs in the range tens of seconds up to minutes. The skeletons extracted with this executable are shown in Figure 6.



(a) Human

(b) Armadillo

(c) Pig

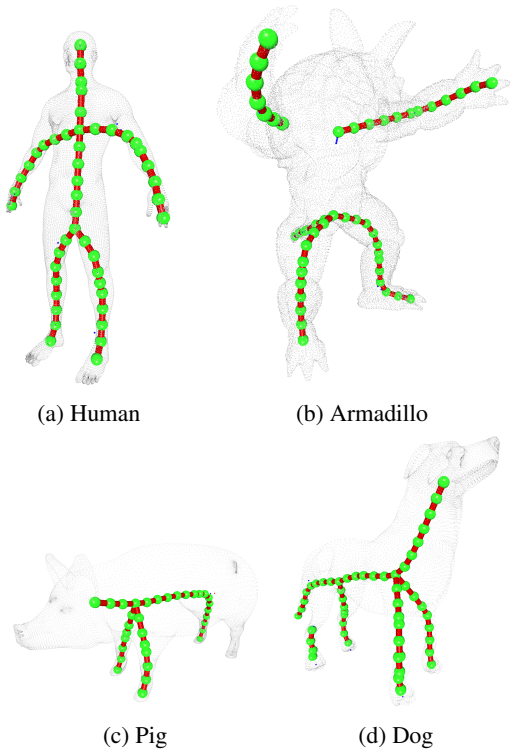
(d) Dog

Figure 5: Our proposed skeletonization pipeline (Medial surface, L1 contracted vertices, Curve skeleton) for various meshes

Produced skeleton quality We now evaluate our skeletons based on the criteria for skeleton quality evaluation as proposed by Cornea et al. [5]. Our skeletons show good

Mesh	Vertex count	Medial surface	Vertex sampling	L1 Contraction	Skeleton reconstruction	Total	L1 reference implementation
Suzanne	7 830	16 ms	<1 ms	<1 ms	<1 ms	16 ms	11 311 ms
Frog	17 484	156 ms	<1 ms	<1 ms	1 ms	157 ms	15 547 ms
Dog	18 114	150 ms	<1 ms	<1 ms	1 ms	151 ms	12 306 ms
Pig	20 098	146 ms	<1 ms	<1 ms	1 ms	147 ms	19 659 ms
Plant	22 570	257 ms	<1 ms	<1 ms	<1 ms	257 ms	8 303 ms
Human	24 461	170 ms	<1 ms	<1 ms	1 ms	171 ms	9 569 ms
Armadillo	34 594	418 ms	<1 ms	<1 ms	3 ms	421 ms	35 311 ms
Bunny	34 817	510 ms	<1 ms	<1 ms	5 ms	515 ms	60 360 ms
Hand	66 848	824 ms	<1 ms	<1 ms	8 ms	832 ms	142 425 ms
Horse	120 660	2 169 ms	1 ms	<1 ms	12 ms	2 182 ms	134 400 ms
Motherhood	202 690	9 684 ms	1 ms	<1 ms	91 ms	9 776 ms	563 590 ms

Table 1: Algorithm execution times for various meshes



(a) Human (b) Armadillo

(c) Pig (d) Dog

Figure 6: Skeletons extracted with the provided executable based on the approach proposed by Huang et al. [6]

qualities in homotopy, thinness, robustness and connectivity. Our skeletons are fully connected, as opposed to the reference implementation with well formed branches in all input meshes. The medial surface extraction step helps achieve moderate centeredness, but an additional re-centering step is required. Due to the reconstruction step, the produced skeletons are not completely smooth. Our approach should perform well under isometric transformations of the input, however our process of skeletonization is data-destructive and does not provide the reconstructability property.

HW	Notebook MSI GS63VR RF7
CPU	Intel i7-7700 HQ, 2.8GHz, quad core
RAM	16GB 2400MHz
GPU	Nvidia GTX 1060 6GB, Pascal
OS	Win 10 Home
CUDA	10.2

Table 2: Hardware used for testing

Shortcomings of our approach Our approach has its shortcomings with the unbound contraction radius growth being the most pressing one. We were not capable to eliminate this effect due to the limitations in the GPGPU computation model. This makes our algorithm semi-automatic, as it requires the user to specify a fixed number of iterations for the L1 contraction method based on their satisfaction with the resulting skeleton.

We encountered a second problem which is that the parallel vertex sampling is biased due to differences in the input medial surface density. We mitigate this effect by introducing a relaxed contraction constraint to the L1 phase as proposed by Huang et al. [6]. This constraint relies on weighting the input medial surface vertices based on density and is shown in:

$$d_i = 1 + \sum_{i' \in I \setminus \{i\}} \theta(||c_i - c_{i'}||)$$
$$x_j^{k+1} = \frac{\sum_{i \in I} c_i \alpha_{ij}^k / d_i}{\sum_{i \in I} \alpha_{ij}^k / d_i} + \mu \sigma_j^k \frac{\sum_{j' \in J \setminus \{j\}} (x_j^k - x_{j'}^k) \beta_{jj'}^k}{\sum_{j' \in J \setminus \{j\}} \beta_{jj'}^k} \quad (3)$$

The density weight is then incorporated to the contraction, so denser regions have a more relaxed contraction speed than sparser regions.

5 Conclusion

In this paper we presented a hybrid approach for fast curve skeleton extraction from 3D meshes and point clouds built on the GPGPU platform. Our method combines reliable

algorithms from the state of the art domain and introduces modifications which improve said algorithms and adapt them for a GPGPU environment. We demonstrated the capability of our implementation to run in near-real time with satisfying results.

In the future we would like to improve the overall shortcomings of our algorithm and apply enhancements to further increase the quality of our results.

References

[1] L1 skeleton extraction reference implementation - author home page. `http://shihaowu.net/`.

[2] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. In *ACM transactions on graphics (TOG)*, volume 27, page 44. ACM, 2008.

[3] Andrea Baldacci, Rastislav Kamenickỳ, Adam Riečickỳ, Paolo Cignoni, Roman Ďurikovič, Roberto Scopigno, and Martin Madaras. Gpu-based approaches for shape diameter function computation and its applications focused on skeleton extraction. *Computers & Graphics*, 59:151–159, 2016.

[4] Harry Blum et al. A transformation for extracting new descriptors of shape. *Models for the perception of speech and visual form*, 19(5):362–380, 1967.

[5] Nicu D Cornea, Deborah Silver, and Patrick Min. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization & Computer Graphics*, (3):530–548, 2007.

[6] Hui Huang, Shihao Wu, Daniel Cohen-Or, Minglun Gong, Hao Zhang, Guiqing Li, and Baoquan Chen. L1-medial skeleton of point cloud. *ACM Trans. Graph.*, 32(4):65–1, 2013.

[7] Andrei C Jalba, Jacek Kustra, and Alexandru C Telea. Surface and curve skeletonization of large 3d models on the gpu. *IEEE transactions on pattern analysis and machine intelligence*, 35(6):1495–1508, 2013.

[8] In-Kwon Lee. Curve reconstruction from unorganized points. *Computer aided geometric design*, 17(2):161–177, 2000.

[9] Lei Li and Wencheng Wang. Improved use of lop for curve skeleton extraction. In *Computer Graphics Forum*, volume 37, pages 313–323. Wiley Online Library, 2018.

[10] Shengren Li and Nina Amenta. Brute-force k-nearest neighbors search on the gpu. In *International Conference on Similarity Search and Applications*, pages 259–270. Springer, 2015.

[11] Jaehwan Ma, Sang Won Bae, and Sunghee Choi. 3d medial axis point approximation using nearest neighbors and the normal field. *The Visual Computer*, 28(1):7–19, 2012.

[12] Kálmán Palágyi and Attila Kuba. A parallel 3d 12-subiteration thinning algorithm. *Graphical Models and Image Processing*, 61(4):199–221, 1999.

[13] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer*, 24(4):249, 2008.

[14] Andrea Tagliasacchi, Thomas Delame, Michela Spagnuolo, Nina Amenta, and Alexandru Telea. 3d skeletons: A state-of-the-art report. In *Computer Graphics Forum*, volume 35, pages 573–597. Wiley Online Library, 2016.