EFFICIENT TECHNIQUE FOR SIMULATING MULTIPLE LAYERS OF SOFT TISSUES USING CUDA BASED GPU

Jayasudha.K[1], Mohan.G.Kabadi[2]
[1]Visvesvaraya Technological University , [2]Presidency University
jayanataraja@gmail.com[1], mohankg@presidencyuniversity.in[2]

Abstract
With the introduction of VR(Virtual Reality) based simulators, surgeons keep learning these procedures that are different from traditional surgery procedures. In fact VR environments are promising medium to carry out training and practicing surgery techniques efficiently. This paper investigates the key component of simulation of multiple layers of 3D soft tissues of human skin. CUDA based GPU computing is adopted to speed up the simulation performance. The parallel computation is achieved using necessary data structures and algorithms. The performance evaluation of the model is done using vtkPython and CUDA programming language implementations. The comparative analysis of the models performance shows that there is a significant increase in speed up at a fraction of the cost with GPU equivalent to ten times the traditional CPU cores.
Index Terms: dermis, epidermis, mesh, soft tissues, subcutaneous.

I. INTRODUCTION

In recent years, the real time performance of a mesh model in surgical simulation applications requires costly algorithms. To increase the speed ability of such algorithms, many application developers explore parallelization techniques described for a mesh model with large number of nodes. GPU architecture is the newest generation architecture that eases the programmer with increased generality by providing tremendous memory bandwidth and computational power. This paper describes the comparative analysis of the multiple layers of soft tissue models performance used in virtual surgery applications

II. RELATED WORK

This paper presents few accounts of work done on GPU parallel computations previously. Many researchers have performed analyses on this area to gain more knowledge with different perspectives. Here are some of the related works presented in the field of parallelization.

A. Finite Element Modeling

Nonlinear finite element equations comprise of intense arithmetic computations makes it suitable for implementations on GPU with parallel computing platform. Taylor et al.[1][2] explains GPU scheme for FE (Finite Element) equations to achieve speed up gain for up to 16000 tetrahedral elements. Author Joldes et al.[3] uses both hexahedral and tetrahedral elements on CUDA based GPU for real time nonlinear FE computations to neurosurgical simulation, whereas the same author in paper[4] compares several approaches with different element types for FE algorithms on GPU using CUDA. Author Oldfield et al.[5] gives detail description of deep needle insertions by FEM into a soft tissue phantom. Comas et al.[6] describes simulation of cataract surgery model based on algorithm using GPU implementation and SOFA framework based on CUDA.
Huthwaite et al.[7] explains partitioning the mesh based on arrangement of nodes, implemented by accelerated FE algorithm on GPU. Mafi and sirouspour [8] explains simulation of large deformations and strains for dynamic nonlinear deformation analysis using GPU based implementation of FEM equations. Author Johnsen et al.[9] describes soft tissue simulation capabilities into biomedical applications using niftysim(an open source FE toolkit) based on GPU.

B. Haptic feedback forces

Haptic Feedback Force is a simulator, although virtual makes user to feel as if actual forces are exerted by the organs on virtual tools. Author De Pascale et al.[10] describes real time haptic rendering of complex objects deformed through multiple contacts where as author Altomonte et al.[11] explains the same using mass-spring model. Author Courtecuisse H et al. in paper[12] explains deformation with cutting using haptics based on corotational FEM formulation where as author Zerbato D et al. in paper[13] explains the same by taking into account the real tissue properties.

C. Ultrasound Simulations

Ultrasound Simulations are a framework of virtual reality based simulators. This stitches together the ultrasound volumes with different scan angles to generate ultrasound panorama. Kutter.O et al. in paper[14] presents ultrasound simulation and visualization in real time using ray based method. Reichl.T et al. in paper[15] explains simulation of ultrasound reflection, shadowing artifacts, speckle noise and radial image blurring. Rosenzweig. S et al. in paper [16] proposes ultrasonic research systems in cubic spline interpolation and Loupas 2D autocorrelation for displacement estimation.

Although the above mentioned papers uses standard techniques like FEM (Finite Element Modeling), haptic feedback, ultrasound simulations etc, but does not propose computations of hybrid model comprising of triangular and tetrahedral meshes. The next section describes the methodology used to build the hybrid model.

III. METHOD

According to Jean Christophe[17] human skin is made up of three layers: epidermis, dermis and subcutaneous layers as shown in fig.1. Epidermis is thin and soft layer with hair follicles or hair shaft. Dermis is the center most layer consisting of capillaries, sweat glands, touch receptors etc. Dermis layer is quite thick and it supports epidermis. Subcutaneous is bottom most layer connected to muscles and bones. From fig.1 it is clear that human skin consists of multiple layers. The thickness and behavior of each layer is different from each other. Hence it is required to develop a prototype of multiple layers of soft tissue model that mimics human skin. This is achieved using hybrid mesh model based on Delaunay triangulation concept and marching cube concept applied to cube data structures.

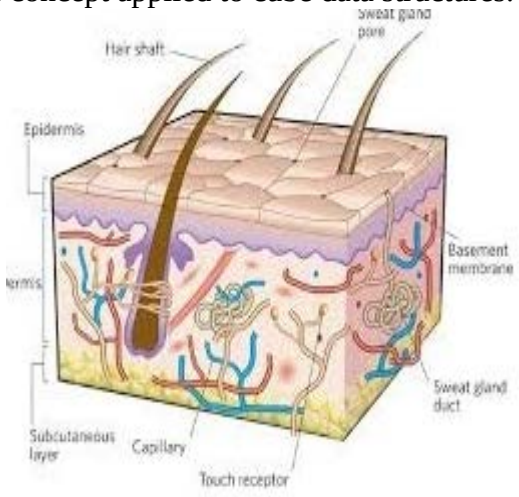

Fig.1. Human skin structure

The cube data structures of Delaunay triangulation concept contain vertex-list, edge-list and triangle-list shown in fig.2. The Delaunay cube algorithm is built based on this concept shown in fig.3. The cube data structures of marching cube concept contain vertex-list, edge-list, face-list and tetra-list as shown in fig.4. The marching cube algorithm is built based on this concept shown in fig.5. the next section describes the comparative study to analyze the model implementation techniques using VTKpython and CUDA programming languages.

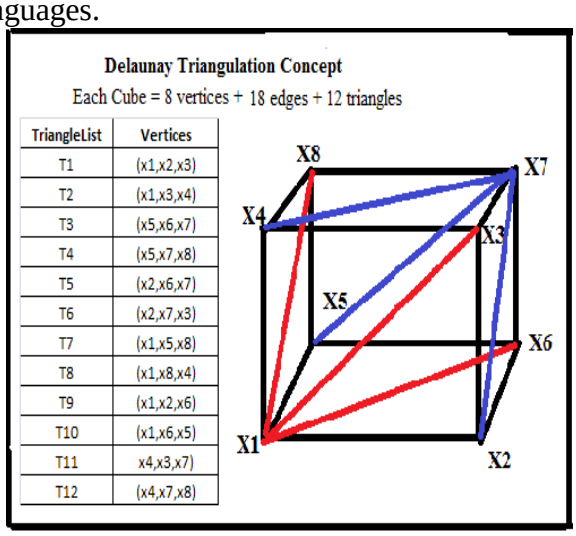

Fig.2. Delaunay Triangulation Concept

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-6, ISSUE-6, 2019
DOI: 10.21276/ijcesr.2019.6.6.25

Delaunay Cube Algorithm

```
for all vertices v1 to v8 from input set do
        if ( number of vertices == 3) then
        get triangle configuration
                if ( triangle is not DelaunayCube) then
                change the configuration
                Result = triangulatedCube ( v1 to v8 )
                end if
        end if
end for
```

Fig.3. Delaunay cube algorithm

Marching Cube Concept

Each Cube = 8 vertices + 5 tetras + 16 faces + 18 edges

TetraList	Vertices
T1	(x5,x6,x8,x1)
T2	(x6,x2,x1,x3)
T3	(x8,x1,x4,x3)
T4	(x6,x8,x3,x7)
T5	(x6,x8,x1,x3)

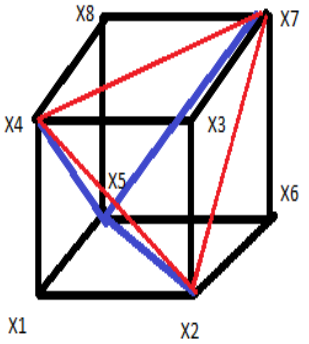

Fig.4. Marching Cube Concept

Marching Cube Algorithm

```
for all vertices v1 to v8 from input set do
        if ( number of vertices == 4) then
        get  tetra  configuration
                if (  tetra  is not Marching Cube) then
                change the configuration
                Result = tetrahedral Cube ( v1 to v8 )
                end if
        end if
end for
```

Fig.5. Marching cube algorithm

IV. VTKPYTHON IMPLEMENTATION

The Visualization Toolkit (VTK) is an open source, freely available software system for 3D computer graphics, image processing, and visualization. Although VTK is freely available, commercial support is available from Kit ware, Inc. [18]. VTK includes a C++ class library, and several interpreted interface layers including Tcl/ITk, Java, and Python. VTK has been implemented on nearly every Unix-based platform, PC's (Windows 95/98/NT/2000/XP) and Mac OSX Jaguar and later. VTK supports a wide variety of visualization algorithms including scalar, vector, tensor, texture, and volumetric methods and advanced modeling techniques like implicit modeling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation. The design and implementation of the library has been strongly influenced by object-oriented principles.

In VTK, applications can be written directly in C++, Tcl, Java, or Python. Python is a popular programming and scripting language with object-oriented programming concepts. Python supports multithreading and multiprocessing. It also provides GUI (Graphical User Interface) functionality. And therefore VTKpyhton is used for implementation. Fig.6. depicts the VTKPython architecture.

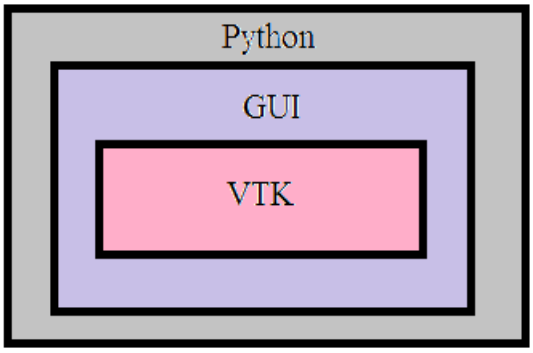

Fig.6. VTKPython Architecture

VTK compatible with python supports a wide variety of visualization algorithms including scalar, vector, tensor, texture, and volumetric methods and advanced modeling techniques like implicit modeling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation. The sample code for VTKpython is shown below:

```
void compute_python(float *a, float b, int N)
{
 for(int Idx=0;Idx<N;Idx++)
 a[Idx] = a[Idx] + b;
}

Void main()
{
 ….
 compute_python(a,b,N);

}
```

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-6, ISSUE-6, 2019
DOI: 10.21276/ijcesr.2019.6.6.25

The above code shows a simple example of adding an element to the array, till required size of array (N). It is normal programming function where array elements are incremented sequentially.

V. CUDA IMPLEMENTATION

The simulation performance of multiple layers of soft tissues is evaluated using NVIDIA's CUDA (Compute Unified Device Architecture) based on GPU. Parallel computing in CUDA takes place corresponding to the process flow steps shown in fig.7.

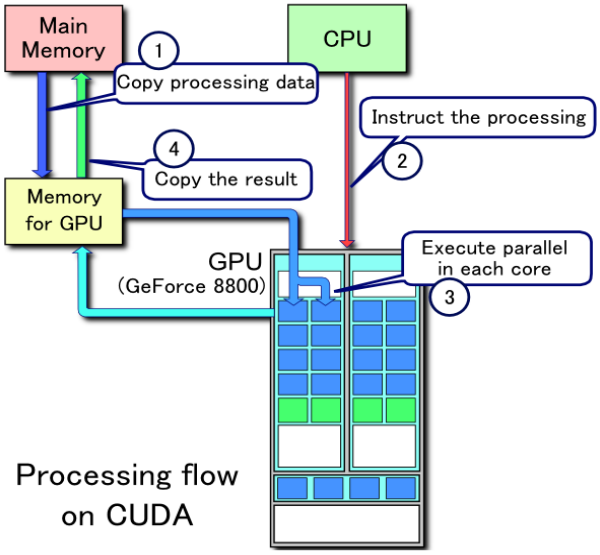

Fig.7.Parallel computing in CUDA

The above figure shows parallel computation in CUDA consists of the following steps:

1) The data to be processed is copied from CPU (main memory) to GPU memory.
2) CPU instructs GPU to perform parallel processing
3) Parallel execution takes place in GPU memory
4) Results are copied from GPU memory to CPU.

The key performance of CUDA[19] is that it utilizes large number of cores and hides global memory latency to perform massive multithreading. The optimization strategy for parallelization of algorithms using CUDA manages the following:

- Number of threads for multiprocessor
- Number of registers
- On-chip memory used per thread
- Global memory bandwidth

The sample code for CUDA parallel computation is shown below:

```
__global__void compute_gpu(float *a, float b, int N)
{
 int Idx = blockIdx.x * blockDim.x + threadIdx.x;
 if (Idx < N )
  a[Idx] = a[Idx] + b;
 }

Void main()
{
 ….
Dim3 dimBlock(blocksize);
Dim3 dimGrid(ceil (N/(float)blocksize));
compute_gpu<<<dimGrid,dimBlock>>>(a,b,N);
}
```

The above code shows the same example of adding an element to the array using CUDA parallel programming technique. CUDA memory contains many grids in which each grid is a collection of blocks and in turn each block is a collection of threads. The terms used in the above code are as follows:

__global__: syntax of routine executed on the GPU.
threadIdx : thread index within the block
blockIdx : block index within the grid
blockDim : dimension of block in threads
Dim3 :Three dimension (x,y,z)
dimBlock : Number of threads in block (1D or 2D or 3D)
dimGrid : Number of blocks in Grid (1D or 2D)
<<<…>>> : syntax for kernel calls

All the threads synchronize the execution parallel. Each thread will have an Id so as to compute memory address and take control decisions. However the differences between CUDA and VTKpython are shown in table I with respect to multithreading aspects. The next section debriefs the experimental results obtained.

Table I. Comparison of multithreading concepts

Sl.no	CUDA	VTKPython
1	Parallel execution of threads	Sequential execution of threads
2	Threads are light weighted	Threads are heavy weight
3	Instant switching of threads	No switching of threads
4	Uses 100's of threads to achieve	Uses few threads to achieve

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-6, ISSUE-6, 2019
DOI: 10.21276/ijcesr.2019.6.6.25

	efficiency	efficiency
5	Drastic reduction in memory bandwidth	No reduction in memory bandwidth
6	Uses shared memory concept	No shared memory concept
7	Thread co-operation is a powerful feature	Thread co-operation is less
8	Achieves fine grain parallelism	Achieves coarse grain parallelism

VI. RESULTS

The results presented are based on cubical data structures applied to hybrid mesh model. It is called hybrid mesh model because the epidermal layer is thin, hence it is made up of triangulated mesh model using Delaunay triangulation concept. Dermis layer is thick and 10 times more than epidermis[20], hence it is made up of tetrahedral mesh model using marching cube concept. The subcutaneous layer is also thick, but its thickness differs in different parts of the body. Hence it is also made up of tetrahedral mesh as they are dense and thick. Since the epidermis layer is thin, it modeled as single horizontal layer with twenty adjacent cubes placing next to each other. The dermis layer is quite thick and modeled as double horizontal layers with twenty adjacent cubes placing next to each other. The subcutaneous layer is not only thick but also hard as it is connected to muscles and bones. Therefore it is also modeled as double horizontal layers with twenty adjacent cubes placing next to each other. The visual appearance of all the three layers is shown in fig.8(a) and 8(b) in solid and wireframe models. The model represented in which all surfaces meet is called solid model. The model represented in which it contains lines and curves is called wireframe model.

The simulation results are analyzed for parallel computations performed on hybrid mesh model using VTKpython and CUDA with the following computing environments:

VTKPython computing environment:

Intel®core (TM) i3 CPU M380 @2.53Ghz
VTK 6.1.0 Python 64-bit(Intel) on windows7
Python compiler

CUDA computing environment:

Intel quad core Xeon processor (3.7 GHz), 2GB NVIDIA
Quadro K2000

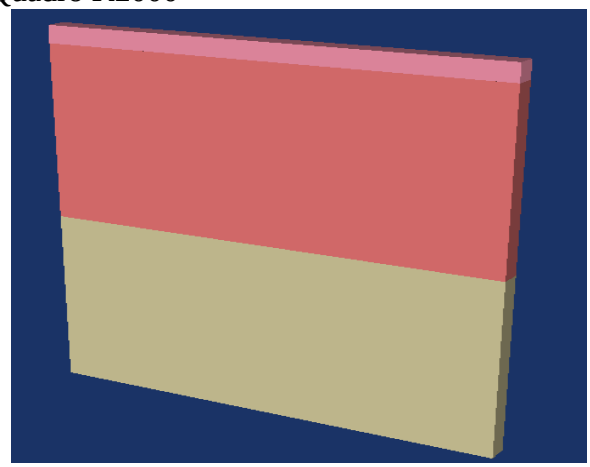

Fig.8(a). Multilayered skin in solid model (20X 1X 21)

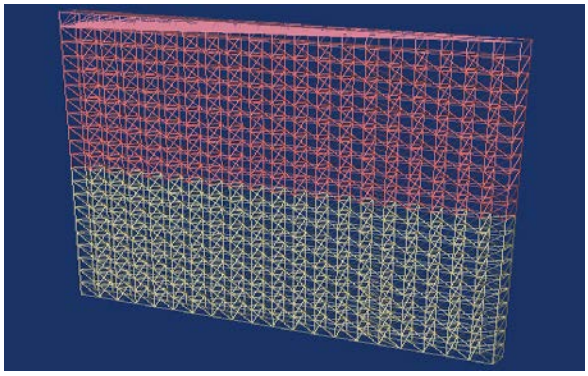

Fig.8(b). Multilayered skin in wireframe model (20X 1X 21)

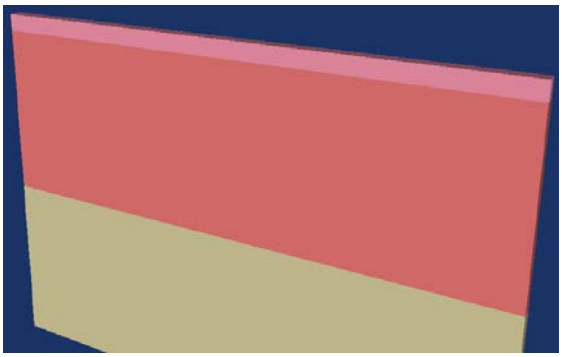

Fig.9(a). Multilayered skin in solid model (40X 1X 21)

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-6, ISSUE-6, 2019
DOI: 10.21276/ijcesr.2019.6.6.25

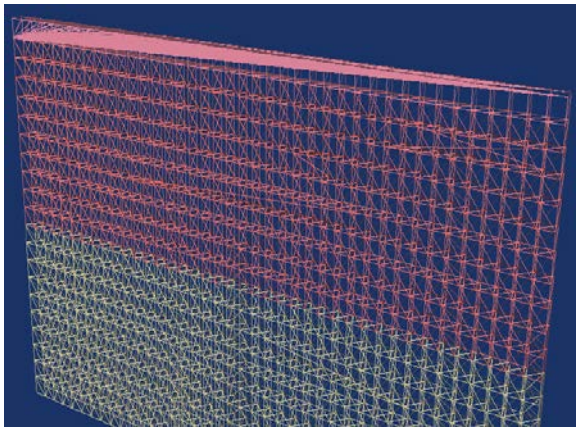

Fig.9(b). Multilayered skin in wireframe model (40X 1X 21)

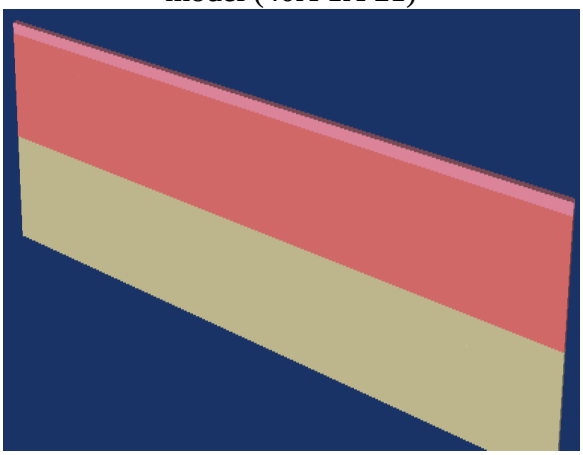

Fig.10(a). Multilayered skin in solid model (100X 1X 21)

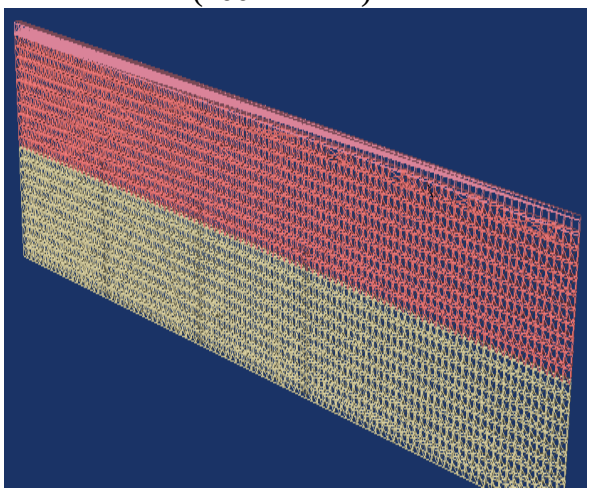

Fig.10(b). Multilayered skin in wireframe model (100X 1X 21)

Initially the hybrid mesh model's size is 20X1X21 by length, width and breadth as shown in fig.8(a) and (b). That means 20 cubes are placed horizontally to form the length of the mesh applicable to all the three layers. Here 1 represents the width of the mesh. Similarly 21 represents the breadth of the mesh including one layer of epidermis, 10 layers of dermis and 10 more layers of subcutaneous. Mesh size is gradually scaled horizontally to 40, 60, 100 etc up to 500 as shown in fig.9(a) and (b), fig.10(a) and (b) to study the models performance.

Table II. Execution time comparisons

Mesh Size (lXWXd)	Execution time(ms) using VTK Python	Execution time(ms) using NVIDIA's CUDA
20X1X21	0.4374	0.162
40X1X21	1.0202	0.324
60X1X21	1.2946	0.486
100X1X21	2.1648	0.81
140X1X21	3.2816	1.134
200X1X21	5.3322	1.62
300X1X21	9.1937	2.43
400X1X21	12.0256	3.24
500X1X21	15.1543	4.05

The comparative analysis of the multilayered soft tissue model is done using C and python programming languages [21]. This paper is an extended version in which it incorporates hybrid soft tissue model and also to increase the speed of execution at a faster rate CUDA based GPU is employed. The hybrid model's corresponding execution timings are noted down as shown in table II. The execution time is recorded in milliseconds. The simulation performance of the model is analyzed and compared using VTKpython and CUDA. It is observed that the simulation performance is enhanced using CUDA than VTKpython. This is because of CUDA consists of dedicated parallel computation framework. The same performance is presented using graphical chart shown in fig.11. It is understood from the graph that for smaller number of mesh size up to 200, there is no drastic variation in the performance. But as the mesh size increases say up to 500, the performance increases tremendously. This is because when the mesh size is small, the time taken for data processing is also small. As the mesh size increases, the time taken for data processing is also increases. From the graph it is clearly visible that NVIDIA's CUDA takes less execution time compared to VTKpython. The reason behind this is that CUDA GPU architecture utilizes data parallel co-processors that allows efficient mapping of computation to graphics hardware.

ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, VOLUME-6, ISSUE-6, 2019
DOI: 10.21276/ijcesr.2019.6.6.25

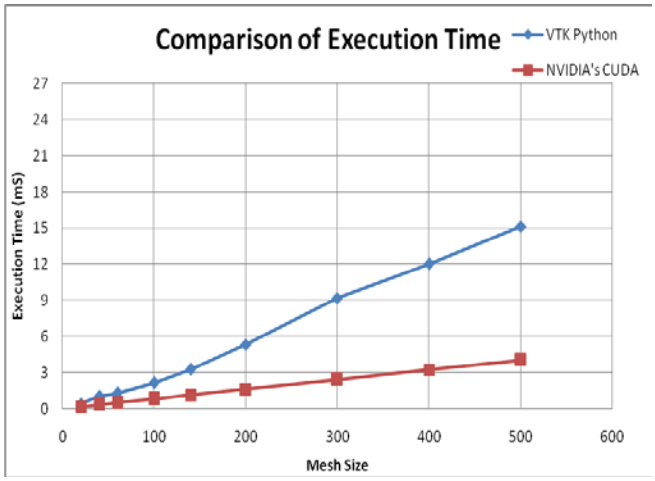

Fig.11. Comparison of execution times

VII. CONCLUSION

The presented framework simulates multilayered soft tissue model for human skin. It is the mesh model that mimics the human skin. It is modeled in an hybrid manner consisting of triangular and tetrahedral meshes. Performances of the mesh model are analyzed using VTKpython and CUDA. It can be inferred from the graph that CUDA enabled GPU enhances the efficiency of the model compared to VTKpython in terms of reducing the time consumption of models execution in the field of parallelization. It can be concluded that with CUDA implementation mesh can be generated easily with more number of elements, represent better geometry and also produces more accurate results.

REFERENCES

[1] Taylor ZA, Cheng M, Ourselin S. Real-time nonlinear finite element analysis for surgical simulation using graphics processing units. In International Conference on Medical Image Computing and Computer-Assisted Intervention 2007 Oct 29 (pp. 701-708). Springer, Berlin, Heidelberg.

[2] Taylor ZA, Cheng M, Ourselin S. High-speed nonlinear finite element analysis for surgical simulation using graphics processing units. IEEE transactions on medical imaging. 2008 May;27(5):650-63.

[3] Joldes GR, Wittek A, Miller K. Real-time nonlinear finite element computations on GPU–Application to neurosurgical simulation. Computer methods in applied mechanics and engineering. 2010 Dec 15;199(49-52):3305-14.

[4] Joldes GR, Wittek A, Miller K. Real-time nonlinear finite element computations on GPU: handling of different element types. In Computational biomechanics for medicine 2011 (pp. 73-80). Springer, New York, NY.

[5] Oldfield M, Dini D, Giordano G, Rodriguez y Baena F. Detailed finite element modeling of deep needle insertions into a soft tissue phantom using a cohesive approach. Computer methods in biomechanics and biomedical engineering. 2013 May 1;16(5):530-43.

[6] Comas O, Taylor ZA, Allard J, Ourselin S, Cotin S, Passenger J. Efficient nonlinear FEM for soft tissue modeling and its GPU implementation within the open source framework SOFA. In International Symposium on Biomedical Simulation 2008 Jul 7 (pp. 28-39). Springer, Berlin, Heidelberg.

[7] Huthwaite P. Accelerated finite element elastodynamic simulations using the GPU. Journal of Computational Physics. 2014 Jan 15;257:687-707.

[8] Mafi R, Sirouspour S. GPU-based acceleration of computations in nonlinear finite element deformation analysis. International journal for numerical methods in biomedical engineering. 2014 Mar 1;30(3):365-81.

[9] Johnsen SF, Taylor ZA, Clarkson MJ, Hipwell J, Modat M, Eiben B, Han L, Hu Y, Mertzanidou T, Hawkes DJ, Ourselin S. NiftySim: A GPU-based nonlinear finite element package for simulation of soft tissue biomechanics. International journal of computer assisted radiology and surgery. 2015 Jul 1;10(7):1077-95.

[10] De Pascale M, De Pascale G, Prattichizzo D, Barbagli F. A GPU-friendly method for haptic and graphic rendering of deformable objects. In Proceedings of Eurohaptics 2004 (Vol. 2004, pp. 44-51).

[11] Altomonte M, Zerbato D, Botturi D, Fiorini P. Simulation of deformable environment with haptic feedback on GPU. In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on 2008 Sep 22 (pp. 3959-3964). IEEE.

[12] Courtecuisse H, Jung H, Allard J, Duriez C, Lee DY, Cotin S. GPU-based real-time soft tissue deformation with cutting and haptic feedback. Progress in biophysics and molecular biology. 2010 Dec 1;103(2-3):159-68.

[13] Zerbato D, Baschirotto D, Baschirotto D, Botturi D, Fiorini P. GPU-based physical cut in interactive haptic simulations. International journal of computer assisted radiology and surgery. 2011 Mar 1;6(2):265-72.

[14] Kutter O, Shams R, Navab N. Visualization and GPU-accelerated simulation of medical ultrasound from CT images. Computer methods and programs in biomedicine. 2009 Jun 1;94(3):250-66.

[15] Reichl T, Passenger J, Acosta O, Salvado O. Ultrasound goes GPU: real-time simulation using CUDA. In Medical Imaging 2009: Visualization, Image-Guided Procedures, and Modeling 2009 Mar 13 (Vol. 7261, p. 726116). International Society for Optics and Photonics.

[16] Rosenzweig S, Palmeri M, Nightingale K. GPU-based real-time small displacement estimation with ultrasound. IEEE transactions on ultrasonics, ferroelectrics, and frequency control. 2011 Feb;58(2).

[17] Jean-Christophe Nebel, “Soft tissue modelling from 3D scanned data”, Department of Computing Science, University of Glasgow, G128QQ, Glasgow, UK., pp 85-97, 2000.

[18] http://www.vtk.org

[19] http://en.wikipedia.org/wiki/CUDA

[20] Genzer J, Groenewold J. Soft matter with hard skin: From skin wrinkles to templating and material characterization. Soft Matter. 2006;2(4):310-23.

[21] Jayasudha.K and Dr.K.G.Mohan,“ Realizing Multilayered Soft Tissue Model Using Multithreading in Python ”, International Journal of Engineering and Technology(IJET), (ISSN 2227-524X, Volume 7(3.12), pgn0.589-593, Issue 31, July 2018).