VOXELIZED SKELETAL MODELING TECHNIQUES VIA COMPLEMENTAL SKELETONS

Tyler J. Williams, Prof. Mark A. Ganter, Prof. Duane W. Storti
University of Washington Mechanical Engineering Department
Seattle, WA/USA

ABSTRACT

Skeletal modeling is a modeling methodology based on finding and manipulating the medial surfaces of a 3D model. One of the key strengths of skeletal modeling is the simplicity of manipulating the thickness of objects, a task that can be difficult with standard 3D modeling techniques. However, due to the nature of medial surfaces, changes made to one section of a skeletal model can alter multiple regions on the surface of the resulting object, making it difficult to perform small edits to the model without adding more branches to the skeleton. This paper develops a novel twist on skeletal modeling: the 'Complemental Skeleton.' Through the usage of a complemental skeleton, localized changes to object thickness can be made to the model by modifying the radial data, allowing the modeler to leave the geometry of the skeleton unaltered. This feature greatly lowers the amount of work required to create asymmetrical edits.

Corresponding author: Tyler Williams (tjwill@uw.edu)

1. INTRODUCTION

There are two main types of skeletons that are used for 3D objects. The curve skeleton, a collection of 1D curves that follow the shape of the object, lends itself well to surface reconstruction, shape classification, and 3D animation. The curve skeleton lacks a well-defined methodology of creation, and, more importantly, is often a one-way transformation; the original object cannot be reconstructed from the curve skeleton alone [1-3].

The alternative to curve skeletons, often referred to as surface skeletons, are collections of 2D sheets, 1D curves, and 0D points that follow the shape of the object. They lend themselves well to surface modification, geometry editing, and thickness modification [4-8]. In contrast to the curve skeleton, the surface skeleton is strictly defined as the collection of the object's medial surfaces. In addition, the skeletal data can be used to completely recreate the original object, allowing for an object to be turned into its surface skeleton, modified in the skeletal state, and then transformed back into a fully 3D model [7-9]. An example of this process can be seen in Figure 1. The radial data associated with the skeleton can be scaled numerically in order to increase or decrease the overall thickness of the object.

Copyright 2020. Used by the Society of the Advancement of Material and Process Engineering with permission.

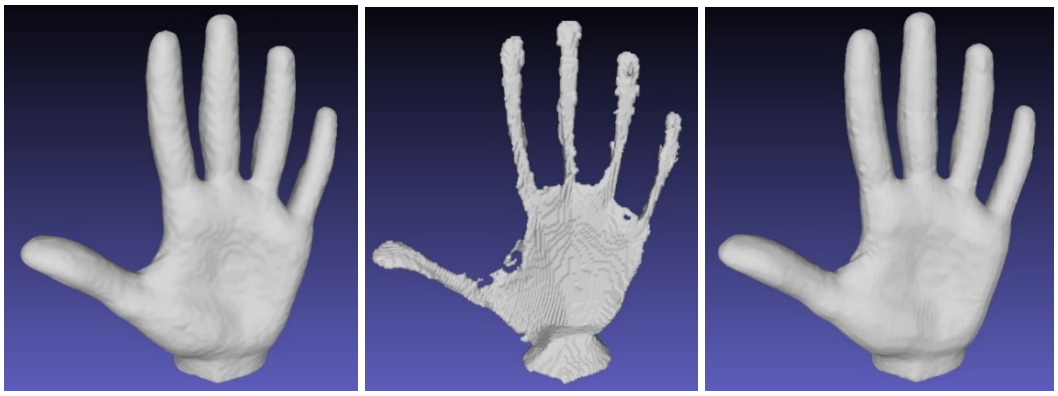


Figure 1. Left to Right: Input Model, Skeleton, Reconstruction from Skeleton [10].

1.1 Definitions

In this section, we define the foundational concepts and keywords used in this paper.

1.1.1 Medial Surface

The medial surface can be defined as the collection of points inside the object that have multiple 'equally closest' points on the surface of the object. For any point on the medial surface, the largest sphere centered on that point that is fully contained within the object will touch at least two points on the surface of the object. We will term these spheres to be 'Maximal Spheres'. For the purposes of this paper, the medial surfaces of an object can be seen as the 'skeleton' of the 3D model. The medial surface itself is a four-dimensional object, as each point on the surface has the three spatial coordinates (x,y,z) and a radial component (r). Recreating the object, termed 'refleshing' in this paper, consists of unioning together skeletal spheres centered on each point along the skeleton with their radii set to match the 'r' coordinate at that point in space [7].

1.1.2 Voxel Model

There are many ways that 3D geometry can be stored and manipulated. Voxel geometry stores 3D geometry based on a discrete grid, where each voxel in the grid is assigned a value that states whether the voxel is inside the object, outside the object, or on the surface of the object. Conventionally, internal voxels have negative values, external voxels have positive values, and voxels that land on the surface of the object are set to zero. The magnitude of the value is irrelevant for the purposes of storing the object itself, allowing for the values to store some additional data about the object, such as color, material properties, or distance information [9, 11]. An example of a relatively low resolution voxel model is shown in Figure 2. As only the sign of the voxel value determines the geometry of the object, a given matrix storing a voxel model is not unique.

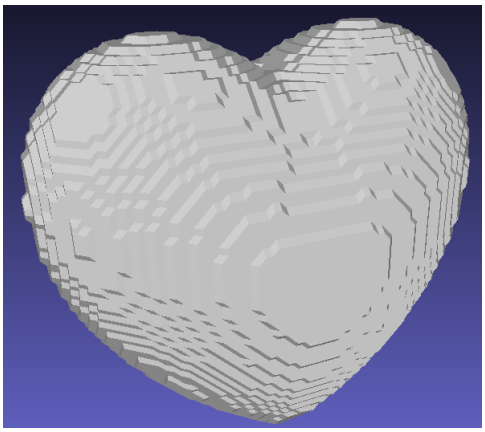


Figure 2. Taubin's Heart Surface in 75^3 Voxelspace, converted to an STL for rendering via the Marching Cubes algorithm. [13]

1.1.3 Signed Distance Field

For our purposes, a signed distance field (SDF) can be used to describe a voxel model. It follows the same sign conventions as standard voxel models, but now the magnitude of each voxel stores the distance from its point in space to the nearest point on the surface of the object [12]. This added restriction enforces uniqueness. Any given voxel model has only one SDF.

1.1.4 Parallel Computing

Normally, a computer functions in a serial manner, doing one task then moving to the next and repeating until it is finished. Parallel computing allows for tasks to be divvied up into independent computations that can be run simultaneously. This practice works particularly well when a single algorithm must be solved for each cell of a large matrix, allowing for all the solutions to be computed in lock-step and cutting down on computation time dramatically [9, 12]. For this paper, PyCUDA, a Python binding for the CUDA parallel computing platform developed by Nvidia, was used as our primary language for implementation.

1.1.5 Skeletal Data

For simplicity and accuracy in naming, the skeleton of a 3D model is going to be divided into two distinct parts. The 'Skeleton Geometry' will refer to the geometric location data. This represents the shape of the skeleton itself. The 'Radial Field' is going to refer to the radii of the maximal spheres as defined by the spatial coordinates. The 'Skeletal Data' refers to the combination of these parts, as both are required to recreate the object.

1.2 Features of Using Skeletal Modeling

Some editing tasks are most easily performed by transforming the model into its skeletal data, making modifications to the radial data, then refleshing the skeletal data to retrieve the desired object. Changing object thickness is often one of the easiest modifications that can be performed via skeletal modeling, as it consists of changing the magnitude of the Radial Field by adding some constant or multiplying any number of values by some constant. An example is visible in Figure 3.

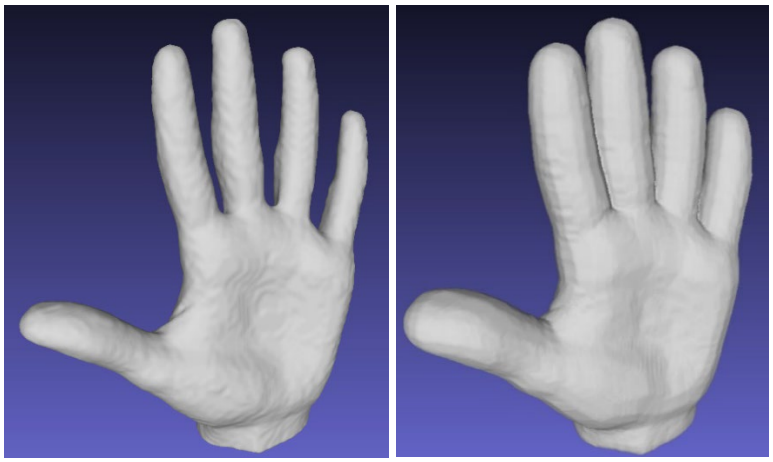


Figure 3. Left: Original input model [10], Right: Recreated after increasing each radial value by a constant.

It is important to note that changes made in this manner can affect multiple regions on the surface of the resulting model, as the Skeleton Geometry remains centered within the object. Given this limitation, using skeletal modeling techniques to make localized edits can be cumbersome. New surfaces, curves, or points would have to be added to the Skeleton Geometry if the desired modifications are intended to only affect specific regions. This is not as simple to perform as modifying the Radial Field.

1.3 Proposed Alternative

This paper introduces the concept of the Complemental Skeleton, a new type of skeletal entity that allows for localized modifications to the surface of the resulting object. The complemental skeleton lies between the original medial surface and the original surface and has its skeletal dataset. When combined with the original skeletal data, it allows for the complete object to be reconstructed. Images from the process are visible in Figure 4.

The key benefit of the complemental skeleton is that modifications applied to its Radial Field will generally only affect the regions of the surface on the corresponding side of the object, leaving the opposite side untouched. This property allows for asymmetric edits to be made without requiring modifications to the Skeletal Geometry.

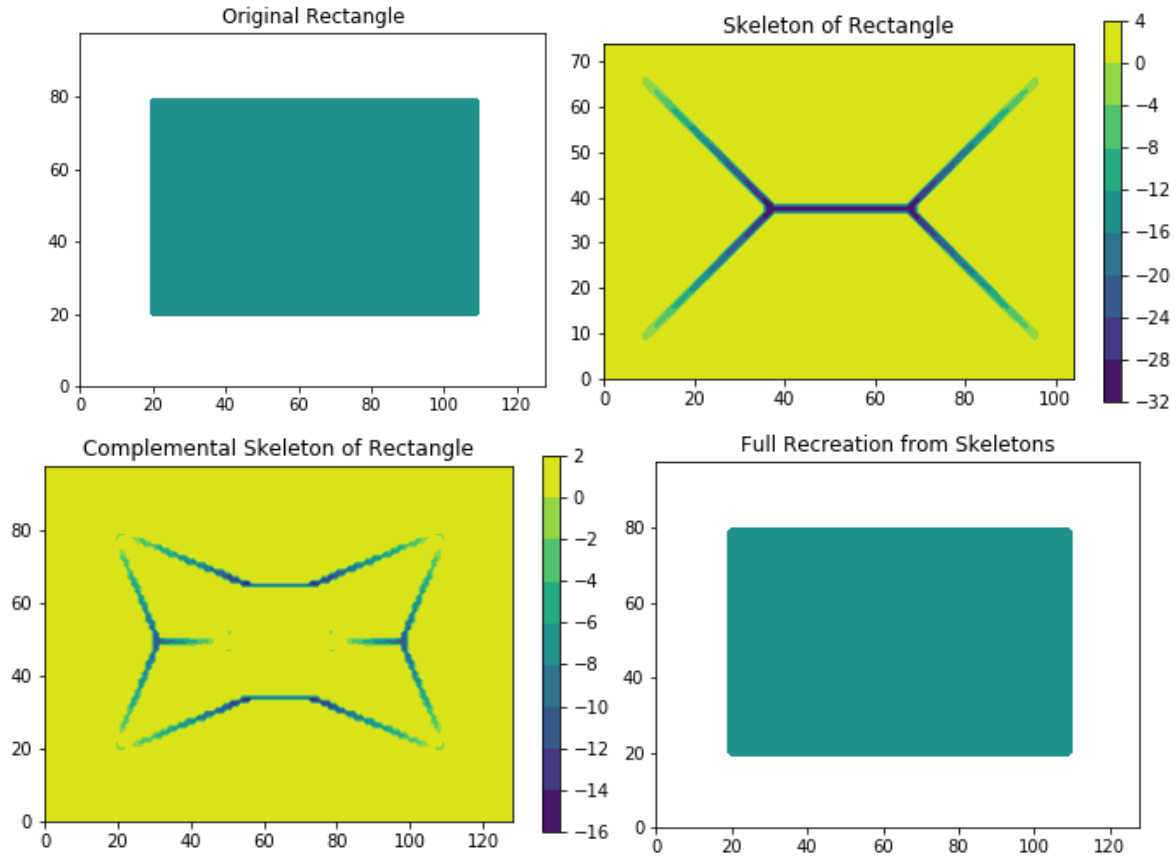


Figure 4. Left to Right, Top to Bottom: Original Object, Original Skeleton, Complemental Skeleton, Object recreated from Complemental Skeleton.

1.4 Motivations

As mentioned earlier, skeletal modeling is particularly useful in modifying the thickness of objects [4-8]. One application where model thickness is incredibly important is 3D printing, as 3D printers often have a minimum model thickness. Using the complemental skeleton in tandem with standard skeletal modeling techniques increases the utility of the methodology and allows for a larger variety of changes to be made to the model without switching from skeletal modeling to a different modeling technique. In short, the complemental skeleton has the potential to smooth the workflow of editing 3D models for 3D printing.

2. METHODS OF IMPLEMENTATION

For brevity in the main body of the paper, pseudo-code for each **bolded** algorithm is available in the appendix. All code used is freely available for download and inspection at https://github.com/tjwill95/complemental_skeletonizer. Additionally, this process also functions for finding the curve skeleton (and complemental curve skeleton) of 2D geometry. For clarity, the 2D geometry is used in the diagrams, but the process does operate on 3D data, as shown in the pseudocode.

2.1 Finding the Skeletal Data

Let's begin with a voxel model. The definition of the medial surface gives us a clue as to how to find it: If we can find the SDF of the object, the medial surface will pass through all points along the 'peaks' of the SDF. The rectangle and the SDF of the rectangle is shown in Figure 5.

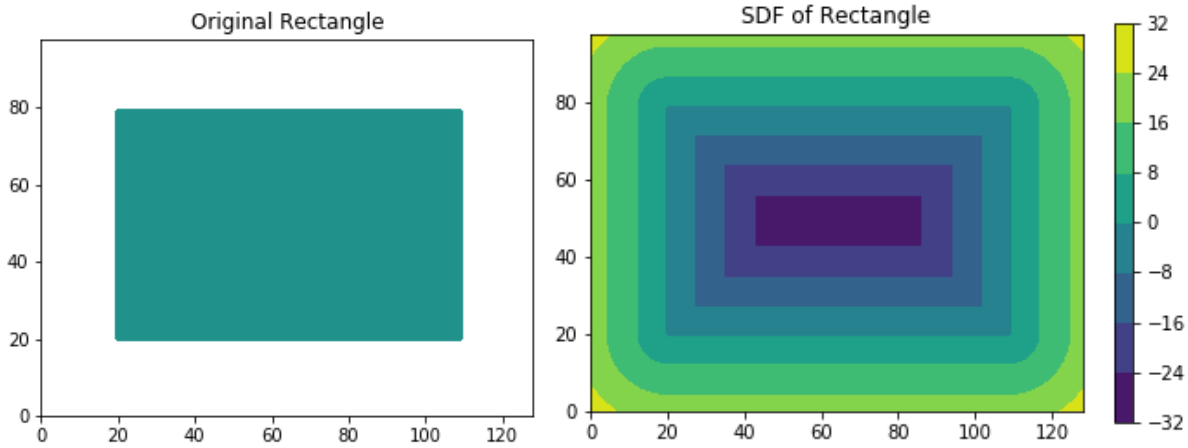


Figure 5. Left: Original Rectangle. Right: Contour Plot of the SDF of the Rectangle.

These 'peaks' can be found mathematically by discovering where the gradient of the SDF does not exist. The gradient of the rectangle's SDF is shown on the left in Figure 6, and it represents the Skeletal Geometry.

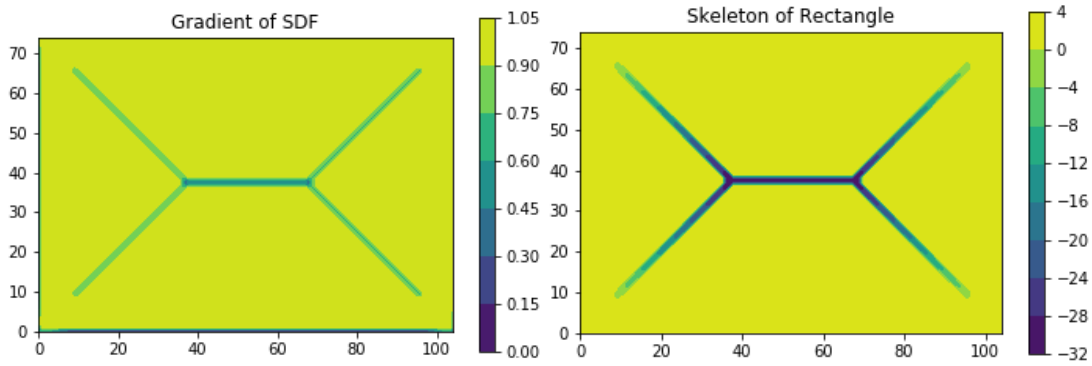


Figure 6. Left: Contour Plot of the magnitude of the gradient of the SDF of a rectangle. Right: Skeleton of a rectangle.

However, as voxel models are defined in discrete space rather than in continuous space, the gradient function will always return some value, even if inaccurate. To solve this problem, we will instead define the Skeletal Geometry as all points where the magnitude of the gradient of the distance field at that point is less than a threshold value. As the distance field usually has a gradient magnitude of one (moving one voxel away from the surface should increase the distance by one unit), we should set the threshold to be a value less than one. Empirically, the value of 0.85 leads to good results, as this includes the relevant points while removing many of the numerical errors that can arise in more complex models.

If we stop here, we are left with a loss of information, making it impossible to reconstruct the object without the Radial Data. As the magnitudes of the voxels that make up the Skeletal Geometry have no effect on the skeleton's shape, we can store the negative of the Radial Field in the voxels of the Skeletal Geometry, allowing for the original object to be recreated. A plot of the skeletal data is shown on the right of Figure 6.

2.2 Modifying the Skeleton

We now have both the Radial Field and the Skeletal Geometry. The simplest modification we can make is to the Radial Field. Changing the magnitude of the Radial Field at any point is going to result in changes to the thickness of the object based on that portion of the skeleton. Depending on the object, many voxels in the Radial Field may need to be modified in order to achieve the desired effect, especially if the magnitude is being reduced. The internal geometry of a model may be within the maximal spheres of several points on the Skeletal Geometry, so reducing the Radial Field at just one point on the skeleton may have nearly imperceivable effects on the resulting geometry. As can be seen in Figure 7, reducing the magnitude of the entire central section of the skeletal data to zero had a much lower effect on the resulting shape than increasing the magnitude of one point in the Radial Field.

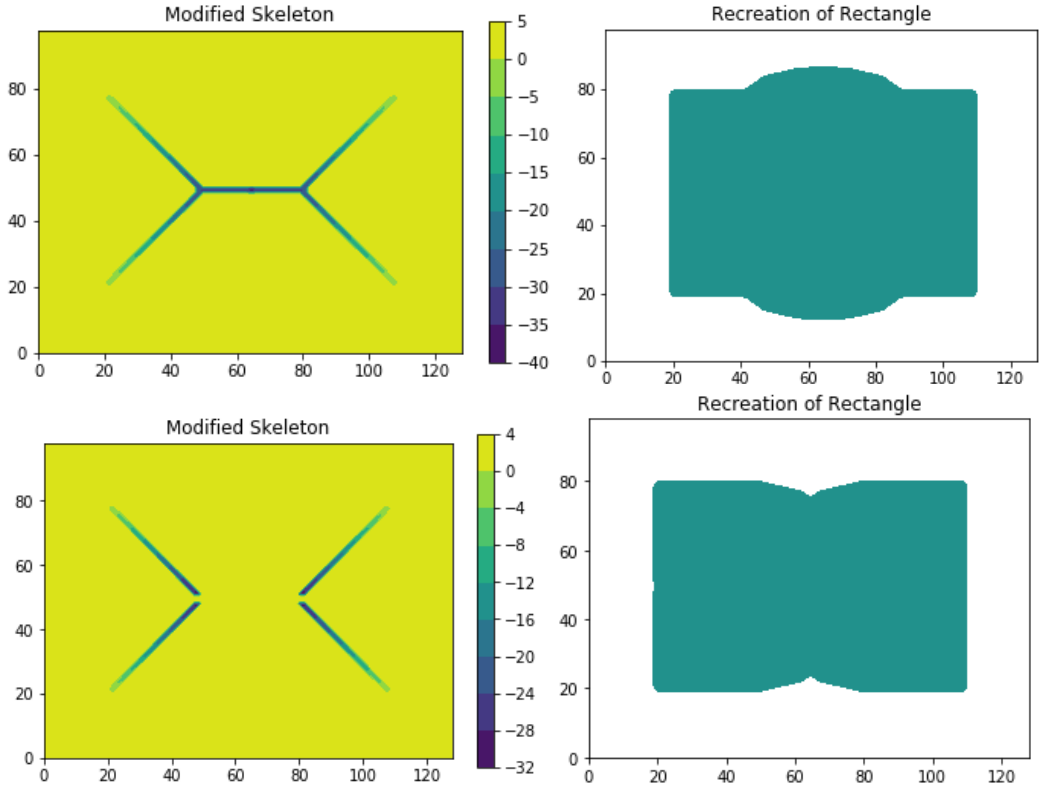


Figure 7. Left: Modified Skeletons. Right: Resulting Models. Top: The central voxels have their magnitudes increased. Bottom: The central row of voxels has been removed.

It should be noted that making any modifications to the Skeletal Geometry or the Radial Field may result in an object that no longer counts as a 'legal' skeleton, i.e. maximal spheres could be drawn that no longer center on the Skeletal Geometry. The true skeletal data of the

refleshed object is generally going to be different from the modified skeletal data used to generate it. This is especially apparent when adding voxels to the Skeletal Geometry, as the true medial surface will generally have branches to help define any protrusions, or may bend to accommodate for changes. An example where new voxels have been added to the Skeletal Geometry can be seen in Figure 8.

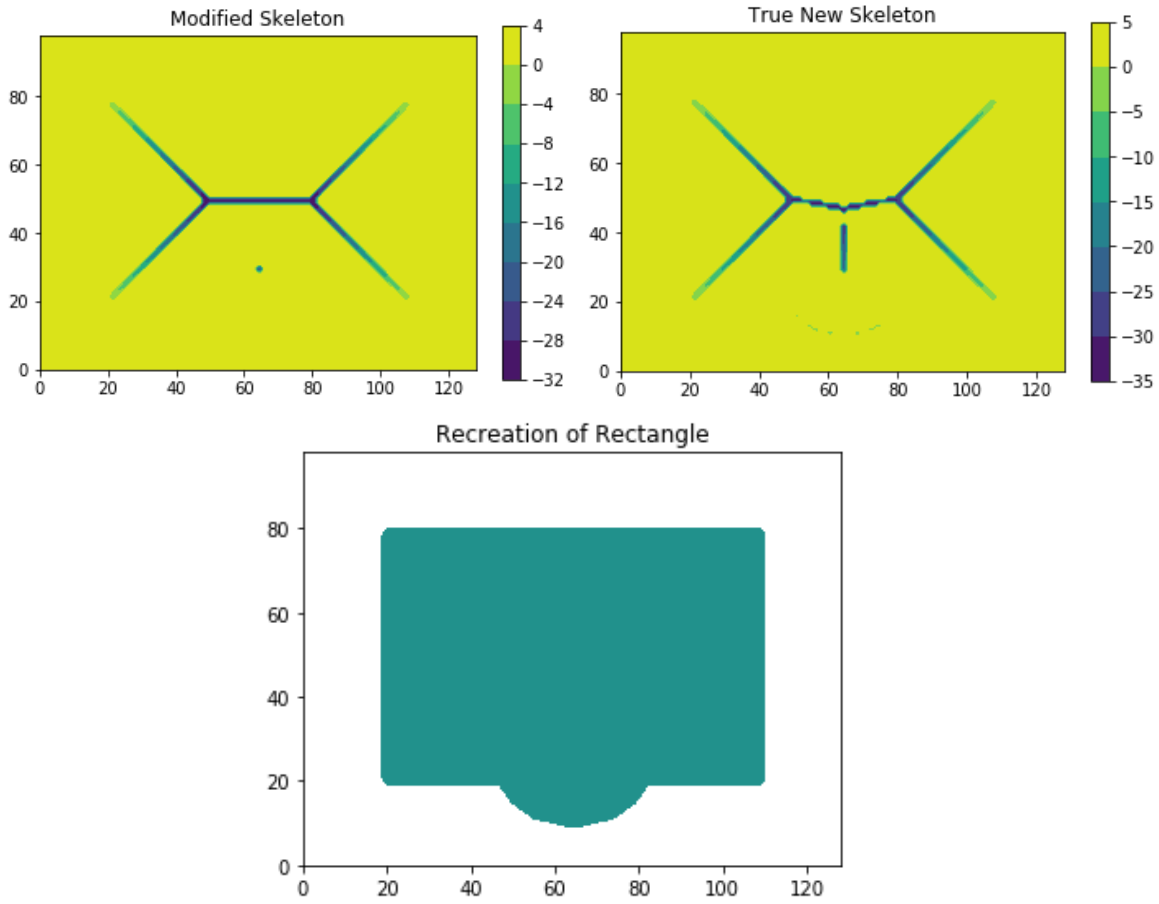


Figure 8. A dot has been manually added to the original skeletal data of the rectangle (top left), resulting in a circular bulge to the bottom of the shape (bottom). The 'true' skeletal data of the updated shape is shown to have more complex changes (top right).

As can be seen in Figure 8, new voxels had to be added to the skeletal data in order to produce the asymmetric modification to the rectangle.

2.3 Recreating the Object

Now that we have our adjusted skeleton, we would like to reconstruct our original object with the changes that would result from the modifications that we've made to the skeletal data. As mentioned earlier, we could generate a sphere centered on each voxel of the medial surface, using the values stored in the voxels to determine their radii, and union them all together. However, this solution is inelegant and computationally expensive. A more elegant solution is to propagate outwards from the medial surface. This process lends itself well to being parallelized and iterated, allowing for each voxel to compute its own intermediary value independently by

checking the input states of its neighbors. The resulting matrix is then sent back through the algorithm several more times as the data gets propagated outwards.

2.4 Finding the Complemental Skeleton

Changes to the Radial Field are often going to affect the resulting object in multiple regions. This enforced symmetry is not always the desired behavior, as it makes it difficult to implement asymmetrical changes without modifying the Skeletal Geometry itself, which can be cumbersome for the modeler. The complemental skeleton, as mentioned earlier, acts as an intermediary between the object's skeletal data and the complete object. The complemental skeleton of the afore shown rectangle is visible in Figure 9 in combination with the original skeletal data.

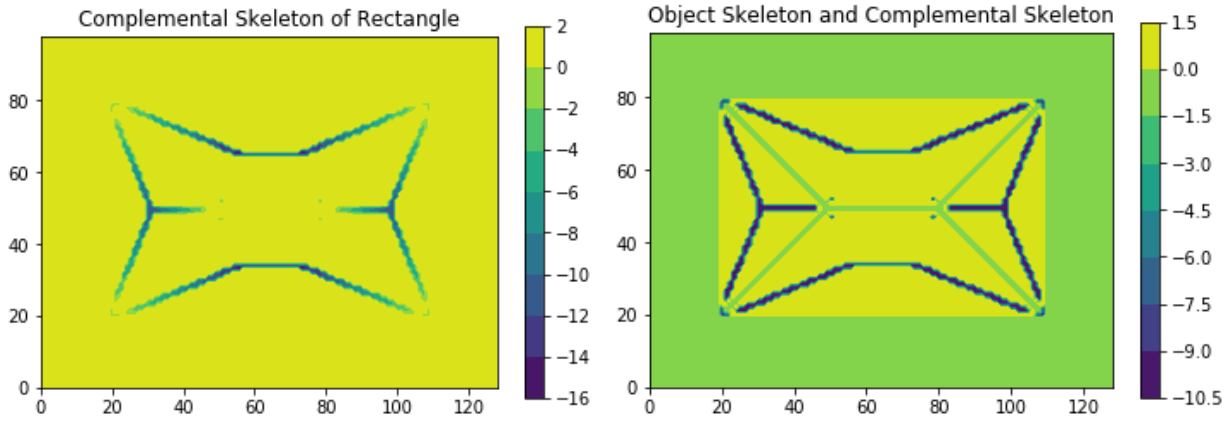


Figure 9. Left: Complemental Skeleton of Rectangle. Right: Complemental Skeletal Geometry of Rectangle (Blue) superimposed over Skeletal Geometry of Rectangle and Inverse of Rectangle (Green)

To find the complemental skeleton, we must take the initial voxelized model, remove every voxel that it shares with its Skeletal Geometry (performing Boolean **subtraction**), then use the resulting geometry as the input for our skeleton-finding algorithm. The output of this process is our complemental skeleton; the skeletal data of the complement of the original skeletal data.

2.5 Modifying the Complemental Skeleton

Modifying the Complemental Skeleton

The complemental skeletal data can be modified in all the same ways as the standard skeletal data. The Radial Field can be modified, and voxels can be added to or removed from the Skeletal Geometry of the complemental skeleton. The key concept to keep in mind is that the Skeletal Geometry of the complemental skeleton now represents opposing regions of the object, so small changes to one section of the complemental skeleton's Radial Field are unlikely to affect the opposite region of the model's surface when recreated. However, this new freedom does have limits. Reducing the magnitude of the Radial Field on the complemental skeleton may now produce voids within the object. Some examples of modifications are shown in Figure 10.

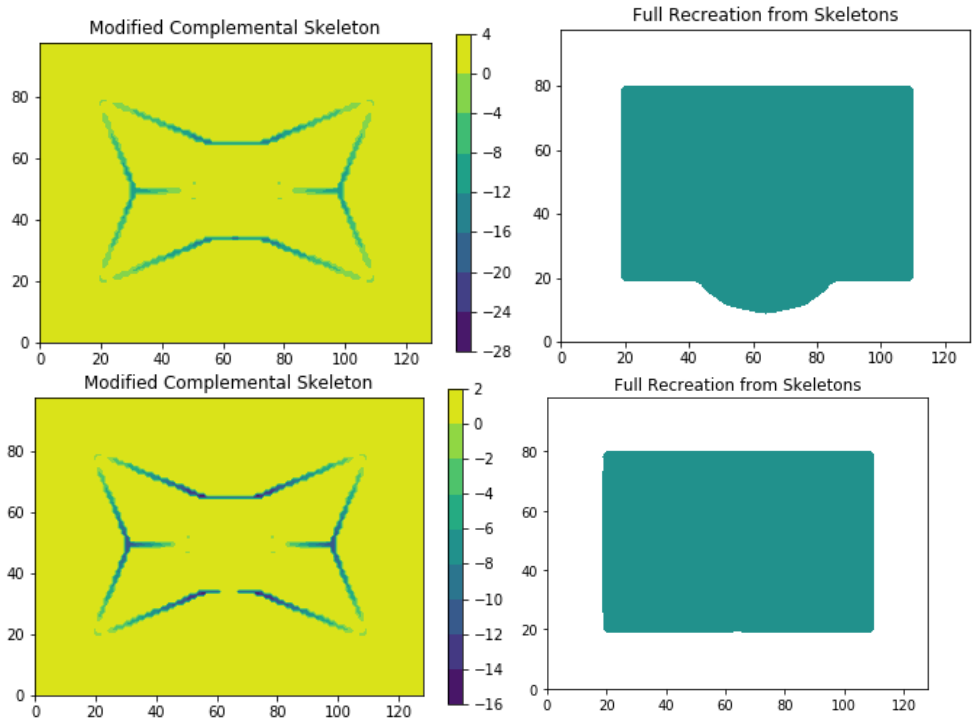


Figure 10. Left: Modified Radial Fields of the complemental skeletons. Right: Reconstructed models. Top: Several voxels on the lower portion have had their magnitudes increased. Bottom: Those voxels have been removed, to little effect.

2.6 Recreating the Object

After modifications have been made to the complemental Radial Field, the complemental skeletal data can then be used as the input for the same refleshing algorithm used for the original skeletal data. In continuous space, this process would result in creating the original However, in our discretized domain, repeating the same process as before leaves a void in the output object of the same shape as the initial Skeletal Geometry. This is seen on the left in Figure 11.

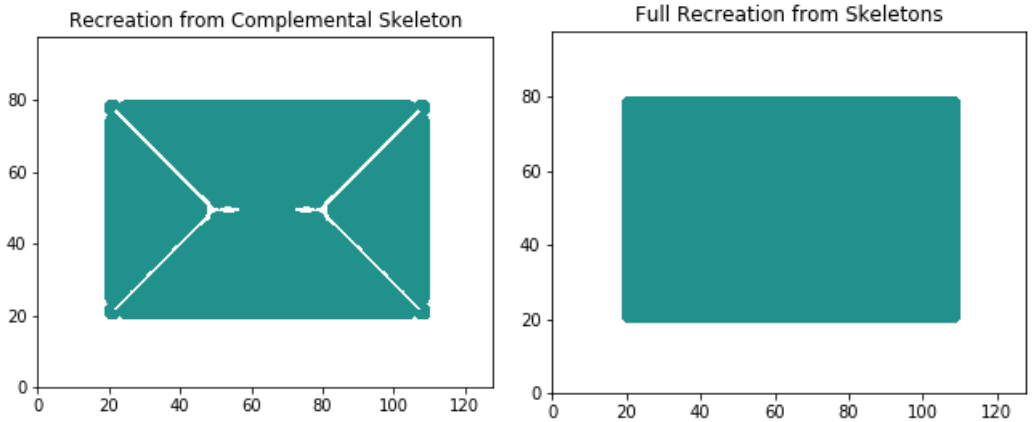


Figure 11. Left: Direct Reflesh of Complemental Skeletal Data. Right: Complete Model Reflesh.

Fortunately, this void can be repaired by simply performing a **union** operation between the original Skeletal Geometry with the geometry that results from using the **refleshing** algorithm on the complemental skeletal Data. The resulting object is seen on the right in Figure 11.

3. DISCUSSION

3.1 Strengths

The process of using the complemental skeleton provides skeletal modelers with the ability to make more changes to the resulting object without modifying the Skeletal Geometry. In particular, it is difficult to make changes to the Radial Field that won't result in changes to multiple regions of the object. Such changes traditionally require modifications to be made to the Skeletal Geometry in order to affect only one side of the object. However, simple changes made to the Radial Field of the complemental skeleton will only affect their corresponding regions, allowing for these asymmetric changes to be made without modifying the Skeletal Geometry. An example is shown in Figure 12.

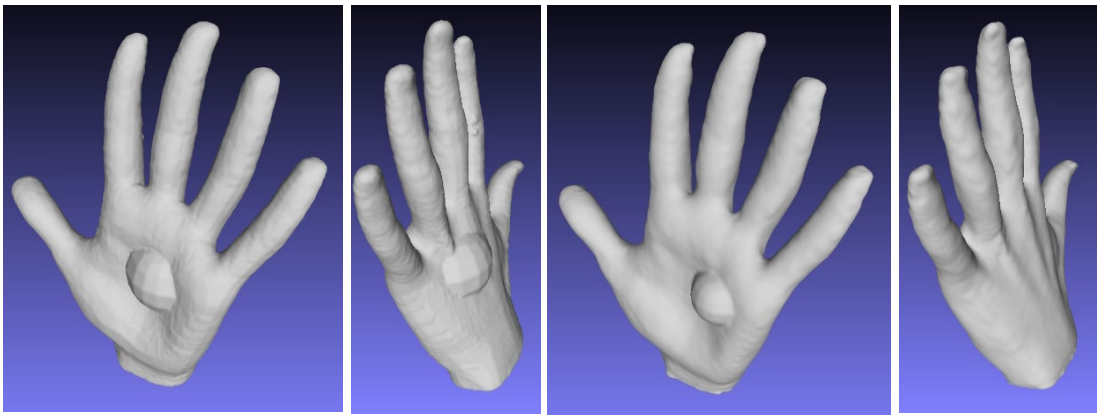


Figure 12. In both sets of images, a modification is made by altering the magnitude of a single voxel in the Radial Field. For the left two, the altered voxel is on the skeleton. For the right two, the altered voxel is on the complemental skeleton.

As seen in Figure 12, the resulting object has an area of higher thickness protruding from the palm, caused by the change to the magnitude of the Radial Field at that point. When the sphere is centered on the Skeletal Geometry, the extents of the sphere protrude from both sides of the object. When the sphere is centered on the complemental Skeletal Geometry, the extents of the sphere only protrude from one side of the object, leaving the other side untouched.

3.2 Current Limitations

The fundamental issue with this methodology is that it is discretized, which means that the Skeletal Geometry is made of voxels, each with a finite size. The true Skeletal Geometry of an object is infinitely thin, which does lead to more accurate results and recreations. The basic concept of the complemental skeleton could be created for continuous space, but that goes beyond the scope of this report.

As mentioned previously, the complemental skeleton works well for increasing local thicknesses, but has difficulties when reducing object thickness. As there is now an internal set

of faces that must be accounted for, reducing the thickness can lead to voids within the object. These voids still result in a valid object, but may not be intended. Additionally, the same conditions that apply to standard skeletal model manipulation apply to the complemental skeleton. Modifications made to complemental Skeleton Geometry are unlikely to result in an accurate Complemental Skeleton for that object.

On an algorithmic level, it is important to know of the limitations of the gradient function, as it is key to this method of locating the Skeletal Geometry. It is very sensitive to surface texture, meaning noise or fine details on the surface of the model can lead to extraneous voxels being classified as part of the Skeletal Geometry. A partial solution to this is to smooth the model before applying the gradient function, which reduces the noise and significantly lowers the number of false positives. However, this methodology does not discriminate between noise and intended detail, and can lead to thicker skeletal geometry than is necessary.

Finally, skeletal modeling in discrete space has limitations. In particular, models recreated from voxelized skeletal data tend to have corners rounded off. This is because the true medial surfaces of the model go completely up to any edges or corners, but the extents of these skeletal branches can be difficult to accurately capture in voxel space.

3.3 Future Improvement

The most impactful next step is to improve the algorithm used to generate the Skeletal Geometry. The gradient function works reasonably well, but its limitations require accommodations that are sometimes less than ideal. To minimize noise, the surface should be smoothed, which decreases the accuracy of the SDF. Increasing the grid size will produce thinner skeletons, allowing for a closer approximation of the true medial surfaces. The thinner the Skeletal Geometry, the more accurate the reflesh of the complemental skeletal data can be.

4. CONCLUSIONS

The inclusion of the complemental skeleton allows skeletal modeling to be used for asymmetric edits without necessitating cumbersome modifications to the Skeletal Geometry. To achieve similar results, new voxels would have to be added to the Skeletal Geometry, where the complemental skeleton allows the modeler to simply modify the Radial Field for similar results. This is not a replacement for standard skeletal modeling practices, but instead a new way to use some of the same basic techniques of skeletal modeling to make localized edits to 2D and 3D models. As the complemental skeleton is still a new construct, its full set of uses has yet to be explored.

5. REFERENCES

[1] Cornea, N.D., Silver, D., & Min, P., "Curve-Skeleton Properties, Applications and Algorithms." *IEEE Trans Vis Comput. Graph., 13*(3) (2007): pp.530-48

[2] Huang, H., Wu, S., Cohen-Or, D., Gong, M., Zhang, H., Li, G., & Chen, B., "L_1-Medial Skeleton of Point Cloud." *ACM Trans. Graph.*, *32*(4) (2013): pp.65-1

[3] Attali, D. & Montanvert, A., "Computing and Simplifying 2D and 3D Continuous Skeletons." *Computer Vision and Image Understanding 67*(3) (1997): pp.261-273

[4] Lee, Y. & Lee, K., "Computing the medial surface of a 3-D boundary representation model." *Advances in Engineering Software 28* (1997): pp.593-605

[5] Dey, T.K. & Zhao, W., "Approximate Medial Axis as a Voronoi Subcomplex." *Proceedings of the seventh ACM symposium on Solid Modeling and Applications* (2002): pp. 356-366
[6] Puig Puig, Anna, "Discrete Medial Axis Transform for Discrete Objects." (1997)
[7] Delamé, T., Roudet, C., & Faudot, D., "From A Medial Surface To A Mesh" *Eurographics Symposium on Geometry Processing 31*(5) (2012): pp.1637-46
[8] Blanding, R., Brooking, C., Ganter, M., & Storti, D., "A Skeletal-Based Solid Editor." *Proceedings of the fifth ACM symposium on Solid modeling and applications* (1999): pp. 141-50
[9] Zhang, Di, *A GPU Accelerated Signed Distance Voxel Modeling System*. 2016. University of Washington Seattle, PhD dissertation. <http://hdl.handle.net/1773/38175>
[10] Artec Space Spider, "Hand" *Artec3D* Santa Clara, CA, Accessed 1/7/2020 <www.artec3d.com/3d-models/hand>.
[11] Ensz, M., Storti, D., & Ganter, M., "Implicit Methods for Geometry Creation." *International Journal of Computational Geometry & Applications 8*(3) (1998): pp.509-36
[12] Rong, G., & Tan, T., "Jump Flooding in GPU with Applications to Voronoi Diagram and Distance Transform." *Symposium on Interactive 3D graphics and Games* (2006): pp.109-16
[13] van der Walt, S., Schönberger, J.L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T., & The scikit-image contributors, "scikit-image: Image processing in Python." PeerJ 2:e453 (2014)
[14] Williams, T., Langehennig, S., Ganter, M., & Storti, D., "Using Parallel Computing Techniques to Algorithmically Generate Voronoi Support and Infill Structures for 3D Printed Objects." *Solid Freeform Fabrication Symposium Proceedings 30* (2019): pp.1830-52

6. APPENDICIES

Several algorithms are mentioned in the Methods of Implementation that were not fully described in order to preserve the flow of the section. Here is a description of the process for each of the referenced algorithms in the following format:

Main Function (Inputs: A, B, C; Outputs: X, Y, Z):
PSEUDOCODE

Sub Function (Inputs: A, B, C; Outputs: X, Y, Z):
PSEUDOCODE

We are using Python as a baseline for pseudocode, meaning that the index of an array is dictated by square brackets (u[i,j,k] = element stored at index i,j,k), a colon selects all values within that index, (u[i,:,:] = 2D array), and the # symbol is used to dictate comments.

SDF (Inputs: Voxel Model of Object; Outputs: SDF of Object):

The Signed Distance Field function used for this paper uses the methodology described in [12], as it uses the Jump Flooding technique on a parallel computing platform. Detailed pseudocode can be found in [14], as the same code was used between the projects.

Boolean Operations (Inputs: Voxel Model 1, Voxel Model 2; Outputs: Voxel Model 3):

The Boolean operations (Union and subtraction) used in this paper follow the same methodology as described in [9] and [11]. Detailed pseudocode can be found in [14], as the same code was used between the projects.

Gradient (Inputs: SDF of Object; Outputs: Gradient Magnitude Field):

```
Gradient Magnitude Field = Copy of SDF of Object
Gradient Kernel (SDF of Object, Gradient Magnitude Field)
#Done in parallel, exact setup depends on the language/platform
Return Gradient Magnitude Field

Central Finite Difference (P, N, H):
#P = value one step forward from our coordinate
#N = value one step backward from our coordinate
#H = step size
return ((P-N)/(2*H))^2 #Standard central difference equation for first derivative

Gradient Kernel (Output, Input):
i, j, k = thread indices
dx = Central Finite Difference(Input[i-1,j,k], Input[i+1,j,k], 1)
dy = Central Finite Difference(Input[i,j-1,k], Input[i,j+1,k], 1)
dz = Central Finite Difference(Input[i,j,k-1], Input[i,j,k+1], 1)
#1D derivatives
dPxPy = Central Finite Difference(Input[i-1,j-1,k], Input[i+1,j+1,k], 2^(0.5))
dNxPy = Central Finite Difference(Input[i-1,j+1,k], Input[i+1,j-1,k], 2^(0.5))
dPyPz = Central Finite Difference(Input[i,j-1,k-1], Input[i,j+1,k+1], 2^(0.5))
dNyPz = Central Finite Difference(Input[i,j-1,k+1], Input[i,j+1,k-1], 2^(0.5))
dPxPz = Central Finite Difference(Input[i-1,j,k-1], Input[i+1,j,k+1], 2^(0.5))
dNxPz = Central Finite Difference(Input[i-1,j,k+1], Input[i+1,j,k-1], 2^(0.5))
#Diagonal (2D) derivatives
d1 = (dx+dy+dz)
d2 = (dPxPy+dNxPy+dz)
d3 = (dx+dPyPz+dNyPz)
d4 = (dPxPz+dy+dNxPz)
Output[i,j,k] = min(d1,d2,d3,d4)^(0.5)
#Finds the squares of the magnitudes, then compares them to find the minimum and
#takes the square root, storing the output in the corresponding cell of the output matrix.
```

Skeleton (Input: SDF of Model, Grad Magnitude Field, Grad Threshold, SDF Threshold; Output: Skeleton):

```
#Gradient threshold is usually 0.85, as a magnitude below this indicates that the gradient
#function would return an 'invalid' value.
    #SDF Threshold helps get rid of false positives, if the SDF value is greater than the
#Threshold (ie, positive or low magnitude negative), the voxel can be ignored with
#minimal impact on the resulting skeleton. Usually set to around -1 voxel lengths.
Skeleton = Copy of SDF of Model
```

```
    Skeleton Kernel(Skeleton, Grad Magnitude Field, Grad Threshold, SDF Threshold)
    #Done in parallel, exact setup depends on the language/platform
    Return Skeleton

Skeleton Kernel(Skeleton, Grad Field, Grad Threshold, SDF Threshold):
    i, j, k = thread indices
    if (skeleton[i,j,k] > SDF Threshold) or (Grad[i,j,k]>Grad Threshold):
            Skeleton[i,j,k] = 1
            #If the gradient is above the threshold, we are necessarily outside of the skeleton,
    #so any positive value would remove us from the object.
    #If the skeleton value at that point (currently set to match the SDF value) is above
    #the threshold, we are either outside the object, so any positive value maintains
    #that, or we are inside with a low magnitude and can be safely ignored.

Reflesh (Input: Skeleton, Iterations; Output: Recreated Object):
    Read = Skeleton
    Write = Skeleton
    For Iterations number of loops:
            Reflesh Kernel (Read, Write)
    #Done in parallel, exact setup depends on the language/platform
            Read, Write = Write, Read
            #Ensures that the input value isn't being overwritten while it's still being used,
    #then swaps the matrices around to make the loop easy to keep track of.
    Recreated Object = Write
    Return Recreated Object

Reflesh Kernel (Read, Write):
    i, j, k = thread indices
    For index from 0 to 26:
            checkPos = (i+Mod3(index//9)-1, j+Mod3(index//3)-1, k+Mod3(index)-1)
                    #Double backslash is integer division, divide then round down to nearest
integer.
                    #That line cycles through all coordinates in a 3x3 grid centered on i,j,k
                    Updated = Read[checkPos]+
            ((i-checkPos[0])^2+(j-checkPos[1])^2+(k-checkPos[2])^2)^(1/2)
            #Calculates the distance from the current cell to the checked cell, then adds the
            #result to the depth value at the checked point in Read.
            Write[i,j,k]=min(Write[i,j,k],Updated)
```