SAT-1.405B-1-MIP-02

GPU Accelerated Planar Grid Generation for Cloth Simulation

Stanislav Kostadinov

Създаване на повърхнинни решетки за виртуални симулации на плат чрез графичен процесор

Станислав Костадинов

Abstract: *This paper presents an approach for GPU acceleration of planar grid generation with a rectangular topology. To accelerate computing analyses, graphics units (GPU) are widely used as a low-cost, low-memory algorithms and high-performance computing platforms. The algorithm proposed in this paper is executed over a graphics processor and its main purpose is fast generation of large grids for cloth simulation with less memory usage.*

Key words: *Cloth Simulation, Grid Generation, GPU acceleration.*

Резюме: *Докладът представя подход за създаване на повърхнинни решетки с правоъгълна топология посредством програми, предназначени за паралелно програмиране. С цел увеличаването на изчислителната мощ се използват програми, изпълнявани върху графичен процесор. Основните причини са широкото им разпространение, ниската цена и най-вече високата изчислителна мощ. Алгоритъмът, представен в този доклад, се изпълнява именно върху графичен процесор и основните му цели са малко заемана памет и бързо генериране на големи по размери решетки за целите на виртуалните симулации на плат.*

Ключови думи: *Виртуални Симулации на Плат, Създаване на Решетки, Графично ускорение*

INTRODUCTION

In the recent years many developers in the field of Computer Graphics work on algorithms executing over large amount of data. One of these algorithms is generation of large grids used in cloth simulation. It requires a rectangular topology grid to be generated on a cloth piece, so that the cloth simulation of the masses and springs can run [1] [2] [3]. One of the most precise solutions is to use direct optimization method [4], but as shown in the next sections, it is not very fast for large grids.

One of the newest research areas within numerical analysis is the parallel mesh generation. It is union between two scientific computing disciplines: computational geometry and parallel computing. Parallel mesh generation methods are using multiple processors or threads. One of the challenges in parallel mesh generation is to develop parallel meshing algorithm that generates meshes by using graphics units. The approach presented in this paper implements the improved length functional with non-uniform weights [5] and is executed over GPU. Detailed description of the solution is presented in the third section. The rest of the paper is organized as follows. The next section reviews previous work on grid generation. Section 3 describes in details the proposed approach. The last section concludes the paper and gives ideas for future work.

PREVIOUS WORK ON GRID GENERATION

As above mentioned one of the best quality meshes are generated with variational optimization approach by Castillo and Otto [6] [7]. They treat the grid generation problem as discrete from the very beginning. Several discrete functionals (length, area, orthogonal) are defined and their variation is minimized. They define the following generalized length functional (1). This produces linear systems, which can be solved much faster than for the elliptic generator. If only the length functional is used, then the x and y coordinates are

decoupled and two independent systems should be solved, one for x and one for y coordinates, which have the same system matrix. This results in faster generation, but minimization only of the length variation sometimes produces folded grids. That is why area is added too. However, x and y coordinates are coupled and the number of unknowns is increased twice.

$$F_L = \frac{1}{2}\sum_{i,j}\frac{(x_{i+1,j}-x_{i,j})^2}{A_{ij}^{\ 2}}+\frac{(y_{i+1,j}-y_{i,j})^2}{B_{ij}^{\ 2}}+\frac{(x_{i,j+1}-x_{i,j})^2}{C_{ij}^{\ 2}}+\frac{(y_{i,j+1}-y_{i,j})^2}{D_{ij}^{\ 2}} \quad (1)$$

Another approach is implements the improved length functional with non-uniform weights [5]. The idea of this approach is to use non-uniform weights, which are small in the center of the grid and gradually increase as moving away from the center and reach their maximum at the boundary curves. In this way the addends in functional (1), which correspond to edges that are closer to the boundary curves, will have greater influence in the sum and the edges will follow the shape of the boundary curves (2).

$$L_{ui} = 0.01+\frac{[0.5(u_{\max}-1)-i])^2}{u_{\max}}, i=1,...,u_{\max}$$
$$L_{vj} = 0.01+k\frac{[0.5(v_{\max}-1)-j]^2}{v_{\max}}, j=1,...,v_{\max} \quad (2)$$

The main idea of the approach, proposed in this work, is to provide a mechanism for execution of the improved length functional with non-uniform weights through different graphics units.

GPU ACCELERATED METHOD

This main idea of the GPU approach is to use graphics pipeline for parallel computation. There are two stages within the graphics unit pipeline where we can plug code: vertex shader and pixel (fragment) shader (Fig. 1). Pixel shader computes each pixel color and passes them to the frame buffer – buffer data shown on the display.

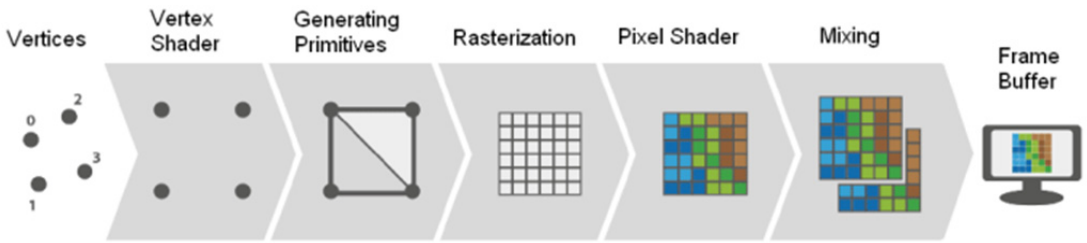

Fig. 1 Graphics unit pipeline

Frame buffers are two-dimensional array of pixels and they are similar as a structure to the topology grids. Each pixel contains three color components: red, green and blue color. Analogically, each 3D point contains three floating point components: x, y and z.

The approach passes the initial grid (it contains only grid contours) as floating point texture and accesses the frame buffer with the result grid which is the same structure – floating point two-dimensional array. The main idea is to use graphics parallel computation by substituting textures with grids and shader programs for manipulation of colors with the improved length functional algorithm. The initial grid contains the contours and initial values of the inner nodes. Using numerical analysis, algorithm solves the problem and find the results with a limited error – ε. The pixel shader implements algorithm that computes

each inner node (each node which is not part of the contour) of the grid by the neighbor nodes and equation 1 and 2. The output frame buffer is a result that partially satisfies the need of our cloth simulation purposes. Fragment shader is implemented with GLSL (OpenGL Shading Language) and each inner node is computed by using equation 3 and values from the neighbor nodes within the input two-dimensional array.

$$R(i,j)=\left[\frac{R(i-1,j)+R(i+1,j)}{L_u^2}+\frac{R(i,j-1)+R(i,j+1)}{L_v^2}\right]/\left(\frac{2}{L_u^2}+\frac{2}{L_v^2}\right) \quad (3)$$

Fig.2 presents the neighbors of two different nodes. The result value of the first node will be closer to the real one because there are two precise neighbor nodes (part of the contour) and the second one won't be so close to the expected result. Numeric analysis prescribes infinite iteration of the algorithm over the grid otherwise the result won't be absolutely accurate. This iterative approach identifies a cycle where each iteration is one execution of the graphics pipeline and input for each pass through the pipeline is the output from the previous one.

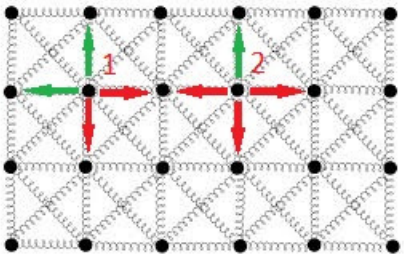

Fig. 2 Initial planar grid and node connections: 1 – node with two precise neighbour nodes, 2- node with one precise neighbour node

The result grid will be used in cloth simulation and that is why it is not necessary to generate the real grid nodes according to the CPU approach [5]. GPU approach has to satisfy the cloth simulation restrictions. The distance between the real grid nodes and the GPU algorithm nodes should be less or equal to specific length – ε (numerical analysis error). For the purposes of cloth simulation this error can be defines as half of a millimeter. The average human body is less than 2 meters, so the error can be defined as $\varepsilon = 2^{-12}$ (0.5mm/2m). So the infinite cycle can be limited to cycle that satisfies this error.

RESULTS

The initial values of the inner nodes of the grid can be all zeroes or a little optimization can be implemented in the first pass of the algorithm. The first GPU pipeline executing can initialize node to values that are closer to the real one. One example is the average x, y and z component of the contour grid – Fig. 3. Better solution which ensures less iterations is the algebraic interpolation of the nodes of the grid (4) – Fig. 4.

$$R(u,v)=\begin{bmatrix}1-u & u\end{bmatrix}\begin{bmatrix}R(0,v)\\R(1,v)\end{bmatrix}+\begin{bmatrix}R(u,0) & R(u,1)\end{bmatrix}\begin{bmatrix}1-v\\v\end{bmatrix}-\begin{bmatrix}1-u & u\end{bmatrix}\begin{bmatrix}R(0,0) & R(0,1)\\R(1,0) & R(1,1)\end{bmatrix}\begin{bmatrix}1-v\\v\end{bmatrix} \quad (4)$$

Experimental results from the average values and algebraic interpolation prove that the less distance between initial and result values are, the less iterations has to be passes. The example with average values is executed 943 times, and with algebraic interpolation values only 312 times.

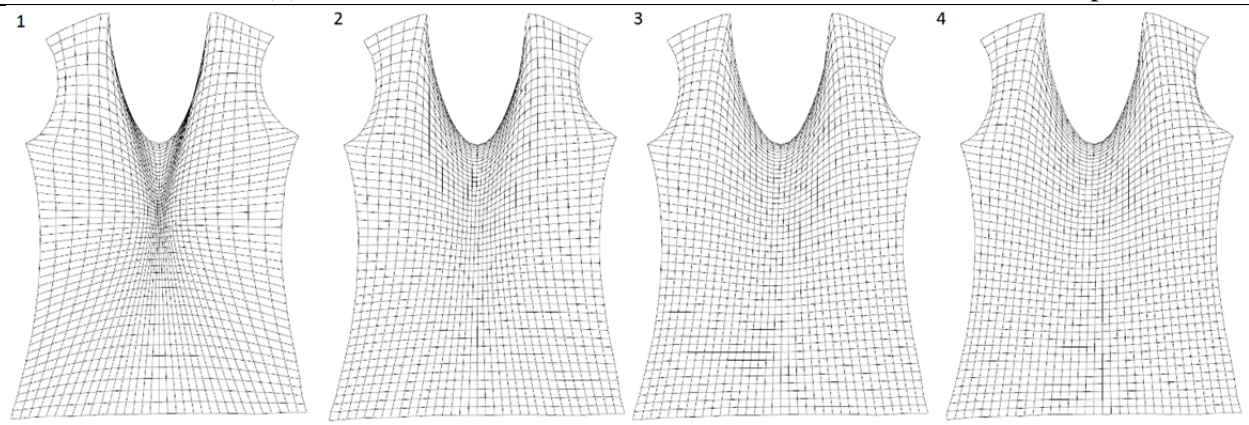

Fig. 3 Four stages of transformation of grid data with average x, y and z component

Test results prove that GPU acceleration approach is faster for grids bigger than 900 nodes (Fig. 5). The time for calculation of the grid with the CPU approach grows exponentially and that's why it becomes useless for grids with huge number of nodes. The required memory for the CPU approach grows exponentially, too. For example, solving the problem with 1024x1024 nodes requires 1024x1024x1024x1024 points and such RAM size (few terabytes) are inconsistent with the personal computers and other devices like smartphones and tablets. Current approach requires memory enough to collect the data about the grid (1024x1024 points).

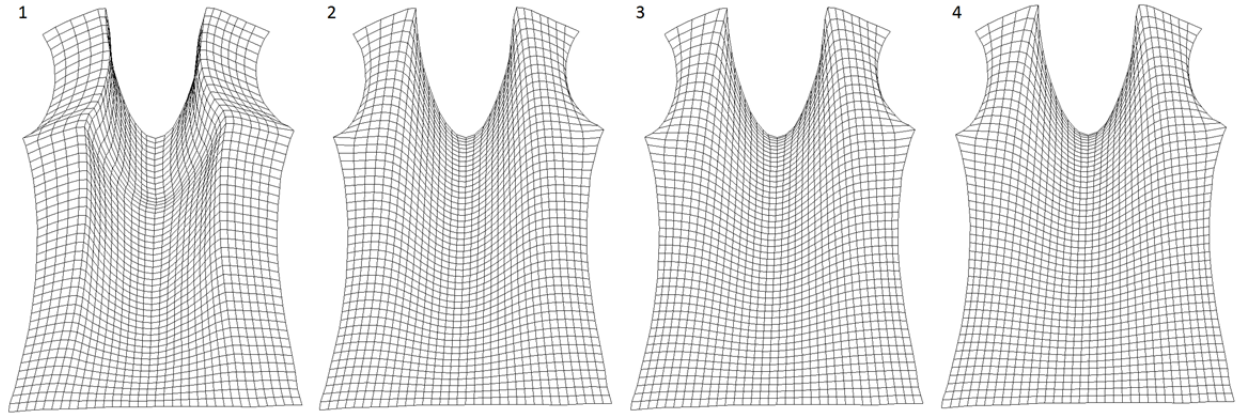

Fig. 4 Four stages of transformation of grid data with algebraic interpolation of the nodes

On the other side, GPU algorithm provides powerful tool for calculation of huge grids. Test ran on personal computers for grids with size 1024x1024 nodes end for a less than a minute. The limitation of this approach comes from the maximum size of the frame buffer. For example, 95% of all the devices have graphics unit with frame buffer bigger of equal to 4096x4096 (1% have unit with frame buffer 16384x16384). All of them have memory bigger than the required one (the frame buffers 16384x16384 require less than 1 GB RAM).

The input of the algorithm is implemented with Java code but it can be done with any other programming language. It can be implemented for every kind of device (PC, Smartphone, tablet, etc.) that has graphics unit. Tests are executed on personal computer with this hardware: CPU Intel i7-4720HQ 2.60 GHz, RAM 8 GB, GPU NVIDIA GeForce 940M.

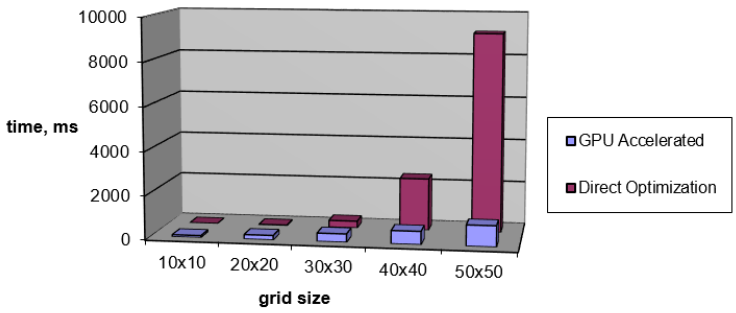

Fig. 5 Comparison between direct optimization algorithm and GPU accelerated

CONCLUSIONS AND FUTURE WORK

This paper presented an approach for GPU accelerated planar grid generation. The following conclusions can be drawn:

- current approach is faster for grids with more than 900 knots
- GPU algorithm requires minimal amount of memory
- GPU algorithm can be executed with large grids as input over every kind of device that has graphics unit

This approach can be extended and it can generate planar grids with different non-uniform weights for each node. This can be done by simply generating the same size texture as the grid with four weights for each node (instead of red, green, blue and alpha component of the texture). This powerful algorithm can be used for calculation of other problems which are using numerical methods for solving partial differential equations. Such problem is heat transfer within non-uniform environment, etc.

REFERENCES

[1] Provot X., Deformation constraints in a mass-spring model to describe rigid cloth behaviour, Proceedings of Graphics Interface, 1995, 141-155.

[2] Vassilev T., Spanlang B., Chrysanthou Y., Fast Cloth Animation on Walking Avatars, Computer Graphics Forum, 2001, Vol. 3, 260-267.

[3] Vassilev T.I., Comparison of Several Parallel API for Cloth Modelling On Modern GPUs, Proceedings of 11th International Conference CompSysTech 2010, Sofia, 2010.

[4] Smith E., Algebraic Grid Generation, Numerical Grid Generation, Vol. 137, 1982.

[5] Vassilev T.I., Kostadinov S., Planar Grid Generation for Simulation and Visualization, Proceedings of the Bulgarian Academy of Sciences, 2014, Vol. 8, 1061-1068.

[6] Castillo E., Otto J., A Practical Guide to Direct Optimization for Planar Grid Generation, Computers and Mathematics with Applications, 1999, 123-156.

[7] Castillo E., A discrete variational grid generation method, SIAM Journal on Scientific and Statistical Computing, Vol. 12, 1991, 454-468.

About the author:

Stanislav Dimchev Kostadinov, PhD, senior developer at ForkPoint, phone: 0898 237753, e-mail: sdkostadinov@uni-ruse.bg