ISSN 2464-4617 (print)
ISSN 2464-4625 (online)
Computer Science Research Notes - CSRN 3301
http://www.wscg.eu
WSCG 2023 Proceedings

Using the Adaptive HistoPyramid to Enhance Performance of Surface Extraction in 3D Medical Image Visualisation

Antony Padinjarathala
School of Electronic Engineering
Dublin City University
Dublin 9, Ireland
antony.padinjarathala2@mail.dcu.ie

Robert Sadleir
School of Electronic Engineering
Dublin City University
Dublin 9, Ireland
robert.sadleir@dcu.ie

ABSTRACT

There are currently a range of different approaches for extracting iso-surfaces from volumetric medical image data. Of these, the HistoPyramid appears to be one of the more promising options. This is due to its use of stream compaction and expansion which facilitates extremely efficient traversal of the HistoPyramid structure. This paper introduces a novel extension to the HistoPyramid concept that entails incorporating a variable reduction between the HP layers in order to better fit volumes with arbitrary dimensions, thus saving memory and improving performance. As with the existing HistoPyramid techniques, the adaptive version lends itself to implementation on the GPU which in turn leads to further performance improvements. Ultimately, when compared against the best performing existing HistoPyramids, the adaptive approach yielded a performance improvement of up to 20% without any impact on the accuracy of the extracted mesh.

Keywords

Marching cubes, surface extraction, HistoPyramid, Parallel Processing, CUDA

1 INTRODUCTION

3D Medical imaging often involves extracting surfaces from volumetric datasets obtained using modalities like MRI & CT. These datasets are stored as 2D images of pixels which when combined build a 3D volume of voxels.

The surfaces in a medical image have constant density and so are called iso-surfaces. The Marching Cubes Algorithm (MCA) [Lor87], developed by Lorensen & Cline can be used to extract these surfaces. It is a robust algorithm that subdivides a volume into smaller $2 \times 2 \times 2$ overlapping neighbourhoods and processes each neighbourhood individually. This 'divide and conquer' approach is well suited to parallel processing. This method is consistent and accurate but can be quite slow.

The MCA performs computations on the whole volume, however, only a fraction of this volume will produce geometry. Alternative solutions to the MCA attempt to reduce the number of unnecessary computations that are performed. In this paper, one such solution that will be looked at is the HistoPyramid (HP) [Dyk08; Dyk10; Smi12]. The HP is a data structure that is used to transform the problem from input-centric to output-centric. This is a much more efficient solution. However, the HP approach can be further improved to be more space-efficient and further optimised.

This paper introduces the Adaptive HistoPyramid (AHP), as a novel alternative solution to existing formations of the HP. The AHP further enhances and extends the HP such that it can operate on any arbitrary volume without the need for extra padding. It is a more flexible structure in comparison to the standard HP. Less padding should have the effect of reducing the memory required to store the AHP and also the amount of time to create the AHP.

Both the HP and AHP can be shown to be highly parallelizable and will be implemented using NVIDIA's Compute Unified Device Architecture (CUDA) which is a parallel programming platform [NVI20]. With this all of the steps involved can be executed on the GPU alone without requiring additional transfers between the GPU and CPU which adversely affects performance.

2 PRIOR WORK

2.1 Marching Cubes and Extensions

The MCA introduces key concepts that make it ideal for the task of surface extraction. First, the

3D grid of voxels is split into $2 \times 2 \times 2$ neighbourhoods formed by adjacent voxels. From this each individual neighbourhood is matched against one of the 15 Marching Cubes which approximate the geometry within the neighbourhood.

The MCA does have some limitations in its ability to handle 'sharp' shapes and those with ambiguity in certain sections. For this reason, more sophisticated methods based on the MCA such as Marching Cubes 33 [Che95] and Neural Marching Cubes [CZ21] were proposed. Marching Cubes 33 aims to improve the MCA by introducing more patterns so that there are more possible scenarios that can be modelled in each cube, thus reducing the potential for ambiguity.

Alternatively, the Neural Marching Cubes algorithm uses a cube with internal vertices and deep learning in order to create a model to recover more accurate geometry from the cube. While any of these quality-centric marching cubes options would be compatible with the performance-centric approach that is the focus of this paper, the standard MCA will be used as the starting point in order to provide the most accessible description of the technique that is being proposed.

2.2 Prefix Sum (Parallel Scan)

The Prefix Sum [Har07] is a common parallel algorithm. It can perform many parallel additions very quickly. It has many uses but the one that is most applicable for these purposes is stream compaction. Stream compaction involves modelling the problem as a series of streams and then removing unwanted or unnecessary streams. This is done by turning the MCA into streams that produce triangles with several streams for each neighbourhood and then culling the streams that don't produce triangles. The scan done with the Prefix Sum creates a cumulative sum of triangles over the entire volume and then a scatter operation is executed that selects only the streams that produce output triangles.

The MCA processes every neighbourhood which translates to outputting five streams per neighbourhood. In practice, the average number of triangles per neighbourhood will be less than one as typical medical scans feature a large amount of empty space. The Prefix Sum method uses stream compaction which ensures there are exactly as many streams as triangles. Until version 11.6 of CUDA, a sample was packaged with the CUDA toolkit that extracted iso-surfaces using the Prefix Sum. This did not scale well with large volumes so a better solution is needed.

2.3 HistoPyramid

Another possible solution is using the HP method. The HP extends the prefix sum and creates a new data structure, a tree structure that maintains distribution information, instead of using a single compacted array of streams. Similar to the Prefix Sum it uses stream compaction to reduce the problem into a stream of triangles. The HP base layer has an entry for each neighbourhood which is the number of triangles that will be produced by the neighbourhood. It makes upper layers by summing entries at each layer until there is only one entry at the top layer which is the total number of triangles. These triangles are passed through each layer of the HP to find the position and dimensions of that triangle. Often a neighbourhood may produce multiple triangles, but each traversal of the HP produces exactly one triangle. These triangles make up the surface as before.

2.4 Parallel Processing

A great benefit associated with the MCA is that it is highly parallelizable. Even running the standard MCA on a GPU can result in a reduction in processing time. Parallel implementations of the MCA have been tested [Arc11] and can take between four and fifty times less time to process by utilising the GPU. However, memory issues may become a problem which limits the benefit that can be gained when processing larger volumes.

One reason is that parallel processing on a GPU tends to require that each parallel process has a fixed allocation of memory to deal with all possible input/output scenarios. i.e., each neighbourhood outputs the maximum number of triangles which is five even though in most cases no triangles will be produced. This will inevitably lead to inefficiencies in terms of memory requirements and processing speed. On some GPUs it may not be possible to process larger volumes in a single pass without having to do additional slow GPU to CPU transfers creating a bottleneck.

The HPs are highly parallelizable and the output is a compact set of triangles so it doesn't require as much memory despite having to store a full

HP data structure as well as storing the full set of underlying $2 \times 2 \times 2$ voxel neighbourhoods. The layers of the structure are composed of summing sections of previous layers, a process which can be carried out in a computationally efficient manner. The HP is a Pyramid of partial sums. With the top layer being the sum of all entries in the base layer. This is similar to a Prefix Sum operation that outputs intermediate layers. The HP layers split the work done by the Prefix Sum method and allow it to be traversed much faster while taking the exact same time to create. And, since the traversal is a simple algorithm that iterates over a number of triangles it is easy to implement in parallel.

3 DESIGN AND IMPLEMENTATION

3.1 Marching Cubes Algorithm

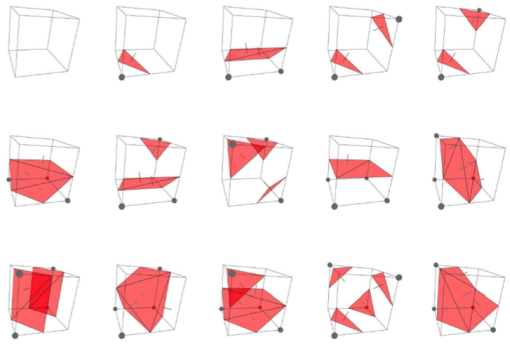

Figure 1: The 15 marching cubes cases.

To implement the MCA, the 3D grid of voxels is first split into $2 \times 2 \times 2$ neighbourhoods formed by adjacent voxels. Each neighbourhood is a $2 \times 2 \times 2$ overlapping region from the original dataset. Every voxel forms a corner of the neighbourhood and has a domain-specific value for its density.

3.1.1 *Thresholding*

For each neighbourhood, voxel densities are compared with a threshold. This threshold is the density of the surface to be extracted. Voxels are thus assigned as internal or external to this surface.

3.1.2 *Identify Intersected Edges*

If a neighbourhood contains voxels that are both external and internal to the surface, then it must be intersected by the surface. There are 256 ways that a neighbourhood can be intersected. However, using rotations and complementary cases, this can be reduced to only 15 unique scenarios as shown in Fig. 1. Using a lookup table produced by Lorensen and Cline, a set of intersected edges was found in that neighbourhood that must contain vertices of the surface.

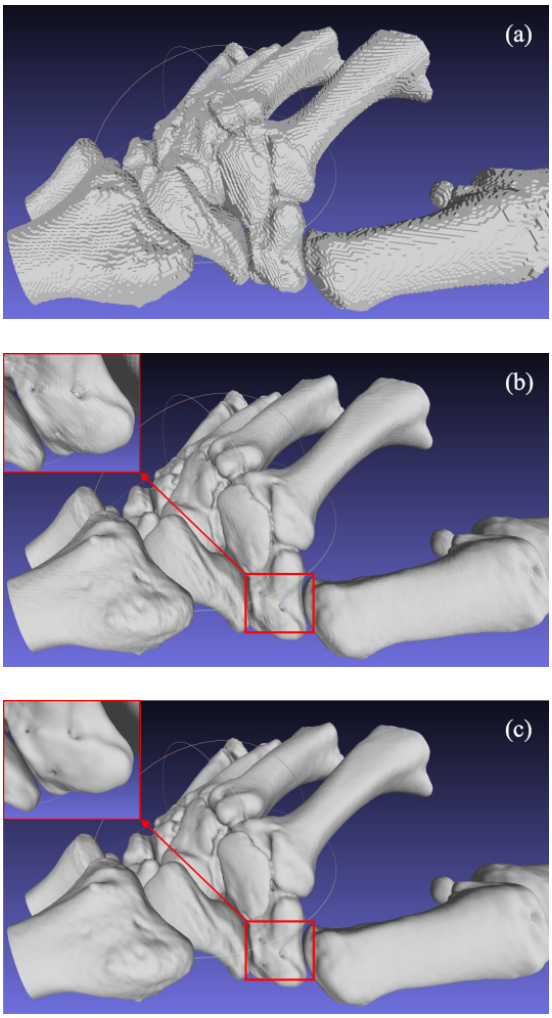

Figure 2: Evolution of the mesh extracted using the MCA. (a) A surface extracted before any interpolation occurs. (b) A surface extracted without calculating vertex normals. (c) A fully extracted surface with vertex normals for lighting calculations.

3.1.3 *Interpolation of Points*

Linear Interpolation is applied using the densities of the voxels on either end of an intersected edge and the threshold value to approximate the exact position of the vertex along this edge. Interpolation stops the rendered surface from looking 'blocky' as in Fig. 2(a) and instead look as they do in Fig. 2(b). After this the vertices are triangulated.

3.1.4 *Calculating Vertex Normals*

At this point the mesh is accurate but does not have the information required to facilitate per fragment lighting calculations. Consequently, normals must be calculated for each vertex. This is done for each voxel by finding the rate of change in voxel density along each axis local to that voxel using neighbouring voxels. Vertices lying on an edge containing those two voxels have their normals found using linear interpolation as before.
The results were examined to ensure that a smooth, high detail surface was produced as illustrated in Fig. 2(c). After the MCA was implemented on the CPU, the process was repeated on the GPU using CUDA and the resulting surface was verified to be identical to the surface produced by the CPU bound version of the algorithm.

3.2 HistoPyramid

The HistoPyramid is a data structure that will allow a faster extraction phase and is ideal for GPU implementations. It requires additional setup time; however this time is negligible in comparison to the gains made during the extraction phase.

3.2.1 *First Pass*

As before, the volume is split into overlapping neighbourhoods and thresholding is applied. However, the voxels from this are used with the lookup tables to quickly calculate the number of triangles that will be produced by each neighbourhood without performing any time-consuming calculations.

3.2.2 *Constructing the HP*

It has been suggested that accessing the data as a series of tiled 2D slices [Har07; Dyk08] rather than the original 3D sub-volumes has the potential to significantly reduce the computational overhead associated with indexing through the HP data . The HP base layer in this case will be a 2D array with each entry representing the number of triangles in a given neighbourhood. The layer above this is constructed by summing entries in the layer below. This reduces the size of each upper layer by a factor of four until one entry remains at the top layer that contains the total number of triangles produced by the surface.

This is a reduction factor r of four or 2×2. It should be noted that for this reduction to be possible, the bottom layer must have sides of equal length that are of size 2^k and will form $k+1$ layers. The layers are padded to fit these constraints. The formation of the HP can be thought of as a Prefix Sum with Intermediate Partial Sum layers.

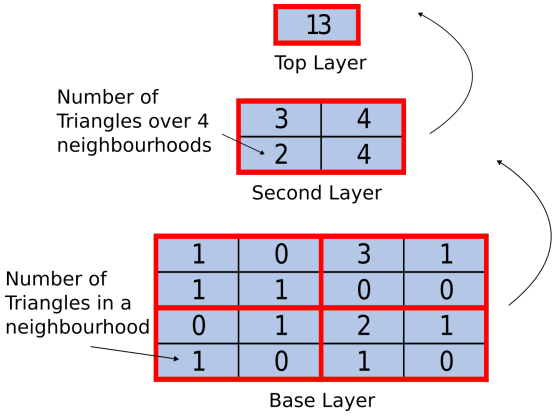

Figure 3: Construction of a 2D HP with 16 entries in its base layer.

For the above case, the reduction factor is four which equates to a 2×2 square. However, it is possible to sum over any area that is an $n \times m$ rectangle. This requires that the base layer is of area $n^k \times m^k$ and that there are $k+1$ layers. Above, in Fig. 3, a 4×4 area is used to represent a possible $2 \times 2 \times 4$ volume.

3.2.3 *Traversing the HP*

Next the HP is traversed to find each triangle in the surface. The number of triangles equals the top entry since it is a sum of all base layer entries. The HP is traversed using indices from 0 to $N-1$, where N is the top entry in the HP to find all N triangles. Instead of iterating on an entire volume like the MCA, the HP iterates on all the triangles in a volume and then the HP is traversed to find the exact position of each triangle.

This is done by finding which sub-area of a lower layer that a triangle belongs to. Once the base layer is reached, the neighbourhood containing the triangle along with its corresponding vertex positions are found. Each parallel process in this is a stream that produces exactly one triangle. This stream compaction transforms the basis of the problem such that it is based on the number of triangles instead of the size of the volume. This new basis means that the complexity is related to the length of the output which is why it is referred to as an output-centric method. MCA on the other

hand has complexity related to the input volume making it input-centric.

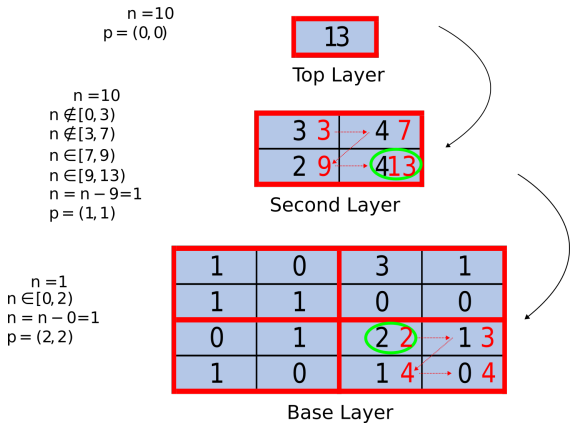

Figure 4: Traversal of a HP to find the location of the 10th of 13 triangles in a surface.

In Fig. 4, the n^{th} triangle is being extracted for $n = 10$. The entries in each 2×2 square are ordered starting at the top left and following the arrow in the figure. At each layer a cumulative sum of the entries is calculated as shown by the number in red to calculate the ranges contained within each entry. At the top layer, n must be within the area beneath the first entry. At the second layer the ranges contained by each entry are created. In the example, n belongs to the range created by the last entry. n is updated to be local to that entry and given a value of one. The position value p is updated also to point at this entry by multiplying the position vector by the reduction factor. In the base layer, n is within the first entry. The triangle is within this neighbourhood in the base layer. This neighbourhood produces multiple triangles and n decides which triangle to extract from the neighbourhood.

Unlike the construction phase, the number of instructions to retrieve the position of the triangle in a lower layer is not always constant since the process may need to check multiple entries. This can potentially result in slightly slower processing because conditional logic may lead to branch divergence and as a consequence the streams may vary in terms of the time required to execute. These processes run on threads, and in CUDA, threads run in collections of 32 called warps. These warps must be synchronised and if the execution time varies greatly the performance is affected negatively. Execution of a warp will continue until its longest running member thread has completed its operation.

3.3 HP Modifications

Different formations of the HistoPyramid are possible by changing the way that reductions are applied.

3.3.1 3D HistoPyramid

The HistoPyramid can have 3D layers [Smi12]. In this scenario, the 3D base layer is not flattened into a tiled area. Instead, the 3D data is processed directly with constraints similar to those that apply to the 2D case. It is possible then to divide any volume of $n^k \times m^k \times l^k$ where n, m and l are the reduction factors in each dimension and $k+1$ is the number of layers. Only $2 \times 2 \times 2$ reduction factors were considered in this case which is equivalent to an overall reduction factor of eight.

3.3.2 1D HistoPyramid

This HistoPyramid can also be flattened down to 1D layers [Dyk10]. This has the potential to reduce the indexing overhead even more than the 2D case. It also loosens the constraints on the size of the base layer to be any length that can be written in the form r^k where r is the reduction between each layer and $k+1$ is the number of layers. A series of 1D HPs were evaluated and the best performing of these were considered further. As observed from experiments carried out, the best performing HPs had reduction factors of five or eight.

3.4 Adaptive HP

The HP makes the traversal phase and surface extraction phase of the process optimally efficient as they are output-centric. This is not the case for the first pass over the volume and the creation of the HP. As noted from repeated experimentation, most of the processing time is associated with the input-centric computations. These computations took on average 84% of the total computation time for surface extraction. The input-centric computations took more time for sparser datasets, especially where a large amount of padding of the HP structure is required to accommodate the HistoPyramid size constraints, while the opposite was the case in datasets that contained more extensive meshes. Reducing the effect of input-centric processes requires removing the padding created from the HP. To do this, the AHP

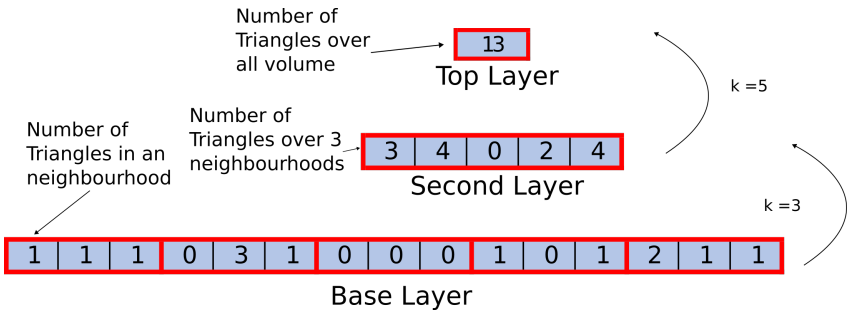

Figure 5: AHP removing the need for padding in a volume of fifteen neighbourhoods by using reduction factors of three and five.

can be used.

Starting with a 1D HP, the AHP takes advantage of the fact that a constant reduction between layers is not necessary. This is possible as long as each layer is treated the same at construction and traversal. Fig. 5 shows how the example used in Fig. 3 can be reshaped in the form of an AHP. With this the layers have a reduction of three and then five to make up the size of exactly fifteen which could not be expressed in the form r^k or by any of the HPs explored thus far.

The AHP must be traversed in the reverse order to construction to find the correct positions. As seen in Fig. 6, the traversal is identical to the HP and selects the same entry for $n = 10$ and for any other n, the main difference is how the layers are divided. The AHP aims to reduce the padding by a range of different reduction factors and finding an arrangement of these that will produce the least padding. The constraints on this structure are that the base layer is of size $\prod_0^K r_k$ where r_k is the reduction at layer k.

Any size base layer can theoretically be accommodated given that it is not a prime number. This significantly reduces the need for padding compared to other similar techniques. In fact, it could be possible to take only a subset of an overall volume using this method and thereby reducing the memory requirements further. Larger reductions at each layer will produce fewer layers however it will also increase the amount of data at each layer which would slow down processing at that layer.

A method is required for determining the reduction factors. The maximum reduction factor found in existing works is eight [Smi12] but tended to be lower e.g. four or five [Dyk08; Dyk10]. With the AHP it will be possible to use larger reductions without possibly creating a large amount of padding. This is because, for a regular HP the HP can only take a limited set of sizes which are dependent on the reduction factor and this set of sizes becomes sparser as the reduction factor increases as the base layer size must be exactly r^k. Conversely, the AHP uses multiple reduction factors so does not have the same constraints on size as is the case with HP, so this is not an issue.

The reduction factors for a given volume were selected from a set of numbers ranging from 4-16. Any factor is a valid choice, even so, it is important to select factors that are not too large as it was determined that this created layers with a large possibility of branch divergence which

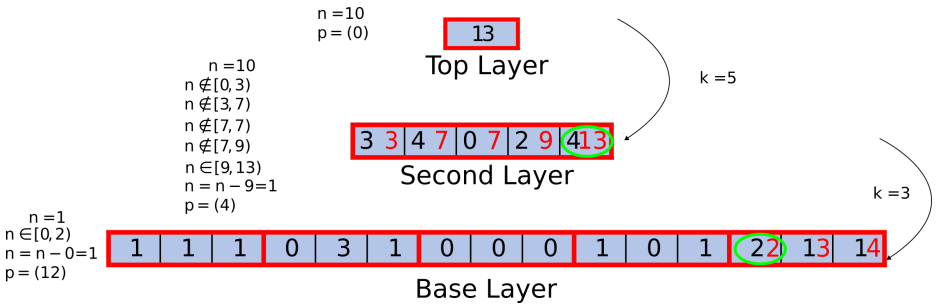

Figure 6: Traversal of a AHP to find the 10th of 13 triangles of a surface.

would negatively affect performance.

The reduction factors were determined empirically from the range of values from this set for this initial implementation. This was deemed adequate as it was generally possible to find an outcome that was close to ideal with very little computation overhead. The HP could be a possible configuration of the AHP so, in theory, an ideal AHP would at most take as much storage as most space-efficient regular HP.

4 RESULTS AND ANALYSIS

The algorithms were run on nine different datasets [Sta87; Ack98; Kik14; McC14] obtained from a range of sources. Table 1 presents a summary of each of the methods compared with the standard MCA as well as the additional memory requirements to store the base layer of the HP over all datasets. Additionally, detailed results are provided for three exemplar datasets that show the performance of the technique across a range of different modalities, dataset dimensions and surface morphologies.

All algorithms were evaluated on a PC with an Intel Core i7 2.60GHz CPU with six cores and 32 GB RAM, with a 4GB NVIDIA Quadro P2000 Graphics Card. The algorithms were implemented using version 4.8 of the .NET framework which incorporated CUDA 11.6 through the NuGet Package ILGPU [ILG22].

Table 1 shows the average relative performance from timing the various algorithms. The standard MCA implementations are denoted as MCA CPU and MCA GPU. 1D HP implementations are denoted by HP and then a number relating to the reduction. Only 2 of these were considered, HP5 and HP8, which have reduction factors of five and eight respectively. These two were the best performing of the 1D HPs. AHP refers to the adaptive approach and the final two algorithms, HP2D and HP3D refer to implementations that have 2D or 3D layers respectively.

The HistoPyramids reduce the number of computations and performs these computations in parallel, so a large improvement is expected. This approach was found to be 50 to 500 times faster than the MCA to extract the same surface. The benefit of this technique is clear from these results.

The AHP performs better than any of the other implementations. On average it performed 4.5% faster than the next best performing method for each individual case, for the volumes that exceeded $256 \times 256 \times 256$ this improvement increases to an average of 10%. In the best case from the dataset this improvement increased to 20%. The performance of the AHP was also found to improve as the size of the volume increased.

In addition to this the AHP was generally found to be the best method across datasets. In one outlier case, the AHP was not the top performer as it took slightly longer than the HP5 method. In this case, the AHP used a large amount of padding and has not tended to the best formulation of layers. The HP5 method uses less padding for this case. There is at least one better solution to the arrangement of AHP layers since any 1D HP is also a possible solution to the AHP. Using a different algorithm for determining the reduction factors of the AHP would reduce the padding used and improve performance further. This outlier demonstrates that padding is a relevant factor in performance.

The 1D implementation tends to perform better than 2D or 3D alternatives. Using 1D layers eases the constraints on the shape and size of the original volume. Additionally, the index overhead of using 3D or 2D indices for each layer is not present with the 1D implementation. For irregularly shaped data this can be particularly apparent. The AHP tends to use even less padding especially when averaging results over several datasets. It will produce a consistently low amount of padding while with other forms of the HP, the amount of padding can vary significantly from dataset to dataset. This may be because it can model any arbitrary volume easily. Because of this the AHP is particularly fast in the construction phase.

5 CONCLUSIONS

The AHP evaluated was an initial unrefined implementation and yet it regularly performed better than the other methods. It enhances the HP using a variable reduction factor between layers of the data structure. This results in a performance boost of up to 20% with an additional benefit of using less of the finite memory available on a GPU. These performance boosts do not sacrifice the accuracy of the mesh extracted. In the outlier case, where the AHP doesn't give a benefit over a 1D HP, which is quickly calculable before creating any structure, a hybrid solution might be used

	MCA CPU	MCA GPU	HP5	HP8	AHP	HP2D (HP4)	HP3D (HP8)
Average Performance over 9 datasets							
Relative Time to MCA	1.000000	0.244867	0.005974	0.006015	0.005722	0.006558	0.007677
Padding Required			86.56%	139.06%	8.98%	105.93%	258.50%
MRHead (130 x 256 x 256)							
Relative Time to MCA	1.000000	0.341110	0.004098	0.004229	0.004232	0.004216	0.004643
Padding Required			14.85%	98.44%	22.03%	98.44%	98.44%
F_Head (234 x 512 x512)							
Relative Time to MCA	1.000000	0.229836	0.006884	0.006498	0.005701	0.009239	0.011601
Padding Required			300.45%	243.90%	0.16%	9.48%	119.78%
Wrist CT (251 x 440 x 440)							
Relative Time to MCA	1.000000	0.485414	0.009634	0.010373	0.009393	0.015005	0.018657
Padding Required			0.49%	177.72%	7.60%	38.43%	177.72%

Table 1: The relative time taken for each method relative to the Baseline MCA CPU and the extra padding required for each HP and AHP. The best performing algorithms for each surface are highlighted.

instead that could force the use of the most appropriate conventional HistoPyramid-based approach.

There are some areas in which the AHP could be improved. Namely, the final structure is often not the optimal solution and, moreover, uses more padding than needed. This happens because the method for selecting factors reaches an acceptable solution but there are often better solutions available, possibly by refining an initial guess or by using a more analytical approach. Additionally, the AHP can readily select subsets of the volume. For example, a pre-processing step could crop only important parts of the volume as much of it is empty space. Or a computer vision task might quickly detect subvolumes from larger datasets that might need to be selectively extracted, something that the AHP would easily accommodate. Future work might see a better method for selecting factors while also taking advantage of the AHP's ability to model arbitrary volumes to see more significant gains.

As with the HP, the AHP is a parallel-first approach making it ideal for GPU implementations. This avoids any CPU to GPU transfers which will cause a bottleneck. Each step of the HP consists of only additions or comparisons. It is therefore easy to comply with the SIMD or SIMT models of parallel computing and ensure that maximum use is being made of the GPU.

REFERENCES

[Ack98] Ackerman, M. "The Visible Human Project". In: *Proceedings of the IEEE* 86 (Mar. 1998), pp. 504–511.

[Arc11] Archirapatkave, V., Sumilo, H., See, S. C. W., and Achalakul, T. "GPGPU Acceleration Algorithm for Medical Image Reconstruction". In: *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications*. 2011, pp. 41–46.

[Che95] Chernyaev, E. V. "Marching Cubes 33: Construction of topologically correct isosurfaces". In: (Nov. 1995).

[CZ21] Chen, Z. and Zhang, H. "Neural Marching Cubes". In: *ACM Trans. Graph.* 40.6 (Dec. 2021).

[Dyk08] Dyken, C., Ziegler, G., Theobalt, C., and Seidel, H.-P. "High-speed Marching Cubes using HistoPyramids". In: *Computer Graphics Forum* 27.8 (2008), pp. 2028–2039.

[Dyk10] Dyken, C. and Ziegler, G. "GPU-accelerated data expansion for the Marching Cubes algorithm". In: Proc. PGU Technol. Conf., 2010.

[Har07] Harris, M., Sengupta, S., and Owens, J. "Parallel prefix sum (scan) with CUDA". In: vol. 39. Aug. 2007, pp. 851–.

[ILG22] *ILGPU*. https://www.ilgpu.net/.

[Kik14] Kikinis, R., Pieper, S. D., and Vosburgh, K. G. "3D Slicer: A Platform for Subject-Specific Image Analysis, Visualization, and Clinical Support". In: *Intraoperative Imaging and Image-Guided Therapy*. Ed. by F. A. Jolesz. New York, NY: Springer New York, 2014, pp. 277–289.

[Lor87] Lorensen, W. and Cline, H. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *ACM SIGGRAPH Computer Graphics* 21 (Aug. 1987), pp. 163–.

[McC14] McCormick, M., Liu, X., Jomier, J., Marion, C., and Ibanez, L. "ITK: Enabling Reproducible Research and Open Science". In: *Frontiers in neuroinformatics* 8 (Feb. 2014), p. 13.

[NVI20] NVIDIA, Vingelmann, P., and Fitzek, F. H. *CUDA, release: 10.2.89*. 2020.

[Smi12] Smistad, E., Elster, A., and Lindseth, F. "Real-Time Surface Extraction and Visualization of Medical Images using OpenCL and GPUs". In: Jan. 2012.

[Sta87] *The Stanford Volume Data Archive*. http://graphics.stanford.edu/data/voldata/.