Deformation of 3D Object of Human Body Internal Organs Using Finite Element Method Approach Accelerated by GPU

Cakra Adipura Wicaksana[1,*] Wahyuni Martiningsih[1]

[1] *Department of Electrical Engineering, University of Sultan Ageng Tirtayasa, Cilegon, Indonesia*
[*] *Corresponding author. Email: cakraadipura@untirta.ac.id*

ABSTRACT
Few years ago, there are some research or publication about 3D simulation especially virtual surgery that still continuous growing, that is 3D deformable object. The 3D deformable object resembles or mimic some object. The purpose of the research is to continue or add previous research especially in term of deformable 3D object by using OpenGL and as well as startup especially in Indonesia in term of soft body simulation. To make object deform, one of the numeric approach methods would be used i.e. Finite Element Method. this approach is quite common or used by another researchers because it is stable, detail, but also quite heavy in modelling some objects. This approach would implement by using OpenGL GPU-based with eight kernels or functions. Result of this research show that this simulation of 3D of human body internal organs object produces the good FPS of simulation with 3.07 times faster rather than not using GPU.

Keywords: *Finite Element Method, Deformation, Game Engine, Simulation, GPU, Virtual Surgery*

1. INTRODUCTION

Another scientific fields like the medical field, using computers in their daily activity. A Simulator will be used by surgeons or other to skillful expertise and a surgeon to practice their ability before doing the actual patient. The simulator modeled on human organs body that resembles the original internal body organs, like liver, heart, lungs, pairs of kidneys, two eyes, teeth, and so on.

By using simulator, they can perform some variety of things like the interaction of touching the object being simulated, perform collision between two objects, slicing or cutting objects. Interaction with objects usually using haptic or controllers like razer hydra.

Today there have been some virtual simulator for doing surgery such as da Vinci Surgical System. Price is little expensive approximately 1,75 million USD according to [1] and [2] so that not all of many hospitals would be able to buy. it is important to have some breakthroughs to create or design some simulators that have fine quality with affordable price so that some hospitals or medical schools can have it so simulator can be useful.

Creating and designing a simulator with affordable price, has been done in previous studies, namely by [3] and [4], simulated 3D objects by using application of game engine like Unity and razer hydra act as controller with affordable price, while [3] would continue such that research into a deformation of 3D objects and cutting objects 3D simulation.

Seeing two studies that have been there before, this research can continue and improve the research, especially research conducted by [3] in terms of deformation of 3D objects and implementation if the simulated organs in the human body. Thus, this paper is only show and explain on deformation of the 3D object.

There have been two previous research related to this current research they are following:

1.1. Robotic Surgery Simulation with Low-Cost

Simulation of surgical robots already existed before, but the price is still quite expensive so the research in [3] makes and designs an application with some interactions that could be reached with affordable price. The main purpose of application is conducted by [3] which is with a low cost budget, it could effectively simulate the

control of an existing robot of surgical. Applications are made by [3] using applications that version of the game engine like unity and Razer Hydra as controller.

This research would use Razer Hydra controller with two controllers on right and left side, laptop, and software game engine like Unity. the application is made by [3] used two virtual arms as a clamp.

All Scene at unity game engine made by [3] includes some important models, as follows:

- Robot Arms or virtual arms have been equipped by clamp or pincer.
- The beads as the object moved.
- Board and pegboard that act as the attachment of beads.

1.2. Cutting in Deformable Bodies

To mimic a virtual surgical simulator, [3] create a module, which cutting 3D objects that have properties of deformation or deformation. Cutting an objects in field of virtual surgery simulation is very important features [5] because almost from the surgery requires cutting process.

In this cutting 3D objects, [4] continues to research conducted by [3] by adding features objects on the object deformation cuts a 3D object. In this study, the algorithm used is spring mass Method for deformation, and for creating the object on surface side into a volume mesh are used Delaunay Algorithm Tetrahedralization through TetGen applications that integrate with the Game Engine.

In research in [5] add some additional arms that act to cut the 3D object so that the user interface of the application is similar with research made by [5] it is just robot arms that is left side to hold the object to be cut or grabbing, right side acting as a cutter. On Unity scene there is an object that has box-shaped deformation properties.

Implementation made by [5] using tools like Razer Hydra as a controller, Unity Game Engine as its application, and one personal computer. In this application only some cubical box objects are simulated use two robotic arms.

Before simulate 3D object, we need a file named surface mesh converted into volumetric mesh. File types would be used, namely obj file or stl file. An integration between softawre named TetGen [6] with some software, TetGen functions of the application itself, which is to make the input surface mesh into a volumetric mesh using Delaunay Tetrahedralization Algorithm.

2. METHOD

There are many approach methods that had been implemented by many researchers. According to [7] there are four common methodologies for deformable modelling. The deformable object can be modeled by using the physically based model which defines the behavior of model from mathematical equations and laws of physics [8]. In the previous research, [4] use Spring Mass system as the approach method and in this research Finite Element Method Approach would be used. On the other hand, FEM is powerful, highly efficient, and versatile numerical [9].

2.1. Mass Spring System

It is one method to create and design 3D objects. It is quite simple compared with other methods like the Finite Element Method. In the Spring-Mass method, edges on each tetrahedral act as Spring. Each vertex of tetrahedral mass. One tetrahedral has four vertices, each vertex relates to other tetrahedral. So that each vertex can be influenced by the style of the vertices of the other tetrahedral.

In a study conducted by [4] Spring-Mas system they pleased to be a style that is obtained from each vertex then summed back in style at one vertex or thereafter referred to as Spring Neighbor. The goal is improving the stability of the geometry of mesh and to prevent instability of the structure geometry. Neighbor Spring can also be integrated with the force of gravity. Without a Neighbor's Spring, mesh would collapse because of the mass of the mesh itself.

2.2. Finite Element Method (Will be used)

This method is numerical technique for solving problems. It is described by partial differential equations or formulated as functional minimization according to [10]

According to [11] the entire step of deformable object by using FEM as follow:

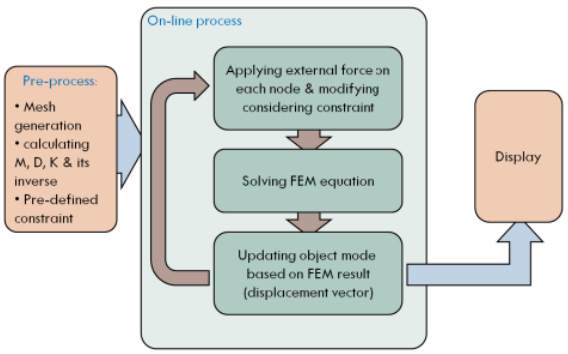

Figure 1 Entire Simulation of FEM [10]

In order to simulate an object, mimic the original shape, need to be added some of elastic and plastic force as mentioned in [12] and [13] beside external force as depicture on [10]. Until now, FEM is still undergoing further research like in [14]

Different from Mass Spring System, Finite Element Method have an objective is to get the last velocity of each node. After getting it, it should be adding to the function of time integration method that would be explained later.

As mention on [15], [10], [13], [12], and [10]. The external force is governed by this equation:

$$F_{ext} = M\ddot{u} + D\dot{u} + Ku \quad (1)$$

M is global mass matrix, D is damping matrix, K is the stiffness matrix, u is displacement vector of each node, finally 1st and 2nd derivate from u are velocity and acceleration, respectively. Where symbol D = αM + βK.

Stiffness matrix symbolled with symbol K from Ke that from entire element or tetrahedral in mesh object. The formula of Ke = Ve BeT C Be. Ke is stiffness matrix that have ordo 12x12.

Following is the explanation how to get Ve, Be, and C. C is called the material matrix, Be is called strain matrix which is constant and can be precomputed, and Ve is called volume of tetrahedral. Symbol E is elasticity modulus, therefore symbol ν for Poisson's ratio.

$$6V = \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{vmatrix} \quad (2)$$

$$a_1 = \begin{vmatrix} x_2 & x_3 & x_4 \\ y_2 & y_3 & y_4 \\ z_2 & z_3 & z_4 \end{vmatrix} \quad b_1 = -\begin{vmatrix} 1 & 1 & 1 \\ y_2 & y_3 & y_4 \\ z_2 & z_3 & z_4 \end{vmatrix} \quad (3)$$

$$c_1 = \begin{vmatrix} 1 & 1 & 1 \\ x_2 & x_3 & x_4 \\ z_2 & z_3 & z_4 \end{vmatrix} \quad d_1 = -\begin{vmatrix} 1 & 1 & 1 \\ x_2 & x_3 & x_4 \\ y_2 & y_3 & y_4 \end{vmatrix} \quad (4)$$

$$\boldsymbol{B}^e = \frac{1}{6V}\begin{bmatrix} b_1 & 0 & 0 & b_2 & 0 & 0 & b_3 & 0 & 0 & b_4 & 0 & 0 \\ 0 & c_1 & 0 & 0 & c_2 & 0 & 0 & c_3 & 0 & 0 & c_4 & 0 \\ 0 & 0 & d_1 & 0 & 0 & d_2 & 0 & 0 & d_3 & 0 & 0 & d_4 \\ c_1 & b_1 & 0 & c_2 & b_2 & 0 & c_3 & b_3 & 0 & c_4 & b_4 & 0 \\ 0 & d_1 & c_1 & 0 & d_2 & c_2 & 0 & d_3 & c_3 & 0 & d_4 & c_4 \\ d_1 & 0 & b_1 & d_2 & 0 & b_2 & d_3 & 0 & b_3 & d_4 & 0 & b_4 \end{bmatrix} \quad (5)$$

$$\boldsymbol{C} = \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix} \quad (6)$$

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}$$
$$\lambda = \frac{\nu E}{1-\nu^2}$$
$$\mu = \frac{E}{2(1+\nu)} \quad (7)$$

Here E is the elasticity modulus and ν is the Poisson's ratio. According to [12], the following are all algorithm to simulate deformable object:

forall *elements e compute* B_e, P_e, K_e, *from* x_0
Initialize x^0, v^0
$i=0$
loop
  forall *elements e compute* R_e *based on* x^i
  forall *elements e update* $\epsilon_{plastic,\ e}$
    $K = \sum_e R_e K_e R_e^{-1}$
    $f_0 = -\sum_e R_e K_e x_0$
    $f_{plastic} = \sum_e R_e P_e \epsilon_{plastic,\ e}$
    external forces f_{ext}
  solve $(M + \Delta t C + \Delta t^2 K)\, v^{i+1} = Mv_i - \Delta t(Kx^i + f_0 + f_{plastic} - f_{ext})$
    for v^{i+1}
  update $x^{i+1} = x^i + \Delta t v^{i+1}$
  endfor
  $i=i+1$
endloop

Figure 2 Algorithm FEM [11]

2.3. Method of Integration Time)

The dynamic system is aims to integrate over time domain to simulate the deformation. This could be done by many explicit or implicit time integration. Previous research that had been conducted by [4] is using verlet solver. The verlet solver is governed by this equation:

$$X_{j+1} = x_j - x_{(j-1)} + a_j dt^2 \quad (8)$$

Research on [16] and [15] use Implicit Euler form of Finite Element Method as follow:

$$(M + \Delta tC + \Delta t^2 K)\, v^{i+1} = Mv_j - \Delta t(Kx^j + f_0 + f_{plastic} - f_{ext}) \quad (9)$$

In this simulation we would modify a numerical integration method as follow so that it would be easy to implement in programming CUDA kernel.

$$(1 + \alpha\Delta t)\, M + (\beta\Delta t + \Delta t^2)\, v^{j+1} = \Delta t\, fj^{i+1} + Mv - \Delta t\, K\, u \quad (10)$$

2.4. FEM Calculation Process Using GPU

For heavy calculation processes such as matrix operations that have large orders of up to thousands, parallel programming is required. For parallel programming CUDA is used to do parallel programming using Graphics devices from NVIDIA and as references from [17]. The GPU calculation process is found in the process of calculating numerical integration methods and the process of finding solutions to the numerical integration equation because this process is quite heavy when run on the CPU.

We created eight kernels running on GPU. They are is a breakdown of equation 13 described as follows:

2.4.1. Kernel_Computegravityforce

This kernel functions to calculate the force of gravity. In this case, the gravitational force functions as an

external force which makes the object appear to be falling if not given a constraint.

```
__global__
void kernel_computegravityforce(float *vec_m, float *out, const int M)
{
   int tIndex = threadIdx.x + blockIdx.x * blockDim.x;
   if (tIndex <M) { if (tIndex % 3 == 1) {out[tIndex] = vec_m[tIndex] * - 9.8;} else {
out[tIndex] = 0;}}
}
```

Figure 3 Kernel of Compute Gravity Force

2.4.2. Kernel_Force_Mul_Deltatime

This kernel functions to calculate the multiplication process between external force vectors and delta time in the form of scalar vectors.

```
__global__
void kernel_vectormuldeltatime(float *vec, float *out, const int N, float scalar)
{
   int tIndex = blockIndeX.x*blockDim.x + threadIndeX.x;
   if (tIndex < N) {out[tIndex] = vec[tIndex] * scalar;}
}
```

Figure 4 Kernel of Compute Gravity Force

2.4.3. Kernel_Massa_Mul_Velocity

This kernel functions to calculate the multiplication process between mass vectors and velocity vectors.

```
__global__
void kernel_massa_mul_velocity(float *massa, float *vel, float *out, const int N)
{
    int tIndex = blockIndeX.x*blockDim.x + threadIndeX.x;
    if (tIndex < N) { out[tIndex] = massa[tIndex] * vel[tIndex];}
}
```

Figure 5 Kernel of Massa Mutiply by Velocity

2.4.4. Kernel_Displacement_Mul_Stiff

This kernel functions to calculate the multiplication process between the kernel transfer and matrix stiffness.

```
__global__
void kernel_displacement_mul_stiff
(float *displ, float *stiff, float *out, const int N, float scalar)
{
    int tIndex = threadIndeX.x + blockIndeX.x * blockDim.x;
    float Sum = 0;
    if (tIndex <N) { for (int i = 0; i < N; i++) { sum += displ[i] * stiff[(i*N) + tid];}
    out[tid] = sum * scalar;}
}
```

Figure 6 Kernel of Kernel Transfer Mutiply by stiffness

2.4.5. Kernel_Add_Right_Side

This kernel functions to calculate the summing process of kernel_force_mul_deltatime, kernel_massa_mul_velocity, and kernel_displacement_mul_stiff.

```
__global__
void kernel_add_right_side(float *vec1, float *vec2, float *vec3, float *out, const int N)
{
   int tIndex = blockIdx.x*blockDim.x + threadIndeX.x;if (tIndex < N)
    {
        out[tIndex] = vec1[tIndex] + vec2[tIndex] - vec3[tIndex];
    }
```

Figure 7 Kernel of Add Right Side

2.4.6. Kernel_New_U

This kernel functions to calculate and store new displacement vector values for later use in the next time step.

```
__global__
void kernel_new_u(float *vec_u, float *vec_newv, float *out, const int N, float scalar)
{
   int tIndex = blockIndeX.x*blockDim.x + threadIndeX.x;
    if (tIndex < N){ out[tIndex] = vec_u[tIndex] + (vec_newv[tIndex] * scalar);}
`
```

Figure 8 Kernel of New Displacement

2.4.7. Kernel_Mul_Left_And_Right_Side

This kernel functions to calculate solutions from numerical integration methods. The output of this kernel is in the form of a displacement vector as in this formula Ax = B, the value of x is what the solution would look for.

```
__global__
void kernel_mul_left_and_right_side(float *left, float *right, float *out, const int N)
{
   int tIndex = threadIndeX.x + blockIdx.x * blockDim.x; float sum = 0;
    if (tIndex <N)
    {
        for (int i = 0; i < N; i++){sum += right[i] * left[(i*N) + tIndex]; }
        out[tIndex] = sum ;
    }
```

Figure 9 Kernel of Calculations left and Right Side

2.4.8. Kernel_Set_Zero_Force

This kernel functions to calculate the value of the force vector to make it 0.

```
__global__
void kernel_set_zero_force(float *out_u, float *out_v, const int N)
{ int id = blockIdx.x*blockDim.x + threadIdx.x; if (id < N)
    { out_u[id] = 0; out_v[id] = 0; }
}
```

Figure 10 Kernel of Set to Zero

3. RESULT AND DISCUSSION

Before running simulation there are parameters that must be set to default as show on Table 1.

Table 1. 3D Simulation Parameters

No	Parameters	Symbol	Value
1	Young Modulus	E	10.000-1.000.000
2	Possion's Ratio	V	0.2-0.33
3	Density	P	1000
4	Spring Force	F spring	350-1000
5	Damper Force	F damper	0-450
6	Alpha	A	0.65
7	Betha	B	0.45
8	Gravity	A	9.8
9	Degree of Freedom (DOF)	-	3
10	Delta Time	Δt	0.02

In this simulation the value of alpha, acceleration of gravity, and DOF remain unchanged, while the other values are adjusted to the object to be simulated. Young Modulus value is greater, the object would be more rigid. The value of the spring force and the damper force are adjusted to the object to be simulated because these two variables function when the object hits the floor.

The following is a table that displays some results of performance of the approach with the Finite Element Method by using OpenGL.

Table 2. 3D Simulation Performance

No	Object (3D)	Vertex	Tetrahedon	Triangles	FPS CPU	FPS GPU
1	Rectangle	265	807	1837	18	59
2	Liver	965	4738	9894	10	59
3	Kidney	1027	4889	10276	11	60
4	Lungs	1535	5656	12490	15	60
5	Stomach	1437	6451	13659	20	59
6	A pair of kidneys	558	1991	4414	20	58

As we can see on table 2, we tested six number the three of dimension object. The Speed stated as (Frame Per Second) or FPS in m/s. It shown that using GPU faster than using CPU. It is because the eight kernels work well as planned.

The following are the results of a three-dimensional object simulation using OpenGL C++ on GPU CUDA v.7.5:

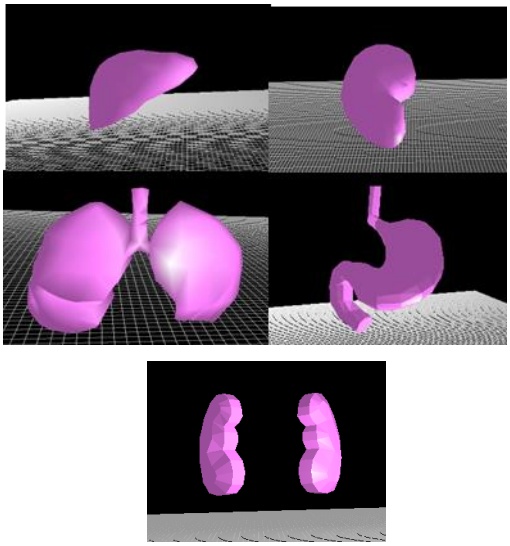

Figure 11 3D Object Using OpenGL and GPU

Performance in terms of speed, this Finite Element Method approach shows a low FPS, but when compared to previous methods such as Spring Mass shows better visualization. When the vertex and tetrahedron values of the object increases, the FPS speed would decrease. The simulation video as a whole, can be seen on YouTube with a link as follow:

https://www.youtube.com/channel/UCHJh62OUKoC2FXR4QeAD5gw.

Removing the gravitational force would make the object float. The object is dropped at a certain height and would hit the floor. In addition the object is also given a constraint at certain points so that the object would not move anywhere. For collisions with the floor, the spring and damper methods are used. So when the object has hit or through the floor, the spring and damper forces would apply. To make an object not move at a certain point is to make the force at that point zero and the stiffness matrix at that point zero.

4. CONCLUSION

The results obtained by using Finite Element Method that running on NVDIA GEFORCE 840M are more visible objects, if the average CPU FPS is around 10-20 FPS, while on the GPU ranges between 58-60 FPS. Object Rectangle is 2.27 faster; Object Liver is 4.90 faster; Object Kidney is 4.45 faster; Object Lungs is 3.00 faster; Object Stomach is 1.95 faster; Object a pair of kidneys is 1.90 faster; the Average is 3.07 times faster. Running on GPU is faster than running on CPU. It is because every vertex is computed on the same time and the eight kernel works well. Thus, programming with the GPU is one solution.

REFERENCES

[1] G. Aston, “Surgical Robots Worth the Investment,” 2012. [Online]. Available: http://www.hhnmag.com/.

[2] G. Ostrovsky, “POTUS Tries Out Da Vinci Surgical Robot,” 2009. [Online]. Available: http://www.medgadget.com.

[3] K. Grande, R. S. Jensen, M. Kraus, and M. Kibsgaard, “Low-cost Simulation of Robotic Surgery,” in Proceedings of the Virtual Reality International Conference: Laval Virtual, 2013, pp. 6:1--6:4, doi: 10.1145/2466816.2466823.

[4] M. Kibsgaard, K. K. Thomsen, and M. Kraus, “Cutting in Deformable Bodies,” Aalborg University, 2013.

[5] L. Jeřábková and T. Kuhlen, “Stable Cutting of Deformable Objects in Virtual Environments Using XFEM,” IEEE Comput. Graph. Appl., vol. 29, no. 2, pp. 61–71, 2009, doi: 10.1109/MCG.2009.32.

[6] H. Si, “A quality tetrahedral mesh generator and a 3d delaunay triangulator,” no. 13, 2010, p. 104.

[7] X. Wu and M. Downes, “Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes,” Comput. Graph. …, vol. 20, no. 3, pp. 349–358, 2001, doi: 10.1111/1467-8659.00527.

[8] S. Natsupakpong and M. C. Cavusoglu, “Comparison of Numerical Integration Methods for Simulation of Physically-Based Deformable Object Models in Surgical Simulation,” Electr. Eng., p. 6, 2009.

[9] D. Marinkovic and M. Zehn, “Survey of Finite Element Method-BasedReal-Time Simulations,” MDPI - Appl. Sci., vol. 9, no. 14, pp. 2076–3417, 2019, doi: https://doi.org/10.3390/app9142775.

[10] G. P. Nikishkov, “Introduction To the Finite Element Method,” in University of Aizu, 2004, pp. 1–45.

[11] H. Kim and W. K. Chung, “FEM based simulation of a deformable object,” 2012 9th Int. Conf. Ubiquitous Robot. Ambient Intell. URAI 2012, no. 2, pp. 496–497, 2012, doi: 10.1109/URAI.2012.6463050.

[12] M. Matthias, M. Gross, and M. Müller, “Interactive virtual materials,” Proc. Graph. Interface 2004, pp. 239–246, 2004.

[13] E. Lindquist, F. Lundell, and J. Peterson, “Deformation of Tetrahedral Meshes Using the Finite Element Method,” TSBK03-Advanced Game Program., pp. 1–16, 2011.

[14] J. M. Peña, A. LaTorre, and A. Jérusalem, “SoftFEM: The Soft Finite Element Method,” Int. J. Numer. Methods Eng., 2019, doi: 10.1002/nme.6029.

[15] Morten Bro-Nielsen, “Finite element modeling in surgery simulation,” Proc. IEEE, vol. 86, no. 3, pp. 490–503, 1998, doi: 10.1109/5.662874.

[16] M. Müller, J. Dorsey, and L. McMillan, “Stable Real-time Deformations,” in Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2002, pp. 49–54, doi: 10.1145/545261.545269.

[17] Brian Tuomanen, Hands-On GPU Programming with Python and CUDA: Explore high-performance parallel computing with CUDA. Birmingham: Packt Publishing, 2018.